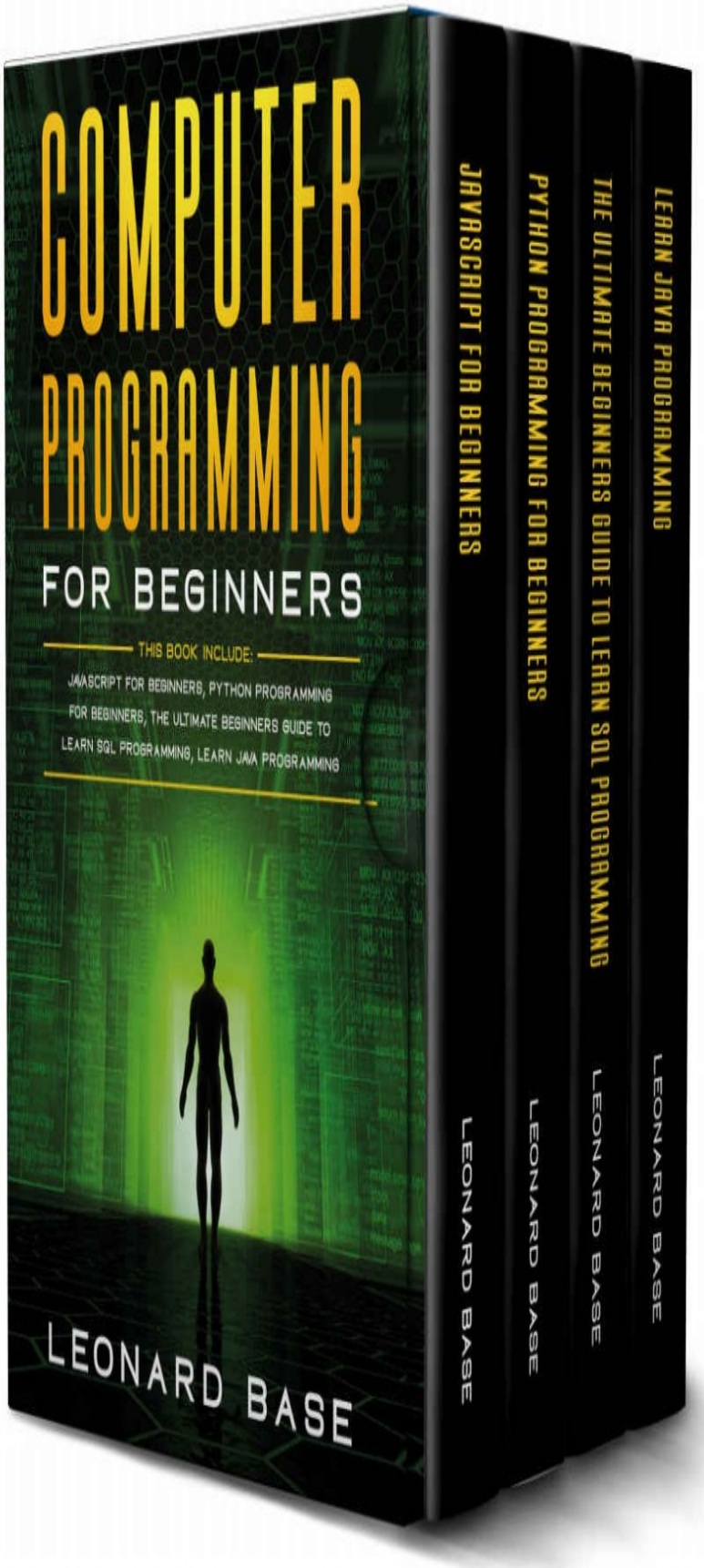

COMPUTER PROGRAMMING
FOR BEGINNERS
THIS BOOK INCLUDE:
JAVASCRIPT FOR BEGINNERS, PYTHON PROGRAMMING FOR BEGINNERS, THE ULTIMATE BEGINNERS GUIDE TO LEARN SQL PROGRAMMING, LEARN JAVA PROGRAMMING
LEONARD BASE
JAVASCRIPT FOR BEGINNERS
LEONARD BASE
PYTHON PROGRAMMING FOR BEGINNERS
LEONARD BASE
THE ULTIMATE BEGINNERS GUIDE TO LEARN SQL PROGRAMMING
LEONARD BASE
LEARN JAVA PROGRAMMING
LEONARD BASE

COMPUTER PROGRAMMING FOR BEGINNERS

4 Manuscript: JavaScript for Beginners, Python Programming for Beginners, The Ultimate Beginners Guide to Learn SQL Programming, Learn Java Programming

By

Leonard Base

Javascript for beginners:

The complete modern guide to start learn quickly and easily Javascript language. Coding and program with tips and tricks.

Python programming for beginners:

The ultimate crash course in python programming. A comprehensive guide to mastering the powerful programming language and learn machine learning.

The ultimate beginners guide to learn SQL programming:

A smart step by step guide to learn SQL database and server. How to building an advanced and elite level in SQL.

Learn Java programming:

A definitive crash course for beginners to learn java fast. Secrets, Tips and tricks to programming with Java Code and the fundamentals to creating your first program.

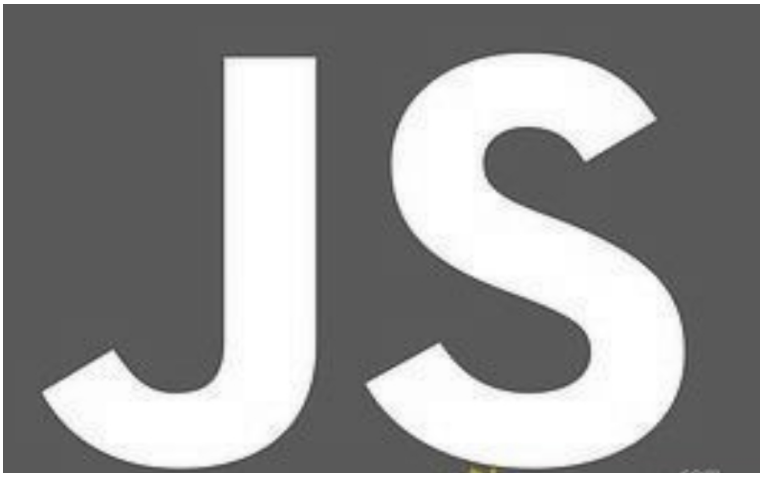
Table of Contents

MAKING OUR PROGRAM INTERACTIVE

INTRODUCTION TO JAVA

JAVASCRIPT FOR

BEGINNERS:



The Complete Modern Guide To Start Learn Quickly And Easily Javascript Language. Coding And Program With Tips And Tricks

Introduction

There are plenty of books on this subject on the market, thanks again for choosing this one! Every effort was made to ensure it is full of as much useful information as possible, please enjoy!

What is JavaScript?

JavaScript is an interpreted programming language, so it is not necessary to compile the programs to execute them. In other words, programs written with JavaScript can be tested directly in any browser without the need for intermediate processes.

Despite its name, JavaScript has no direct relationship with the Java programming language.

How to include JavaScript in XHTML documents

The integration of JavaScript and XHTML is very flexible since there are at least three ways to include JavaScript code in web pages.

Include JavaScript in the same XHTML document

The JavaScript code is enclosed between <script> tags and is included anywhere in the document. Although it is correct to include any block of code in any area of the page, it is recommended to define the JavaScript code within the header of the document (within the <head> tag).

Example of JavaScript code in the document itself </title>

<script type = "text / javascript"> alert ("A test message");

</script>

</head>

<body>

 A paragraph of text.

</body>

</html>

In order for the resulting XHTML page to be valid, it is necessary to add the type attribute to the < script>. The values included in the type attribute are standardized and in the case of JavaScript, the correct value is text / javascript.

This method is used when defining a small block of code or when you want to include specific instructions in a specific HTML document that complete the instructions and functions that are included by default in all documents on the website.

The main drawback is that if you want to make a modification to the code block, it is necessary to modify all the pages that include that same block of JavaScript code.

Define JavaScript in an external file

JavaScript instructions can be included in an external JavaScript file that XHTML documents link using the <script> tag. You can create all the necessary JavaScript files and each XHTML document can link as many JavaScript files as you need.

In addition to the type attribute, this method requires defining the src attribute, which indicates the URL corresponding to the JavaScript file to be linked. Each <script> tag can only link a single file, but on the same page you can include as many <script> tags as necessary.

JavaScript files are normal text documents with the extension .js, which can be created with any text editor such as Notepad, Wordpad, EmEditor, UltraEdit, Vi, etc.

The main advantage of linking an external JavaScript file is that the XHTML code of the page is simplified, that the same JavaScript code can be reused on all pages of the website and that any modification made to the JavaScript file is immediately reflected in all the XHTML pages that link it.

Include JavaScript in XHTML elements

This last method is the least used, since it consists of including JavaScript pieces within the XHTML code of the page:

<! DOCTYPE html PUBLIC "- // W3C // DTD XHTML 1.0 Transitional // EN" "http://www.w3.org/TR/ xhtml1 / DTD / xhtml1-transitional.dtd">

<html xmlns = "http://www.w3.org/1999/xhtml">

<head>

<meta http -equiv = "Content-Type" content = "text / html; charset = iso-8859-1" />

<title> Example of JavaScript code in the document itself </title>

</head>

<body>

 A paragraph of text.

</body>

</html>

The biggest drawback of this method is that it unnecessarily soils the page's XHTML code and complicates maintenance of the JavaScript code. In general, this method is only used to define some events and in some other special cases, as will be seen later.

Noscript Tag

Some browsers do not have full JavaScript support, other browsers allow you to partially block it and even some users completely block the use of JavaScript because they believe they are browsing more securely.

In these cases, it is common that if the web page requires JavaScript for its correct operation, a warning message to the user is included indicating that you should activate JavaScript to fully enjoy the page. The following example shows a JavaScript-based web page when accessed with JavaScript enabled and when accessed with JavaScript completely disabled.

The following code shows an example of using the <noscript> tag:

<head> ... </head>

<body>

<noscript>

 Welcome to My Site

 The page you are viewing requires the use of JavaScript for its operation.

If you have intentionally disabled it, please enable it again.

</noscript>

</body>

The <noscript> tag must be included inside the <body> tag (usually included at the beginning of <body >). The message that shows <noscript> can include any XHTML element or tag.

Basic Glossary

Script: each of the programs, applications or pieces of code created with the JavaScript programming language. A few lines of code form a script and a file of thousands of lines of JavaScript is also considered a script. Sometimes it is translated into Spanish directly as "escribir", although script is a more appropriate and commonly accepted word.

Sentence: each of the instructions that form a script.

Reserved words: these are the words (in English) that are used to construct JavaScript statements and therefore cannot be used freely.

Syntax

The syntax of a programming language is defined as the set of rules that must be followed when writing the source code of the programs to be considered correct for that programming language.

The basic rules that define JavaScript syntax are as follows:

Blank spaces and new lines are not taken into account: as is the case with XHTML, the JavaScript interpreter ignores any remaining

blank space, so the code can be properly sorted to better understand it (tabulating the lines, adding spaces, creating new lines, etc.)

The case is case sensitive: as is the case with the syntax of the XHTML tags and elements. However, if an XHTML page is used interchangeably, the page is displayed correctly, the only problem being the non-validation of the page. On the other hand, if JavaScript is exchanging upper and lower case letters, the script does not work.

The type of the variables is not defined: when creating a variable, it is not necessary to indicate the type of data that will be stored. In this way, the same variable can store different types of data during script execution.

It is not necessary to end each sentence with the semicolon character (;): in most programming languages, it is mandatory to finish each sentence with the character; Although JavaScript does not require you to do so, it is convenient to follow the tradition of ending each sentence with the semicolon character (;).

Comments can be included: comments are used to add information in the source code of the program. Although the content of the comments is not displayed on the screen, it is necessary to take precautions on the information included in the comments.

JavaScript defines two types of comments: those of a single line and those that occupy several lines.

Example of a single line comment:

// below is an alert message ("test message");

Single line comments are defined by adding two slashes (//) at the beginning of the line.

Example of multi-line comment:

/ * Multi-line comments are very useful when you need to include enough information in comments * / alert ("test message");

Multiline comments are defined by enclosing the text of the comment between the symbols / * and * /.

Possibilities And Limitations

Since its inception, JavaScript has always been used massively by most Internet sites. The appearance of Flash diminished its popularity since Flash allowed to perform some actions impossible to carry out through JavaScript.

However, the appearance of AJAX applications programmed with JavaScript has returned an unparalleled popularity within web programming languages.

As for the limitations, JavaScript was designed to run in a very limited environment that would allow users to rely on the execution of the scripts.

In this way, JavaScript scripts cannot communicate with resources that do not belong to the same domain from which the script was downloaded. Nor can scripts close windows that have not opened those same scripts. The windows that are created cannot be too small or too large or placed outside the user's view (although the specific details depend on each browser).

In addition, scripts cannot access the files of the user's computer (neither in read mode nor in write mode) and cannot read or modify browser preferences.

Finally, if the execution of a script lasts too long (for example due to a programming error), the browser informs the user that a script is consuming too many resources and gives the possibility to stop its execution.

In spite of everything, there are alternatives to be able to skip some of the above limitations. The most used and known alternative is to digitally sign the script and ask the user for permission to perform these actions.

Javascript And Browsers

The most modern browsers currently available include JavaScript support up to the version corresponding to the third edition of the ECMA-262 standard. The biggest difference lies in the dialect used

since while Internet Explorer uses JScript, the rest of browsers (Firefox, Opera, Safari, Konqueror) use JavaScript.

Javascript In Other Environments

The unparalleled popularity of JavaScript as a web application programming language has been extended to other applications and other non-web related environments. Tools like Adobe Acrobat allow you to include JavaScript code in PDF files. Other Adobe tools such as Flash and Flex use ActionScript, a dialect of the same JavaScript standard. Photoshop allows you to make small scripts using JavaScript and Java version 6 includes a new package (called javax.script) that allows you to integrate both languages.

Chapter 1 –

Why Javascript?

Why Learn Javascript?

Here are a few points to consider:

It is the programming language of web browsers (all the most important ones support it and have it activated by default: Firefox, Chrome, IE, Opera, Safari ...), which make it the most popular language on the Internet. There are other options, but they require installation and activation through plugins or only work in specific browsers.

There is a real competition between browsers to optimize their engines and provide better support for JavaScript and for your code to run faster. JavaScript is therefore increasingly stable and has better performance.

It is fully integrated with HTML and CSS. Each plays a definite role in a web page: HTML is the markup language to create the structure, CSS the language of styles to configure the presentation and JavaScript provides interactive and dynamic behavior.

It will play an important role, enhancing its performance, with the arrival of HTML5 / CSS3

It is very powerful and expressive, with syntax that keeps similarities with other very popular languages, but with particular characteristics.

Is easy to learn. They have a fast learning curve and require little time investment to start using it.

It is very easy to work with him. You do not need a web server or a special development environment. It is interpreted so we don't have to compile code. We do not need to acquire licenses. A text editor (or a code editor that facilitates the task) and a browser is all you need. To that, we can add an add-on such as Firebug to debug and we are fully equipped with free and free tools.

It is the basis for using AJAX on our pages, a programming technique that well used allows us to build 'new generation' websites greatly improving the user experience (we will see it in a next post).

Once we know the basics of JavaScript, we have open and free frameworks available (work environments / libraries) such as JQuery, Prototype, Dojo,…. which make our work even easier and allow us to develop very advanced elements with extraordinary simplicity.

It allows us to work in an open and standardized environment without being tied to the criteria of any software manufacturer. In addition, it can interact with other popular technologies such as Flash or Java.

It is true that some of these reasons have also been used to criticize JavaScript, as it is so simple that it is used without knowing it. The label of 'language for amateurs' for its use to overload visual effects web pages, has changed radically. Opinions like those of Douglas Crockford helped to understand it better and generate a great interest in knowing and applying its possibilities also among advanced programmers.

For those who start and have an interest in programming (web) is a language that makes it easy for them to take the first steps. In fact, the low complexity to work with him has encouraged its use as a reference language to teach programming concepts. Like any tool, it has its flaws and can be misused, but knowing where its limitations are is one of the first learning options that anyone who wants to start in web programming should consider.

Python Vs Javascript: Which One Is Better For Web Development?

It is difficult to find a person, company or brand that currently lacks a website. Many of these sites are programmed “to measure”, and in most cases they fulfill functions superior to the demonstrative and informative ones, with which the –even less static– corporate sites were born.

Then it is there where questions appear that although they are simple to answer, they can be somewhat disturbing at some time for those of us who are in one way or another involved in the IT sector. How to develop a different website? What is better? What language should I use for the web development of a site to make it look more professional? Will Python suit me better or do I focus on learning JavaScript?

It is good to start any comparison (and particularly this one) indicating that both are languages of different syntaxes, but the paradigms are the same and the pre conclusion is that similar results are obtained.

But although these similarities can be discussed, in terms of code and when it comes to web development they are totally different. For this reason, here we will know a little about each one of them, what are their advantages and which one you should choose.

What is Python?

Python is a minimalist programming language, which contains a syntax that makes it quite simple. It is an interpreted language, that is to say, not compiled, in addition this serves for all types of development especially to give dynamics to objects in different programs and / or paradigms.

Undoubtedly Python is one of the best options to develop a website, especially when you know the basic elements of language.

Let's see Python what offers us.

Python Features

Before continuing, we will point out some important Python features and why you should learn it.

Minimalist Dode

Yes, the code and the simple syntax are perfect for developing websites, facilitating the work and writing of it.

Well Paid

That's right, if you are going to develop a Python website, prepare your bank account, since the benefit you will receive from developing a Python website will be very profitable, as you can see in Medium.

Multiplatform

Python can not only run it in an operating system, so you can take it anywhere, from free operating systems such as Linux and through the already known Windows or Mac, in addition to other devices that have systems based on the aforementioned distributions.

Extensive Libraries

An advantage that comes very well from Python is the amount of libraries or libraries you can find to develop.

There is a wide variety of reusable code, from game creation to large websites and quality.

What Is Javascript?

Javascript is a very simple language, which can hardly be interpreted with a browser. This means that it does not need to be compiled for execution.

Javascript is highly recommended for the realization ofpages websince it allows the development of the user in it. How? Javascript It allows the website to be as static as possible, and comes hand in hand with other compulsory learning languages, such as HTML and CSS.

What gives interactivity to the web with animations, which allows the user to feel more at ease, since he will not observe a flat website, all this is included in JavaScript.

Normally with JavaScript, programs or applications are created and then inserted into the website to be used. Likewise, they are used to develop mobile applications and complex programs, from the Backend and Front End point of view.

Characteristics of JavaScript

Very Requested

Most of the potential clients ask that their websites be developed in JavaScript, because it is simple and also economical compared to Python.

Easy To Learn

Undoubtedly, this is an advantage that JavaScript has, since it has been very well documented in its different stages and has a large community, in which the novice developer can be supported to achieve a learning with solid bases.

Fast And Versatile

Since this is executed with the browser it is very easy to develop and test improvements, it can also be used to give dynamism and interactivity to a web page and then use that code to create a mobile version of that site.

Good Integration

High level of integration with cloud platforms, such as Amazon Cloud and Heroku. This means that in a large number of cases we have facilities to upload our program with some basic steps unlike Python, which when working with different frameworks requires specific actions that are not always provided by the support, or in any case, not regularly found on platforms that use Javascript.

Personal Recommendation

Once the characteristics of each language have been analyzed, the most sensible thing to our experience is to immerse yourself in one of them and learn in a staggered way to see if it is to our liking and comfort. It is important in programming learning that you have a solid foundation and this is achieved only if you understand the paradigms that make up a language and how to apply these concepts in large-scale projects.

Javascript is an eminently web language. But with Python we can develop many projects of a similar nature, in addition to allowing us to build desktop programs and do additional things, such as scripting or big data.

Once the primary rudiments are known, it is the programmer's duty to delve into things of greater complexity. And although there is contradiction in the subject, the vast majority of people find in learning a framework not only a fast and easy way to create specific programs, but also a lot of learning since knowing a framework allows to delve into programming concepts.

In this sense, the ideal is to start building basic applications or websites to better understand web development and then delve into more complex projects; There are sites like Awwwards where award-winning sites are seen and you can access fresh ideas as well as content that motivates you to continue learning.

There are no strict differences between one language and the other, there is only one gap when the programmer is able to build lines of code in true works of modern art and in binary.

Advantages And Disadvantages Of Javascript

As we have said, JavaScript is a universal language present in numerous HTML pages, in a complementary way to this code. Thanks to JavaScript, HTML pages are more enjoyable and have many additional features.

Knowing how to write scripts in JavaScript means allowing users of your HTML pages to access other functionalities and other services, thereby improving the professionalism of a website. Even recently, when a user first chose a username, it was necessary to click on a button and wait for a response from the server that sometimes asked to restart the procedure since the username already belonged to another person. However today, thanks to the use of AJAX technology, the control is carried out in the background at the same time that the user completes the file. It is undeniable that JavaScript contributes greatly to the ease of use of a website and also increases user loyalty.

Taking into account this important dissemination, knowing how to program in JavaScript has become a basic knowledge for every web developer today.

However, the use of JavaScript is not exclusive to the network; In fact, many programs on the market such as Adobe Photoshop or Adobe Illustrator use versions very similar to JavaScript for automating many tasks.

Tools Conception

Inserted tools that allow JavaScript code are many. There are from the simple text editor such as Windows WordPad to specialized tools such as Aptana Studio, through HTML code editors such as Dreamweaver or FrontPage, which allow you to insert blocks of JavaScript code. The use of these programs allows to have a certain number of tools that facilitate the writing of the code. For example, it is very simple:

Verify a syntax thanks to the automatic coloring of the source code;

Have the function of automatic completion (proposition of available methods or properties of the object);

Know the value of a variable once the script is executed.

Once the design tool is selected, it is convenient to create a programming and test environment to waste as little time as possible in the search for errors that will inevitably arise.

Parameters And Ideal Test Environment

It is necessary to keep in mind that to start working with JavaScript, you need a minimum of HTML knowledge, especially the notion of tags that allow you to place yourself on the page. To refresh the memory, we will simply remember that an HTML page is divided into two main parts:

The head where the data corresponding to the description of the content are located;

The body where the code that makes possible the construction of objects on the page (form fields, text areas, images, etc.) appears.

A JavaScript script can be located, as desired, in one or the other of these two parts. However, in principle, scripts are generally found in the head part of the page. Its execution can be immediate (when the

page is loaded) or deferred (click on a button, for example). In this case, it will be necessary to use event-based programming and functions for the code to execute. These points are discussed in the Functions and Events chapter of this book. However, placing the scripts in the head part does not mean that they will be indexed by the search engines. In fact, until now, search engines such as Yahoo or Google do not propose any content from these elements of the code, but with the development of Web 2.0 they will do so sooner or later. At the moment, in the case...

HTML And JavaScript

We have previously explained that JavaScript and HTML were closely related, with the HTML code that generally serves as a container for the JavaScript instruction block. Once the HTML page is loaded, the browser executes the JavaScript instructions thus allowing it to be enriched with new features. However, there is another type of JavaScript execution.

The Two Types Of JavaScript Execution

The JavaScript code blocks can be directly present in the source code of the HTML page between two tags (one start and one end), or written in a JavaScript file with the extension .js, totally independent of the HTML code of the page. The first case is known as internal JavaScript as opposed to the second called external JavaScript.

Neither is better than the other, it is just a programming option. The second option, however, has the advantage of allowing the code to be reused in other HTML pages without the need to rewrite or copy it.

Specifically, the script is written in a special document using a text editor and is saved without formatting under the extension .js. If the editor does not propose this extension by default, just add it when saving the document. It is recommended to explicitly name the role that your script plays in the HTML page so that it is easier to find it on another occasion.

Once these actions have been carried out, it is very easy to designate the JavaScript file in the HTML page respecting the following syntax:

<script src = "javascript_file.js"> </script>

Obviously, the file must be present in the same folder on your disk hard or from the server where the corresponding HTML file is located.

The Code Syntax Rules

In view of the fact that JavaScript is a non-flexible language that does not authorize errors, as we can see in the following lines, respecting these rules is essential to start on JavaScript.

1. Upper and lower case

One of the main difficulties of JavaScript is to be a programming language that distinguishes the use of upper and lower case. It is a rule that becomes very important when working with variables and objects.

Specifically, in JavaScript Myobject is not the same as myobject.

This applies to all keywords (properties, methods, JavaScript functions) and the use of design tools such as Aptana or the Dreamweaver editor facilitates the identification of this syntax since they are almost instantly identified with colors.

Another syntactic rule refers to the insertion of comments.

2. Inserting comments

As in most programming languages, inserting comments into your scripts can be extremely useful. In fact, apart from being able to more easily find the instruction blocks that you have created, the comments can be of immense help the day you have to retake the code. The readability of the code is even one of the main criteria determining the quality of a JavaScript code. Because, after all, how about...

Creating The Test Page

To write effectively in the HTML pages, it is best to create a model page where you must include the labels that indicate the beginning and end of the script .

Being the head part of the page where the JavaScript code is usually inserted, look at this example of the HTML code of the model page that will be used for writing all your scripts.

The first two lines determine the type of document, its presence is essential for the proper functioning of DHTML instructions, as we will see in the chapter Improve interactivity with JavaScript and CSS. The fourth line indicates the beginning of the head tag that interests us in a particular way. The fifth line allows you to add a meta tag that indicates the characters used, the sixth gives a title to the page (in our case, JavaScript Model Page).

Creating A Personal Library Of Javascript Scripts

Over time, you will have to develop numerous scripts that can be reused later. To facilitate this reuse, identify your pages with names that clearly indicate the purpose of your JavaScript script.

Beware of confusing personal library scripts and common libraries, abundant in the network, and that enrich the classic JavaScript operation. The installation of new JavaScript libraries will be discussed in the chapter Improving interactivity with JavaScript and CSS.

Error Messages And Tips For Debugging The Code (Debug)

The fact that browsers interpret JavaScript differently, imposes the need to perform tests on each of them. However, the best advice is to test the scripts first in Firefox / Mozilla which has a more powerful error resolution tool and then perform a test in Internet Explorer. To help you a little in processing possible errors; we can divide these into different categories:

First, it may be the case that nothing happens loading the page. It is necessary to know that JavaScript controls the script before displaying anything. If it finds an error, the script is interrupted

without going too far. These errors are usually due to approximate syntax or keyboard errors.

It is also possible to find errors not when loading the page but when it is executed. This generally means that objects, their properties, their methods and even functions do not correspond or are misused.

Finally, the most difficult errors to detect are the purely logical errors that arise when the script tests have not been sufficient. In these cases, the script can work in one case and cause an error with other values or other circumstances. Do not hesitate to test your scripts with different values and carefully observe the results obtained. By prudence, if you do not have a tool that allows you to control the state of the script (such as Aptana ...

JavaScript Debugging Tools

Although it is possible to write a script in a very simple way, the use of a design tool It can be useful, especially if it is the resolution of an error. Its use will allow us to have breakpoints, know the value of the variables and other aids that will be very useful in this crucial stage such as the search for an error.

We are going to introduce some of these tools:

1. Microsoft Script Debugger

With the use of Microsoft Script Debugger, you have a tool to help with syntax and problem solving, when the Internet Explorer browser encounters a problem in the development of a script, in Microsoft Script Debugger it is possible to go directly to the line that presents the problem.

2. Microsoft FrontPage / Adobe Dreamweaver

The HTML code editors allow you to view the code of the page and thus access the JavaScript part. But they do not have functionalities capable of adding breakpoints and knowing the value of the variables.

Chapter 2

Basic Programming

Before starting to develop programs and utilities with JavaScript, it is necessary to know the basic elements with which the applications are built. If you already know how to program in some programming language, this chapter will help you to know the specific syntax of JavaScript.

If you have never programmed, this chapter explains in detail and starting from scratch the basic knowledge necessary to be able to later understand the advanced programming, which is the one used to create the real applications.

Variables

The variables in the programming languages follow a logic similar to the variables used in other fields such as mathematics. A variable is an element that is used to store and reference another value. Thanks to the variables it is possible to create "generic programs", that is, programs that always work the same regardless of the specific values used.

In the same way that if in Mathematics the variables did not exist the equations and formulas could not be defined, in programming one could not make really useful programs without the variables. If there were no variables, a program that adds two numbers could be written as: result = 3 + 1

The previous program is so unhelpful that it only applies to the case where the first number of the sum is 3 and the second number is 1. In any other case, the program obtains an incorrect result.

However, the program can be remade as follows using variables to store and refer to each number:

number_1 = 3 number_2 = 1

result = number_1 + number_2

Elements number_1 and number_2 are variables that store the values used by the program. The result is always calculated based

on the value stored by the variables, so this program works correctly for any number of indicated numbers. If the value of the variables number_1 and number_2 is modified, the program continues to function correctly.

Variables in JavaScript are created using the reserved word var. In this way, the previous example can be done in JavaScript as follows:

```
var number_1 = 3; var number_2 = 1;

var result = number_1 + number_2;
```

The reserved word var should only be indicated when defining the variable for the first time, which is called declaring a variable. When the variables are used in the rest of the script instructions, it is only necessary to indicate their name. In other words, in the previous example it would be a mistake to indicate the following:

```
var number_1 = 3; var number_2 = 1;

var result = var number_1 + var number_2;
```

If a value is also assigned when a variable is declared, it is said that the variable has been initialized. In JavaScript, it is not mandatory to initialize the variables since they can be declared by one party and assign them a value later. Therefore, the previous example can be remade as follows:

```
var number_1; var number_2;

number_1 = 3; number_2 = 1; var result = number_1 + number_2;
```

One of the most surprising features of JavaScript for programmers accustomed to other programming languages is that it is not necessary to declare the variables. In other words, you can use variables that have not been previously defined by the reserved word var. The previous example is also correct in JavaScript as follows:

```
var number_1 = 3; var number_2 = 1;

result = number_1 + number_2;
```

The result variable is not declared, so JavaScript creates a global variable (the differences between local and global variables will be

seen later) and assigns it the corresponding value. In the same way, the following code would also be correct:

number_1 = 3; number_2 = 1;

result = number_1 + number_2;

In any case, it is recommended to declare all the variables to be used.

The name of a variable is also known as an identifier and must comply with the following rules:

▪ It can only consist of letters, numbers, and the symbols $ (dollar) and _ (underscore).

▪ The first character cannot be a number.

Therefore, the following variables have correct names:

var $ number1; var _ $ letter; var $$$ otherNumber; var $ _ to $ 4;

However, the following variables have incorrect identifiers:

var 1number; // Start with a number var number; 1_123; // Contains a character ";"

Types Of Variables

Although all JavaScript variables are created in the same way (using the reserved word var), the way in which they are assigned a value depends on the type of value to be stored (numbers, texts, etc.)

Numeric

They areused to store integer numerical values (called integer in English) or decimals (called float in English). In this case, the value is assigned by directly indicating the integer or decimal number. Decimal numbers use the character. (period) instead of, (comma) to separate the whole part and the decimal part:

var iva = 16 // integer variable var total = 234.65; // decimal type variable

Text Strings

Used to store characters, words and/or text phrases. To assign the value to the variable, the value is enclosed in double or single quotes, to delimit its beginning and its end:

var message = "Welcome to our website"; var productName = 'Product ABC'; var Selected letter = 'c';

Sometimes, the text that is stored in the variables is not so simple. If, for example, the text itself contains single or double quotes, the strategy followed is to enclose the text with quotes (single or double) that the text does not use:

/ * The content of text1 has single quotes, so it is enclosed with double quotes * /

var text1 = "A phrase with 'single quotes' inside";

/ * The content of text2 has double quotes, so it is enclosed with single quotes * /

var text2 = 'A phrase with "double quotes" inside';

However, sometimes text strings contain both single and double-quotes. In addition, there are other characters that are difficult to include in a text variable (tab, ENTER, etc.). To solve these problems, JavaScript defines a mechanism to easily include special and problematic characters within a text string.

The mechanism consists of replacing the problematic character with a simple combination of characters. In this way, the previous example that contained single and double quotes within the text can be redone as follows:

var text1 = 'A phrase with \' single quotes \ 'inside'; var text2 = "A phrase with \" double quotes \ "inside";

Arrays

Sometimes, arrays are called vectors, matrices, and even arrays. However, the term array is the most used and is a commonly accepted word in the programming environment.

An array is a collection of variables, which can be all of the same type or each of a different type. Its utility is better understood with a simple example: if an application needs to handle the days of the week, seven variables of type text could be created:

var day1 = "Monday"; var day2 = "Tuesday"; ... var day7 = "Sunday";

Although the previous code is not incorrect, it is inefficient and complicates the programming excessively. If instead of the days of the week you had to save the name of the months of the year, the name of all the countries in the world or the daily temperature measurements of the last 100 years, you would have to create tens or hundreds of variables.

In these types of cases, all related variables can be grouped into a collection of variables or array. The previous example can be remade as follows: var days = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"];

Now, a single variable called days stores all related values, in this case, the days of the week. To define an array, the characters [and] are used to delimit its beginning and end and the character, (comma) is used to separate its elements: var array_name = [value1, value2, ..., valueN];

Once an array is defined, it is very easy to access each of its elements. Each element is accessed indicating its position within the array. The only complication, which is responsible for many errors when starting to program, is that the positions of the elements begin to be counted at 0 and not at 1:

var day Selected = days [0]; // selected day = "Monday" var otherDay = days [5]; // otherDay = "Saturday"

In the previous example, the first instruction wants to get the first element of the array. To do this, the name of the array is indicated and in square brackets the position of the element within the array. As mentioned, the positions begin to be counted at 0, so the first element occupies the position 0 and is accessed through days [0].

The days value [5] refers to the element that occupies the sixth position within the days array. As the positions begin to be counted at 0, position 5 refers to the sixth element, in this case, the Saturday value.

Booleans

Booleans or Boolean type variables are also known as logical type variables. Although to really understand its usefulness you should study the advanced programming with JavaScript in the following chapter, its basic operation is very simple.

A variable of type boolean stores a special type of value that can only take two values: true (true) or false (false). It cannot be used to store numbers, nor can it save text strings.

The only values that can store these variables are true and false, so the true and false values cannot be used. Below are a couple of Boolean variables:

var Registered client = false; var vatIncluded = true;

Operators

Variables alone are of little use. Until now, we have only seen how to create variables of different types and how to show their value through the alert () function. To make really useful programs, other tools are necessary.

Operators allow you to manipulate the value of variables, perform mathematical operations with their values and compare different variables. In this way, operators allow programs to perform complex calculations and make logical decisions based on comparisons and other types of conditions.

Assignment

The assignment operator is the most used and the easiest. This operator is used to store a specific value in a variable. The symbol used is = (not to be confused with the operator == that will be seen later): var number1 = 3;

To the left of the operator, the name of a variable must always be indicated. To the right of the operator, you can indicate variables, values, logical conditions, etc:

var number1 = 3; var number2 = 4;

/ * Error, the assignment is always made to a variable, so on the left, you cannot indicate a number * /

5 = number1;

// Now, variable number1 is worth 5 number1 = 5;

// Now, the variable number1 is worth 4 number1 = number2;

Increase And Decrease

These two operators are only valid for numerical variables and are used to increase or decrease the variable value by one unit.

Example:

var number = 5; ++ number;

alert (number); // number = 6

The increment operator is indicated by the prefix ++ in the variable name. The result is that the value of that variable is increased by one unit. Therefore, the previous example is equivalent to:

var number = 5; number = number + 1; alert (number); // number = 6

Equivalently, the decrement operator (indicated as a prefix - in the name of the variable) is used to decrease the value of the variable:

var number = 5; --number;

alert (number); // number = 4

The previous example is equivalent to:

var number = 5; number = number - 1; alert (number); // number = 4

The increment and decrement operators can not only be indicated as a prefix of the variable name, but it is also possible to use them as a suffix. In this case, their behavior is similar but very different. In the following example:

var number = 5; number ++;

alert (number); // number = 6

The result of executing the previous script is the same as when the ++ number operator is used, so it may seem equivalent to indicate the ++ operator in front of or behind the variable identifier. However, the following example shows their differences:

var number1 = 5; var number2 = 2; number3 = number1 ++ + number2; // number3 = 7, number1 = 6

var number1 = 5; var number2 = 2; number3 = ++ number1 + number2;

// number3 = 8, number1 = 6

If the ++ operator is indicated as a prefix of the variable identifier, its value is increased before performing any other operation. If the ++ operator is indicated as a variable identifier suffix, its value is increased after executing the statement in which it appears.

Therefore, in the instruction number3 = number1 ++ + number2 ;, the value of number1 is increased after the operation (first it is added and number3 is worth 7, then the value of number1 is increased and is worth 6). However, in the instruction number3 = ++ number1 + number2 ;, first the value of number1 is increased and then the sum is made (first it increases number1 and voucher 6, then the sum is made and number3 voucher 8).

Logical

Logical operators are essential for complex applications, as they are used to make decisions about the instructions that the program should execute based on certain conditions.

The result of any operation that uses logical operators is always a logical or Boolean value.

Denial

One of the most used logical operators is that of negation. It is used to obtain the opposite value of the variable:

var visible = true;

alert (! visible); // Show "false" and not "true"

Logical denial is obtained by presetting the symbol! to the variable identifier. If the original variable is of Boolean type, it is very simple to obtain its negation. However, what happens when the variable is a number or a text string? To obtain the negation in this type of variables, it is first carried out its conversion to a Boolean value:

▪ If the variable contains a number, it becomes false if it is 0 and true for any other number (positive or negative, decimal or whole).

▪ If the variable contains a text string, it becomes false if the string is empty ("") and true in any other case.

var quantity = 0; empty =! quantity; // empty = true

quantity = 2; empty =! quantity; // empty = false var message = ""; empty message =! message; // empty= true message; message = "Welcome";

empty message =! message; // Empty message = false

AND

The logical AND operation obtains its result by combining two Boolean values. The operator is indicated by the && symbol and its result is only true if the operands hold true.

OR

The logical OR operation also combines two Boolean values. The operator is indicated by the symbol || and its result is true if either of the two operands is true.

Mathematics

JavaScript allows you to perform mathematical manipulations on the value of numerical variables. The operators defined are: addition (+), subtraction (-), multiplication (*) and division (/). Example:

var number1 = 10; var number2 = 5;

result = number1 / number2; // result = 2 result = 3 + number1; // result = 13 result = number 2-4; // result = 1 result = number1 * number 2; // result = 50

In addition to the four basic operators, JavaScript defines another mathematical operator that is not easy to understand when it is first studied, but which is very useful on some occasions.

This is the "module" operator, which calculates the rest of the entire division of two numbers. If you divide 10 and 5 for example, the division is exact and gives a result of 2. The rest of that division is 0, so module 10 and 5 equals 0.

However, if you divide 9 and 5, the division is not exact, the result is 1 and the remainder 4, so module 9 and 5 equals 4.

The JavaScript module operator is indicated by the% symbol, which should not be confused with the percentage calculation :

var number1 = 10; var number2 = 5; result = number1% number2; // result = 0

number1 = 9; number2 = 5;

result = number1% number2; // result = 4

Mathematical operators can also be combined with the assignment operator to abbreviate their notation:

var number1 = 5; number1 + = 3; // number1 = number1 + 3 = 8 number1 - = 1; // number1 = number1 - 1 = 4 number1 * = 2; // number1 = number1 * 2 = 10 number1 / = 5; // number1 = number1 / 5 = 1 number1% = 4; // number1 = number1% 4 = 1

Relational

The relational operators defined by JavaScript are identical to those that define mathematics.

Operators that relate variables are essential for any complex application, as will be seen in the next chapter on advanced

programming. The result of all these operators is always a Boolean value.

Special care must be taken with the equality operator (==), as it is the source of most programming errors, even for users who already have some experience developing scripts. The == operator is used to compare the value of two variables, so it is very different from the = operator, which is used to assign a value to a variable:

// The operator "=" assigns values var number1 = 5; result = number1 = 3; // number1 = 3 and result = 3

// The operator "==" compares variables var number1 = 5;

result = number1 == 3; // number1 = 5 and result = false

Relational operators can also be used with variables of type text string:

var text1 = "hello"; var text2 = "hello"; var text3 = "goodbye";

result = text1 == text3; // result = false result = text1! = text2; // result = false result = text3> = text2; // result = false

When text strings are used, the operators "greater than" (>) and "less than" (<) follow a non-intuitive reasoning: letter by letter is compared starting from the left until a difference is found Between the two text strings. To determine if one letter is greater or less than another, uppercase letters are considered lower than lower case letters and the first letters of the alphabet are smaller than the last ones (a is less than b, b is less than c, A is less than a, etc.)

Flow Control Structures

Programs that can be performed using only variables and operators are a simple linear sequence of basic instructions.

However, programs that show a message cannot be carried out if the value of a variable is equal to a certain value and does not show the message in all other cases. Nor can the same instruction be repeated efficiently, such as adding a certain value to all the elements of an array.

To carry out this type of programs, flow control structures are necessary, which are instructions of the type "if this condition is met, do it; if it is not met, do this another". There are also instructions such as "repeat this while this condition is met".

If flow control structures are used, the programs cease to be a linear sequence of instructions to become intelligent programs that can make decisions based on the value of the variables.

If Structure

The structure most used in JavaScript and in most programming languages is the if structure. It is used to make decisions based on a condition. Its formal definition is:

```
if (condition) {...
}
```

If the condition is met (that is, if its value is true), all instructions within {...} are executed. If the condition is not met (that is if its value is false), no instruction contained in {...} is executed and the program continues executing the rest of the instructions in the script.

Example: var show Message = true;

```
if (show Message) {
alert ("Hello World");
}
```

Here the message is shown to the user since the variable ShowMessage has a value of true and therefore, the program enters into the if instruction block.

The example could also be rewritten as:

```
var show Message = true;
if (show Message == true) {
alert ("Hello World");
}
```

In this case, the condition is a comparison between the value of the variable Show message and the value true. As the two values coincide, the equality is met and therefore the condition is true, its value is true and the instructions contained in that if block are executed.

The comparison of the previous example is usually the origin of many programming errors, by confusing the == and = operators. Comparisons are always made with the operator == since the operator = only assigns values: var show Message = true;

// The two if values are compared (show Message == false) {...

}

// Error - The value "false" is assigned to the variable if (show Message = false) {...

}

The condition that controls the if () you can combine the different logical and relational operators shown above:

var shown = false;

if (! shown) {

alert ("This is the first time the message is displayed"); }

The AND and OR operators allow you to chain several simple conditions to build complex conditions:

var shown = false; var user PermitMessages = true;

if (! displayed && user Allow Messages) {

alert ("This is the first time the message is displayed"); }

The previous condition is formed by an AND operation on two variables. In turn, the negation operator is applied to the first variable before performing the AND operation. Thus, since the value of displayed is false, the value! Displayed would be true. Since the variable UserPermitMessages is true, the result of! Displayed && userPermitMessages would be equal to true && true, so the final

result of the condition of the if () would be true and therefore, the instructions inside the block are executed of if (). Exercise 5

Complete the conditions of the if in the following script so that the messages of the alerts () are always displayed correctly:

var number1 = 5; var number2 = 8;

if (...) {

alert ("number1 is not greater than number2");

} if (...) {

alert ("number2 is positive");

} if (...) {

alert ("number1 is negative or nonzero");

} if (...) {alert ("Increasing the value ofby 1 unit does not make it greater than or equal to

number1number2");

}

Structure if ... else

Sometimes, the decisions to be made are not of the type "if the condition is met, do it; if it is not met, do nothing." Normally the conditions are usually of the type "if this condition is met, do it; if it is not met, do this another".

For this second type of decision, there is a variant of the if structure called if ... else. Its formal definition is as follows:

if (condition) {...

} else {...

}

If the condition is met (that is if its value is true), all the instructions found within the if () are executed. If the condition is not met (that is, if its value is false), all instructions contained in else {} are executed. Example:

```
var age = 18;
if (age> = 18) {
alert ("You are of legal age");
} else {
alert ("You are still a minor");
}
```

If the value of the variable age is greater than or equal to the numerical value 18, the condition of the if () is fulfilled and therefore, its instructions are executed and the message "You are of age" is displayed. However, when the value of the age variable is not equal to or greater than 18, the condition of the if () is not met, so all the instructions in the else {} block are automatically executed. In this case, the message "You are still a minor" would be displayed.

The following example compares variables of type text string:

```
var name = "";
if (name == "") {
alert ("You haven't told us your name yet");
} else {
alert ("We have saved your name");
}
```

The condition of the previous if () is constructed using the == operator, which is used to compare two values (not to be confused with the operator = used to assign values). In the previous example, if the text string stored in the variable name is empty (that is, it is equal to ""), the message defined in the if () is displayed. Otherwise, the message defined in the else {} block is displayed.

The if ... else structure can be chained to perform several checks in a row:

```
if (age <12) {
```

```
alert ("You are still too small");
} else if (age <19) {
alert ("You are a teenager");
} else if (age <35) {
alert ("You are still young");
} else {
alert ("Think about taking care of yourself a little more"); }
```

It is not mandatory that the combination of structures if ... else ends with the else statement, since it can end with an else type instruction if ().

Structure for

The structures if and if ... else are not very efficient when you want to repeatedly execute an instruction. For example, if you want to display a message five times, you might think about using the following if:

```
var times = 0;
if (times <4) {
alert ("Message"); times ++;
}
```

It is checked if the variable times is less than 4. If it is fulfilled, it enters the if (), the message is displayed and the value of the variable is increased times. This should continue to run until the message is displayed five times as desired.

However, the actual operation of the previous script is very different from the desired one, since the message is only shown once per screen. The reason is that the execution of the if () structure is not repeated and the condition check is only done once, regardless of whether the value of the variable used in the condition is modified within the if ().

The for structure allows you to perform this type of repetitions (also called loops) in a very simple way. However, its formal definition is not as simple as that of if ():

for (initialization; condition; update) {...

}

The idea of the operation of a for loop is as follows: "as long as the indicated condition is still fulfilled, repeats the execution of the instructions defined within the for. In addition, after each repetition, it updates the value of the variables used in the condition ".

▪ "Initialization" is the area in which the initial values of the variables that control the repetition are established.

▪ The "condition" is the only element that decides whether the repetition continues or stops.

▪ The "update" is the new value that is assigned after each repetition to the variables that control the repetition.

var message = "Hi, I'm in a loop";

for (var i = 0; i <5; i ++) {

alert (message);

}

The part of the loop initialization consists of:

var i = 0;

Therefore, the variable i is created first and assigned the value of 0. This initialization zone is only taken into account just before starting to execute the loop.

The following repetitions do not take into account this initialization part.

The condition zone of the loop is:

i <5

The loops continue to run as long as the conditions are met and stop executing just after verifying that the condition is not met. In this case, while the variable i is worth less than 5 the loop is executed indefinitely.

Since the variable i has been initialized to a value of 0 and the condition to exit the loop is that i is less than 5, if the value of i is not modified in any way, the loop would be repeated indefinitely.

For this reason, it is essential to indicate the update zone, in which the value of the variables that control the loop is modified: i ++

In this case, the value of the variable i is increased by one unit after each repetition. The update zone is executed after the execution of the instructions included in the for.

Thus, during the execution of the fifth repetition the value of i will be 4. After the fifth execution, the value of i is updated, which will now be worth 5. Since the condition is that i is less than 5, the condition is no longer complies and for instructions are not executed a sixth time.

Normally, the variable that controls the loops for is called i, since it remembers the word index and its short name saves a lot of time and space.

The previous example that showed the days of the week contained in an array can be redone more easily using the structure for:

```
var days = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday Sunday"];

for (var i = 0; i <7; i ++) {

alert (days [i]);

}
```

Structure for ... in

A control structure derived from for is the structure for ... in. Its exact definition implies the use of objects, which is an advanced programming element that is not going to be studied. Therefore, only

the for ... in structure adapted to its use in arrays will be presented. Its formal definition adapted to arrays is:

for (index in array) {...

}

If you want to traverse all the elements that form an array, the structure for ... in is the most efficient way to do it, as shown in the following example:

var days = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"];

for (i in days) {

alert (days [i]);

}

The variable indicated as an index is the one that can be used within the for ... in loop to access the array elements. In this way, in the first repetition of the loop the variable i is worth 0 and in the last voucher 6.

This control structure is the most suitable for traversing arrays (and objects) since it avoids having to indicate the initialization and conditions of the loop for simple and works correctly whatever the length of the array. In fact, it still works the same even if the number of array elements varies.

Functions And Basic Properties Of Javascript

JavaScript incorporates a series of tools and utilities (called functions and properties, as will be seen later) for handling variables. In this way, many of the basic operations with the variables can be performed directly with the utilities offered by JavaScript.

Useful functions for text strings

Below are some of the most useful functions for handling text strings:

length, calculates the length of a text string (the number of characters that make it up)

var message = "Hello World" ;

var numberLetters = message.length; // numberLetters = 10

+, is used to concatenate several text strings

var message1 = "Hello"; var message2 = "World";

var message = message1 + message2; // message = "Hello World"

In addition to the + operator, you can also use the concat () function

var message1 = "Hello";

var message2 = message1.concat ("World"); // message2 = "Hello World" Text strings can also be joined with numerical variables:

var variable1 = "Hello"; var variable2 = 3;

var message = variable1 + variable2; // message = "Hello 3"

When several text strings are joined, it is usual to forget to add a space between the words:

var message1 = "Hello"; var message2 = "World";

var message = message1 + message2; // message = "HelloWorld"

Blank spaces can be added at the end or beginning of the strings and can also be explicitly indicated:

var message1 = "Hello"; var message2 = "World";

var message = message1 + "" + message2; // message = "Hello World"

toUpperCase (), transforms all the characters in the string to their corresponding characters in uppercase:

var message1 = "Hello";

var message2 = message1.toUpperCase (); // message2 = "HELLO"

toLowerCase (), transforms all the characters in the string to their corresponding characters in lowercase:

var message1 = "HolA";

var message2 = message1.toLowerCase (); // message2 = "hello"

charAt (position), get the character that is in the indicated position:

var message = "Hello"; var letter = message.charAt (0); // letter = H letter = message.charAt (2);// letter = l

indexOf (character), calculates the position where the indicated character is within the text string. If the character is included several times within the text string, its first position is returned starting to search from the left. If the string does not contain the character, the function returns the value -1:

var message = "Hello"; var position = message.indexOf ('a'); // position = 3 position = message.indexOf ('b'); // position = -1 Its analogous function is lastIndexOf ():

lastIndexOf (character), calculates the last position in which the indicated character is within the text string. If the string does not contain the character, the function returns the value -1:

var message = "Hello"; var position = message.lastIndexOf ('a'); // position = 3 position = message.lastIndexOf ('b'); // position = -1

The lastIndexOf () function starts its search from the end of the string to the beginning, although the position returned is correct starting from the beginning of the word. substring (start, end), extracts a portion of a text string. The second parameter is optional. If only the start parameter is indicated, the function returns the part of the corresponding original string from that position to the end:

var message = "Hello World"; var portion = message.substring (2); // portion = "the World" portion = message.substring (5); // portion = "World" portion = message.substring (7); // portion = "ndo"

If a negative start is indicated, the same original string is returned:

var message = "Hello World";

var portion = message.substring (-2); // portion = "Hello World"

When the beginning and end are indicated, the part of the original chain between the initial position and the one immediately preceding the final position is returned (that is, the start position is included and the position final no):

var message = "Hello World"; var portion = message.substring (1, 8); // portion = "wave Mun" portion = message.substring (3, 4); // portion = "a"

If an ending smaller than the beginning is indicated, JavaScript considers them in reverse, since it automatically assigns the smallest value at the beginning and the largest at the end:

var message = "Hello World"; var portion = message.substring (5, 0); // portion = "Hello" portion = message.substring (0, 5); // portion = "Hello"

split (separator), convert a text string into an array of text strings. The function starts the text string by determining its chunks from the indicated separator character:

var message = "Hello World, I am a text string!"; var words = message.split ("");

// words = ["Hello", "World,", "I am", "a", "string", "of", "text!"]; With this function you can easily extract the letters that form a word:

var word = "Hello"; var letters = word.split (""); // letters = ["H", "o", "l", "a"]

Useful functions for numbers

Below are some of the most useful functions and properties for handling numbers.

NaN, (from English, "Not a Number") JavaScript uses the NaN value to indicate an undefined numerical value (for example, division 0/0).

var number1 = 0; var number2 = 0;

alert (number1 / number2); // the value NaN

isNaN () is shown, it allows to protect the application from possible undefined numerical values

var number1 = 0; var number2 = 0; if (isNaN (number1 / number2)) {

alert ("The division is not defined for the indicated numbers");

} else {

alert ("The division is equal to =>" + number1 / number2); }

Infinity, refers to an infinite and positive numerical value (there is also the –Infinity value for negative infinities)

var number1 = 10; var number2 = 0;

alert (number1 / number2); // the Infinityvalue is displayed

toFixed (digits), returns the original number with as many decimals as indicated by the digits parameter and performs the necessary rounding. It is a very useful function for example to show prices.

var number1 = 4564.34567; number1.toFixed (2); // 4564.35 issue1.toFixed (6); // 4564.345670 issue1.toFixed (); // 4564

Chapter 3

Advanced Programming

The control structures, the operators and all the JavaScript utilities that have been seen in the previous chapters, allow creating simple and medium complexity scripts.

However, other elements such as functions and other more advanced control structures are required for more complex applications, which are described in this chapter.

Functions

When developing a complex application, it is very common to use the same instructions over and over again. A script for an e-commerce store, for example, has to calculate the total price of the products several times, to add taxes and shipping costs.

When a series of instructions are repeated over and over again, the source code of the application becomes too complicated, since:

▪ The application code is much longer because many instructions are repeated.

▪ If you want to modify any of the repeated instructions, you should make as many modifications as you have written that instruction, which becomes very heavy work and very prone to mistakes.

Functions are the solution to all these problems, both in JavaScript and in other programming languages. A function is a set of instructions that are grouped together to perform a specific task and that can be easily reused.

In the following example, the instructions that add the two numbers and show a message with the result are repeated over and over again:

var result;

var number1 = 3; var number2 = 5;

// The numbers are added and the result result is shown = number1 + number2; alert ("The result is" + result);

number1 = 10; number2 = 7;

// The numbers are added and the result result is shown = number1 + number2; alert ("The result is" + result);

number1 = 5; number2 = 8;

// The numbers are added and the resultis shown

result= number1 + number2; alert ("The result is" + result); ...

Although it is a very simple example, it seems clear that repeating the same instructions throughout the code is not recommended. The solution proposed by the functions consists in extracting the instructions that are repeated and replacing them with an instruction of the type "at this point, the instructions that have been extracted are executed":

var result;

var number1 = 3; var number2 = 5;

/ * At this point, the function that adds

2 numbers is called and shows the result * /

number1 = 10; number2 = 7;

/ * At this point, the function that adds

2 numbers is called and displays the result * /

number1 = 5; number2 = 8;

/ * At this point, the function that adds

2 numbers is called and shows the result * / ...

For the solution of the previous example to be valid, the common instructions must be grouped into a function that can be Indicate the numbers to add before displaying the message.

Therefore, you must first create the basic function with the common instructions. Functions in JavaScript are defined by the reserved word function, followed by the name of the function. Its formal definition is as follows:

```
function function_name () {...
}
```

The function name is used to call that function when necessary. The concept is the same as with the variables, which are assigned a unique name to be able to use them within the code. After the name of the function, two parentheses are included whose meaning is detailed below. Finally, the symbols {and} are used to enclose all the instructions that belong to the function (similar to how the instructions are enclosed in the if or for structures).

Returning to the previous example, a function called sum_and_sample is created as follows:

```
function sum_and_sample () {
result = number1 + number2;
alert ("The result is" + result); }
```

Although the previous function is created correctly, it does not work as it should because the "arguments" are missing, which are explained in the next section. Once the function has been created, the function can be called from any point in the code to execute its instructions

(in addition to "calling the function", the expression "invoke the function" is also used).

The function call is made simply by indicating its name, including the end brackets and the character; to finish the instruction:

```
function sum_and_sample () {
result = number1 + number2; alert ("The result is" + result);
} var result;
var number1 = 3; var number2 = 5; sum_and_sample ();
number1 = 10; number2 = 7; sum_and_sample ();
number1 = 5; number2 = 8;
sum_and_sample (); ...
```

The code in the previous example is much more efficient than the first code that was shown since there are no repeated instructions. The instructions that add and display messages have been grouped under one function, which allows them to be executed at any point in the program simply by indicating the name of the function.

The only thing missing from the previous example to function correctly is to be able to indicate to the function the numbers to add. When data needs to be passed to a function, the "arguments" are used, as explained in the next section.

Arguments And Return Values

The simplest functions do not need any information to produce their results. However, most functions of real applications must access the value of some variables to produce their results.

The variables that functions need are called arguments. Before you can use them, the function must indicate how many arguments you need and what is the name of each argument. In addition, when invoking the function, the values that will be passed to the function must be included. The arguments are indicated within the parentheses that go after the name of the function and are separated by a comma (,).

Following the previous example, the function must indicate that it needs two arguments, corresponding to the two numbers that it has to add: function sum_and_sample (firstNumber, secondNumber) {...}

Next, to use the value of the arguments within the function, the same name with which the arguments were defined should be used:

function sum_and_sample (firstNumber, secondNumber) {...}

var result = firstNumber + secondNumber; alert ("The result is" + result); }

Within the function, the value of the variable firstNumber will be equal to the first value that is passed to the function and the value of the variable secondNumber will be equal to the second value that is

passed to it. To pass values to the function, they are included in the parentheses used when calling the function: Function

//definition function sum_and_sample (firstNumber, secondNumber) {...}

var result = firstNumber + secondNumber; alert ("The result is" + result);

}

// Declaration of the variables var number1 = 3; var number2 = 5;

// Call to the function sum_and_sample (number1, number2);

In the previous code, it should be taken into account that:

▪ Although variables are almost always used to pass the data to the function, the value of these variables could have been used directly: sum_and_sample (3, 5);

▪ The number of arguments passed to a function should be the same as the number of arguments indicated by the function. However, JavaScript does not show any errors if more or less arguments are passed than necessary.

▪ The order of the arguments is fundamental since the first data indicated in the call will be the first value expected by the function; The second value indicated in the call is the second value that the function expects and so on.

▪ An unlimited number of arguments can be used, although if their number is very large, the function call is complicated too much.

▪ It is not mandatory to match the name of the arguments used by the function and the name of the arguments. Previously, the arguments passed are number1 and number2 and the arguments used by the function are firstNumber and secondNumber.

Below is another example of a function that calculates the total price of a product from its basic price:

// Function definition function calculates Total Price (price) {

var taxes = 1.16; var expenses Shipping = 10; var Total price = (price * taxes) + expenses Shipping;

}

// Call to the function calculates Total Price (23.34);

The previous function takes as an argument a variable called price and adds taxes and shipping costs to obtain the total price. When calling the function, the value of the basic price is passed directly by number 23.34.

However, the previous code is not too useful, since the ideal would be that the function could return the result obtained to save it in another variable and be able to continue working with this total price:

function calculates Total Price (price) {

var taxes = 1.16; var expenses Shipping = 10; var Total price = (price * taxes) + expenses Shipping;

}

// The value returned by the function is stored in a variable var Total price = calculates Total Price (23.34);

// Continue working with the variable "Total price"

Fortunately, functions can not only receive variables and data but can also return the values they have calculated. To return values within a function, the reserved word return is used. Although functions can return values of any type, they can only return one value each time they are executed.

function calculates Total Price (price) {

var taxes = 1.16; var expenses Shipping = 10; var Total price = (price * taxes) + expenses Shipping; return total price;

} var Total price = calculates Total Price (23.34);

// Continue working with the variable "total price"

In order for the function to return a value, it is only necessary to write the reserved word return together with the name of the variable to be

returned. In the previous example, the execution of the function arrives at the statement return total price; and at that time, it returns the value that contains the variable priceTotal.

As the function returns a value, at the point where the call is made, the name of a variable in which the returned value is stored must be indicated: var Total price = calculates Total Price (23.34);

If the name of any variable is not indicated, JavaScript does not show any errors and the value returned by the function is simply lost and therefore will not be used in the rest of the program. In this case, it is also not mandatory that the name of the variable returned by the function matches the name of the variable in which that value is to be stored.

If the function reaches an instruction of type return, the indicated value is returned and the execution of the function ends. Therefore, all instructions that are included after a return are ignored and for that reason, the return statement is usually the last of most functions.

To make the above example more complete, another argument can be added to the function that indicates the percentage of taxes that must be added to the price of the product. Obviously, the new argument must be added to both the definition of the function and its call:

function calculates Total Price (price, percentage) {

Taxesvar expenses Shipping = 10; var price Taxes = (1 + percentage Taxes / 100) * price; var Total price = price with Taxes + expenses Shipping; return total price;

}

var Total price = calculates Total Price (23.34, 16); var other Total Price = calculates Total Price (15.20, 4);

To complete the previous exercise, the total price returned by the function can be rounded to two decimals:

function calculates Total Price (price, percentage) {

Taxesvar expenses Shipping = 10; var price Taxes = (1 + percentage Taxes / 100) * price; var Total price = price with Taxes + expenses Shipping; return priceTotal.toFixed (2);

} var Total price = calculates Total Price (23.34, 16);

Scope Of The Variables

The scope of a variable (called "scope" in English) is the area of the program in which the variable is defined. JavaScript defines two scopes for the variables: global and local.

The following example illustrates the behavior of the scopes:

function creates Message () {

var message = “Test message”;

} createMessage (); alert (message);

The previous example first defines a function called createMessage that creates a variable called message. Then, the function is executed by the call createMessage (); and then, the value of a variable called message is shown by the alert () function.

However, when executing the above code, no message is displayed on the screen. The reason is that the message variable has been defined within the createMessage () function and is, therefore, a local variable that is only defined within the function.

Any instruction that is within the function can make use of that variable, but all instructions that are in other functions or outside of any function will not have the message variable defined. Thus, to display the message in the previous code, the alert () function must be called from within the createMessage ():

function function createMessage () {

var message = "Test message"; alert (message);

} createMessage ();

In addition to local variables, there is also the concept of a global variable, which is defined at any point in the program (even within any function). var message = "Test message";

```
function sample Message () {
alert (message);
}
```

The code above is the inverse example to the one shown above. Within the sample message function () you want to use a variable called message and that has not been defined within the function itself. However, if the previous code is executed, the message defined by the message variable is shown.

The reason is that in the previous JavaScript code, the message variable has been defined outside of any function. These types of variables are automatically transformed into global variables and are available at any point in the program (even within any function).

Thus, although no variable called message has been defined inside the function, the previously created global variable allows the alert () instruction within the function to display the message correctly.

If a variable is declared outside of any function, it is automatically transformed into a global variable regardless of whether it is defined using the reserved word var or not. However, the variables defined within a function can be global or local.

If within a function, the variables are declared by var are considered local and the variables that have not been declared by var, are automatically transformed into global variables.

Therefore, you can redo the code from the first example so that it displays the message correctly. To do this, simply define the variable within the function without the reserved word var, so that it becomes a global variable:

```
function creates Message () {
message = "Test message";
}
createMessage (); alert (message);
```

What happens if a function defines a local variable with the same name as a global variable that already exists? In this case, the local variables prevail over the global ones, but only within the function:

var message = "wins the one outside";

function sample Message () {

var message = "win the one inside"; alert (message);

}

alert (message); Sample Message (); alert (message);

The previous code shows the following messages on the screen:

the outside one wins the inside one wins the outside one

Within the function, the local variable called message has more priority than the global variable of the same name, but only within the function.

What happens if a global variable is defined within a function with the same name as another global variable that already exists? In this other case, the global variable defined within the function simply modifies the value of the global variable defined above: variable

var message = "wins the outside"; function sample Message () {

message = "win the inside"; alert (message);

}

alert (message); Sample Message (); alert (message);

In this case, the messages shown are:

the one from outside wins the one from inside wins the one from inside

The general recommendation is to define as local variables all the variables that are exclusively used to perform the tasks entrusted to each function. Global variables are used to share variables between functions easily.

Break And Continue Statements

The control structure for is very simple to use, but it has the disadvantage that the number of repetitions that can be performed can only be controlled by the variables defined in the loop update zone.

The break and continue statements allow you to manipulate the normal behavior of the for loops to stop the loop or to skip some repetitions. Specifically, the break statement allows you to abruptly terminate a loop and the continue statement allows you to skip some repetitions of the loop.

The following example shows the use of the break statement:

var string = "In a spot of the Stain whose name I don't want to remember ..."; var letters = string.split (""); var result = "";

for (i in letters) {

if (letters [i] == 'a') {

break; } else {

result + = letters [i];

}}

alert (result); // shows "In a lug"

If the program reaches a break type instruction, it immediately exits the loop and continues executing the rest of the instructions outside the for loop. In the previous example, all the letters of a text string are traversed and when it encounters the first letter "a", the execution of the for loop is stopped.

The utility of break is to end the execution of the loop when a variable takes a certain value or when some condition is met.

Sometimes, what is desired is to skip some repetition of the loop when some conditions occur. Following the previous example, it is now desired that the output text remove all the letters "a" from the original text:

stringvar string = "In a spot of the Stain whose name I don't want to remember ..."; var letters = string.split (""); var result = "";

```
for (i in letters) {
if (letters [i] == 'a') {
continue; } else {
result + = letters [i];
}} alert (result);
// shows "In a lugr of l Mnch whose name I don't want to cordr ..."
```

In this case, when a letter "a" is found, the loop is not terminated, but the instructions of that repetition are not executed and Go directly to the next repetition of the for loop.

The utility of continue is that it allows you to use the for loop to filter the results based on some conditions or when the value of a variable matches a certain value.

Other Control Structures

The flow control structures that have been seen (if, else, for) and the sentences that modify their behavior (break, continue) are not sufficient to perform some complex tasks and other types of repetitions. For this reason, JavaScript provides other flow control structures that are different and in some cases more efficient.

While Structure

The while structure allows you to create loops that run no more or more times, depending on the condition indicated. Its formal definition is:

```
while (condition) {...
}
```

The operation of the while loop is summarized in: "While the indicated condition is fulfilled, repeat the instructions included within the loop indefinitely".

If the condition is not met even the first time, the loop is not executed. If the condition is met, the instructions are executed once and the condition is checked again. If the condition is still fulfilled, the

loop is executed again and so it continues until the condition is not met.

Obviously, the variables that control the condition must be modified within the loop itself, since otherwise, the condition would always be met and the while loop would be repeated indefinitely.

Structure do ... while

The loop of type do ... while is very similar to the while loop, except that in this case the instructions of the loop are always executed at least the first time. Its formal definition is:

do {

} while (condition);

In this way, as the condition is checked after each repetition, the loop instructions are always executed the first time. It is important not to forget that after the while () the character must be added; (contrary to what happens with the simple while loop).

Using this loop you can easily calculate the factorial of a number:

var result = 1; var number = 5;

do {

result * = number; // result = result * number number--; } while (number> 0); alert (result);

In the previous code, the result is multiplied in each repetition by the value of the variable number. In addition, in each repetition, the value of this variable number is decremented. The condition of the do ... while loop is that the number value is greater than 0 since the factorial of a number multiplies all numbers less than or equal to itself, but up to number 1 (the factorial of 5, for example, is 5 x 4 x 3 x 2 x 1 = 120).

As in each repetition, the value of the variable number is decremented and the condition is that number is greater than zero, in the repetition in which number is worth 0, the condition is no longer fulfilled and the program leaves the do loop ... while.

Switch Structure

The if ... else structure can be used to perform multiple checks and make complex decisions. However, if all conditions always depend on the same variable, the resulting JavaScript code is too redundant:

if (number == 5) {...} else if (number == 8) {...} else if (number == 20) {...

} else {...

}

In these cases, the switch structure is the most efficient, since it is specially designed to easily handle multiple conditions on the same variable. Its formal definition may seem complex, although its use is very simple.

The switch structure is defined by the reserved keyword switch followed, in parentheses, by the name of the variable to be used in the comparisons. As usual, the instructions that are part of the switch are enclosed in quotes {and}.

Within the switch, all the comparisons that you want to make on the value of the variable are defined. Each comparison is indicated by the reserved word case followed by the value with which the comparison is made.

What happens if no value of the switch variable matches the values defined in the cases? In this case, the default value is used to indicate the instructions that are executed in the case where no case is met for the indicated variable.

Although default is optional, switch structures usually include it to define at least one default value for some variable or to display a message on the screen.

Chapter 4

DOM Document Object Model

The creation of the Document Object Model or DOM is one of the innovations that has most influenced the development of dynamic web pages and the most complex web applications.

DOM allows web programmers to access and manipulate XHTML pages as if they were XML documents. In fact, DOM was originally designed to easily manipulate XML documents.

Despite its origins, DOM has become a utility available for most programming languages (Java, PHP, JavaScript) and whose only differences are in the way of implementing it.

Node Tree

One of the usual tasks in programming web applications with JavaScript is the manipulation of web pages. In this way, it is usual to obtain the value stored by some elements (for example the elements of a form), create an element (paragraphs, , etc.) dynamically and add it to the page, apply an animation to a item (to appear / disappear, to scroll, etc.).

All these usual tasks are very simple to perform thanks to DOM. However, in order to use the DOM utilities, it is necessary to "transform" the original page. A normal HTML page is nothing more than a succession of characters, so it is a very difficult format to manipulate. Therefore, web browsers automatically transform all web pages into a more efficient structure to manipulate.

This transformation is done by all browsers automatically and allows us to use the DOM tools very easily. The reason why the operation of this internal transformation is shown is that it determines the behavior of DOM and therefore, the way in which pages are manipulated.

DOM transforms all XHTML documents into a set of elements called nodes, which are interconnected and that represent the contents of web pages and the relationships between them. Because of its appearance, the union of all nodes is called a "node tree".

It becomes the next tree of nodes. The root of the node tree of any XHTML page is always the same: a special type node called "Document".

From that root node, each XHTML tag is transformed into a node of type "Element". The conversion of tags into nodes is done hierarchically. In this way, only the HEAD and BODY nodes can derive from the root node. From this initial derivation, each XHTML tag is transformed into a node that derives from the node corresponding to its "parent tag".

The transformation of the usual XHTML tags generates two nodes: the first is the "Element" type node (corresponding to the XHTML tag itself) and the second is a "Text" type node that contains the text enclosed by that XHTML tag.

Thus, the following XHTML tag:

<title> Simple page </title> Generates the following two nodes:

 This page is very simple   It generates the following nodes:

▪ Type node "Element "corresponding to the tag.

▪ Node of type "Text" with the textual content of the tag.

▪ Since the content of includes another XHTML tag inside it, the inner tag is transformed into a node of type "Element" that represents the tag and derived from the previous node.

▪ The content of the tag generates another node of type "Text" that derives from the node generated by .

The automatic transformation of the page into a node tree always follows the same rules:

▪ The XHTML tags are transformed into two nodes: the first is the tag itself and the second node is the son of the first and consists of the textual content of the label.

▪ If an XHTML tag is inside another one, the same procedure as above is followed, but the generated nodes will be child nodes of its

parent tag.

As you can guess, the usual XHTML pages produce trees with thousands of nodes. Even so, the transformation process is fast and automatic, being the functions provided by DOM (which will be seen later) the only ones that allow access to any node of the page easily and immediately.

Node Types

The full DOM specification defines 12 types of nodes, although the usual XHTML pages can be manipulated by handling only four or five types of nodes:

▪ Document, root node from which all other nodes in the tree derive.

▪ Element, represents each of the XHTML tags. It is the only node that can contain attributes and the only one from which other nodes can derive.

▪ Attr, a node of this type is defined to represent each of the attributes of the XHTML tags, that is, one for each attribute = value pair.

▪ Text, node that contains the text enclosed by an XHTML tag.

▪ Comment, represents the comments included in the XHTML page.

The other types of existing nodes that are not to be considered are DocumentType, CDataSection, DocumentFragment, Entity, EntityReference, ProcessingInstruction, and Notation.

Direct Access To The Nodes

Once the complete DOM node tree is built automatically, it is now possible to use the DOM functions to directly access any node in the tree. How to access a node in the tree is equivalent to accessing "a piece" of the page, once the tree is built, it is already possible to easily manipulate the page: access the value of an element, set the value of an element, move an element of the page, create and add new elements, etc.

DOM provides two alternative methods to access a specific node: access through its parent nodes and direct access.

The functions that DOM provides to access a node through its parent nodes consist of accessing the root node of the page and then its child nodes and the child nodes of those children and so on until the last node of the branch terminated by The searched node. However, when you want to access a specific node, it is much faster to directly access that node and not reach the node through all its parent nodes.

For this reason, the functions necessary for hierarchical node access will not be presented and only those that allow direct access to the nodes are shown.

Finally, it is important to remember that access to the nodes, their modification and their elimination are only possible when the DOM tree has been completely built, that is, after the XHTML page is fully loaded.

Normally the name attribute is unique to HTML elements that define it, so it is a very practical method to directly access the desired node. In the case of HTML radiobutton elements, the name attribute is common to all radiobuttons that are related, so the function returns a collection of elements.

Internet Explorer 6.0 does not correctly implement this function, since it only takes it into account for elements of type <input> and <img>. In addition, it also takes into consideration elements whose id attribute is equal to the function parameter.

getElementById ()

The getElementById () function is the most used when developing dynamic web applications. This is the preferred function to directly access a node and be able to read or modify its properties.

The getElementById () function returns the XHTML element whose id attribute matches the parameter indicated in the function. Since the id attribute must be unique for each element of the same page,

the function returns only the desired node. var header = document.getElementById ("header");

 ...

The getElementById () function is so important and so used in all web applications, that almost all the examples and exercises that follow use it constantly.

Internet Explorer 6.0 also incorrectly interprets this function, since it also returns those elements whose name attribute matches the parameter provided to the function.

Creating And Deleting Nodes

Accessing the nodes and their properties (which will be seen later) is only part of the usual manipulations on the pages. The other usual operations are to create and delete nodes from the DOM tree, that is, create and delete "chunks" from the web page.

Creating simple XHTML elements

As we have seen, a simple XHTML element, such as a paragraph, generates two nodes: the first node is of the Element type and represents the <p> tag and the second node is of the Text type and represents the textual content of the <p> tag.

For this reason, creating and adding a new simple XHTML element to the page consists of four different steps:

1. Creation of an Element type node that represents the element.
2. Creation of a node of type Text that represents the content of the element.
3. Add the Text node as the child node of the Element node.
4. Add the Element node to the page, in the form of a child node of the node corresponding to the place where you want to insert the element.

Thus, if you want to add a simple paragraph at the end of an XHTML page, it is necessary to include the following JavaScript code:

// Create node of type Element var paragraph = document.createElement ("p");

// Create node of type Text var content = document.createTextNode ("Hello World!");

// Add the Text node as a child of the Element node parrafo.appendChild (content);

// Add the Element node as a child of the document.body.appendChild (paragraph) page;

The process of creating new nodes can become tedious since it involves the use of three DOM functions:

▪ createElement (tag): it creates a node of type Element that represents the XHTML element whose tag is passed as a parameter.

▪ createTextNode (content): Creates a node of type Text that stores the textual content of the XHTML elements.

▪ parent node.appendChild (son node): add a node as a child of another node.

Deleting Nodes

Fortunately, removing a node from the DOM tree on the page is much easier than adding it. In this case, it is only necessary to use the removeChild () function:

var paragraph = document.getElementById ("provisional"); paragraph.parentNode.removeChild (paragraph);

 ...

The removeChild () function requires the node to be deleted as a parameter. In addition, this function must be invoked from the parent element of that node to be deleted. The safest and fastest way to access the parent node of an element is through the nodeHijo.parentNode property.

Thus, to remove a node from an XHTML page, the removeChild () function is invoked from the parentNode value of the node to be deleted. When a node is deleted, all the child nodes it has, are automatically deleted, so it is not necessary to manually delete each child node.

Direct Access To XHTML Attributes

Once a node has been accessed, the next natural step is to access and/or modify its attributes and properties. Through DOM, it is possible to easily access all XHTML attributes and all CSS properties of any element of the page.

The XHTML attributes of the page elements are automatically transformed into properties of the nodes. To access its value, simply indicate the name of the XHTML attribute behind the name of the node.

Chapter 5

Events

So far, all the applications and scripts that have been created have something in common: they are executed from the first instruction to the last sequentially. Thanks to the flow control structures (if, for, while) it is possible to slightly modify this behavior and repeat some pieces of the script and skip other pieces depending on some conditions.

These types of applications are not very useful since they do not interact with users and cannot respond to the different events that occur during the execution of an application. Fortunately, web applications created with the JavaScript language can use the event-based programming model.

In this type of programming, the scripts are dedicated to waiting for the user to "do something" (press a key, move the mouse, close the browser window). Next, the script responds to the user's action by normally processing that information and generating a result.

Events make it possible for users to transmit information to programs. JavaScript defines numerous events that allow a complete interaction between the user and the web pages/applications. Pressing a key constitutes an event, as well as clicking or moving the mouse, selecting an element of a form, resizing the browser window, etc.

JavaScript allows you to assign a function to each of the events. In this way, when any event occurs, JavaScript executes its associated function. These types of functions are called "event handlers" in English and are usually translated as "event handlers".

Event Models

Creating web pages and applications has always been much more complex than it should be due to incompatibilities between browsers. Although there are dozens of standards for the technologies used, browsers do not fully support them or even ignore them.

The main incompatibilities occur in the XHTML language, in the support of CSS stylesheets and, above all, in the implementation of JavaScript. Of all of them, the most important incompatibility occurs precisely in the browser's event model. Thus, there are up to three different models to handle events depending on the browser in which the application runs.

Basic Event Model

This simple event model was introduced for version 4 of the HTML standard and is considered part of the most basic level of DOM. Although its features are limited, it is the only model that is compatible in all browsers and therefore, the only one that allows you to create applications that work the same way in all browsers.

Standard Event Model

The most advanced versions of the DOM standard (DOM level 2) define a completely new and much more powerful event model than the original. All modern browsers include it, except Internet Explorer.

Internet Explorer Event Model

Internet Explorer uses its own event model, which is similar but incompatible with the standard model. It was first used in Internet Explorer 4 and Microsoft decided to continue using it in the other versions, despite the fact that the company had participated in the creation of the DOM standard that defines the standard event model.

Basic Event Model

Types Of Events

In this model, each XHTML element or tag defines its own list of possible events that can be assigned. The same type of event (for example, clicking the left mouse button) can be defined for several different XHTML elements and the same XHTML element can have several different events associated.

The name of each event is constructed using the prefix on, followed by the English name of the action associated with the event. Thus, the event of clicking an element with the mouse is called onclick and

the event associated with the action of moving the mouse is called onmousemove.

The most used events in traditional web applications are onload to wait for the page to load completely, the events onclick, onmouseover, onmouseout to control the mouse and onsubmit to control the submission of forms.

The typical actions that a user performs on a web page can lead to a succession of events. Pressing for example on a button of type <input type = "submit"> triggers the events onmousedown, onclick, onmouseup and onsubmit consecutively.

Event Handlers

A JavaScript event by itself lacks utility. For events to be useful, JavaScript functions or code must be associated with each event. In this way, when an event occurs, the indicated code is executed, so the application can respond to any event that occurs during its execution.

The functions or JavaScript code defined for each event are called "event handler" and since JavaScript is a very flexible language, there are several different ways to indicate the handlers:

- Handlers as attributes of the XHTML elements.
- Handlers as external JavaScript functions.
- "Semantic" handlers.

Event handlers as XHTML attributes

This is the simplest and least professional method of indicating the JavaScript code that should be executed when an event occurs. In this case, the code is included in an attribute of the XHTML element itself. In the following example, we want to show a message when the user clicks on a button:

<input type = "button" value = "Click me and you will see" onclick = "alert ('Thanks for clicking');" />

In this method, XHTML attributes are defined with the same name as the events to be handled. The previous example only wants to control the event of clicking with the mouse, whose name is onclick. Thus, the XHTML element for which you want to define this event must include an attribute called onclick.

The content of the attribute is a text string that contains all the JavaScript instructions that are executed when the event occurs. In this case, the JavaScript code is very simple (alert ('Thanks for clicking');), since it is only about displaying a message.

In this other example, when the user clicks on the element a message is displayed and when the user hovers the mouse over the element, another message is displayed:

You can click on this element or just hover over the mouse

This other example includes one of the most used instructions in older JavaScript applications :

<body onload = "alert ('The page has been fully loaded');"> ...

</body>

The previous message is displayed after the page has been fully loaded, that is after it has been loaded downloaded your HTML code, your images and any other object included in the page.

The onload event is one of the most used since, as seen in the DOM chapter, the functions that allow access and manipulation of the nodes of the DOM tree are only available when the page has been fully loaded.

Event handlers and ‘this’ Variable

JavaScript variable defines a special variable called this that is created automatically and used in some advanced programming

techniques. In events, the variable this can be used to refer to the XHTML element that caused the event.

Event handlers as external functions

The definition of event handlers in XHTML attributes is the simplest but least advisable method of dealing with events in JavaScript. The main drawback is that it is complicated in excess as soon as a few instructions are added, so it is only recommended for the simplest cases.

If complex applications are made, such as the validation of a form, it is advisable to group all the JavaScript code into an external function and call this function from the XHTML element.

Following the previous example that shows a message when clicking on a button:

<input type = "button" value = "Click me and you will see" onclick = "alert ('Thanks for clicking');" /> Using external functions can be transformed into:

function sample Message () {

alert ('Thanks for clicking');

}

<input type = "button" value = "Click me and you will see" onclick = "sample Message ()" />

This technique consists of extracting all JavaScript instructions and grouping them into an external function. Once the function is defined, the function name is included in the attribute of the XHTML element, to indicate that it is the function that is executed when the event occurs.

The function call is made in the usual way, indicating its name followed by the parentheses and optionally, including all the necessary arguments and parameters.

The main drawback of this method is that in the external functions it is not possible to continue using the variable this and therefore, it is

necessary to pass this variable as a parameter to the function.

In the previous example, the external function is called with the parameter this, which within the function is called element. The complexity of the example is mainly due to the way in which different browsers store the value of the borderColor property.

While Firefox stores (in case the four edges match in color) the black value, Internet Explorer stores it as black black black black and Opera stores its hexadecimal representation # 000000.

Semantic event handlers

The methods that have been seen to add event handlers (as XHTML attributes and as external functions) have a serious drawback: they "dirty" the XHTML code of the page.

As is known, one of the basic good practices in the design of web pages and applications is the separation of content (XHTML) and its appearance or presentation (CSS). Whenever possible, it is also recommended to separate the contents (XHTML) and its behavior or programming (JavaScript).

Mixing the JavaScript code with the XHTML elements only helps to complicate the source code of the page, make it difficult to modify and maintain the page and reduce the semantics of the final document produced.

Fortunately, there is an alternative method to define JavaScript event handlers. This technique is an evolution of the method of external functions, since it is based on using the DOM properties of XHTML elements to assign all external functions that act as event handlers. So, the following example:

<input id = "clickable" type = "button" value = "Click me and you will see" onclick = "alert ('Thanks for clicking');" />

It can be transformed into:

// External function function sample Message () {

alert ('Thanks for clicking');

}

// Assign the external function to the document.getElementById ("clickable") element. Onclick = sample Message;

// XHTML element

<input id = "pinchable" type = "button" value = "Click and see" /> The technique of semantic handlers consists of:

1. Assign a unique identifier to the XHTML element using the id attribute.
2. Create a JavaScript function responsible for handling the event.
3. Assign the external function to the corresponding event in the desired element.

The last step is the key to this technique. First, you get the element to which you want to associate the external function: document.getElementById ("clickable");

Next, a property of the element with the same name as the event to be handled is used. In this case, the property is onclick: document.getElementById ("clickable"). Onclick = ...

Finally, the external function is assigned by its name without parentheses. The most important thing (and the most common cause of errors) is to indicate only the name of the function, that is, dispense with the parentheses when assigning the function: document.getElementById ("clickable"). Onclick = sample Message;

If the parentheses are added after the name of the function, the function is actually running and saving the value returned by the function in the element onclick property.

// Assign an external function to an event of a document.getElementById ("pinchable") element. Onclick = sample Message;

// Execute a function and save its result in a property of a document.getElementById ("pinchable") element. Onclick =

sampleMessage ();

The great advantage of this method is that the resulting XHTML code is very "clean" since it does not mix with the JavaScript code. In addition, within the assigned external functions, the variable this can be used to refer to the element that causes the event.

The only drawback of this method is that the page must be fully loaded before the DOM functions assigned by the handlers to the XHTML elements can be used. One of the easiest ways to ensure that certain code is to be executed after the page is fully loaded is to use the onload event:

window.onload = function () {

document.getElementById ("clickable"). Onclick = sample Message ;
}

The prior art uses the concept of anonymous functions, which is not going to be studied, but which allows to create a compact and very simple code. To ensure that a JavaScript code is to be executed after the page has been fully loaded, you only need to include those instructions between the {and} symbols:

window.onload = function () {...

}

In the following example, you add events to elements of type input = text of a complex form:

function highlights () {

// JavaScript code}

window.onload = function () {

var form = document.getElementById ("form"); var fieldsInput = form.getElementsByTagName ("input");

for (var i = 0; i <fieldsInput.length; i ++) {

if (fieldsInput [i] .type == "text") {fieldsInput [i] .onclick = highlights;

}

```
}
}
```

Obtaining Event Information (Event Object)

Normally, event handlers require additional information to process their tasks. If a function, for example, is responsible for processing the onclick event, you may need to know what position the mouse was at the time of clicking the button.

However, the most common case in which it is necessary to know additional information about the event is that of the events associated with the keyboard. Normally, it is very important to know the key that has been pressed, for example, to differentiate the normal keys from the special keys (ENTER, tab, Alt, Ctrl., Etc.).

JavaScript allows you to obtain information about the mouse and keyboard using a special object called event. Unfortunately, different browsers have very notable differences in the treatment of information about events.

The main difference lies in the way in which the event object is obtained. Internet Explorer considers that this object is part of the window object and the rest of browsers consider it as the only argument that the event handling functions have.

Although it is a behavior that is very strange at first, all modern browsers except Internet Explorer magically and automatically create an argument that is passed to the managing function, so it is not necessary to include it in the call to the managing function. Thus, to use this "magic argument", it is only necessary to assign it a name, as browsers automatically create it.

In summary, in Internet Explorer type browsers, the event object is obtained directly by: var event = window.event;

On the other hand, in the rest of browsers, the event object is obtained magically from the argument that the browser automatically creates:

```
function handlerEvents (theEvent) {
```

var event = theEvent;

}

If you want to program an application that works correctly in all browsers, it is necessary to obtain the event object correctly according to each browser. The following code shows the correct way to obtain the event object in any browser:

function handlerEvents (elEvento) {

var event = elEvento || window.event; }

Once the event object is obtained, all the information related to the event can be accessed, which depends on the type of event produced.

Information about the event

The type property indicates the type of event produced, which is useful when the same function is used to handle several events: var type = event.type;

The type property returns the type of event produced, which is equal to the name of the event but without the prefix on.

Using this property, the previous example in which a section of contents was highlighted when you hover the mouse over.

Information about keyboard events

Of all the events available in JavaScript, keyboard-related events are the most incompatible between different browsers and therefore, the most difficult to handle. First, there are many differences between browsers, keyboards, and user operating systems, mainly due to differences between languages.

In addition, there are three different events for the keystrokes (onkeyup, onkeypress, and onkeydown). Finally, there are two types of keys: normal keys (such as letters, numbers and normal symbols) and special keys (such as ENTER, Alt, Shift, etc.)

When a user presses a normal key, three events occur in a row and in this order: onkeydown, onkeypress and onkeyup. The onkeydown

event corresponds to the fact of pressing a key and not releasing it; the onkeypress event is the key press itself and the onkeyup event refers to the release of a key that was pressed.

The easiest way to obtain information about the key that has been pressed is through the onkeypress event. The information provided by the onkeydown and onkeyup events can be considered as more technical since they return the internal code of each key and not the character that has been pressed.

Below is a list with all the different properties of all keyboard events in both Internet Explorer and other browsers:

Keydown event:

- Same behavior in all browsers:
- KeyCode property: internal code of the key
- CharCode property: not defined
- Keypress event:
- Internet Explorer:
- keyCode property: the character code of the key that was pressed
- charCode property: not defined

Other browsers:

- keyCode property: for normal keys, not defined. For special keys, the internal code of the key.
- charCode property: for normal keys, the character code of the key that was pressed. For special keys, 0.
- Keyup event:
- Same behavior in all browsers:
- KeyCode property: internal code of the key
- charCode property: not defined

To convert the code of a character (not to be confused with the internal code) when character representing the key that was pressed, the String.fromCharCode () function is used.

When you press the a key in the Firefox browser, the following sequence of events is displayed:

---------- ----------

Event type: keydown KeyCode

property: 65

charCode property: 0

Character pressed:?

Event type: keypress

KeyCode property: 0

charCode property: 97

Character pressed: a

Event type: keyup KeyCode

property : 65

charCode property: 0 Character pressed:?

Pressing the A key (the same key, but having previously activated the capital letters) shows the following sequence of events in the Firefox browser:

---------- ----------

Event type: keydown KeyCode

property: 65

charCode property: 0 Character pressed:?

Event type: keypress

KeyCode property: 0

charCode property: 65

Character pressed: A

Event type: keyup KeyCode

property : 65

charCode property: 0 Character pressed:?

In the keydown and keyup events, the keyCode property is still valid in both cases. The reason is that keyCode stores the internal code of the key, so if the same key is pressed, the same code is obtained, regardless of the fact that the same key can produce different characters (for example, upper and lower case).

In the keypress event, the value of the charCode property varies, since character a is not the same as character A. In this case, the value of charCode matches the ASCII code of the pressed character.

Following the Firefox browser, if a special key is pressed, such as the tab, the following information is displayed:

Event type: keydown KeyCode

property: 9

charCode property: 0 Character pressed:?

Event type: keypress

KeyCode property: 9

charCode property: 0

Character pressed:?

Event type: keyup KeyCode

property: 9

charCode property: 0 Character pressed:?

The special keys do not have the charCode property, since only the internal code of the key pressed in the keyCode property is saved, in this case, code 9. If the Enter key is pressed, code 13 is obtained, the key the upper arrow produces code 38, etc. However, depending on the keyboard used to press the keys and depending on the arrangement of the keys depending on the language of the keyboard, these codes may vary.

The result of the execution of the same example above in the Internet Explorer browser is shown below. Pressing the a key, the following information is obtained:

Event type: keydown KeyCode

property: 65

charCode property: undefined Character pressed:

--- ----

Event type: keypress

KeyCode property: 97

charCode property: undefined Character pressed:

--- --------

Event type: keyup KeyCode

property: 65

charCode property: undefined Character pressed:

The keyCode property in the keypress event contains the ASCII code of the key character, so the character can be obtained directly using String.fromCharCode (keyCode).

If the A key is pressed, the information shown is identical to the previous one, except that the code that shows the keypress event changes to 65, which is the ASCII code of the A key:

---------- -------------------------

Event type: keydown KeyCode

property: 65

charCode property: undefined Character pressed:

------- -----------------------------

Event type: keypress

KeyCode property: 65

charCode property: undefined Character pressed:

--- ---------------------------------

Event type: keyup KeyCode

property: 65

charCode property: undefined Character pressed :

When you press a special key like the tab, Internet Explorer displays the following information:

------------------------------ -----

Event type: keydown KeyCode

property: 9

charCode property: undefined Character pressed:

The codes shown for the special keys match those of Firefox and other browsers but remember that they may vary depending on the keyboard that is used and in the function of the arrangement of the keys for each language.

Finally, the altKey, ctrlKey and shiftKey properties store a Boolean value that indicates whether any of those keys were pressed when the keyboard event occurred. Surprisingly, these three properties work the same way in all browsers:

```
if (event.altKey) {
alert ('The ALT key was pressed'); }
```

Below is the case in which the Shift key is pressed and without releasing it, you press on the key that contains the number 2 (in this case, it refers to the key that is at the top of the keyboard and by therefore, it does not refer to the one found on the numeric keypad). Both Internet Explorer and Firefox show the same sequence of events:

Event type: keydown KeyCode

property: 16

charCode property: 0 Character pressed:?

Event type: keydown KeyCode

property: 50

charCode property: 0 Character pressed:?

Event type: keypress

KeyCode property: 0

charCode property: 34

Character pressed: "

Event type: keyup KeyCode

property : 50

charCode property: 0 Character pressed:?

Type of event: keyup

Property keyCode: 16

Property charCode: 0 Character pressed:?

The keypress event is the only one that allows to obtain the really pressed character, since when pressing on key 2 having previously pressed the Shift key, the character is obtained ", which is precisely the one that shows the keypress event.

The following JavaScript code allows you to correctly obtain in any browser the character corresponding to the key pressed:

function handler (elEvento) {

var event = elEvento || window.event; var character = event.charCode || event.keyCode; alert ("The clicked character is:" + String.fromCharCode (character));

} document.onkeypress = handler;

Information about mouse events

The most relevant information about mouse-related events is the coordinates of the mouse pointer position. Although the origin of the coordinates is always in the upper left corner, the point taken as a reference of the coordinates may vary.

In this way, it is possible to obtain the position of the mouse with respect to the computer screen, with respect to the browser window and with respect to the HTML page itself (which is used when the user has scrolled over the page). The simplest coordinates are those that refer to the position of the pointer with respect to the browser window, which are obtained through the clientX and clientY properties:

function showsInformation (the Event) {

var event = the Event || window.event; var coordinateX = event.clientX; var coordinateY = event.clientY; alert ("You clicked on the position:" + coordinate X + "," + coordinate Y);

} document.onclick = sampleInformation;

The coordinates of the position of the mouse pointer with respect to the full screen of the user's computer are obtained in the same way,

using the screenX and screenY properties:

var coordinateX = event.screenX; var coordinateY = event.screenY;

In many cases, it is necessary to obtain another pair of different coordinates: those corresponding to the position of the mouse with respect to the origin of the page. These coordinates do not always coincide with the coordinates regarding the origin of the browser window since the user can scroll over the web page. Internet Explorer does not provide these coordinates directly, while other browsers do. In this way, it is necessary to detect if the browser is Internet Explorer type and if so, perform a simple calculation.

The ie variable is true if the browser in which the script is run is of type Internet Explorer (any version) and false otherwise. For the rest of the browsers, the coordinates regarding the origin of the page are obtained using the pageX and pageY properties. In the case of Internet Explorer, they are obtained by adding the position with respect to the browser window (clientX, clientY) and the page scrolling (document.body.scrollLeft, document.body.scrollTop).

Chapter 6

Forms

Programming applications that contain web forms has always been one of the fundamental tasks of JavaScript. In fact, one of the main reasons why the JavaScript programming language was invented was the need to validate the form data directly in the user's browser. In this way, it was avoided reloading the page when the user made mistakes when filling out the forms.

However, the appearance of AJAX applications has relegated the treatment of forms as the main activity of JavaScript. Now, the main use of JavaScript is that of asynchronous communications with servers and that of dynamic manipulation of applications. In any case, the handling of the forms is still an essential requirement for any JavaScript programmer.

Basic Properties Of Forms And Elements

JavaScript has numerous properties and functions that facilitate the programming of applications that handle forms. First, when a web page is loaded, the browser automatically creates an array called forms and that contains the reference to all forms on the page.

To access the forms array, the document object is used, so document.forms is the array that contains all the forms on the page. As it is an array, access to each form is done with the same syntax of the arrays. The following instruction accesses the first form of the page: document.forms [0];

In addition to the forms array, the browser automatically creates an array called elements for each of the forms on the page. Each array element contains the reference to all the elements (text boxes, buttons, drop-down lists, etc.) of that form. Using the syntax of the

arrays, the following instruction obtains the first element of the first form of the page: document.forms [0] .elements [0];

Array syntax is not always so concise. The following example shows how to directly obtain the last element of the first form of the page: document.forms [0] .elements [document.forms [0] .elements.length-1];

Although this way of accessing the forms is quick and simple, it has a very serious inconvenience. What happens if the page layout changes and in the HTML code the order of the original forms is changed or new forms are added? The problem is that "the first form of the page" could now be another form different from what the application expects.

In an environment as changing as web design, it is very difficult to trust that the order of forms remains stable on a web page. For this reason, access to forms on a page through the document.forms array should always be avoided.

One way to avoid the problems of the previous method is to access the forms of a page through its name (name attribute) or through its id attribute. The document object allows direct access to any form through its name attribute:

var formPrincipal = document.formulario; var Secondary form = document.otro_formulario;

<form name = "form"> ...

</form>

<form name = "other_form"> ...

</form>

Accessing the forms on the page in this way, the script works correctly even if the forms are reordered or new forms are added to the page. The elements of the forms can also be accessed directly through their name attribute:

var formPrincipal = document.form; var first Element = document.form.element;

<form name = "form">

<input type = "text" name = "element" />

</form>

Obviously, you can also access the forms and their elements using the DOM functions of direct access to the nodes. The following example uses the usual document.getElementById () function to directly access a form and one of its elements:

var formPrincipal = document.getElementById ("form"); var firstElement = document.getElementById ("element");

<form name = "form" id = "form">

<input type = "text" name = "element" id = "element" /> </form>

Regardless of the method used to obtain the reference to a form element, Each element has the following useful properties for application development:

- **type:** indicates the type of element being treated. For elements of type <input> (text, button, checkbox, etc.), it matches the value of its type attribute. For normal drop-down lists (<select> element) their value is select-one, which allows them to be differentiated from the lists that allow selecting several elements at once and whose type is select-multiple. Finally, in elements of type <textarea>, the value of type is textarea.
- **form:** is a direct reference to the form to which the element belongs. Thus, to access the form of an element, you can use document.getElementById ("element_id"). Form
- **name:** get the value of the name attribute of XHTML. Only its value can be read, so it cannot be modified.
- **value:** allows you to read and modify the value of the value attribute of XHTML. For the text fields (<input type = "text"> and <textarea>) get the text that the user has written. For the buttons you get the text shown on the button. For the elements checkbox and radiobutton it is not very useful, as will be seen later.

Finally, the most used events in the handling of the forms are the following:

▪ **onclick:** event that occurs when you click with the mouse on an element. Normally it is used with any of the types of buttons that allow to define XHTML

(<input type = "button">, <input type = "submit">, <input type = "image">).

▪ **onchange:** event that occurs when the user changes the value of a text element (<input type = "text"> or <textarea>). It also occurs when the user selects an option from a drop-down list (<select>). However, the event only occurs if after making the change, the user moves to the next field of the form, which is technically known as "the other form field has lost focus."

▪ **onfocus:** event that occurs when the user selects an element of the form.

▪ **onblur:** complementary onfocus event, since it occurs when the user has deselected an element by selecting another element of the form. Technically, it is said that the previous element "has lost focus."

Basic Utilities For Forms

Obtain the value of form fields

Most JavaScript techniques related to forms require reading and / or modifying the value of form fields. Therefore, below is how to get the value of the most used form fields.

Text box and textarea

The value of the text displayed by these elements is obtained and set directly by the value property.

Radiobutton

When a group of radiobuttons is available, one generally does not want to obtain the value of the value attribute of any of them, but the important thing is to know which of all the radiobuttons has been selected. The checked property returns true for the selected radiobutton and false in any other case.

Checkbox

Checkbox elements are very similar to radiobuttons, except that in this case, you should check each checkbox independently of the rest. The reason is that radiobutton groups are mutually exclusive and only one of them can be selected at a time. For their part, checkboxes can be selected independently from the others.

If the following checkboxes are available:

<input type = "checkbox" value = "conditions" name = "conditions" id = "conditions" /> I have read and accept the conditions

<input type = "checkbox" value = "privacy" name = "privacy" id = "privacy" /> I have read the privacy policy

Using the checked property, it is possible to check if each checkbox has been selected:

var element = document.getElementById ("conditions"); alert ("Element:" + element.value + "\ n Selected:" + element.checked);

element = document.getElementById ("privacy");

alert ("Element:" + element.value + "\ n Selected:" + element.checked);

Select

Drop-down lists (<select>) are the elements in which it is more difficult to obtain their value. If a drop-down list such as the following is available:

<select id = "options" name = "options">

<option value = "1"> First value </option>

<option value = "2"> Second value </ option>

<option value = "3"> Third value </option>

<option value = "4"> Fourth value </option>

</select>

In general, what is required is to obtain the value of the value attribute of the option (<option>) selected by the user. Obtaining this

value is not easy, since a series of steps must be performed. In addition, to obtain the selected value, the following properties must be used:

▪ options, is an array created automatically by the browser for each drop-down list and containing the reference to all the options in that list. In this way, the first option in a list can be obtained through document.getElementById ("list_id"). Options [0].

▪ selectedIndex, when the user selects an option, the browser automatically updates the value of this property, which saves the index of the selected option. The index refers to the array options automatically created by the browser for each list.

// Get the reference to the list var list = document.getElementById ("options");

// Get the index of the option selected var indexSelected = list.selectedIndex;

// With the index and array "options", get the option selected var optionSelected = list.options [indexSelected];

// Get the value and the text of the selected option varSelected text = optionSelected.text; var valueSelected = optionSelected.value;

alert ("Selected option:" + selected text + "\ n Option value:" + selected value);

As has been seen, to obtain the value of the value attribute corresponding to the option selected by the user, it is necessary to perform several steps. However, all necessary steps are usually abbreviated in a single instruction: var list = document.getElementById ("options");

// Get the value of the selected option var Value Selected = list.options [list.selectedIndex] .value;

// Get the text that shows the selected option var value Selected = list.options [list.selectedIndex] .text;

The most important thing is not to confuse the value of the selectedIndex property with the value corresponding to the value

property of the selected option. In the previous example, the first option has a value equal to 1. However, if this option is selected, the value of selectedIndex will be 0, since it is the first option of the array options (and the arrays start counting the elements in the number 0).

Set The Focus On An Element

In programming, when an element is selected and you can write directly to it or you can modify any of its properties, it is said to have the focus of the program.

If a text box of a form has the focus, the user can write directly on it without having to previously click with the mouse inside the box. Likewise, if a drop-down list has the focus, the user can select an option directly by going up and down with the arrow keys.

By repeatedly pressing the TABULATOR key on a web page, the different elements (links, images, form fields, etc.) get the focus of the browser (the selected element each time usually shows a small dotted border).

If on a web page the form is the most important element, such as on a search page or on a page with a form to register, it is considered a good usability practice to automatically assign the focus to the first element of the form when load the page

To assign the focus to an XHTML element, the focus () function is used.

Error Detection And Correction

JavaScript is an interpreted programming language, which means that most errors in the code cannot be detected until the scripts are executed. In this way, before considering a script as correct, it is necessary to test it in all the browsers on which it will be used.

When errors occur during the execution of a script, browsers provide some useful information to discover the exact point at which the error occurred and its possible solution. To solve the errors of a script is called "debug the script" or "debug the script" (term that comes from

the English word "debug", which means "to eliminate the errors of an application").

Unfortunately, not all browsers provide the same useful information, which complicates the solution of errors for each type of browser. Below are the tools provided by each browser to detect and correct errors in JavaScript.

Error correction with Internet Explorer

Depending on your configuration, the Internet Explorer browser may have JavaScript error notification disabled. For this reason, it may first be necessary to activate warning messages about JavaScript errors. To activate notifications, access the Tools> Options menu, Advanced Options tab and activate the option.

Avoid sending a duplicate form

One of the usual problems with the use of web forms is the possibility of the user pressing twice in a row on the "Send" button. If the user's connection is too slow or the response of the server is waiting, the original form is still displayed in the browser and for that reason, the user is tempted to click on the "Send" button again.

In most cases, the problem is not serious and it is even possible to control it on the server, but it can be complicated in important application forms such as those involving economic transactions.

For this reason, a good practice in designing web applications is usually to disable the send button after the first press. The following example shows the necessary code:

<form id = "form" action = "#"> ...

<input type = "button" value = "Send" onclick = "this.disabled = true; this.value = ' Sending ... '; this.form.submit () "/> </form>

When you click on the form submit button, the onclick event occurs on the button and therefore, the JavaScript instructions contained in the onclick attribute:

1. First, the button is disabled by the instruction this.disabled = true ;. This is the only instruction necessary if you only

want to disable a button.

2. Next, the message shown by the button is changed. From the original "Send" is passed to the most appropriate "Sending ..."
3. Finally, the form is sent using the submit () function in the following instruction: this.form.submit ()

The button in the previous example is defined using a button of type <input type = "button" />, since the JavaScript code shown does not work correctly with a button of type <input type = "submit" />. If a submit type button is used, the button is disabled before submitting the form and therefore the form ends without being sent.

Limit the character size of a textarea

The most important lack of form fields of type textarea is the impossibility of limiting the maximum number of characters that can be entered, similar to the maxlength attribute of normal text boxes.

JavaScript allows you to add this feature very easily. First of all, it should be remembered that with some events (such as onkeypress, onclick, and onsubmit), normal behavior can be avoided if false is returned.

Avoiding normal behavior is equivalent to completely modifying the usual behavior of the event. If for example the value false is returned in the onkeypress event, the key pressed by the user is not taken into account. If false is returned in the onclick event of an item as a link, the browser does not load the page indicated by the link.

If an event returns the value true, its behavior is the usual one:

```
<textarea onkeypress = "return true;"> </textarea>
```

In the textarea of the previous example, the user can type any character, since the onkeypress event returns true and therefore, its behavior is normal and the pressed key becomes a character within the textarea.

However, in the following example:

```
<textarea onkeypress = "return false;"> </textarea>
```

Since the value returned by the onkeypress event is equal to false, the browser does not execute the default behavior of the event, that is, the Pressed key is not transformed into any character within the textarea. No matter how many times the keys are pressed and the pressed key does not matter, that textarea will not allow you to type any character.

Taking advantage of this feature, it is easy to limit the number of characters that can be written in a textarea type element: it is checked if the maximum number of characters allowed has been reached and if so, the usual behavior of the event is avoided and therefore, additional characters are not added to the textarea:

```
function limit (maximum Characters) {

var element = document.getElementById ("text"); if (element.value.length> = maximum Characters) {

return false;

} else {

return true;

}

}

<textarea id = "text" onkeypress = "return limits (100);"> </textarea>
```

In the previous example, with each key pressed, the total number of characters in the textarea is compared with the maximum number of characters allowed. If the number of characters is equal to or greater than the limit, the value false is returned and therefore, the default behavior of onkeypress is avoided and the key is not added.

Restrict allowed characters in a text box

Sometimes, it may be useful to block certain specific characters in a text box. If for example, a text box expects a number to be entered, it may be interesting not to allow the user to enter any character that is not numeric.

When a key is pressed, it is checked whether the character of that key is within the characters allowed for that <input> element.

If the character is within the allowed characters, true is returned and therefore the onkeypress behavior is usual and the key is written. If the character is not within the allowed characters, false is returned and therefore normal onkeypress behavior is prevented and the key is not written to the input.

In addition, the previous script always allows the pressing of some special keys. In particular, the BackSpace and Delete keys to delete characters and the Left Arrow and Right Arrow keys to move in the text box can always be pressed regardless of the type of characters allowed.

Validation

The main utility of JavaScript in the handling of forms is the validation of the data entered by users. Before sending a form to the server, it is recommended to validate the data inserted by the user through JavaScript. In this way, if the user has made an error when filling out the form, he can be notified instantly, without waiting for the response from the server.

Reporting errors immediately through JavaScript improves user satisfaction with the application (what is technically known as "improving the user experience") and helps reduce the processing load on the server.

Normally, the validation of a form consists of calling a validation function when the user clicks on the form submit button. In this function, it is checked whether the values entered by the user meet the restrictions imposed by the application.

Although there are as many possible checks as different form elements, some checks are very common: that a mandatory field is filled in, that the value of a drop-down list is selected, that the indicated email address is correct, that the date entered is logical, that a number has been entered where required, etc.

Below is the basic JavaScript code needed to incorporate validation into a form:

<form action = "" method = "" id = "" name = "" onsubmit = "return validation ()"> ...

</ form >

And the validation function scheme () is as follows:

function validation () {if (condition that the first field of the form must meet) {// If the condition is not met ... alert ('[ERROR] The field must have a value of ... '); return false;

}

else if (condition that the second field of the form must meet) {// If the condition is not met ... alert ('[ERROR] The field must have a value of ...'); return false;

} ... else if (condition that the last field of the form must meet) {// If the condition is not met ... alert ('[ERROR] The field must have a value of ...'); return false;

}

// If the script has reached this point, all conditions // have been met, so the value true return true is returned;

}

The operation of this validation technique is based on the behavior of the JavaScript onsubmit event. Like other events such as onclick and onkeypress, the event onsubmit varies its behavior depending on the value that is returned.

Thus, if the onsubmit event returns true, the form is sent as it normally would. However, if the onsubmit event returns the value false, the form is not sent. The key to this technique is to check each and every element of the form. When an incorrect item is found, the value false is returned. If no error is found, the value true is returned.

Therefore, the onsubmit event of the form is defined first as: onsubmit = "return validation ()"

Since the JavaScript code returns the value resulting from the validation function (), the form will only be sent to the server if that function returns true In the event that the validation () function returns false, the form will remain unsent.

Within the validation function () all conditions imposed by the application are checked. When a condition is not met, false is returned and therefore the form is not sent. If the end of the function is reached, all conditions have been met correctly, so true is returned and the form is sent.

The notification of mistakes made depends on the design of each application. In the code in the previous example, messages are simply shown by the alert () function indicating the error produced. The best designed web applications show each error message next to the corresponding form element and also usually display a main message indicating that the form contains errors.

Once the scheme of the validation function () is defined, the corresponding code must be added to this function to all the checks carried out on the elements of the form. Below are some of the most common validations of form fields.

Validate a mandatory text field

This is to force the user to enter a value in a text box or text box in which it is mandatory.

For a mandatory text field to be completed, it is checked that the value entered is valid, that the number of characters entered is greater than zero, and that only blank spaces have not been entered.

The reserved word null is a special value that is used to indicate "no value". If the value of a variable is null, the variable does not contain any value of type object, array, numeric, text string or Boolean.

The second part of the condition requires that the text entered be longer than zero characters, that is, that it is not an empty text.

Finally, the third part of the condition (/^\s+$/.test(value)) requires that the value entered by the user is not only made up of blanks. This verification is based on the use of "regular expressions", a common resource in any programming language but which due to its great complexity will not be studied. Therefore, it is only necessary to literally copy this condition, taking special care not to modify any character of the expression.

Validate a text field with numerical values

This involves forcing the user to enter a numerical value in a text box. The JavaScript condition consists of:

value = document.getElementById ("field"). Value; if (isNaN (value)) {

return false;

}

If the content of the value variable is not a valid number, the condition is not met. The advantage of using the internal isNaN () function is that it simplifies checks, since

JavaScript takes care of taking into account decimals, signs, etc.

Validate an email address

This is to force the user to enter an email address with a valid format. What is checked is that the address seems valid, since it is not checked if it is a real and operational email account. The JavaScript condition consists of:

value = document.getElementById ("field"). Value; if (! (/ \ w {1,} [@] [\ w \ -] {1,} ([.] ([\ w \ -] {1,})) {1,3} $ /. test (value))) {

return false;

}

The check is done again using regular expressions since the valid email addresses can be very different. Although the condition is very complex, if it is copied literally, it can be applied to any case in which you need to check an email address.

Validate a date

Dates are usually the most complicated form fields to validate for the multitude of different ways in which they can be entered. The following code assumes that in some way the year, month and day entered by the user have been obtained:

var year = document.getElementById ("year"). Value; var month = document.getElementById ("month"). value; var dia = document.getElementById ("dia"). value; value = new Date (year, month, day);

if (! isNaN (value)) {

return false;

}

The Date function (year, month, day) is an internal JavaScript function that allows you to build dates from the year, month and day of the date. It is very important to keep in mind that the month number is indicated from 0 to 11, with 0 being the month of January and 11 the month of December. The days of the month follow a different numbering since the minimum allowed is 1 and the maximum 31.

The validation consists of trying to construct a date with the data provided by the user. If the user data is not correct, the date cannot be constructed correctly and therefore the form validation will not be correct.

Validate a DNI number

This is to verify that the number provided by the user corresponds to a valid National Identity Document or DNI number. Although for each country or region the requirements of the identity document of the people may vary, a generic example easily adaptable is shown below. The validation must not only verify that the number consists of eight digits and one letter, but it is also necessary to verify that the indicated letter is correct for the entered number:

value = document.getElementById ("field"). Value; var letters = ['T', 'R', 'W', 'A', 'G', 'M', 'Y', 'F', 'P', 'D', 'X', 'B ',' N ',' J ',

' Z ',' S ',' Q ',' V ',' H ',' L ',' C ',' K ',' E ',' T '] ;

if (! (/ ^ \ d {8} [AZ] $ /. test (value))) {

return false;

}

if (value.charAt (8)! = letters [(value.substring (0, 8))% 23]) {

return false;

}

The first check ensures that the format of the number entered is correct, that is, that it consists of 8 consecutive numbers and one letter. If the letter is at the beginning of the numbers, the check would be / ^ [AZ] \ d {8} $ /. If instead of eight numbers and one letter, ten numbers and two letters are required, the check would be / ^ \ d {10} [AZ] {2} $ / and so on.

The second check applies the algorithm for calculating the letter of the DNI and compares it with the letter provided by the user. The algorithm of each identification document is different, so this part of the validation must be adapted accordingly.

Validate a telephone number

Telephone numbers can be indicated in very different ways: with a national prefix, with an international prefix, grouped in pairs, separating numbers with dashes, etc.

The following script considers that a telephone number consists of nine consecutive digits and without spaces or dashes between the figures:

value = document.getElementById ("field"). Value; if (! (/ ^ \ d {9} $ /. test (value))) {

return false;

}

Again, the JavaScript condition is based on the use of regular expressions, which check if the indicated value is a succession of

nine consecutive numbers.

Validate that a checkbox has been selected

If an element of type checkbox is mandatory, JavaScript allows you to check it very easily:

element = document.getElementById ("field"); if (! element.checked) {

return false;

}

Validate that a radiobutton has been selected

Although it is a case similar to that of the checkboxes, the validation of the radiobuttons presents an important difference: in general, the verification that is made is that the user has selected some radiobutton from those that form a certain group Using JavaScript, it is easy to determine if a radiobutton has been selected from a group.

Clocks, Counters And Time Intervals

Sometimes, some web pages display a clock with the current time. If the clock must be updated every second, the time cannot be displayed directly on the HTML page generated by the server. In this case, although there are alternatives made with Java and Flash, the easiest way to do this is to show the time of the user's computer using JavaScript.

To create and display a clock with JavaScript, you must use the internal Date () object to create dates/times and the utilities that allow you to define counters, to update the clock every second.

The Date () object is a utility that provides JavaScript to create dates and times. Once a date type object has been created, it can be manipulated to obtain information or perform calculations with dates. To obtain the current date and time, it is only necessary to create a Date () object without passing any parameters: var dateTime = new Date ();

Using the code above, you can build a very basic clock that does not update its content:

var dateHour = new Date (); document.getElementById ("clock"). innerHTML = dateTime;

When the page is loaded, the previous example would show a text similar to the following in the reserved for the clock:

Thu Aug 02 2007 19:35:19 GMT + 0200 (Summer time romance)

This first built clock has many differences from the clock you want to build. First, it shows much more information than is necessary. In addition, its value is not updated every second, so it is not a very practical clock.

The Date () object provides very useful functions for obtaining information about the date and time. Specifically, there are functions that directly obtain the hour, minutes and seconds.

Using the getHours (), getMinutes () and getSeconds () functions of the Date object, the clock can only display time information. The result of the previous example would be a clock like the following:

20: 9: 21

If the hour, minute or second is less than 10, JavaScript does not add 0 ahead, so the result is not entirely satisfactory.

To complete the clock, you just need to update its value every second. To achieve this, you must use special JavaScript functions that allow you to execute certain instructions when a certain period of time has elapsed.

The setTimeout () function allows you to execute a function after a specified period of time has elapsed. The definition of the function is: setTimeout (functionName, milliseconds);

The function to be executed must be indicated by its name without parentheses and the time that must elapse until it is executed is indicated in milliseconds. In this way, if a function is created to show the time of the clock and it is called Sample Clock (), it can be indicated that it is executed within 1 second.

However, the code simply shows the contents of the clock 1 second after the page loads, so it is not very useful. To execute a function periodically, you use a JavaScript function very similar to setTimeout () that is called setInterval (). Its definition is: setInterval (functionName, milliseconds);

The definition of this function is identical to the setTimeout () function, except that in this case, the programmed function is executed infinitely periodically with a period of time between executions of as many milliseconds as they have been established.

Thus, to build the complete clock, a periodic execution of the sample function Clock () is established every second.

Using the Date object and its functions, it is possible to build "countdown", that is, clocks that show the time left until an event occurs. In addition, the setTimeout () and setInterval () functions can be very useful in other programming techniques.

Conclusion

Thank you for making it through to the end of Javascript For Beginners, let's hope it was informative and able to provide you with all of the tools you need to achieve your goals whatever they may be.

The purpose of this book was to help with Java programming and to ensure that everyone learns to use this language to the fullest. This book is meant to enlighten the knowledge and ensure that there is very little confusion as far as Javascript is concerned. Here's a bit of history on Javascript to end the book.

ECMA (European Computer Manufacturers Association) created the TC39 committee with the objective of "standardizing a cross-platform scripting language and independent of any company". The first standard created by the TC39 committee was called ECMA-262, in which the ECMAScript language was defined for the first time.

For this reason, some programmers prefer the ECMAScript designation to refer to the JavaScript language. In fact, JavaScript is nothing more than the implementation made by the company Netscape of the ECMAScript standard.

The international organization for standardization (ISO) adopted the ECMA-262 standard through its IEC commission, giving rise to the ISO / IEC-16262 standard.

ECMA has published several standards related to ECMAScript.The fourth version of ECMA-262 is currently under development, which could include news such as packages, namespaces, explicit definition of classes, etc.

ECMA has also defined several standards related to ECMAScript, such as the ECMA-357 standard, which defines an extension known as E4X and allows the integration of JavaScript and XML.

Finally, if you found this book useful in any way, a good review on Amazon is always appreciated!

PYTHON PROGRAMMING FOR BEGINNERS:

The ultimate crash course in Python programming. A comprehensive guide to mastering the powerful programming language and learn machine learning.

Introduction

In this Python Beginner's Guide, you're about to learn...

- The Most Vital Basics of Python programming. Rapidly get the dialect and begin applying the ideas to any code that you compose.
- The Useful features of Python for Beginners—including some ideas you can apply to in real world situations and even other programs.
- Different mechanics of Python programming: control stream, factors, records/lexicons, and classes—and why taking in these center standards are essential to Python achievement.
- Protest arranged programming, its impact on present-day scripting languages, and why it makes a difference.

This guide has been composed specifically for Newbies and Beginners. You will be taken through each step of your very first program, and we will explain each portion of the script as you test and analyze the data.

Machine learning is defined as a subset of something called *artificial intelligence* (AI). The ultimate goal of machine learning is to first comprehend the structure of the presented data and align that data into certain models that can then be understood and used by anyone.

Despite the fact that machine learning is a department in the computer science field, it truly is different from normal data processing methods.

In common computing programs, formulas are groups of individually programmed orders that are used by computers to determine outcomes and solve problems. Instead, machine-learning formulas allow computers to focus only on data that is inputted and use

proven stat analysis in order to deliver correct values that fall within a certain probability. What this means is that computers have the ability to break down simple data models which enables it to automate routine decision-making steps based on the specific data that was inputted.

Any innovation client today has profited from machine learning. Facial acknowledgment innovation enables internet based life stages to enable clients to tag and offer photographs of companions.

Optical character acknowledgment (OCR) innovation changes over pictures of content into portable kind. Proposal motors, controlled by machine learning, recommend what motion pictures or TV programs to watch next in view of client inclinations. Self-driving autos that depend on machine learning on how to explore may soon be accessible to shoppers.

Machine learning is a ceaselessly growing field. Along these lines, there are a few things to remember as you work with machine learning philosophies, or break down the effect of machine learning forms.

In this book, we'll look at the normal machine learning strategies for managed and unsupervised learning, the basic algorithmic methodologies including the k-closest neighbor calculation, specific decision tree learning, and deeply impactful techniques. We will also investigate which programming is most used in machine learning, giving you a portion of the positive and negative qualities

Moreover, we'll talk about some important biases that are propagated by machine learning calculations, and consider what can be done to avoid biases affecting your algorithm building.

There are plenty of books on this subject on the market. Thanks for choosing this one! Every effort was made to ensure it’s full of useful information as possible, please enjoy!

Chapter 1:

Basics of the Python Programming Language

Python is an awesome decision on machine learning for a few reasons. Most importantly, it's a basic dialect at first glance. Regardless of whether you're not acquainted with Python, getting up to speed is snappy in the event that you at any point have utilized some other dialect with C-like grammar.

Second, Python has an incredible network which results in great documentation and inviting and extensive answers in Stack Overflow (central!).

Third, coming from the colossal network, there are a lot of valuable libraries for Python (both as "batteries included" an outsider), which take care of essentially any issue that you can have (counting machine learning).

Wait I thought this machine language was slow?

Unfortunately, it is a very valid question that deserves an answer. Indeed, Python is not at all the fastest language on the planet.

However, here's the caveat: libraries can and do offload the costly computations to the substantially more performant (yet much harder to use) C and C++ are prime examples. There's NumPy, which is a library for numerical calculation. It is composed in C, and it's quick. For all intents and purposes, each library out there that includes serious estimations utilizes it—every one of the libraries recorded next utilize it in some shape. On the off chance that you read NumPy, think quick.

In this way, you can influence your computer scripts to run essentially as quick as handwriting them out in a lower level dialect. So there's truly nothing to stress over with regards to speed and agility.

If you want to know which Python libraries you should check out. Try some of these.

“Scikit-learn”

Do you need something that completely addresses everything from testing and training models to engineering techniques?

Then scikit-learn is your best solution. This incredible bit of free programming gives each device important to machine learning and information mining. It's the true standard library for machine learning in Python; suggested for the vast majority of the 'old' ML calculations.

This library does both characterization and relapse, supporting essentially every calculation out there (bolster vector machines, arbitrary timberland, Bayes, you name it). It allows a simple exchanging of calculations in which experimentation is a lot simpler. These 'more seasoned' calculations are shockingly flexible and work extremely well in a considerable amount of problems and case studies.

In any case, that is not all! Scikit-learn additionally does groupings, plural dimensionalities, and so on. It's likewise exceedingly quick since it keeps running on NumPy and SciPy.

Look at a few cases to see everything this library is prepared to do, the instructional exercises on the website, and the need to figure out if this is a good fit.

“NLTK”

While not a machine learning library essentially, NLTK is an unquestionable requirement when working with regular computer language. It is bundled with a heap of Datasets and other rhetorical data assets, which is invaluable for preparing certain models. Aside from the libraries for working with content, this is great for determining capacities, for example, characterization, tokenization, stemming, labeling, and parsing—that's just the beginning.

The handiness of having the majority of this stuff perfectly bundled can't be exaggerated. In case you are keen on regular computer language look at a few of their website's instructional exercises!

“Theano”

Utilized generally in research and within the scholarly community, Theano is the granddad of all deeply profound learning systems. Since it is written in Python, it is firmly incorporated with NumPy.

Theano enables you to make neural systems which are essential scientific articulations with multidimensional clusters. Theano handles this so you that you don't need to stress over the real usage of the math included.

It bolsters offloading figures to a considerably speedier GPU, which is an element that everybody underpins today, yet, back when they presented it, this wasn't the situation. The library is extremely developed now and boasts an extensive variety of activities, which is extraordinary with regards to contrasting it and other comparative libraries.

The greatest grievance out there about Theano is the API might be cumbersome for a few, making the library difficult to use for beginning learners. In any case, there are tools that relieve the agony and makes working with Theano pretty straightforward, for example, try using Keras, or Blocks, and even Lasagne.

“TensorFlow”

The geniuses over at Google made TensorFlow for inside use in machine learning applications and publicly released it in late 2015. They needed something that could supplant their more established, non-open source machine learning structure, *DistBelief*. It wasn't sufficiently adaptable and too firmly ingrained into their foundation. It was to be imparted to different analysts around the globe.

Thus, TensorFlow was made. Despite their slip-ups in the past, many view this library as a much needed change over Theano, asserting greater adaptability and more instinctive API. Another great benefit is it can be utilized to create new conditions, supporting tremendous amounts of new GPUs for training and learning purposes. While it doesn't bolster as wide a scope of functionality like Theano, it has better computational diagram representations.

TensorFlow is exceptionally famous these days. In fact, if you are familiar with every single library on this list, you can agree that there has been a huge influx in the number of new users and bloggers in this library and its functionality. This is definitely a good thing for beginners.

“Keras”

Keras is a phenomenal library that gives a top-level API to neural systems and is best for running alongside or on top of Theano or TensorFlow. It makes bridling full intensity of these intricate bits of programming substantially simpler than utilizing them all by themselves. The greatest benefit of this library is its exceptional ease of understanding, putting the end users’ needs and experiences as its number one priority. This cuts down on a number of errors.

It is also secluded; which means that individual models like neural layers and cost capacities can be grouped together with little to no limitations. This additionally makes the library simple to include new models and interface them with the current ones.

A few people have called Keras great that it is similar to cheating on your exam. In case you're beginning with higher learning in this area, take the illustrations and examples and discover what you can do with it. Try exploring.

Furthermore, by chance that you need to START learning, it is recommended that you begin with their instructional exercises and see where you can go from that point.

Two comparative choices are Lasagne and Blocks; however, they just keep running on Theano. If you attempted Keras and have difficulty, perhaps, experiment with one of these contrasting options to check whether they work out for you.

“PyTorch”

If you are looking for a popular deep learning library, then look no further than Torch, which is written in the language called Lua. Facebook recently open-sourced a Python model of Torch and named it PyTorch, which allows you to easily use the exact same

libraries that Torch uses, but from Python, instead of the original language, Lua.

PyTorch is significantly easier for debugging because of one major difference between Theano, TensorFlow, and PyTorch. The older versions use allegorical computation while the newer does not. Allegorical computation is simply a way of saying that coding an operation, for example, ‘a + b’, will not be computed when that line is read. Before it is executed it must be translated into what is called CUDA or C. This makes the debugging much harder to execute in Theano/TensorFlow since this error is more difficult to pinpoint with a specific line of code. It’s basically harder to trace back to the source. Debugging is not one of this library’s strongest features.

This is extremely beginner-friendly; as your learning increases, try some of their more advanced tutorials and examples.

HISTORY OF PYTHON

Python was invented in the later years of the 1980s. Guido van Rossum, the founder, started using the language in December 1989. He is Python's only known creator and his integral role in the growth and development of the language has earned him the nickname "Benevolent Dictator for Life". It was created to be the successor to the language known as ABC.

The next version that was released was Python 2.0, in October of the year 2000 and had significant upgrades and new highlights, including a cycle-distinguishing junk jockey and back up support for Unicode. It was most fortunate, that this particular version, made vast improvement procedures to the language turned out to be more straightforward and network sponsored.

Python 3.0, which initially started its existence as Py3K. Funny right? This version was rolled out in December of 2008 after a rigorous testing period. This particular version of Python was hard to roll back to previous compatible versions which are the most unfortunate. Yet, a significant number of its real highlights have been rolled back to versions 2.6 or 2.7 (Python), and rollouts of Python 3 which utilizes

the two to three utilities, that helps to automate the interpretation of the Python script.

Python 2.7's expiry date was originally supposed to be back in 2015, but for unidentifiable reasons, it was put off until the year 2020. It was known that there was a major concern about data being unable to roll back but roll FORWARD into the new version, Python 3. In 2017, Google declared that there would be work done on Python 2.7 to enhance execution under simultaneously running tasks.

BASIC FE ATURES OF PYTHON

Python is an unmistakable and extremely robust programming language that is object-oriented based almost identical to Ruby, Perl, and Java.

A portion of Python's remarkable highlights:

- Python uses a rich structure, influencing, and composing projects that can be analyzed simpler.
- It is a simple to utilize dialect that makes it easy to get your program working. This makes Python perfect for model improvement and other specially appointed programming assignments, without trading off viability.
- It accompanies a huge standard library that backs tons of simple programming commands, for example, extremely seamless web server connections, processing and handling files, and the ability to search through text with commonly used expressions and commands.
- Python's easy to use interactive interface makes it simple to test shorter pieces of coding. It also comes with IDLE which is a "development environment".
- Python effortlessly extended out by including new modules executed in a source code like C or C++.
- Python can also be inserted into another application to give an easily programmed interface.

- Python will run anyplace, including OS X, Windows Environment, Linux, and even Unix, with informal models for the Android and iOS environments.
- Python can easily be recorded, modified and re-downloaded and distributed, be unreservedly adjusted and re-disseminated. While it is copyrighted, it's accessible under open source. Ultimately, Python is a free software.

Common Programming Language Features of Python

- A huge array of common data types: floating point numbers, complex numbers, infinite length integers, ASCII strings, and Unicode, as well as a large variety of dictionaries and lists.
- Python is guided in an object-oriented framework, with multiple classes and inheritance.
- Python code can be bundled together into different modules and packages.
- Python is notorious for being a much cleaner language for error handling due to the catching and raising of exceptions allowed.
- Information is firmly and progressively composed. Blending incongruent data types, for example, adding a string and a number together, raises an exception right away where errors are caught significantly sooner than later.
- Python has advanced coding highlights such as comprehending lists and iterators.
- Python's programmed memory administration liberates you from having to physically remove unused or unwanted code.

Chapter 2:

Installation of Python

Python is both procedural and object-oriented coding language. It has a straightforward syntax. Python is cross-platform implying that it can be run on different Operating Systems environments such as Linux, Windows platform, Mac OS X platform, UNIX platform and can be ported to .NET and Java virtual machines. Python is free and open source. While most recent versions of Mac and Linux have Python preinstalled, it is recommended that one installs and runs the current version.

Installing Python

Most recent versions of Linux and Mac have Python already installed in them. However, you might need to install Python, and the following are the steps for installing Python in Windows, Mac OS X or Linux.

Installing Python in Macintosh Operating System X

a) Visit Download Python page which is the credible site and click “Download Python 3.7.2 (The version may differ from the one stated here).

b) When the download completes, click open the package and follow the instructions given. The installation should complete with “The installation was successful” prompt.

c) Now, visit Download Notepad++ and download the text editor and install it by opening the package and following the message prompts. The Notepad++ text editor is free and suited to help write source code (raw text programming words).

Installing Python in Linux Operating Systems

v. It is now time to issue instructions to run the source code on your OS (Operating System)

Installing Python in Windows Operating System

a) Visit Download Python site which is the recommended site and click “Download Python 3.7.2 (The version may differ from the one stated here).

b) When your download completes, open the package by clicking and follow the guidelines given. The Python installation should complete with “The installation was successful” prompt. When you install Python successfully, it also installs a program known as IDLE along with it. IDLE is a graphical user interface when working with Python.

c) Now, visit Download Notepad++ and download the text editor and install it by opening the package and following the message prompts. The Notepad++ text editor is free and suited to help write source code (raw text programming words).

How to Run Python

Before we start running our first Python program, it is important that we understand how we can run python programs. Running or executing or deploying or firing a program simply means that we are making the computer process instructions/lines of codes. For instance, if the lines of codes (program) require the computer to display some message, then it should. The following are the ways or mode of running python programs. The interpreter is a special program that is installed when installing the Python package and helps convert text code into a language that the computer understands and can act on it (executing).

Immediate Mode

It is a way of running python programs that are not written in a file. We get into the immediate mode by typing the word python in the command line and which will trigger the interpreter to switch to immediate mode. The immediate mode allows typing of expressions

directly, and pressing enter generates the output. The sign below is the Python prompt:

>>>

The python prompt instructs the interpreter to accept input from the user. For instance, typing 2+2 and pressing enter will display 4 as the output. In a way, this prompt can be used as a calculator. If you need to exit the immediate mode, type quit() or exit().

Now type 5 +3, and press enter, the output should be 8. The next mode is the Script Mode.

Script Mode

The script mode is used to run a python program written in a file; the file is called a script.

Integrated Development Environment (IDE)

An IDE provides a convenient way of writing and running Python programs. One can also use text editors to create a python script file instead of an IDE by writing lines of codes and saving the file with a .py extension. However, using an IDE can simplify the process of writing and running Python programs. The IDEL present in the Python package is an example of an IDE with a graphical user interface and gets installed along with the Python language. The advantages of IDE include helping getting rid of repetitive tasks and simplify coding for beginners. IDE provides syntax highlighting, code hinting, and syntax checking among other features. There also commercial IDE such as the PyScripter IDE that performs most of the mentioned functions.

Note

We have presented what Python is, how to download and install Python, the immediate and script modes of Python IDE, and what is an IDE.

Your First Program in Python

The rest of the illustrations will assume you are running the python programs in a Windows environment.

I. Start IDLE
II. Navigate to the File menu and click New Window
III. Type the following:
IV. print (“Hello World!”)
V. On the file, menu click on Save. Type the name of myProgram1.py
VI. Navigate to Run and click Run Module to run the program.

The first program that we have written is known as the “Hello World!” and is used to not only provide an introduction to a new computer coding language but also test the basic configuration of the IDE. The output of the program is “Hello World!” Here is what has happened, the Print() is an inbuilt function, it is prewritten and preloaded for you, is used to display whatever is contained in the () as long as it is between the double quotes. The computer will display anything written within the double quotes.

Assignment

Now write and run the following python programs:

a. print(“I am now a Python Language Coder!”)
b. print(“This is my second simple program!”)
c. print(“I love the simplicity of Python”)
d. print(“I will display whatever is here in quotes such as owyhen2589gdbnz082”)

Now we need to write a program with numbers, but before writing such a program, we need to learn something about Variables and Types.

Remember python is object-oriented and it is not statically typed which means we do not need to declare variables before using them or specify their type. Let us explain this statement; an object-oriented language simply means that the language supports viewing and manipulating real-life scenarios as groups with subgroups that can be linked and shared mimicking the natural order and interaction of

things. Not all programming languages are object-oriented; for instance, Visual C programming language is not object-oriented. In programming, declaring variables means that we explicitly state the nature of the variable. The variable can be declared as an integer, long integer, short integer, floating integer, a string, or as a character including if it is accessible locally or globally. A variable is a storage location that changes values depending on conditions.

For instance, number1 can take any number from 0 to infinity. However, if we specify explicitly that int number1 it then means that the storage location will only accept integers and not fractions for instance, fortunately or unfortunately, python does not require us to explicitly state the nature of the storage location (declare variables) as that is left to the python language itself to figure out that.

Before tackling types of variables and rules of writing variables, let us run a simple program to understand what variables when coding a python program are.

i. Start IDLE

ii. Navigate to the File menu and click New Window

iii. Type the following:

```
num1=4
num2=5
sum=num1+num2
print(sum)
```

iv. On the file, menu click on Save. Type the name of myProgram2.py

v. Navigate to Run and click Run Module to run the program.

i. The expected output of this program should be “9” without the double quotes.

Explanation

At this point, you are eager to understand what has just happened and why the print(sum) does not have double quotes like the first

programs we wrote. Here is the explanation.

The first line num1=4 means that variable num1(our shortened way of writing number1, first number) has been assigned 4 before the program runs.

The second line num2=5 means that variable num2(our shortened way of writing number2, second number) has been assigned 5 before the program runs.

The computer interprets these instructions and stores the numbers given

The third line sum=num1+num2 tells the computer that takes whatever num1 has been given and add to whatever num2 has been given. In other terms, sum the values of num1 and num2.

The fourth line print(sum) means that display whatever sum has. If we put double quotes to sum, the computer will display the word sum and not the sum of the two numbers! Remember that cliché that computers are garbage in and garbage out. They follow what you give them!

Note

+ is an operator for summing variables and has other users that will be discussed later.

Now let us try out three Assignments involving numbers before we explain types of variables and rules of writing variables so that you get more freedom to play with variables. Remember variables values vary for instance num1 can take 3, 8, 1562, 1.

Follow the steps of opening the Python IDE and do the following:

a) The output should be 54

num1=43

num2=11

sum=num1+num2

print(sum)

b) The output should be 167

```
num1=101
num2=66
sum=num1+num2
print(sum)
```

c) The output should be 28

```
num1=9
num2=19
sum=num1+num2
print(sum)
```

Variables

We have used num1, num2, and sum and the variable names were not just random, they must follow certain rules and conventions. Rules are what we cannot violate while conventions are much like the recommended way. Let us start with the rules:

The Rules of When Naming Variables in Python

a) Variable names should always start with a letter or an underscore, i.e.

```
num1
_num1
```

b) The remaining part of the variable name may consist of numbers, letters, and underscores, i.e.

```
number1
num_be_r
```

c) Variable names are case sensitive meaning that capital letters and non-capital letters are treated differently.

Num1 will be treated differently with num1.

Assignment

Write/suggest five variables for:

i) Hospital department.

ii) Bank.

iii) Media House.

Given scri=75, scr4=9, sscr2=13, Scr=18

iv) The variable names above are supposed to represent scores of students. Rewrite the variables to satisfy Python variable rules and conventions.

Conventions When Naming Variables in Python

As earlier indicated, conventions are not rules per se are the established traditions that add value and readability to the way we name variables in Python.

d) Uphold readability. Your variables should give a hint of what they are handling because programs are meant to be read by other people other than the person writing them. Number1 is easy to read compared to n1. Similarly, first_name is easy to read compared to firstname or firstName or fn. The implication of all these is that both are valid/acceptable variables in python, but the convention is forcing us to write them in an easy to read form.

e) Use descriptive names when writing your variables. For instance, number1 as a variable name is descriptive compared to yale or mything. In other words, we can write yale to capture values for number1, but the name does not outrightly hint what we are doing. Remember when writing programs; assume another person will maintain them. The person should be able to quickly figure out what the program is all about before running it.

f) Due to confusion, avoid using the uppercase ‘O,’ lowercase letter ‘l’ and the uppercase letter ‘I’ because they can be confused with numbers. In other terms, using these letters

will not be a violation of writing variables, but their inclusion as variable names will breed confusion.

Assignment 1

Re-write the following variable names to (1) be valid variable names and follow (2) conventions of writing variable names.

i) 23doctor

ii) line1

iii) Option3

iv) Mydesk

v) #cup3

Assignment 2

Write/Suggest variable names that are (1) valid and (2) conventional.

i) You want to sum three numbers.

ii) You want to store the names of four students.

iii) You want to store the names of five doctors in a hospital.

Summary

Variables are storage locations that a user specifies before writing and running a python program. Variable names are labels of those storage locations. A variable holds a value depending on circumstances. For instance, doctor1 can be Daniel, Brenda or Rita. Patient1 can be Luke, William or Kelly. Variable names are written by adhering to rules and conventions. Rules are a must while conventions are optional but recommended as they help write readable variable names. When writing a program, you should assume that another person will examine or run it without your input and thus should be well written. The next chapter will discuss Variables. In programming, declaring variables means that we explicitly state the nature of the variable. The variable can be declared as an integer, long integer, short integer, floating integer, a string, or as a character including if it is accessible locally or globally.

A variable is a storage location that changes values depending on conditions. Use descriptive names when writing your variables.

Chapter 3:

What are Variables?

Variables are names for values. In Python the = symbol assigns the value on the right to the name on the left. The variable is created when a value is assigned to it. Here is a Python program that assigns an age to a variable age and a name in quotation marks to a variable first_name.

```
age = 42

first_name = 'Eunice'
```

Types of Variables

Now that we have defined what are variables are and the rules to write variable names in the last chapter, let us explore different types of variables.

a) Numbers

The Python accommodates two kinds of numbers, namely floating point numbers and integer numbers. Python also supports complex numbers. When you sign a value to a number, then a number object is created. For example:

```
number3 =9

number4=12
```

Different Numerical Types Supported in Python

- long for example 681581903L
- int for example 11, 123, -151
- float for example 0.5, 23.1, -54.2
- complex for example 4.12j

Exercise

Identify the type of numerical below:

a. 234, 19, 312

b. 4.56, 2.9, 9.3

c. 76189251468290127624471

Identify the numerical type suitable for the following contexts:

d. Salary amount.

e. Counting the number of students in a class.

f. Getting the census figure in an entire country of China.

b) Strings

A single or double quote in Python is used to indicate strings. The subsets of strings can be taken by using the slice operator ([:]) and []) with indexes beginning at () in the start of the string and operating their way from -1 at the end. Strings can be joined using the + (plus) sign known as the concatenation operator. The asterisk (*) is used as a repetition operator. Remember counting in programming starts from index zero (the first value)!

Note

The # (hash sign) is used to indicate a single line comment. A comment is a descriptive information about a particular line(s) of code. The comment is normally ignored by when running the program. The comment should be written after the # sign in python. Comments increase the readability of the program written.

Assignment

You will key in/type the following program statement:

str = 'I think I am now a Programmer.'

a. Write a program statement that will display the entire string/statement above.

b. Write a program statement to display characters of the string from the second character to the sixth.

c. Write a single program statement that will display the entire string two times. (use *).

d. Write a program statement that will add the following at the end of the statement above, “ of Python.”

Identifiers and Keywords

At this point, you have been wondering why you must use print and str in that manner without the freedom or knowledge of why the stated words have to be written in that manner. The words print and str constitute a special type of words that have to be written that way always. Each programming language has a set of keywords. In most cases, some keywords are found across several programming languages. Keywords are case sensitive in python meaning that we have to type them in their lowercase form always. Keywords cannot be used to name a function (we will explain what it is later), name of a variable.

There are 33 keywords in Python, and all are in lowercase save for None, False, and True. They must always be written as they appear below:

Note

The print() and str are functions, but they are inbuilt/preloaded functions in Pythons. Functions are a set of rules and methods that act when invoked. For instance, the print function will display output when activated/invoked/called. At this point, you have not encountered all of the keywords, but you will meet them gradually. Take time to skim through, read and try to recall as many as you can.

Assignment

Identify what is wrong with the following variable names (The Assignment requires recalling what we have learned so far)

a. for=1
b. yield=3
c. 34ball
d. m

Comments and Statements

Statements in Python

A statement in Python refers to instructions that a Python interpreter can work on/execute. An example is str=’ I am a Programmer’ and number1=3. A statement having an equal sign(=) is known as an assignment statement. They are other types of statements such as the if, while, and for which will be handled later.

Assignment

a. Write a Python statement that assigns the first number value of 18.

b. Write a programming statement that assigns the second number value of 21.

c. What type of statements are a. and b. above?

Multi-line Python Statement

Spreading a statement over multiple lines is possible. Such a statement is known as a multi-line statement. The termination of a programming statement is denoted by new line character. To spread a statement overs several lines, in Python, we use the backslash (\) known as the line continuation character. An example of a multi-line statement is:

```
sum=3+6+7+\
    9+1+3+\
    11+4+8
```

The example above is also known as an explicit line continuation. In Python, the square brackets [] denotes line continuation similar to parenthesis/round brackets (), and lastly braces {}. The above example can be rewritten as

```
sum=(3+6+7+
    9+1+3+
    11+4+8)
```

Note

We have dropped the backslash(\) known as the line continuation character when we use the parenthesis(round brackets) because the parenthesis is doing the work that the line continuation \ was doing.

Solved Question

Why do you think multi-line statements are necessary we can simply write a single line, and the program statement will run just fine?

Answer

Multi-line statements can help improve formatting/readability of the entire program. Remember, when writing a program always assume that it is other people who will use and maintain it without your input.

Assignment

Rewrite the following program statements using multi-line operators such as the \, [],() or {} to improve readability of the program statements.

a. total=2+9+3+6+8+2+5+1+14+5+21+26+4+7+13+31+24

b. count=13+1+56+3+7+9+5+12+54+4+7+45+71+4+8+5

Semicolons are also used when creating multiple statements in a single line. Assume we have to assign and display the age of four employees in a python program. The program could be written as:

employee1=25; employee2=45; employee3=32; employee4=43.

Indentation in Python

Indentation is used for categorization program lines into a block in Python. The amount of indentation to use in Python depends entirely on the programmer. However, it is important to ensure consistency. By convention, four whitespaces are used for indentation instead of using tabs. For example:

Note

I will explain what kind of program this is later. Indentation in Python also helps make the program look neat and clean. Indentation creates consistency. However, when performing line continuation indentation can be ignored. Incorrect indentation will create an

indentation error. Correct python programs without indentation will still run, but they might be neat and consistent from human readability view.

Comments in Pythons

When writing python programs and indeed any programming language, comments are very important. Comments are used to describe what is happening within a program. It becomes easier for another person taking a look at a program to have an idea of what the program does by reading the comments in it. Comments are also useful to a programmer as one can forget the critical details of a program written. The hash (#) symbol is used before writing a comment in Python. The comment extends up to the newline character. The python interpreter normally ignores comments. Comments are meant for programmers to understand the program better.

Example

i. Start IDLE

ii. Navigate to the File menu and click New Window

iii. Type the following:

```
#This is my first comment
#The program will print Hello World
Print('Hello World')   #This is an inbuilt function to display
```

iv. On the file, menu click on Save. Type the name of myProgram5.py

Navigate to Run and click Run Module to run the program

Assignment

This Assignment integrates most of what we have covered so far.

a. Write a program to sum two numbers 45, and 12 and include single line comments at each line of code.

b. Write a program to show the names of two employees where the first employee is “Daisy” and the second employee is “Richard.” Include single comments at each line of code.

c. Write a program to display the student registration numbers where the student names and their registration are: Yvonne=235, Ian=782, James=1235, Juliet=568.

Multi-line Comments

Just like multi-line program statements we also have multi-line comments. There are several ways of writing multi-line comments. The first approach is to type the hash (#) at each comment line starting point.

For Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

#I am going to write a long comment line

#the comment will spill over to this line

#and finally end here.

The second way of writing multi-line comments involves using triple single or double quotes: ‘’’ or”. For multi-line strings and multi-line comments in Python, we use the triple quotes. Caution: When used in docstrings they will generate extra code, but we do not have to worry about this at this instance.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

“This is also a great

illustration of

a multi-line comment in Python.”

Python’s Docstring

In Python, docstring refers to words offering description and are written as the initial program statement in a function, module, method, or class definition. (We will handle this later on). Docstrings in Python are written using triple quotes.

Assignment

a. This Assignment will utilize several concepts that we covered earlier.

b. Given the following program statement: Color1=’red’; color1=’blue’; CoLor1=’yellow’ explain why all the three will be treated as different variables in Python.

c. Consider the following Python program and identify what is wrong with it.

```
student1_age=23                    #This is the age of the first student
student2_age=19                                  #This is the age of the student
sotal_age=student1_age +student2_age      #Getting the sum of the ages of the
print(age)                         #Displaying their ages
```

d. Rewrite b. above to be a valid Python program

Basic Operators in Python

So far we have been using the summation (+) operator, and it also doubles up as a concatenation operator (appending statements). However, we want to expand our list of operators, and this leads us to basic operators in Python.

Arithmetic Operators

The multiplication (*), division (/), subtraction (-), and addition (+) are the arithmetic operators used to manipulate numbers.

Assignment

Write the following programs and run it

a. Difference

```
number1=35              #declaring first number
number2= 12             #declaring second number
difference=number2-number1    #declaring what the difference does
print(difference)             #Calling the print function to display what difference has
```

b. Multiplication

```
number1=2               #declaring first number
number2= 15             #declaring second number
product=number1*number2       #declaring what the product does
print(product)                #Calling the print function to display what product has
```

c. Division

```
number1=10              #declaring first number
number2= 50             #declaring second number
division=number2/number1      #declaring what the division does
print(division)               #Calling the print function to display what product has
```

Modulus

The modulus operator is used to return the integer remainder after division. The modulus=dividend%divisor.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
number1=2               #declaring first number
number2= 15             #declaring second number
```

remainder=number2%number1 #declaring what the remainder does

print(remainder) #Calling the print function to display remainder has

Squaring and Cubing in Python

Squaring a number-number**2

Cubing a number-number**3

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

Square of 3 in Python will be 3**2

Cube of 5 in Python will be 5**3

a. Square of 3

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

number=3 #declaring variable number and assigning value 3

square=number**2

print(square) #Calling the print function to display what square has

b. Cube of 5

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

number=5 #declaring variable number and assigning value 5

```
cube=number**3
print(cube)                #Calling the print function to display what cube has
```

Assignment

Use python operators to write and run a python program that finds the following:

a. Cube of 7
b. Square of 15
c. Cube of 6
d. Square of 11
e. Cube of 8
f. Square of 13

Note

We can still multiply 2 two times to get the square of 2. The reason for using the square and cube operators is to help us write compact and efficient code. Remember that the interpreter goes through each line including comments only that it ignores comments. Using the cube and square operators helps compact code and increase the efficiency of interpretation including troubleshooting as well as human readability of the code.

Operators with String in Python

In Python certain operators are used to help in concatenating strings. The addition sign is used to concatenate strings in Python.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
status=" I am happy I know" + "how to write programs in Python."
print(status)
```

Python Multiplication of a string to create a sequence

many_words=”Great Programmer” * 5

print(many_words)

Assignment

a. Use a concatenation operator to join the following strings in Python

I have realized

that programming is a passion,

dedication and frequent practice.

b. Use an operator to generate ten times the following string

Happy

Summary

We have covered what constitutes variables in Python. Variables are named storage locations, and for this reason, we have variable names. There are numerical and string variables. These are known as variable types. In handling variables and indeed any other aspect of Python we will encounter and use special words known as reserved words or keywords. Keywords are fixed and must be typed the way specified. Keywords cannot be used as variable names or identifiers. Comments (preceded using #,’’’ or “) are for human readability aspect of a Python program. Indentation is used in Python to group lines of codes into blocks. Different types of operators such as the arithmetic operators and string operators are used to allow for manipulation of variables supported by the user and inbuilt functions in Python.

Chapter 4:

A Deep Dive into Data Types

- **Numbers**

As mentioned earlier, Python accommodates floating, integer and complex numbers. The presence or absence of a decimal point separates integers and floating points. For instance, 4 is integer while 4.0 is a floating point number.

On the other hand, complex numbers in Python are denoted as r+tj where j represents the real part and t is the virtual part. In this context the function type() is used to determine the variable class. The Python function is instance() is invoked to make a determination of which specific class function originates from.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
number=6
  print(type(number))    #should output class int
print(type(6.0))    #should output class float
complex_num=7+5j
print(complex_num+5)
print(isinstance(complex_num, complex)) #should output True
```

Important: Integers in Python can be of infinite length. Floating numbers in Python are assumed precise up to fifteen decimal places.

- **Number Conversion**

This segment assumes you have prior basic knowledge of how to manually or using a calculator to convert decimal into binary, octal and hexadecimal. Check out the Windows Calculator in Windows 10,

Calculator version 10.1804.911.1000 and choose programmer mode to automatically convert.

Programmers often need to convert decimal numbers into octal, hexadecimal and binary forms. A prefix in Python allows denotation of these numbers to their corresponding type.

Number System Prefix

Octal ‘0O’ or '0o'

Binary ‘0B' or '0b'

Hexadecimal '0X or '0x'

Example

```
print(0b1010101)  #Output:85
print(0x7B+0b0101)  #Output: 128 (123+5)
print(0o710)  #Output:710
```

Assignment

Create a program in Python to display the following:

i) 0011 11112

ii) 7478

iii) 9316

Type Conversion

Sometimes referred to as coercion, type conversion allows us to change one type of number into another. The preloaded functions such as float(), int() and complex() enable implicit and explicit type conversions. The same functions can be used to change from strings.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

int(5.3) #Gives 5

int(5.9) #Gives 5

The int() will produce a truncation effect when applied to floating numbers. It will simply drop the decimal point part without rounding off. For the float() let us take a look:

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

float(6) #Gives 6.0

ccomplex('4+2j') #Gives (4+2j)

Assignment

Apply the int() conversion to the following:

a. 4.1
b. 4.7
c. 13.3
d. 13.9

Apply the float() conversion to the following:

e. 7
f. 16
g. 19

- **Decimal in Python**

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

(1.2+2.1)==3.3 #Will return False, why?

Explanation

The computer works with finite numbers and fractions cannot be stored in their raw form as they will create infinite long binary sequence.

- **Fractions in Python**

The fractions module in Python allows operations on fractional numbers.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

Important

Creating my_fraction from float can lead to unusual results due to the misleading representation of binary floating point.

Mathematics in Python

To carry out mathematical functions, Python offers modules like random and math.

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
import math
print(math.pi)   #output:3.14159….
print(math.cos(math.pi)) #the output will be -1.0
print(math.exp(10))  #the output will be 22026.4….
print(math.log10(100)) #the output will be 2
print(math.factorial(5)) #the output will be 120
```

Exercise

Write a python program that uses math functions from the math module to perform the following:

a. Square of 34
b. Log1010000
c. Cos 45 x sin 90
d. Exponent of 20

Random function in Python

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
import math
print(random.shuffle_num(11, 21))
y=['f','g','h','m']
print(random.pick(y))
random.anypic(y)
print(y)
print(your_pick.random())
```

Lists in Python

We create a list in Python by placing items called elements inside square brackets separated by commas. The items in a list can be of mixed data type.

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
list_mine=[]   #empty list
list_mine=[2,5,8]  #list of integers
list_mine=[5,"Happy", 5.2] #list having mixed data types
```

Assignment

Write a program that captures the following in a list: “Best”, 26,89,3.9

Nested Lists

A nested list is a list as an item in another list.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

list_mine=[“carrot”, [9, 3, 6], [‘g’]]

Exercise

Write a nested for the following elements: [36,2,1],”Writer”,’t’,[3.0, 2.5]

Accessing Elements from a List

In programming and in Python specifically, the first time is always indexed zero. For a list of five items we will access them from index0 to index4. Failure to access the items in a list in this manner will create index error. The index is always an integer as using other number types will create a type error. For nested lists, they are accessed via nested indexing.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
list_mine=[‘b’,’e’,’s’,’t’]
print(list_mine[0])   #the output will be b
print(list_mine[2])   #the output will be s
print(list_mine[3])   #the output will be t
```

Exercise

Given the following list:

your_collection=[‘t’,’k’,’v’,’w’,’z’,’n’,’f’]

a. Create a program in Python to display the second item in the list

b. Create a program in Python to display the sixth item in the last

c. Create a program in Python to display the last item in the list.

Nested List Indexing

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

nested_list=[“Best’,[4,7,2,9]]

print(nested_list[0][1]

Python Negative Indexing

For its sequences, Python allows negative indexing. The last item on the list is index-1, index -2 is the second last item and so on.

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

list_mine=[‘c’,’h’,’a’,’n’,’g’,’e’,’s’]

print(list_mine[-1]) #Output is s

print(list_mine [-4]) ##Output is n

Slicing Lists in Python

Slicing operator(full colon) is used to access a range of elements in a list.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

list_mine=[‘c’,’h’,’a’,’n’,’g’,’e’,’s’]

print(list_mine[3:5]) #Picking elements from the 4 to the sixth

Example

Picking elements from start to the fifth

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

print(list_mine[:-6])

Example

Picking the third element to the last.

print(list_mine[2:])

Exercise

Given class_names=[‘John’, ‘Kelly’, ‘Yvonne’, ‘Una’,’Lovy’,’Pius’, ‘Tracy’]

a. Write a python program using slice operator to display from the second students and the rest.

b. Write a python program using slice operator to display first student to the third using negative indexing feature.

c. Write a python program using slice operator to display the fourth and fifth students only.

Manipulating Elements in a List using the assignment operator

Items in a list can be changed meaning lists are mutable.

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

list_yours=[4,8,5,2,1]

list_yours[1]=6

print(list_yours) #The output will be [4,6,5,2,1]

Changing a range of items in a list

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

list_yours[0:3]=[12,11,10] #Will change first item to fourth item in the list

print(list_yours) #Output will be: [12,11,10,1]

Appending/Extending items in the List

The append() method allows extending the items in the list. The extend() can also be used.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

list_yours=[4, 6, 5]

list_yours.append(3)

print(list_yours) #The output will be [4,6,5, 3]

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

list_yours=[4,6,5]

list_yours.extend([13,7,9])

print(list_yours) #The output will be [4,6,5,13,7,9]

The plus operator(+) can also be used to combine two lists. The * operator can be used to perform iteration of a list a given severally.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
list_yours=[4,6,5]
print(list_yours+[13,7,9])  # Output:[4, 6, 5,13,7,9]
print(['happy']*4)   #Output:["happy","happy", "happy","happy"]
```

Removing or Deleting Items from a List

The keyword del is used to delete elements or the entire list in Python.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
list_mine=['t','r','o','g','r','a','m']
del list_mine[1]
print(list_mine)  #t, o, g, r, a, m
```

Deleting Multiple Elements

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
del list_mine[0:3]
```

Example

```
print(list_mine)  #a, m
```

Delete Entire List

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

delete list_mine

print(list_mine) #will generate an error of lost not found

The remove() method or pop() function may be used to remove specified item. The pop() method will remove and return the last item if index is not given and helps implement lists as stacks. The clear() method is used to empty a list.

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

list_mine=['t','k','b','d','w','q','v']

list_mine.remove('t')

print(list_mine)   #output will be ['t','k','b','d','w','q','v']

print(list_mine.pop(1))   #output will be 'k'

print(list_mine.pop())   #output will be 'v'

Assignment

Given list_yours=['K','N','O','C','K','E','D']

a. Pop the third item in the list, save the program as list1.
b. Remove the fourth item using remove() method and save the program as list2
c. Delete the second item in the list and save the program as list3.
d. Pop the list without specifying an index and save the program as list4.

Using Empty List to Delete an entire or specific elements

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

list_mine=['t','k','b','d','w','q','v']

list_mine=[1:2]=[]

print(list_mine) #Output will be [‘t’,’w’,’q’,’v’]

List Methods in Python

Assignment

a. Use list access methods to display the following items in reversed order list_yours=[4,9,2,1,6,7]
b. Use list access method to count the elements in a.
c. Use list access method to sort the items in a. in an ascending order/default.

Inbuilt Python Functions that can be used to manipulate Python Lists

Tuple in Python

A tuple is like a list but we cannot change elements in a tuple.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

tuple_mine = (21, 12, 31)

print(tuple_mine)

tuple_mine = (31, "Green", 4.7)

print(tuple_mine)

Accessing Python Tuple Elements

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

tuple_mine=[‘t’,’r’,’o’,’g’,’r’,’a’,’m’]

print(tuple_mine[1]) #output:’r’

print(tuple_mine[3]) #output:’g’

Negative Indexing

Just like lists, tuples can also be indexed negatively.

Like lists, -1 refers to the last element on the list and -2 refer to the second last element.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

tuple_mine=[‘t’,’r’,’o’,’g’,’r’,’a’,’m’]

print(tuple_mine [-2]) #the output will be ‘a’

Slicing

The slicing operator, the full colon is used to access a range of items in a tuple.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

tuple_mine=[‘t’,’r’,’o’,’g’,’r’,’a’,’m’]

print(tuple_mine [2:5]) #Output: ‘o’,’g’,’r’,’a’

print(tuple_mine[:-4]) #’g’,’r’,’a’,’m’

Note

Tuple elements are immutable meaning they cannot be changed. However, we can combine elements in a tuple using +(concatenation operator). We can also repeat elements in a tuple using the * operator, just like lists.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

print((7, 45, 13) + (17, 25, 76))

print(("Several",) * 4)

Note

Since we cannot change elements in tuple, we cannot delete the elements too. However removing the full tuple can be attained using the kwyword del.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

t_mine=[‘t’,’k’,’q’,’v’,’y’,’c’,’d’]

del t_mine

Available Tuple Methods in Python

They are only two methods available for working Python tuples.

count(y)

When called will give the item numbers that are equal to y.

index(y)

When called will give index first item index that is equal to y.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

t_mine=[‘t’,’k’,’q’,’v’,’y’,’c’,’d’]

print(t_mine.count('t'))

```
print(t_mine.index('l'))
```

Testing Membership in Tuple

The keyword in us used to check the specified element exists in a tuple.

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
t_mine=[‘t’,’k’,’q’,’v’,’y’,’c’,’d’]
print('a' t_mine)  #Output: True
print('k' in t_mine)  #Output: False
```

Inbuilt Python Functions with Tuple

String in Python

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
string_mine = 'Colorful'
print(string_mine)
string_mine = "Hello"
print(string_mine)
string_mine = '''Hello'''
print(string_mine)
string_mine = """I feel like I have
        been born a programmer"""
print(string_mine)
```

Accessing items in a string

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
str = 'Colorful'
print('str = ', str)
print('str[1] = ', str[1])  #Output the second item
print('str[-2] = ', str[-2]) #Output the second last item
print('str[2:4] = ', str[2:4]) #Output the third through the fifth item
```

Deleting or Changing in Python

In Python, strings are immutable therefore cannot be changed once assigned. However, deleting the entire string is possible.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
del string_mine
```

String Operations

Several operations can be performed on a string making it a widely used data type in Python.

Concatenation using the + operator, repetition using the * operator

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
string1=’Welcome’
string2=’Again’
print(‘string1+string2=’,string1+string2)
```

print(' string1 * 3 =', string1 * 3)

Exercise

Given string_a=”I am awake” and string_b=”coding in Python in a pajama”

String Iteration

The ***for control*** statement is used to continually scan through an entire scan until the specified severally are reached before terminating the scan.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

Membership Test in String

The keyword in is used to test if a sub string exists.

Example

‘t’ in “triumph’ #Will return True

Inbuilt Python Functions for working with Strings

They include enumerate() and len().The len() function returns the length of the string.

String Formatting in Python

Escape Sequences

Single and Double Quotes

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

print('They said, "We need a new team?"') # escape with single quotes

```
# escaping double quotes
print("They said, \" We  need a new team\"")
```

Escape Sequences in Python

The escape sequences enable us to format our output to enhance clarity to the human user. A program will still run successful without using escape sequences but the output will be highly confusing to the human user. Writing and displaying output in expected output is part of good programming practices. The following are commonly used escape sequences.

Examples

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
print("D:\\Lessons\\Programming")
print("Prints\n in two lines")
```

Summary

Integers, floating point, and complex numbers are supported in Python. There are integers, floating and complex classes that help convert different number data types. The presence or absence of a decimal point separates integers and floating points. For instance, 4 is integer while 4.0 is a floating point number. Programmers often need to convert decimal numbers into octal, hexadecimal and binary forms. We can represent binary, hexadecimal and octal systems in Python by simply placing a prefix to the particular number. Sometimes referred to as coercion, type conversion allows us to change one type of number into another.

Inbuilt functions such as int() allows us to convert data types directly. The same functions can be used to convert from strings. We create a list in Python by placing items called elements inside square brackets separated by commas. In programming and in Python specifically, the first time is always indexed zero. For a list of five

items we will access them from index0 to index4. Failure to access the items in a list in this manner will create index error.

Chapter 5:

Type Conversion and Type Casting

Type Conversion refers to the process of changing the value of one programming data type to another programming data type. Think of dividing two integers that lead to decimal numbers. In this case, it is necessary to convert force the conversion of an integer into a float number. Python has two types of conversion: implicit type conversion and explicit conversion.

i. Implicit Conversion Type

In this case, Python automatically changes one data type to another data type, and the process does not require user involvement. Implicit type conversion is mainly used with the intent of avoiding data loss.

Example

Converting Integer to Float

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
number_int=451

number_flo=4.51

number_new=number_int+number_flo

print(“the type of data of number_int-“, type(number_int))

print(“the type of data of number_flo-“, type(number_flo))

print(“value of number_new-“number_new)

print(“type of data of number_new-“, type(number_new))
```

Explanation

The programming is adding two variables one of data type integer and the other float and storing the value in a new variable of data type float. Python automatically converts small data types into larger

data types to avoid prevent data loss. This is known as implicit type conversion.

Note

Python cannot, however, convert numerical data types into string data type or string data type into numerical data type implicitly. Attempting such a conversion will generate an error.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
number_int=432          #lower data type
number_str="241"        #higher data type
print("The type of data of number_int-", type(number_int))
print("The type of data of number_str-", type(number_str))
print(number_int+number_str)
```

Challenge: Can you guess why this is occurring? Python interpreter is unable to understand whether we want concatenation or addition precisely. When it assumes the concatenation, it fails to locate the other string. When it assumes the addition approach, it fails to locate the other number. The solution to the error above is to have an explicit type conversion.

ii. Explicit Conversion

Here, programmers convert the data type of a named programming object to the needed data type. Explicit conversion is attained through the functions float(), int(), and str() among others.

The syntax for the specific or explicit conversion is:

(needed_datatype) (expression)

Example

Summing of a string and integer using by using explicit conversion

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
number_int=431

number_str=”231”

print(“Type of data of number_int-“, type(number_int))

print(“Type of data number_str prior to Type Casting-“, type(number_str))

number_str=int(number_str)

number_sum=number_int+number_str

print(“Addition of number_int and number_str-“, number_sum)

print(“Type of data of the sum-“, type(number_sum))
```

Note: Running this program will display the data types and sum and display an integer and string.

Explanation

In the above program, we added number_str and number_int variable. We then converted number_str from string(higher data type) to integer(lower data type)type using int() function to perform the summation. Python manages to add the two variables after converting number_str to an integer value. The number_sum value and data type are an integer data type.

Summary

The conversion of the object from one data type to another data type is known as type conversion. The python interpreter automatically performs implicit type conversion. Implicit type conversion intends to enable Python to avoid data loss. Typecasting or explicit conversion happens when the data types of object are converted using the predefined function by the programmer. Unlike implicit conversion, typecasting can lead to data loss as we enforce the object to a particular data type.

The previous explicit conversion/typecasting can be written as:

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
number_int=431              #int data type
number_str="231"            #string data type
number_str=int(number_str)  #int function converting string into int data type
number_sum=number_int+number_str    #variable number_sum
print("Addition of number_int and number_str-", number_sum)
```

Note

Explicit conversion/Type casting requires some practice to master, but it is easy.

The trick is here:

the variable name to convert=predefined function for the desired data type (variable name to convert)

Assignment

Use the knowledge of type casting/explicit data conversion to write a program to compute the sum of:

a. score_int=76

score_str="61."

b. count_str=231

count_str="24"

Use the knowledge of type casting/explicit data conversion to write a program find the product of:

c. number_int=12

number_str="5."

d. odds_int=6

odds_str=”2”

e. minute_int=45

minute_str=”7”

Input, Output, and Import in Python

In Python input and output (I/O) tasks are performed by inbuilt functions. The print() performs output/display, while input() performs input tasks. The print() is used to output data to standard output devices such as a screen. However, it is also possible to copy data to a file. You are now familiar with the print function written as a print(). It is an inbuilt function because we do not have to write it and specify what and how it works. They are two main types of functions, inbuilt/built-in and user-defined functions that we will handle later. When we want the print function to reference or pick the value of a variable, we do not use double quotes.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

number=6 #variable definition and assignment

print(number) #displaying value stored in variable number

Note: Now assume we want to offer some explanation to the user running this program as he or she will only 6 and does not understand why 6 appears on the screen.

number=6

print(“This is a number stored in the variable number, it is:” number)

Note: We can display string and other data types in the print function. A comma is generally used to demarcate the end of one data type and the beginning of another.

Assignment

a. Given score=5, write a Python program that displays “The score is 5”. The program should reference/extract the value of score in its print function.
b. Given age=26, write a Python to display “Richard’s age is 26”. The program should reference/extract the value of score in its print function.
c. Given marks=87, write a Python program that displays “The average marks in the class is 87”. The program should reference/extract the value of score in its print function.

Note that the new line in Python is gotten by “\n” without the double quotes and outside the string or line code.

Formatting Output

It may become necessary to play around with the layout of the output to make it appealing, formatting. The str.format() method is used to format output and is visible/accessible to any string object.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

lucy=3; brian= 7

print(‘The age of lucy is {} and brian is {}’.format(lucy,brian))

Note

The expected output is “The age of lucy is 3, and Brian is 7”.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

print(‘Brian loves {1} and {0}’.format(‘soccer’, ‘movies’))

#The expected output will be: He likes soccer and moves

print(‘he likes {1} and {0}’.format(‘soccer’, ‘movies’))

#Output: He likes movies and soccer

Note

The use of tuple index/ numbers to specify the order of displaying frees up the programmer as one does not have to retype the entire string when switching the order of the variables.

Assignment

Given print(‘She studies {0} and {1}’.format(‘Health’, ‘ICT’))

a. Rewrite the Python program to display “She studies ICT and Health.”
b. Rewrite the Python program to display “She studies Health and Health.”
c. Rewrite the Python program to display “She studies ICT and ICT.”

In real life the concept above (tuple index/specifying order of display) may be useful where you don’t have to rewrite entire lines of codes. For instance, think of a website that asks users to register for an account but begins with asking surname before the first name. Now assume that the ICT team has recommended that the users should be asked their first name first before the surname. Write a simple Python simulation program to demonstrate how the tuple index concept can increase efficiency, readability, and maintainability of code.

Input in Python

So far our Python programs have been static meaning that we could not key in as everything was given to the variable before running the program. In real life, applications allow users to type in values to a program and wait for it to act on the values given. The input function (input()) is used to accept input of values from the user.

Syntax/way of using it in Python

variable name=input(‘option to include a message to the user on what is happening/can leave it out’).

Example

A program that accepts numerals from users

number=input(‘Type your number here:’) #Will display: Type a number here:

Important: Python interpreter at this stage will treat any numbers entered as a string. For Python interpreter to treat the keyed-in number as the number, we must convert the number using type conversion functions for numbers which are int() and float().

To convert the number into an integer, after input from the user does the following:

int(‘number’)

To convert the number into a float, after input from the user does the following:

float(‘number’)

Assignment

a. Create a program in Python to accept numerical data from users for their age, to capture the age of a user
b. Use explicit type conversion to convert the string entered into a floating value in a.
c. Create a program in Python to accept salary figure input from users.
d. Use explicit type conversion to convert the string into a floating value in c.
e. Write a program to accept the count of students/number of students’ in a class from users.
f. Use explicit type conversion to convert the string entered into an integer data type in e.

Import in Python

The programs we have run so far are small, but in reality, a program can be hundreds to ten thousand lines of code. In this case, a large program is broken into smaller units called modules. These modules are related but on different files normally ending with the extension .py. Python allows importing of a module to another module using the keyword import. Analogy: You probably have some of your certificates scanned and stored in your Google drive, have your notebook in your desk, have a passport photo in your phone external storage, and a laptop in your room. When writing an application for an internship, you will have to find a way of accessing all these resources, but in normal circumstances, you will only work with a few even though all of them are connected. The same is true for programs.

Example

Assume we need to access the math pi that is a different module. The following program will illustrate:

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
import.math        #referencing to the contents of math module
print(math.pi)     #Now utilizing the features found in that referenced math
```

Namespace and Scope in Python

The identifier is simply a named object. Python handles every item as an object.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
number=3
```

print(id(3)=‘, id(3))

print(‘id(number)=’, id(number))

Note: Both are referring to the same object.

Namespace in Python

When we start the Python interpreter, a namespace containing all inbuilt names is created as long the workspace remains application is active. Inbuilt functions such as print() and id() exist throughout the program.

Built-in Namespace: These are functions, methods, and associated data that immediately accessible as soon the Python interpreter loads and as such are accessible to each instance and area of the workspace.

Global Namespace: This involves the contents of a module that are accessible throughout the module. Modules can have several functions and methods.

Local Namespace: Mostly for user-defined functions, a local namespace is restricted to the particular function and outside the function access is not implicitly possible.

Variable Scope

Even though there might be several unique namespaces specified, it may not be possible to access all of the namespaces given because of scope. The concept of scope refers to a segment of the program from where we access the namespace directly without any prefix. The following are the scope types:

i. Scope containing local names, current function.

ii. Scope containing global names, the scope of the module.

iii. Scope containing built-in names of the outermost scope.

Summary

Type Conversion refers to the process of changing the value of one data type to another data type. Think of dividing two integers that lead to decimal numbers. In this case, it is necessary to convert

force the conversion of an integer into a float number. Python has two types of conversion: implicit type conversion and explicit conversion. In Python input and output (I/O) tasks are performed by inbuilt functions. The print() performs output/display, while input() performs input tasks. Namespace in Python refers to a collection of names. While different namespaces can co-exist at an instance, they are completely isolated. When we start the Python interpreter, a namespace containing all inbuilt names is created as long the workspace remains application is active.

Chapter 5:
Functions Demystified

Functions in Python help split large code into smaller units. Functions make a program more organized and easy to manage.

In Python functions will assume the syntax form below:

```
def name_of_function (arguments):
    “””docstring”””
statements(s)
```

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
def welcome(salute):
    """The Python function welcomes you to
    the individual passed in as
    parameter"""
    print("Welcome " + salute + ". Lovely Day!")
```

Calling a Method in Python

We can call a function once we have defined it from another function or program.

Calling a function simply involves typing the function name with suitable parameters.

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
welcome(‘Brenda’)
```

The output will be “Welcome Brenda. Lovely Day!’

Assignment

Write a function that when called outputs“Hello (student name), kindly submit your work by Sunday”.

Docstring

It is placed after the function header as the first statement and explains in summary what the function does. Docstring should always be placed between triple quotes to accommodate multiple line strings.

Calling/Invoking the docstring we typed earlier

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

print(welcome._doc_)

The output will be “This function welcomes you to

the individual passed in as

parameter”.

The syntax for calling/invoking the docstring is:

print(function_name. _doc_)

Python function return statement

Return syntax

return [list of expressions]

Explanation

The return statement can return a value or a None object.

Example

Print(welcome(“Richard”)) #Passing arguments and calling the function

Welcome Richard. Lovely Day!

None #the returned value

Example of Returning a Value other None

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

Variable Scope and Lifetime in Python Functions

Variables and parameters defined within a Python function have local scope implying they are not visible from outside. In Python the variable lifetime is valid as long the function executes and is the period throughout that a variable exists in memory. Returning the function destroys the function variables.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
def function_my()
   marks=15
print(“The value inside the function is:”, marks)
marks=37
function_my()
print”The value outside the function is:”,marks)
```

Function Types

They are broadly grouped into user-defined and built-in functions. The built-in functions are part of the Python interpreter while the user-defined functions are specified by the user.

Assignment

Give three examples of built-in functions in Pythons

Function Argument

Calling a function requires passing the correct number of parameters otherwise the interpreter will generate an error.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
def salute(name,mesage):
   """This function welcomes to
   the student with the provided message"""
   print("Welcome",salute + ', ' + message)
welcome("Brenda","Lovely Day!")
```

Note

The function welcome() has two parameters. We will not get any error as has been fed with two arguments. Let us try calling the function with one argument and see what happens:

```
welcome(“Brenda”)    #only one argument passed
```

Running this program will generate an error saying “TypeError: welcome() missing 1 required positional argument. The same will happen when we pass no arguments to the function.

Example 2

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
welcome()
```

The interpreter will generate an error “TypeError: welcome() missing 2 required positional arguments”.

Chapter 6: Type Conversion and Type Casting in Python

Type Conversion refers to the process of changing the value of one programming data type to another programming data type. Think of dividing two integers that lead to decimal numbers. In this case, it is necessary to convert force the conversion of an integer into a float number. Python has two types of conversion: implicit type conversion and explicit conversion.

Implicit Conversion Type

In this case, Python automatically changes one data type to another data type, and the process does not require user involvement. Implicit type conversion is mainly used with the intent of avoiding data loss.

Example

Converting Integer to Float

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
number_int=451

number_flo=4.51

number_new=number_int+number_flo

print(“the type of data of number_int-“, type(number_int))

print(“the type of data of number_flo-“, type(number_flo))

print(“value of number_new-“number_new)

print(“type of data of number_new-“, type(number_new))
```

Explanation

The programming is adding two variables one of data type integer and the other float and storing the value in a new variable of data type float. Python automatically converts small data types into larger data types to avoid prevent data loss. This is known as implicit type conversion.

Note

Python cannot, however, convert numerical data types into string data type or string data type into numerical data type implicitly. Attempting such a conversion will generate an error.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
number_int=432          #lower data type
number_str="241"        #higher data type
print("The type of data of number_int-", type(number_int))
print("The type of data of number_str-", type(number_str))
print(number_int+number_str)
```

Challenge: Can you guess why this is occurring? Python interpreter is unable to understand whether we want concatenation or addition precisely. When it assumes the concatenation, it fails to locate the other string. When it assumes the addition approach, it fails to locate the other number. The solution to the error above is to have an explicit type conversion.

Explicit Conversion

Here, programmers convert the data type of a named programming object to the needed data type. Explicit conversion is attained through the functions float(), int(), and str() among others.

The syntax for the specific or explicit conversion is:

(needed_datatype) (expression)

Example

Summing of a string and integer using by using explicit conversion

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
number_int=431
number_str=”231”
print(“Type of data of number_int-“, type(number_int))
print(“Type of data number_str prior to Type Casting-“, type(number_str))
number_str=int(number_str)
number_sum=number_int+number_str
print(“Addition of number_int and number_str-“, number_sum)
print(“Type of data of the sum-“, type(number_sum))
```

Note: Running this program will display the data types and sum and display an integer and string.

Explanation

In the above program, we added number_str and number_int variable. We then converted number_str from string(higher data type) to integer(lower data type)type using int() function to perform the summation. Python manages to add the two variables after converting number_str to an integer value. The number_sum value and data type are an integer data type.

Summary

The conversion of the object from one data type to another data type is known as type conversion. The python interpreter automatically performs implicit type conversion. Implicit type conversion intends to enable Python to avoid data loss. Typecasting or explicit conversion happens when the data types of object are converted using the predefined function by the programmer. Unlike implicit conversion, typecasting can lead to data loss as we enforce the object to a particular data type.

The previous explicit conversion/typecasting can be written as:

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
number_int=431              #int data type
number_str=”231”            #string data type
number_str=int(number_str)  #int function converting string into int data type
number_sum=number_int+number_str    #variable number_sum
print(“Addition of number_int and number_str-“, number_sum)
```

Note

Explicit conversion/Type casting requires some practice to master, but it is easy.

The trick is here:

the variable name to convert=predefined function for the desired data type (variable name to convert)

Assignment

Use the knowledge of type casting/explicit data conversion to write a program to compute the sum of:

a. score_int=76

score_str=”61.”

b. count_str=231

count_str=”24”

Use the knowledge of type casting/explicit data conversion to write a program find the product of:

c. number_int=12

number_str=”5.”

d. odds_int=6

odds_str=”2”

e. minute_int=45

minute_str=”7”

Input, Output, and Import in Python

In Python input and output (I/O) tasks are performed by inbuilt functions. The print() performs output/display, while input() performs input tasks. The print() is used to output data to standard output devices such as a screen. However, it is also possible to copy data to a file. You are now familiar with the print function written as a print(). It is an inbuilt function because we do not have to write it and specify what and how it works. They are two main types of functions, inbuilt/built-in and user-defined functions that we will handle later. When we want the print function to reference or pick the value of a variable, we do not use double quotes.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
number=6    #variable definition and assignment

print(number)    #displaying value stored in variable number
```

Note: Now assume we want to offer some explanation to the user running this program as he or she will only 6 and does not understand why 6 appears on the screen.

```
number=6

print(“This is a number stored in the variable number, it is:” number)
```

Note: We can display string and other data types in the print function. A comma is generally used to demarcate the end of one data type and the beginning of another.

Assignment

a. Given score=5, write a Python program that displays “The score is 5”. The program should reference/extract the value of score in its print function.

b. Given age=26, write a Python to display “Richard’s age is 26”. The program should reference/extract the value of score in its print function.

c. Given marks=87, write a Python program that displays “The average marks in the class is 87”. The program should reference/extract the value of score in its print function.

Note that the new line in Python is gotten by “\n” without the double quotes and outside the string or line code.

Formatting Output

It may become necessary to play around with the layout of the output to make it appealing, formatting. The str.format() method is used to format output and is visible/accessible to any string object.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
lucy=3; brian= 7
print(‘The age of lucy is {} and brian is {}’.format(lucy,brian))
```

Note

The expected output is “The age of lucy is 3, and Brian is 7”.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
print(‘Brian loves {1} and {0}’.format(‘soccer’, ‘movies’))
#The expected output will be: He likes soccer and moves
print(‘he likes {1} and {0}’.format(‘soccer’, ‘movies’))
#Output: He likes movies and soccer
```

Note

The use of tuple index/ numbers to specify the order of displaying frees up the programmer as one does not have to retype the entire string when switching the order of the variables.

Assignment

Given print('She studies {0} and {1}'.format('Health', 'ICT'))

a. Rewrite the Python program to display "She studies ICT and Health."
b. Rewrite the Python program to display "She studies Health and Health."
c. Rewrite the Python program to display "She studies ICT and ICT."

In real life the concept above (tuple index/specifying order of display) may be useful where you don't have to rewrite entire lines of codes. For instance, think of a website that asks users to register for an account but begins with asking surname before the first name. Now assume that the ICT team has recommended that the users should be asked their first name first before the surname. Write a simple Python simulation program to demonstrate how the tuple index concept can increase efficiency, readability, and maintainability of code.

Input in Python

So far our Python programs have been static meaning that we could not key in as everything was given to the variable before running the program. In real life, applications allow users to type in values to a program and wait for it to act on the values given. The input function (input()) is used to accept input of values from the user.

Syntax/way of using it in Python

variable name=input('option to include a message to the user on what is happening/can leave it out').

Example

A program that accepts numerals from users

number=input(‘Type your number here:’) #Will display: Type a number here:

Important: Python interpreter at this stage will treat any numbers entered as a string. For Python interpreter to treat the keyed-in number as the number, we must convert the number using type conversion functions for numbers which are int() and float().

To convert the number into an integer, after input from the user does the following:

int(‘number’)

To convert the number into a float, after input from the user does the following:

float(‘number’)

Assignment

a. Create a program in Python to accept numerical data from users for their age, to capture the age of a user
b. Use explicit type conversion to convert the string entered into a floating value in a.
c. Create a program in Python to accept salary figure input from users.
d. Use explicit type conversion to convert the string into a floating value in c.
e. Write a program to accept the count of students/number of students’ in a class from users.
f. Use explicit type conversion to convert the string entered into an integer data type in e.

Import in Python

The programs we have run so far are small, but in reality, a program can be hundreds to ten thousand lines of code. In this case, a large

program is broken into smaller units called modules. These modules are related but on different files normally ending with the extension .py. Python allows importing of a module to another module using the keyword import. Analogy: You probably have some of your certificates scanned and stored in your Google drive, have your notebook in your desk, have a passport photo in your phone external storage, and a laptop in your room. When writing an application for an internship, you will have to find a way of accessing all these resources, but in normal circumstances, you will only work with a few even though all of them are connected. The same is true for programs.

Example

Assume we need to access the math pi that is a different module. The following program will illustrate:

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

import.math #referencing to the contents of math module

print(math.pi) #Now utilizing the features found in that referenced math

Namespace and Scope in Python

The identifier is simply a named object. Python handles every item as an object.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

number=3

print(id(3)=‘, id(3))

print(‘id(number)=’, id(number))

Note: Both are referring to the same object.

Namespace in Python

When we start the Python interpreter, a namespace containing all inbuilt names is created as long the workspace remains application is active. Inbuilt functions such as print() and id() exist throughout the program.

Built-in Namespace: These are functions, methods, and associated data that immediately accessible as soon the Python interpreter loads and as such are accessible to each instance and area of the workspace.

Global Namespace: This involves the contents of a module that are accessible throughout the module. Modules can have several functions and methods.

Local Namespace: Mostly for user-defined functions, a local namespace is restricted to the particular function and outside the function access is not implicitly possible.

Variable Scope

Even though there might be several unique namespaces specified, it may not be possible to access all of the namespaces given because of scope. The concept of scope refers to a segment of the program from where we access the namespace directly without any prefix. The following are the scope types:

i. Scope containing local names, current function.

ii. Scope containing global names, the scope of the module.

iii. Scope containing built-in names of the outermost scope.

Summary

Type Conversion refers to the process of changing the value of one data type to another data type. Think of dividing two integers that lead to decimal numbers. In this case, it is necessary to convert force the conversion of an integer into a float number. Python has two types of conversion: implicit type conversion and explicit

conversion. In Python input and output (I/O) tasks are performed by inbuilt functions. The print() performs output/display, while input() performs input tasks. Namespace in Python refers to a collection of names. While different namespaces can co-exist at an instance, they are completely isolated. When we start the Python interpreter, a namespace containing all inbuilt names is created as long the workspace remains application is active.

Chapter 7:
Classes and Objects in Python

Python supports different programming approaches as it is a multi-paradigm. An object in Python has an attribute and behavior. It is essential to understand objects and classes when studying machine learning using Python object-oriented programming language.

Example

Car as an object:

Attributes: color, mileage, model, age

Behavior: reverse, speed, turn, roll, stop, start.

Class

It is a template for creating an object.

Example

class Car:

Note

By convention, we write the class name with the first letter as uppercase. A class name is in singular form by convention.

Syntax

class Name_of_Class:

From a class, we can construct objects by simply making an instance of the class. The class_name() operator creates an object by assigning the object to the empty method.

Class or Object Instantiation

From our class Car, we can have several objects such as a first car, second care or SUVs.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
my_car=Car()
  pass
```

Assignment

a. Create a class and an object for students.

b. Create a class and an object for the hospital.

c. Create a class and an object for a bank.

d. Create a class and an object for a police department.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
class Car:
category=”Personal Automobile”
    def __init__(self, model, insurance):
        self.model = model
        self.insurance =insurance
subaru=Car(“Subaru”,”Insured”)
toyota=Car(“Toyota”,”Uninsured”)
print(“Subaru is a {}”.format(subaru._class_.car))
print(“Toyota is a {}”.format(toyota._class_.car))
print(“{} is {}”.format(subaru.model, subaru.insurance))
print(“{} is {}”.format(toyota.model, toyota.insurance))
```

Functions

Functions defined within a body of the class are known as methods and are basic functions. Methods define the behaviors of an object.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
def __init__(self, model, insurance):
  self.model = model
  self.insurance =insurance
  def ignite(self, ignite):
  return "{} ignites {}".format(self.model, ignition)
  def stop(self):
  return "{} is now stopping".format(self.model)
subaru=Car(“Subaru”,”Insured”)
print(subaru.ignite("'Fast'"))
print(subaru.stop())
```

Note

The methods ignite() and stop() are referred to as instance methods because they are an instance of the object created.

Assignment

a. Create a class Dog and instantiate it.
b. Create a Python program to show names of two dogs and their two attributes from a.

Inheritance in Python

A way of creating a new class by using details of existing class devoid of modifying it is called inheritance. The derived class or child class is the newly formed class while the existing class is called parent or base class.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

Explanation

We created two Python classes in the program above. The classes were Dog as the base class and Pitbull as the derived class. The derived class inherits the functions of the base class. The method _init_() and the function super() are used to pull the content of _init_() method from the base class into the derived class.

Data Encapsulation/Data Hiding

Encapsulation in Python Object Oriented Programming approach is meant to help prevent data from direct modification. Private attributes in Python are denoted using a single or double underscore as a prefix.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

“__” or “_”.

```
class Tv:
  def __init__(self):
  self.__Finalprice = 800
  def offer(self):
  print("Offering Price: {}".format(self.__finalprice))
  def set_final_price(self, offer):
  self.__finalprice = offer
t = Tv()
t.offer()
t.__finalprice = 950
t.offer()
```

```
# using setter function
t.setFinalPrice(990)
t.sell()
```

Explanation

The program defined a class Tv and used _init_(0 methods to hold the final offering price of the TV. Along the way, we attempted to change the price but could not manage. The reason for the inability to change is because Python treated the _finalprice as private attributes. The only way to modify this value was through using a setter function, setMaxPrice() that takes price as a parameter.

Polymorphism

In Python, polymorphism refers to the ability to use a shared interface for several data types.

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

Explanation

The program above has defined two classes Tilapia and Shark all of which share the method jump() even though they have different functions. By creating common interface jumping_test() we allowed polymorphism in the program above. We then passed objects bonny and biggy in the jumping_test() function.

Assignment

a. In a doctor consultation room suggest the class and objects in a programming context.

b. In a football team, suggest programming class and objects.

c. In a grocery store, suggest programming class and objects.

Definition of a Class

The keyword def is used to define a class in Python. The first string in a Python class is used to describe the class even though it is not always needed.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
class Dog
'''Briefly taking about class Dog using this docstring'''
    Pass
```

Example 2

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
Class Bright:
"My other class"
b=10
def salute(self):
print('Welcome')
print(Bright.b)
print(Bright.salute)
print(Bright.__doc__)
```

Creating an Object in Python

Example from the previous class

Open the previous program file with class Bright

```
student1=Bright()
```

Explanation

The last program will create object student1, a new instance. The attributes of objects can be accessed via the specific object name prefix. The attributes can be a method or data including the matching class functions. In other terms, Bright.salute is a function object and student1.salute will be a method object.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
class Bright:
    "Another class again!"
    c = 20
    def salute(self):
    print('Hello')
student2 = Bright()
print(Bright.salute)
print(student2.salute)
student2.salute()
```

Explanation

You invoked the student2.salute() despite the parameter ‘self’ and it still worked without placing arguments. The reason for this phenomenon is because each time an object calls its method, the object itself is passed as the first argument. The implication is that student2.salute() translates into student2.salute(student2). It is the reason for the ‘self; name.

Constructors

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
class NumberComplex
class ComplexNumber:
    def __init__(self,realnum = 0,i = 0):
    self.real = realnum
    self.imaginarynum = i
    def getData(self):
    print("{0}+{1}j".format(self.realnumber,self.imaginarynum))
complex1 = NumberComplex(2,3)
complex1.getData()
complex2 = NumberComplex(5)
complex2.attribute = 10
print((complex2.realnumber, complex2.imaginarynumber, complex2.attribute))
complex1.attribute
```

Deleting Objects and Attributes

The del statement is used to delete attributes of an object at any instance.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
complex1 = NumberComplex(2,3)
del complex1.imaginarynumber
complex1.getData()
del NumberComplex.getData
complex1.getData()
```

Deleting an Object

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

complex1=NumberComplex(1,3)

del complex1

Explanation

When complex1=NumberComplex(1,3) is done, a new instance of the object gets generated in memory and the name complex1 ties with it. The object does not immediately get destroyed as itt temporarily stays in memory before the garbage collector purges it from memory. The purging of the object helps free resources bound to the object and enhances system efficiency. Garbage destruction Python refers to automatic destruction of unreferenced objects.

Inheritance in Python

In Python inheritance allows us to specify a class that takes all the functionality from the base class and adds more. It is a powerful feature of OOP.

Syntax

class ParentClass:

  Body of parent class

class ChildClass(ParentClass):

  Body of derived class

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

class Rect_mine(Rect_mine):

```
    def __init__(self):
    Shape.__init__(self,4)
    def getArea(self):
    s1, s2, s3,s4 = self.count_sides
    perimeter = (s1+s2+s3+s4)
    area = (s1*s2)
    print('The rectangle area is:' %area)
```

Example 2

```
r = rect_mine()
r.inputSides()
Type b1 : 4
Type l1 : 8
Type b2 : 4
Type l1:    8
r.dispSides()
Type b1 is 4.0
Type l1 is 8.0
Type b2 is 4.0
Type l1 is 8.0
r.getArea()
```

Function Overriding in Python

When a method is defined in both the base class and the derived class, the method in the child class/derived class will override the parent/base class. In the above example, _init_() method in Rectangle class will override the _init_() in Shape class.

Inheritance in Multiple Form

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

MultiInherit is derived from class Parent1 and Parent2.

Multilevel Inheritance

Inheriting from a derived class is called multilevel inheritance.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
class Parent:
    pass
class Multilevel1(Parent):
    pass
class Multilevel2(Multilevel1):
    pass
```

Explanation

Multilevel1 derives from Parent, and Multilevel2 derives from Multilevel1.

Method Resolution Order

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
print(issubclass(list,object))
print(isinstance(6.7,object))
print(isinstance("Welcome",object))
```

Explanation

The specific attribute in a class will be scanned first. The search will continue into parent classes. This search does not repeat searching the same class twice. The approach or order of searching is sometimes called linearization of multi derived class in Python. The Method Resolution Order refers to the rules needed to determine this order.

Operator Overloading in Python

Inbuilt classes can use operators and the same operators will behave differently with different types. An example is the + that depending on context will perform concatenation of two strings, arithmetic addition on numbers, or merge lists. Operating overloading is an OOP feature that allows assigning varying meaning to an operator subject to context.

Making A Class Compatible with Inbuilt Special Functions

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
class Planar:
    def __init__(self, x_axis= 0, y_axis = 0):
    self.x_axis = x_axis
    self.y_axis = y_axis
    def __str__(self):
    return "({0},{1})".format(self.x_axis,self.y_axis)
```

Explanation

```
planar1=Planar(3,5)
print(planar1)          #The output will be (3,5)
```

Additional inbuilt methods

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
class Planar:
    def __init__(self, x_axis= 0, y_axis = 0):
    self.x_axis = x_axis
    self.y_axis = y_axis
str(planar1)
format(planar1)
```

Explanation

It then follows that each time we invoke format(planar1) or str(planar1), Python is in effect executing planar1._str_() thus the name, special functions.

Operator + Overloading

The _add_() function addition in a class will overload the +.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
class Planar:
    def __init__(self, x_axis= 0, y_axis = 0):
    self.x_axis = x_axis
    self.y_axis = y_axis
    def __str__(self):
    return "({0},{1})".format(self.x_axis,self.y_axis)
    def __add__(self,z):
```

```
    x_axis = self.x_axis + z.x_axis
    y_axis = self.y_axis + z.y_axis
    return Planar(x_axis,y_axis)
```

Assignment

a. Print planar1 + planar2 from the example above.

Explanation

When you perform planar1+planar2 in Python, it will call planar._add_(planar2) and in turn Planar._add_(planar1, planar2).

Revisit Logical and Comparison Operators

Assignment

a. Given x=8, y=9, write a Python program that uses logical equals to test if x is equal to y.

b. Write a program that evaluates x!=y in Python programming language.

c. Write and run the following program

```
m = True
n = False
print('m and n is',m and n)
print('m or n is',m or n)
print('not m is',not n)
```

d. From the program in c., which program statement(s) evaluates to True, or False.

e. Write and run the following program in Python

```
m1 = 15
n1 = 15
m2 = 'Welcome'
n2 = 'Welcome'
```

```
m3 = [11,12,13]
n3 = [11,12,13]
print(m1 is not n1)
print(m2 is n2)
print(m3 is n3)
```

f. Which program statement(s) generate True or False states in e.

g. Write and run the following program

```
m = 'Welcome'
n = {11:'b',12:'c'}
print('W' in m)
print('Welcome' not in m)
print(10 in n)
print('b' in n)
```

h. Which program statement(s) in g. return True or False states.

The special functions needed for overloading other operators are listed below.

Overloading Comparison Operators

In Python, comparison operators can be overloaded.

Example

Assignment

a. Perform the following to the example above Planar(1,1)
b. Again perform Planar(1,1) in the above example.
c. Finally, perform Planar(1,1) from the above example.

Functions for Implementing Overloading of Comparison Operators

Summary

Python supports different programming approaches as it is a multi-paradigm. An object in Python has an attribute and behavior. From a class, we can construct objects by simply making an instance of the class. The class_name() operator creates an object by assigning the object to the empty method. The keyword def is used to define a class in Python. The first string in a Python class is used to describe the class even though it is not always needed. When a method is defined in both the base class and the derived class, the method in the child class/derived class will override the parent/base class. In the above example, _init_() method in Rectangle class will override the _init_() in Shape class.

Inbuilt classes can use operators and the same operators will behave differently with different types. An example is the + that depending on context will perform concatenation of two strings, arithmetic addition on numbers, or merge lists. Operating overloading is an OOP feature that allows assigning varying meaning to an operator subject to context.From a class, we can construct objects by simply making an instance of the class. The class_name() operator creates an object by assigning the object to the empty method.

The _init_() function is a special function and gets called whenever a new object of the corresponding class is instantiated. Functions defined within a body of the class are known as methods and are basic functions. Methods define the behaviors of an object. In Python, polymorphism refers to the ability to use a shared interface for several data types. An illustration is a program that has defined two classes Tilapia and Shark all of which share the method jump() even though they have different functions. By creating common interface jumping_test() we allowed polymorphism in the program above. We then passed objects bonny and biggy in the jumping_test() function.

Chapter 8:
Loops in Python

Before tackling flow control, it is important we explore logical operators. Comparison operators are special operators in Python that evaluate to either True or False state of the condition.

Program flow control refers to a way in which a programmer explicitly species the order of execution of program code lines. Normally, flow control involves placing some condition (s) on the program code lines. In this chapter, we will explore and test various flow controls in Python.

if…else Statement

The if..else statement in Python is a decision making when executing the program. The if…else statement will ONLY execute code if the specified condition exists.

The syntax of if…else in Python

if test expression:

Statement(s)

Explanation

The python program will only execute the statements(s) if the test expression is true. The program first evaluates the test expression before executing the statement(s). The program will not execute the statement(s) if the test expression is False. By convention, the body of it is marked by indentation while the first is not indented line signals the end.

Assignment

Think of scenarios, real-life, where the if…else condition is required.

- If you have not enrolled for a course, then you cannot sit for the exam else sit for the exam.

- If you have paid for house rent then you will be issued with acknowledgment receipt else request for more time.
- If you are a licensed driver then you can drive to school else you hire a taxi.
- If you are tired then you can watch movies else can complete the essay.
- If you are an ethical person then you will acknowledge your mistake else you will overlook the damage caused.
- If you are committed to programming then you will practice daily else you will lose interest.
- If you have signed for email alerts you will be updated frequently else you will have to check the website daily.
- If you plead guilty to all accounts you are likely to be convicted else the merit of your case will depend on cross-examination of witnesses and evidence presented.

Note

When we use the if statement alone without the else part, it will only print/display if the condition is true, it will not cater for the alternative, the case where the first condition is not present.

Example 1

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
number=5
if number>0            #The comparison operator
   print(number, “The number is a positive number”)
```

Explanation

The program contains the if the condition that tests if the given number satisfies the if condition, “is it greater than 0” since 5 is

greater than zero, the condition is satisfied the interpreter is allowed to execute the next statement which is to extract and display the numerical value including the string message. The test condition in this program is “number>0. But think of when the condition is not met, what happens? Let us look at Example 2.

Example 2

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
number=-9
if number>0:
   print(number, “This is a positive number”)
```

Explanation

The program contains only the if statement which tests the expression by testing of -9 is greater than zero since it is not the interpreter will not execute the subsequent program code lines. In real life, you will want to provide for an alternate in case the first condition is not met. This program will not display anything when executed because the if the condition has not been met. The test condition in this program is “number>0.

Assignment

Write programs in Python using if statement only to perform the following:

a. Given number=7, write a program to test and display only even numbers.

b. Given number1=8, number2=13, write a program to only display if the sum is less than 10.

c. Given count_int=57, write a program that tests if the count is more than 45 and displays, the count is above the recommended number.

d. Given marks=34, write a program that tests if the marks are less than 50 and display the message, the score is below average.

e. Given marks=78, write a program that tests if the marks are more than 50 and display the message, great performance.

f. Given number=88, write a program that tests if the number is an odd number and displays the message, Yes it is an odd number.

g. Given number=24, write a program that tests and displays if the number is even.

h. Given number =21, write a program that tests if the number is odd and displays the string, Yes it is an odd number.

Note

The execution of statements after the if expression will only happen where the if the expression evaluates to True, otherwise the statements are ignored.

if…else Statement in Python

The if…else syntax

```
if test condition:
    Statements
else:
    Statements
```

The explanation the if statement, the if…else statement will execute the body of if in the case that the tests condition is True. Should the if…else tests expression evaluate to false, the body of the else will be executed. Program blocks are denoted by indentation. The if… else provides more maneuverability when placing conditions on the code.

Example

A program that checks whether a number is positive or negative

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
number_mine=-56
if(number<0):
    print(number_mine, “The number is negative”)
else:
   print(number_mine, “The number is a positive number”)
```

Assignment

Write a Python program that uses if..else statement to perform the following

a. Given number=9, write a program that tests and displays whether the number is even or odd.
b. Given marks=76, write a program that tests and displays whether the marks are above pass mark or not bearing in mind that pass mark is 50.
c. Given number=78, write a program that tests and displays whether the number is even or odd.
d. Given marks=27, write a program that tests and displays whether the marks are above pass mark or not bearing in mind that pass mark is 50.

Assignment

Write a program that accepts age input from the user, explicitly coverts the age into integer data types, then uses if…else flow control to tests whether the person is underage or not, the legal age is 21. Include comments and indentation to improve the readability of the program.

if…elif…else Statement in Python

Now think of scenarios where we need to evaluate multiple conditions, not just one, not just two but three and more. Think of where you have to choose team member, if not Richard, then Mercy,

if not Richard and Mercy then Brian, if not Richard, Mercy, and Brian then Yvonne. Real-life scenarios may involve several choices/conditions that have to be captured when writing a program.

Remember that the elif simply refers to else if and is intended to allow for checking of multiple expressions. The if the block is evaluated first, then elif block(s), before the else block. In this case, the else block is more of a fallback option when all other conditions return false. Important to remember, despite several blocks available in if..elif..else only one block will be executed.

Example

Three conditions covered but the only one can execute at a given instance.

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

Explanation

There are three possibilities but at any given instance the only condition will exist and this qualifies the use of if family flow control statement. For three or more conditions to evaluate, the if…elif..else flow statement merits.

Nested if Statements in Python

Sometimes it happens that a condition exists but there are more sub-conditions that need to be covered and this leads to a concept known as nesting. The amount of statements to nests is not limited but you should exercise caution as you will realize nesting can lead to user errors when writing code. Nesting can also complicate maintaining of code. The only indentation can help determine the level of nesting.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
my_charact=str(input("Type a character here either 'a', 'b' or 'c':"))
if (my_charact='a'):
if(my_charact='a'):
print("a")
else if:
(my_charact='b')
print("b")
else:
  print("c")
```

Assignment

Write a program that uses the if..else flow control statement to check non-leap year and display either scenario. Include comments and indentation to enhance the readability of the program.

for Loop in Python.

Indentation is used to separate the body of for loop in Python.

Note

Simple linear list takes the following syntax:

Variable_name=[values separated by a comma]

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
numbers=[12, 3,18,10,7,2,3,6,1]    #Variable name storing the list
sum=0                    #Initialize sum before usage, very important
for cumulative in numbers:        #Iterate over the list
```

sum=sum+cumulative

print(“The sum is” ,sum)

Assignment

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

Write a Python program that uses the for loop to sum the following lists.

a. marks=[3, 8,19, 6,18,29,15]
b. ages=[12,17,14,18,11,10,16]
c. mileage=[15,67,89,123,76,83]
d. cups=[7,10,3,5,8,16,13]

range() function in Python

The range function (range()) in Python can help generate numbers. Remember in programming the first item is indexed 0.

Therefore, range(11) will generate numbers from 0 to 10.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

print(range(7))

The output will be 0,1,2,3,4,5,6

Assignment

Without writing and running a Python program what will be the output for:

a. range(16)
b. range(8)

c. range(4)

Using range() and len() and indexing

Solved Exercise

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

The output of this program will be:

I prefer omelet

I prefer fish

I prefer jazz

Assignment

Create a program in Python to iterate through the following list and include the message I listen to (each of the music genre). Use the for loop, len() and range(). Refer to the previous example on syntax.

folders=[‘Rumba’, ‘House’, ‘Rock’]

Using for loop with else

It is possible to include a for loop with else but as an option. The else block will be executed if the items contained in the sequence are exhausted.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
marks=[12, 15,17]
for i in marks:
   print(i)
else:
```

print(“No items left”)

Assignment

Write a Python program that prints all prime numbers between 1 and 50.

while Loop in Python

In Python, the while loop is used to iterate over a block of program code as long as the test condition stays True. The while loop is used in contexts where the user does not know the loop cycles that are required. As indicated earlier, the while loop body is determined through indentation.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

Caution: Failing to include the value of the counter will lead to an infinite loop.

Assignment

a. Write a Python program that utilizes the while flow control statement to display the sum of all odd numbers from 1 to 10.
b. Write a Python program that employs the while flow control statement to display the sum of all numbers from 11 to 21.
c. Write a Python program that incorporates while flow control statement to display the sum of all even numbers from 1 to 10.

Using While loop with else

If the condition is false and no break occurs, a while loop’s else part runs.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
track = 0
while track< 4:
    print("Within the loop")
track = track + 1
else:
    print("Now within the else segment")
```

Python’s break and continue

Let us use real-life analogy where we have to force a stop on iteration before it evaluates completely. Think of when cracking/breaking passwords using a simple dictionary attack that loops through all possible character combinations, you will want the program immediately it strikes the password searched without having to complete. Again, think of when recovering photos you accidentally deleted using a recovery software, you will want the recovery to stop iterating through files immediately it finds items within the specified range. The break and continue statement in Python works in a similar fashion.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
for tracker in "bring":
if tracker == "i":
break
print(tracker)
print("The End")
```

continue statement in Python

When the continue statement is used, the interpreter skips the rest of the code inside a loop for the current iteration only and the loop does not terminate. The loop continues with next iteration.

The syntax of Python continue

continue

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
for tracker in "bring":
if tracker == "i":
continue
print(tracker)
print("Finished")
```

The output of this program will be:

```
b
r
n
g
Finished
```

Analogy

Assume that you are running data recovery software and have specified skip word files (.doc, dox extension). The program will have to continue iterating even after skipping word files.

Assignment

a. Write a Python program using for loop that will break after striking “v” in the string “Oliver”.
b. Write a Python program that will continue after skipping “m” in the string “Lemon”.

pass Statement in Python

Like a comment, a pass statement does not impact the program as it leads to no operation.

The syntax of pass

pass

Think of a program code that you plan to use in future but is not currently needed. Instead of having to insert that code in future, the code can be written as pass statements.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
my_list={‘k’,’i’,’n’}
for tracker in my_list:
  pass
```

Chapter 9:
Lifetime of Variables and Functions

Keywords Arguments in Python

Python provides a way of calling functions using keyword arguments. When calling functions using keyword arguments, the order of arguments can be changed. The values of a function are matched to the argument position-wise.

Note

In the previous example function welcome when invoked as welcome(“Brenda”, “Lovely Day!”). The value “Brenda” is assigned to the argument name and “Lovely Day!” to msg.

Calling the function using keywords

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

welcome(name=”Brenda”, msg=”Lovely Day!”)

Keywords not following the order

welcome(msg=”Lovely Day!”, name=”Brenda”)

Arbitrary Arguments

It may happen that we do not have knowledge of all arguments needed to be passed into a function. Analogy: Assume that you are writing a program to welcome all new students this semester. In this case, you do not how many will report.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

*def welcome(*names):*

“””This welcome function salutes all students in the names tuple.”””

for name in names:

print(“Welcome”.name)

welcome("Lucy","Richard","Fridah","James")

The output of the program will be:

Welcome Lucy

Welcome Richard

Welcome Fridah

Welcome James

Recursion in Python

The definition of something in terms of itself is called recursion. A recursive function calls other functions.

Example

A Python program to compute integer factorials

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

Assignment

Create a program in Python to find the factorial of 7.

Python Anonymous Function

Some functions may be specified devoid of a name and this are called anonymous function. The lambda keyword is used to denote an anonymous function. Anonymous functions are also referred to as lambda functions in Python.

Syntax

lambda arguments: expression.

Lambda functions must always have one expression but can have several arguments.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
double = lambda y: y * 2
# Output: 10
print(double(5))
```

Example 2

We can use inbuilt functions such as filter () and lambda to show only even numbers in a list/tuple.

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
first_marks = [3, 7, 14, 16, 18, 21, 13, 32]
fresh_marks = list(filter(lambda n: (n%2 == 0) , first_marks))
# Output: [14, 16, 18, 32]
print(fresh_marks)
```

Lambda function and map() can be used to double individual list items.

Example 3

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
first_score = [3, 7, 14, 16, 18, 21, 13, 32]
fresh_score = list(map(lambda m: m * 2 , first_score))
# Output: [6, 14, 28, 32, 36, 42, 26, 64]
print(fresh_score)
```

Python’s Global, Local and Nonlocal

Python’s Global Variables

Variables declared outside of a function in Python are known as global variables. They are declared in global scope. A global variable can be accessed outside or inside of the function.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
y= "global"
def foo():
    print("y inside the function :", y)
foo()
print("y outside the function:", y)
```

Explanation

In the illustration above, y is a global variable and is defined a foo() to print the global variable y. When we call the foo() it will print the value of y.

Local Variables

A local variable is declared within the body of the function or in the local scope.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
def foo():
    x = "local"
```

```
foo()
print(x)
```

Explanation

Running this program will generate an error indicating ‘x’ is undefined. The error is occurring because we are trying to access local variable x in a global scope whereas foo() functions only in the local scope.

Creating a Local Variable in Python

Example

A local variable is created by declaring a variable within the function.

```
def foo():
```

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
    x = "local"
    print(x)
foo()
```

Explanation

When we execute the code, the output is expected to be:

Local

Python’s Global and Local Variable

The following example shows how to use both local and global variables in the same Python program

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
y = "global"
def foo():
    global y
    x = "local"
    y = y * 2
    print(y)
    print(x)
foo()
```

Explanation

The output of the program will be:

global global

local

Explanation

We declared y as a global variable and x as a local variable in the foo(). The * operator issued to modify the global variable y and finally, we printed both y and x.

Local and Global Variables with the same name

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
y=6
def foo():
y=11
   print(“Local variable y-“, y)
foo()
print("Global variable y-", y)
```

Python’s Nonlocal Variables

A Python’s nonlocal variable is used in a nested function whose local scope is unspecified. It is neither global nor local scope.

Example

This example shows how to create a nonlocal variable.

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
def outer():
    y = "local variable"
    def inner():
        nonlocal y
        y = "nonlocal variable"
        print("inner:", y)
    inner()
    print("outer scope:", y)
outer()
```

Global Keyword in Python

The global keyword I Python allows modification of the variable outside the current scope. The global keyword makes changes to the variable in a local context. There are rules when creating a global keyword:

A global keyword is local by default when we create a variable within a function.

It is global by default when we define a variable outside of a function and you do not need to use the global keyword.

The global keyword is used to read and write a global variable within a function.

The use of global keyword outside a function will have no effect.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
number = 3        #A global variable
def add():
    print(number)
add()
```

The output of this program will be 3.

Modifying global variable from inside the function.

```
number=3                    #a global variable
def add():
    number= number + 4      # add 4 to 3
    print(number)
add()
```

Explanation

When the program is executed it will generate an error indicating that the local variable number is referenced before assignment. The reason for encountering the error is because the global variable can only be accessed but it is not possible to modify it from inside the function. Using a global keyword would solve this.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

Modifying global variable within a function using the global keyword.

```
number = 3            # a global variable
def add():
    global number
    number= number + 1 # increment by 1
    print("Inside the function add():", number)
add()
print("In main area:", number)
```

Explanation

When the program is run, the output will be:

Inside the function add(): 4

In the main area: 4

We defined a number as a global keyword within the function add(). The variable was then incremented by 1, variable number. Then we called the add () function to print global variable c.

Creating Global Variables across Python Modules

We can create a single module config.py that will contain all global variables and share the information across several modules within the same program.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

Create config.py

```
x=0
y=”empty”
```

Then create an update.py file to modify global variables

Import config

config.x=11

config.y=”Today”

Then create a main.py file to evaluate the changes in value

import config

import update

print(config.x)

print(config.y)

Explanation

Running the main.py file will generate:

11

Today

Chapter 10:
Python Modules and Packages

Python Modules

Modules consist of definitions as well as program statements.

An illustration is a file name config.py which is considered as a module. The module name would be config. Modules are sued to help break large programs into smaller manageable and organized files as well as promoting reusability of code.

Example

Creating the First module

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
Def add(x, y):
“””This is a program to add two
   numbers and return the outcome"""
outcome=x+y
return outcome
```

Module Import

The keyword import is used to import.

Example

Import first

The dot operator can help us access a function as long as we know the name of the module.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

first.add(6,8)

Import statement in Python

The import statement can be used to access the definitions within a module via the dot operator.

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

import math

print("The PI value is", math.pi)

Import with renaming

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
import math as h
  print(“The PI value is-“,h.pi)
```

Explanation

In this case, h is our renamed math module with a view helping save typing time in some instances. When we rename the new name becomes valid and recognized one and not the original one.

From…import statement Python.

It is possible to import particular names from a module rather than importing the entire module.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
from math import pi
print("The PI value is-", pi)
```

Importing all names

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
from math import*
print("The PI value is-", pi)
```

Explanation

In this context, we are importing all definitions from a particular module but it is encouraged norm as it can lead to unseen duplicates.

Module Search Path in Python

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
import sys
sys.path
```

Python searches everywhere including the sys file.

Reloading a Module

Python will only import a module once increasing efficiency in execution.

print(“This program was executed”)

import mine

Reloading Code

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

import mine

import mine

import mine

mine.reload(mine)

Dir() built-in Python function

For discovering names contained in a module, we use the dir() inbuilt function.

Syntax

dir(module_name)

Python Package

Files in python hold modules and directories are stored in packages. A single package in Python holds similar modules. Therefore, different modules should be placed in different Python packages.

Chapter 11:
Dictionaries and Data Structures in Python

Dictionary

Believe it or not, a Python dictionary works in a very similar way to a regular dictionary. Python offers many different data structures to hold information, and the dictionary is one of the simplest and most useful. While many things in Python are iterables, not all of them are sequences and a Python dictionary falls in this category. In this article, we will talk about what a Python dictionary is, how it works, and what are its most common applications.

What is a Python Dictionary?

Getting clean and actionable data is one of the key challenges in data analysis. You can't build and fit models to data that isn't usable. A Python dictionary makes it easier to read and change data, thereby rendering it more actionable for predictive modeling.

A Python dictionary is an unordered collection of data values. Unlike other data types which hold only one value as an element, a Python dictionary holds a key: value pair. The Python dictionary is optimized in a manner that allows it to access values when the key is known.

While each key is separated by a comma in a Python Dictionary, each key-value pair is separated by a colon. Moreover, while the keys of the dictionary have to be unique and immutable (tuples, strings, integers, etcetera), the key-values can be of any type and can also be repeated any number of times. An example of a Python dictionary is shown below:

How do Python Dictionaries Work?

While there are several Python dictionary methods, there are some basic operations that need to be mastered. We will walk through the most important ones in this section.

Creating a Python dictionary

To create a Python dictionary you need to put items (each having a key and a corresponding value expressed as key: value) inside curly brackets. Each item needs to be separated from the next by a comma. As discussed above, values can repeat and be of any type. Keys, on the other hand, are unique and immutable. There is also a built-in function dict() that you can use to create a dictionary. For easier understanding note that this built in function is written as diction() in the rest of this book. Here are some examples:

Accessing Items within the Python dictionary

Accessing items in the dictionary in Python is simple enough. All you need to do is put the key name of the item within square brackets. This is important because the keys are unique and non-repeatable.

Example

To get the value of the model key:

k = thisdiction[“model”]

You can also use another of the Python dictionary methods get() to access the item. Here’s what it looks like.

k = thisdiction.get(“model”)

How to Change Values in a Python Dictionary

To change the value of an item, you once again need to refer to the key name. Here is an example.

If you have to change the value for the key “year” from 1890 to 2025:

thisdiction = {

“brand”: “Mitsubishi”,

“model”: “Toyota”,

“year”: 1890

}

thisdiction[“year”] = 2025

How do you loop through a Python Dictionary

You can use a for loop function to loop through a dictionary in Python. By default, the return value while looping through the dictionary will be the keys of the dictionary. However, there are other methods that can be used to return the values.

To print the key names:

```
for k in thisdiction:

print(k)
```

To print the values in the dictionary, one by one:

```
for k in thisdiction:

print(thisdiction[k])
```

Another way of returning the values by using the values() function

```
for k in thisdiction.values():

print(k)
```

If you want to Loop through both the keys and the values, you can use the items() function:

```
for k, m in thisdiction.items():

print(k, m)
```

How Do You Check if a Key Exists in the Dictionary

Here's how you can determine whether a particular key is actually present in the Python dictionary:

Say you have to check whether the key "model" is present in the dictionary:

```
thisdiction = {

"brand": "Mitsubishi",

"model": "Toyota",

"year": 1890
```

}

if “model” in thisdiction:

print(“Yes, ‘model’ is one of the keys in the thisdiction dictionary”)

How do you determine the number of items in the Dictionary

To determine the number of key: value pairs in the dictionary we use one of the most commonly used Python Dictionary methods, len(). Here’s how it works:

print(len(thisdiction))

How to add an item to the Python Dictionary

To add a new key: value pair to the dictionary, you have to use a new index key and then assign a value to it.

For instance,

thisdiction = {

“brand”: “Mitsubishi”,

“model”: “Toyota”,

“year”: 1890

}

thisdiction[“color”] = “pink”

print(thisdiction)

Removing Items from the Python Dictionary

Here are some of the methods to remove an item from the Python dictionary. Each approaches the same goal from a different perspective.

Method 1

This method, pop(), removes the item which has the key name that is being specified. This works well since key names are unique and immutable.

thisdiction = {

“brand”: “Mitsubishi”,

“model”: “Toyota”,

“year”: 1890

}

thisdiction.pop(“model”)

print(thisdiction)

Method 2

The popitem() method removes the item that has been added most recently. In earlier versions, this method used to remove any random item. Here’s how it works:

thisdiction = {

“brand”: “Mitsubishi”,

“model”: “Toyota”,

“year”: 1890

}

thisdiction.popitem()

print(thisdiction)

Method 3

Much like the pop() method, the del keyword removes the item whose key name has been mentioned.

thisdiction = {

“brand”: “Mitsubishi”,

“model”: “Toyota”,

“year”: 1890

}

del thisdiction[“model”]

print(thisdiction)

Method 4

Unlike the pop() method, the del keyword can also be used to delete the dictionary altogether. Here's how it can be used to do so:

```
thisdiction = {
"brand": "Mitsubishi",
"model": "Toyota",
"year": 1890
}
del thisdiction
print(thisdiction) #this will cause an error because "thisdiction" no longer exists.
```

Method 5

The clear() keyword empties the dictionary of all items without deleting the dictionary itself:

```
thisdiction = {
"brand": "Mitsubishi",
"model": "Toyota",
"year": 1890
}
thisdiction.clear()
print(thisdiction)
```

A list of Common Python Dictionary Methods

There are a number of Python Dictionary methods that can be used to perform basic operations. Here is a list of the most commonly used ones.

Method	Description
clear()	This removes all the items from the dictionary

copy()	This method returns a copy of the Python dictionary
fromkeys()	This returns a different directory with only the key : value pairs that have been specified
get()	This returns the value of the key mentioned
items()	This method returns the thuple for every key: value pair in the dictionary
keys()	This returns a list of all the Python dictionary keys in the dictionary
popitem()	In the latest version, this method deletes the most recently added item
pop()	This removes only the key that is mentioned
update()	This method updates the dictionary with certain key-value pairs that are mentioned
values()	This method simply returns the values of all the items in the list

Merits of a Dictionary in Python

Here are some of the major pros of a Python library:

It improves the readability of your code. Writing out Python dictionary keys along with values adds a layer of documentation to the code. If the code is more streamlined, it is a lot easier to debug. Ultimately, analyses get done a lot quicker and models can be fitted more efficiently.

Apart from readability, there’s also the question of sheer speed. You can look up a key in a Python dictionary very fast. The speed of a

task like looking up keys is measured by looking at how many operations it takes to finish. Looking up a key is done in constant time compared with looking up an item in a large list which is done in linear time.

To look up an item in a huge list, the computer will look through every item in the list. If every item is assigned a key-value pair then you only need to look for the key which makes the entire process much faster. A Python dictionary is basically an implementation of a hash table. Therefore, it has all the benefits of the hash table which include membership checks and speedy tasks like looking up keys.

Demerits of a Python dictionary

While a Python dictionary is easily one of the most useful tools, especially for data cleaning and data analysis, it does have a downside. Here are some demerits of using a Python dictionary.

Dictionaries are unordered. In cases where the order of the data is important, the Python dictionary is not appropriate.

Python dictionaries take up a lot more space than other data structures. The amount of space occupied increases drastically when there are many Python Dictionary keys. Of course, this isn’t too much of a disadvantage because memory isn’t very expensive.

Summary

At the end of the day, a Python dictionary represents a data structure that can prove valuable in cleaning data and making it actionable. It becomes even more valuable because it is inherently simple to use and much faster and more efficient as well.

Of course, if you are looking for a career in data science, a comprehensive course with live sessions, assessments, and placement assistance might be just what you need.

Data Structures in Python

Among the basic data types and structures in Python are the following:

Logical: bool

Numeric: int, float, complex

Sequence: list, tuple, range

Text Sequence: str

Binary Sequence: bytes, bytearray, memoryview

Map: dict

Set: set, frozenset

All of the above are classes from which object instances can be created. In addition to the above, more data types/structures are available in modules that come as part of any default Python installation: collections, heapq, array, enum, etc. Extra numeric types are available from modules numbers, decimals and fractions. The built-in function type() allows us to obtain the type of any object.

Explanation

With respect to data types, what are the differences between Python2 and Python3?

The following are important differences:

A division such as 5 / 2 returns integer value 2 in Python2 due to truncation. In Python3, this will evaluate to float value 2.5 even when the input values are only integers.

In Python2, strings were ASCII. To use Unicode, one had to use the unicode type by creating them with a prefix: name = u'Saṃsāra'. In Python3, str type is Unicode by default.

Python2 has int and long types but both these are integrated in Python3 as int. Integers can be as large as system memory allows.

What data structures in Python are immutable and mutable?

Mutable objects are those that can be changed after they are created, such as updating/adding/removing an element in a list. It can be said that mutable objects are changed in place.

Immutable objects can't be changed in place after they are created. Among the immutable basic data types/structures

are bool, int, float, complex, str, tuple, range, frozenset, and bytes.

The mutable counterparts of frozenset and bytes are set and bytearray respectively. Among the other mutable data structures are list and dict.

With immutable objects it may seem like we can modify their values by assignment. What actually happens is that a new immutable object is created and then assigned to the existing variable. This can be verified by checking the ID (using id() function) of the variable before and after assignment.

What data structures in Python are suited to handle binary data?

The fundamental built-in types for manipulating binary data are bytearray and bytes. They support memoryview that makes use of the buffer protocol to access the storage location of other binary objects without making a copy.

The module array supports storage of simple data types such as thirty-two-bit integers and double floating point values. Characters, integers and floats can be stored array types, which gives low-level access to the bytes that store the data.

What containers and sequences are available in Python?

The diagram below shows List data type and its relationship to other data types.

Containers are data structures that contain one or more objects. In Python, a container object can contain objects of different types. For that matter, a container can contain other containers at any depth. Containers may also be called **collections.**

Sequences are containers that have inherent ordering among their items. For example, a string such as str = "hello world" is a sequence of Unicode characters h, e, l, etc. Note that there is no character data type in Python, and the expression "h" is actually a 1-character string.

Sequences support two main operations (for example, sequence variable seq):

Indexing: Access a particular element: seq[0] (first element), seq[-1] (last element).

Slicing: Access a subset of elements with syntax seq[start:stop:step]: seq[0::2] (alternate elements), seq[0:3] (first three elements), seq[-3:] (last three elements). Note that the stop point is not included in the result Among the basic sequence types are list, tuple, range, str, bytes bytearray and memoryview. Conversely, dict, set and frozenset are simply containers in which elements don't have any particular order. More containers are part of collections module.

How can I construct some common containers?

The following examples are self-explanatory:

str: a = '' (empty), a = "" (empty), a = 'Hello'

bytes: a = b'' (empty), a = b"" (empty), a = b'Hello'

list: a = list() (empty), a = [] (empty), a = [1, 2, 3]

tuple: a = tuple() (empty), a = (1,) (single item), a = (1, 2, 3), a = 1, 2, 3

set: a = set() (empty), a = {1, 2, 3}

dict: a = dict() (empty), a = {} (empty), a = {1:2, 2:4, 3:9}

We can construct bytearray from bytes and frozenset from set using their respective built-in functions.

What are iterables and iterators?

An iterable is a container that can be processed element by element. For sequences, elements are processed in the order they are stored. For non-sequences, elements are processed in some arbitrary order.

Formally, any object that implements the **iterator protocol** is an iterable. The iterator protocol is defined by two special methods, __iter__() and __next__(). Calling iter() on an iterable

returns what is called an iterator. Calling next() on an iterator gives us the next element of the iterable. Thus, iterators help us process the iterable element by element.

When we use loops or comprehensions in Python, iterators are used under the hood. Programmers don't need to call iter() or next() explicitly.

Can I convert from one data type to another?

Yes, provided they are compatible. Here are some examples:

int('3') will convert from string to integer

int(3.4) will truncate float to integer

bool(0) and bool([]) will both return False

ord('A') will return the equivalent Unicode code point as an integer value

chr(65) will return the equivalent Unicode string of one character

bin(100), oct(100) and hex(100) will return string representations in their respective bases

int('45', 16) and int('0x45', 16) will convert from hexadecimal to decimal

tuple([1, 2, 3]) will convert from list to tuple

list('hello') will split the string into a list of 1-character strings

set([1, 1, 2, 3]) will remove duplicates in the list to give a set

dict([(1,2), (2,4), (3,9)]) will construct a dictionary from the given list of tuples

list({1:2, 2:4, 3:9}) will return a list based on the dictionary keys.

Should I use a list or a tuple?

If ordering is important, sets and dictionaries should not be used: prefer lists and tuples. Tuples are used to pass arguments and return results from functions. This is because they can contain multiple elements and are immutable. Tuples are also good for storing closely related data. For example, (a, b, c) coordinates or (red, green, blue)

color components can be stored as tuples. Use lists instead if values can change during the lifetime of the object.

If a sequence is to be sorted, use a list for in-place sorting. A tuple can be used but it should return a new sorted object. A tuple cannot be sorted in-place.

For better code readability, elements of a tuple can be named. For this purpose, use collections.namedtuple class. This allows us to access the elements via their names rather than tuple indices.

It's possible to convert between lists and tuples using functions list() and tuple().

When to use a set and when to use a dict?

Sets and dictionaries have no order. However, from Python 3.7, the order in which items are inserted into a dict are preserved.

Sets store unique items. Duplicates are discarded. Dictionaries can contain duplicate values but keys must be unique. Since dict keys are unique, often dict is used for counting. For example, to count the number of times a word appears in a document, words can be keys and counts can be values.

Sets are suited for finding the intersection/union of two groups, such as finding those who live in a neighborhood (set 1) and/or also own a car (set 2). Other set operations are also possible.

Strings, lists and tuples can take only integers as indices due to their ordered nature but dictionaries can be indexed by strings as well. In general, dictionaries can be indexed by any of the built-in immutable types, which are considered **hashable*.*** Thus, dictionaries are suited for key-value pairs such as mapping country names (keys) to their capitals (values). But if capitals are the more common input to your algorithm, use them as keys instead.

How can I implement a linked list in Python?

Linked list is a group of nodes connected by pointers or links**. A node** is one point of statistics or details in the linked list. Not only does it hold data but also it shows direction to the following node in a

linked list that is single. Thus, the definition of a node is recursive. For a double-linked list, the node has two pointers, one that connects to the previous node and another one that connects to the next node. Linked lists can be designed to be ordered or unordered.

The head of the linked list must be accessible. This allows us to traverse the entire list and perform all possible operations. A double-linked list might also expose the tail for traversal from the end. While a Node class may be enough to implement a linked list, it's common to encapsulate the head pointer and all operations within LinkedList class. Operations on the linked lists are methods of the class. One possible implementation is given by Downey. A DoubleLinkedList can be a derived class from LinkedList with the addition of a tail pointer and associated methods.

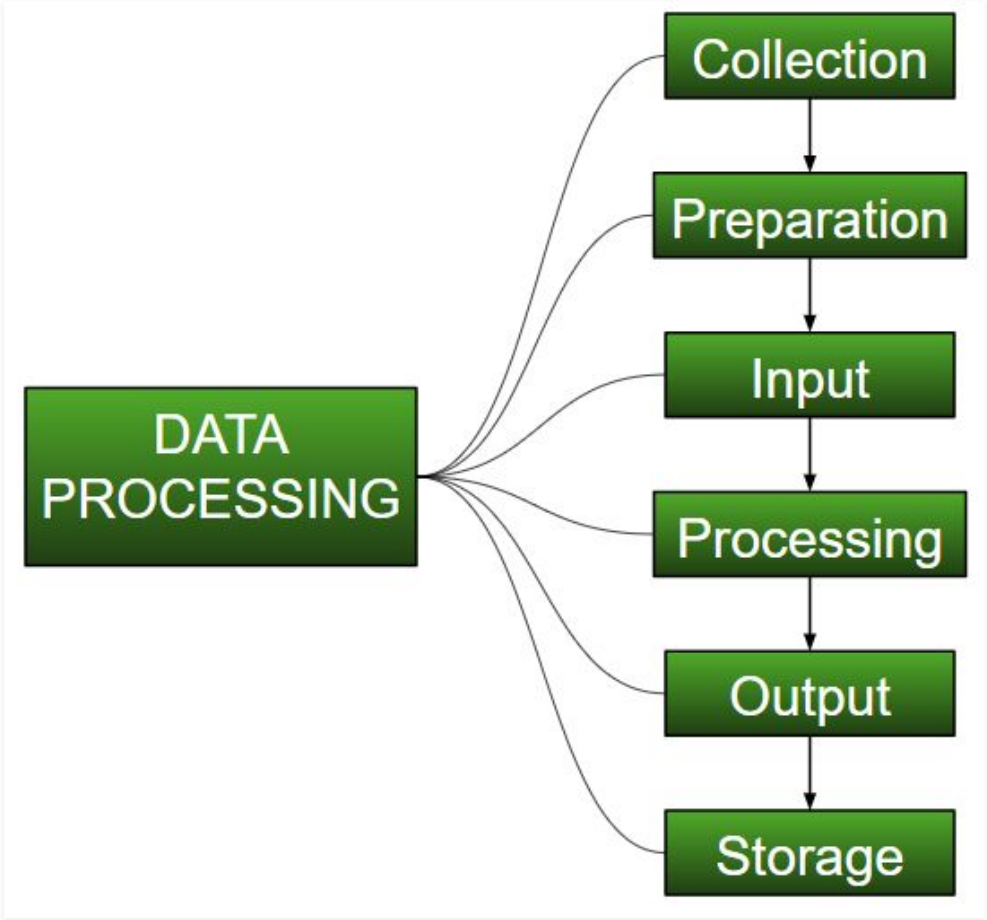
Chapter 12:
Data Processing, Analysis, and Visualization

If you work with data, then Data Visualization is an important part of your daily routine. And if you happen to use Python programming language for your analysis, you must be overwhelmed by the sheer number of choices available in the form of data visualization libraries. There are some libraries such as Matplotlib which are used for initial exploration but are not so useful for showing complex relationships in data. There are some which work well with large datasets while there are still others which majorly focus on 3D renderings. There is not a single visualization library that can be considered perfectly the best. There are certain features in one that is better than the other and vice versa. In short, there are a lot of options, and it is impossible to learn and try them all or maybe, get them all to work together. So how do we get our job done? PyViz definitely has the answer.

Understanding Data Processing

Data processing is the act of changing the nature of data into a form that is more useful and desirable. In other words, it is making data more meaningful and informative. By applying machine learning algorithms, statistical knowledge, and mathematical modeling, one can automate this whole process. The output of this whole process can be in any form like tables, graphs, charts, images, and much more, based on the activity done and the requirements of the machine.

This might appear simple, but for big organizations and companies like Facebook, Twitter, UNESCO, and health sector organizations, this whole process has to be carried out in a structured way. The diagram below shows some of the steps that are followed:



Let’s look in detail at each step:

Collection

The most important step when getting started with Machine Learning is to ensure that the data available is of great quality. You can collect data from genuine sources such as Kaggle, data.gov.in, and UCI dataset repository. Please see the following example, when students are getting ready to take a competitive exam, they always find the best resources to use to ensure they attain good results. Similarly, accurate and high-quality data will simplify the learning process of the model. This means that during the time of testing, the model would output the best results.

A great amount of time, capital, and resources are involved in data collection. This means that organizations and researchers have to select the correct type of data which they want to implement or research.

For instance, to work on the Facial Expression Recognition requires a lot of images that have different human expressions. A good data will make sure that the results of the model are correct and genuine.

Preparation

The data collected can be in raw form. Raw data cannot be directly fed into a machine. Instead, something has to be done on the data first. The preparation stage involves gathering data from a wide array of sources, analyzing the datasets, and then building a new data set for additional processing and exploration. Preparation can be done manually or automatically and the data should be prepared in numerical form to improve the rate of learning of the model.

Input

Sometimes, data already prepared can be in the form which the machine cannot read, in this case, it has to be converted into readable form. For conversion to take place, it is important for specific algorithm to be present.

To execute this task, intensive computation and accuracy is required. Please see the following example, you can collect data through sources like MNIST, audio files, twitter comments, and video clips.

Processing

In this stage, ML techniques and algorithms are required to execute instructions generated over a large volume of data with accuracy and better computation.

Output

In this phase, results get procured by the machine in a sensible way such that the user can decide to reference it. Output can appear in the form of videos, graphs, and reports.

Storage

This is the final stage where the generated output, data model, and any other important information are saved for future use.

Data Processing in Python

Let’s learn something in python libraries before looking at how you can use Python to process and analyze data. The first thing is to be familiar with some important libraries. You need to know how you can import them into the environment. There are different ways to do this in Python.

You can type:

Import math as m

From math import *

In the first way, you define an alias m to library math. Then you can use different functions from the math library by making a reference using an alias m. factorial ().

In the second method, you import the whole namespace in math. You can choose to directly apply factorial () without inferring to math.

Note:

Google recommends the first method of importing libraries because it will help you tell the origin of the functions.

The list below shows libraries that you’ll need to know where the functions originate from.

NumPy: This stands for Numerical Python. The most advanced feature of NumPy is an n-dimensional array. This library has a standard linear algebra function, advanced random number capability, and tools for integration with other low-level programming languages.

SciPy: It is the shorthand for Scientific Python. SciPy is designed on NumPy. It is among the most important library for different high-level science and engineering modules such as Linear Algebra, Sparse matrices, and Fourier transform.

Matplotlib: This is best applied when you have a lot of graphs which you need to plot. It begins from line plots to heat plots and you can apply the Pylab feature in IPython notebook to ensure plotting features are inline.

Pandas: Best applied in structured data operations and manipulations. It is widely used for data preparation and mining. Pandas were introduced recently to Python and have been very useful in enhancing Python's application in the data scientist community.

Scikit-learn: This is designed for machine learning. It was created on matplotlib, NumPy, and SciPy. This specific library has a lot of efficient tools for machine learning and statistical modeling. That includes regression, classification, clustering, and dimensionality community.

StatsModels: This library is designed for statistical modeling. Statsmodels refers to a Python module which permits users to explore data, approximate statistical models, and implement statistical tests.

Other libraries

- Requests used to access the web.
- Blaze used to support the functionality of NumPy and Pandas.
- Bokeh used to create dashboards, interactive plots, and data applications on the current web browsers.
- Seaborn is used in statistical data visualization.
- Regular expressions that are useful for discovering patterns in a text data
- NetWorx and Igraph applied to graph data manipulations.

Now that you are familiar with Python fundamentals and crucial libraries, let's now jump into problem-solving through Python.

An exploratory analysis in Python with Pandas

If you didn't know, Pandas is an important data analysis library in Python. This library has been key at improving the application of Python in the data science community. Our example uses Pandas to read a data set from an analytics Vidhya competition, run exploratory

analysis, and create a first categorization algorithm to solve this problem.

Before you can load the data, it is important to know the two major data structures in Pandas. That is Series and DataFrames.

Series and DataFrames

You can think of series as a 1-dimensional labeled array. These labels help you to understand individual elements of this series via labels.

A data frame resembles an Excel workbook, and contains column names which refer to columns as well as rows that can be accessed by row numbers. The most important difference is that column names and row numbers are referred to as column and row index.

Series and data frames create a major data model for Pandas in Python. At first, the datasets have to be read from data frames and different operations can easily be subjected to these columns.

Practice data set – Loan Prediction Problem

The following is the description of variables:

```
VARIABLE DESCRIPTIONS:
Variable                Description
Loan_ID                 Unique Loan ID
Gender                  Male/ Female
Married                 Applicant married (Y/N)
Dependents              Number of dependents
Education               Applicant Education (Graduate/ Under Graduat
e)
Self_Employed           Self employed (Y/N)
ApplicantIncome         Applicant income
CoapplicantIncome       Coapplicant income
LoanAmount              Loan amount in thousands
Loan_Amount_Term        Term of loan in months
Credit_History          credit history meets guidelines
Property_Area           Urban/ Semi Urban/ Rural
Loan_Status             Loan approved (Y/N)
```

First, start iPython interface in Inline Pylab mode by typing the command below on the terminal:

```
ipython notebook --pylab=inline
```

Import libraries and data set

This chapter will use the following python libraries:

NumPy

Matplotlib

Pandas

Once you have imported the library, you can move on and read the dataset using a function read_csv(). Below is how the code will look till this point.

```
import pandas as pd
import numpy as np
import matplotlib as plt
%matplotlib inline
#Reading the dataset in a dataframe using Pandas
df = pd.read_csv("/home/kunal/Downloads/Loan_Prediction/train.csv")
```

Notice that the dataset is stored in

“/home/kunal/Downloads/Loan_Prediction/train.csv”

Once you read the dataset, you can decide to check a few top rows by using the **function head().**

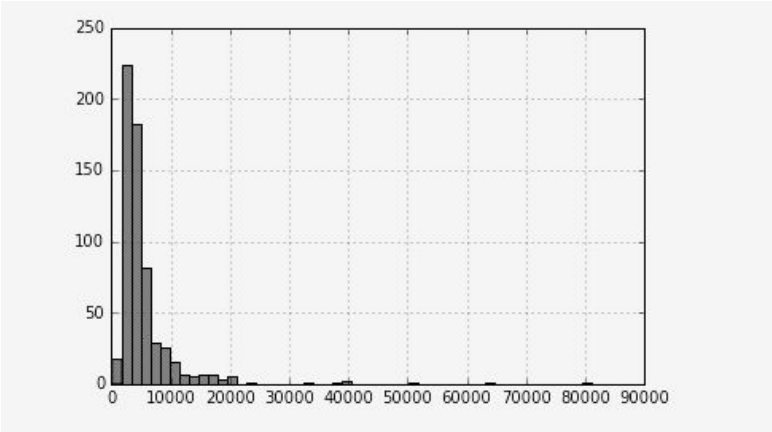
Next, you can check at the summary of numerical fields by using the **describe () function.**

Distribution analysis

Since you are familiar with basic features of data, this is the time to look at the distribution of different variables. Let’s begin with numeric variables-ApplicantIncome and LoanAmount.

First, type the commands below to plot the histogram of ApplicantIncome.

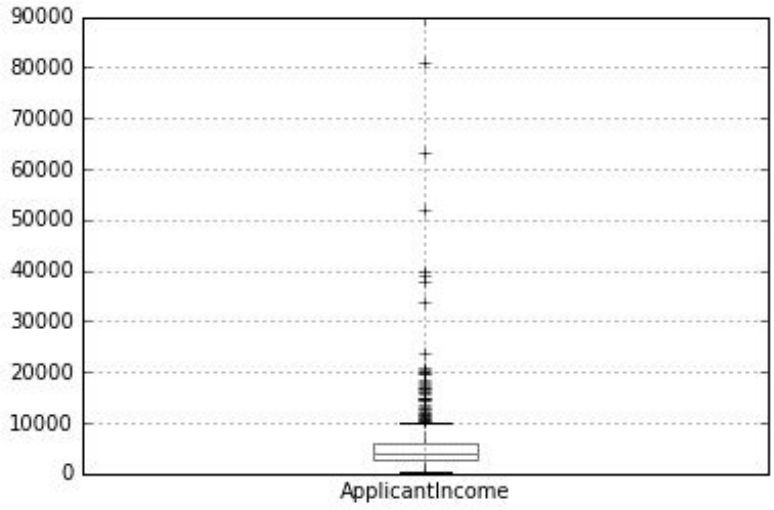
```
df['ApplicantIncome'].hist(bins=50)
```



Notice that there are a few extreme values. This is why 50 bins are needed to represent the distribution clearly.

The next thing to focus on is the box plot. The box plot for fare is plotted by:

```
df.boxplot(column='ApplicantIncome')
```



This is just a tip of an iceberg when it comes data processing in Python.

Let's look at:

Techniques for Preprocessing Data in Python

Here are the best techniques for Data Preprocessing in Python.

Rescaling Data

When you work with data that has different scales, you need to rescale the properties to have the same scale. The properties are rescaled between the range 0 to 1 and refer to it as normalization. To achieve this, the MinMaxScaler class from Scikit-learn is used. Please see the following example:

```
>>> import pandas, scipy, numpy
>>> from sklearn.preprocessing import MinMaxScaler
>>> df=pandas.read_csv( 'http://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red.csv ',sep=';')
>>> array=df.values
>>> #Separating data into input and output components
>>> x=array[:,0:8]
>>> y=array[:,8]
>>> scaler=MinMaxScaler(feature_range=(0,1))
>>> rescaledX=scaler.fit_transform(x)
>>> numpy.set_printoptions(precision=3) #Setting precision for the output
>>> rescaledX[0:5,:]
```

```
>>> rescaledX[0:5,:]
array([[0.248, 0.397, 0.   , 0.068, 0.107, 0.141, 0.099, 0.568],
       [0.283, 0.521, 0.   , 0.116, 0.144, 0.338, 0.216, 0.494],
       [0.283, 0.438, 0.04 , 0.096, 0.134, 0.197, 0.17 , 0.509],
       [0.584, 0.11 , 0.56 , 0.068, 0.105, 0.225, 0.191, 0.582],
       [0.248, 0.397, 0.   , 0.068, 0.107, 0.141, 0.099, 0.568]])
```

After rescaling, you get the values between 0 and 1. By rescaling data, it confirms the use of neural networks, optimization algorithms as well as those which have distance measures such as the k-nearest neighbors.

Normalizing Data

In the following task, you rescale every observation to a specific length of 1. For this case, you use the Normalizer class. Here is an example:

```
>>> from sklearn.preprocessing import Normalizer
>>> scaler=Normalizer().fit(x)
>>> normalizedX=scaler.transform(x)
>>> normalizedX[0:5,:]
```

```
>>> normalizedX[0:5,:]
array([[2.024e-01, 1.914e-02, 0.000e+00, 5.196e-02, 2.079e-03, 3.008e-01,
        9.299e-01, 2.729e-02],
       [1.083e-01, 1.222e-02, 0.000e+00, 3.611e-02, 1.361e-03, 3.472e-01,
        9.306e-01, 1.385e-02],
       [1.377e-01, 1.342e-02, 7.061e-04, 4.060e-02, 1.624e-03, 2.648e-01,
        9.533e-01, 1.760e-02],
       [1.767e-01, 4.416e-03, 8.833e-03, 2.997e-02, 1.183e-03, 2.681e-01,
        9.464e-01, 1.574e-02],
       [2.024e-01, 1.914e-02, 0.000e+00, 5.196e-02, 2.079e-03, 3.008e-01,
        9.299e-01, 2.729e-02]])
```

Binarizing Data

If you use the binary threshold, it is possible to change the data and make the value above it to be 1 while those that are equal to or fall below it, 0. For this task, you use the binarized class.

```
>>> from sklearn.preprocessing import Binarizer
>>> binarizer=Binarizer(threshold=0.0).fit(x)
>>> binaryX=binarizer.transform(x)
>>> binaryX[0:5,:]
```

```
>>> binaryX[0:5,:]
array([[1., 1., 0., 1., 1., 1., 1., 1.],
       [1., 1., 0., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 0., 1., 1., 1., 1., 1.]])
```

As you can see, the python code will label 0 over all values equal to or less than 0, and label 1 over the rest.

Mean Removal

This is where you remove mean from each property to center it on zero.

One Hot Encoding

When you deal with a few and scattered numerical values, you might need to store them before you can carry out the One Hot Encoding. For the k-distinct values, you can change the feature into a k-dimensional vector that has a single value of 1 and 0 for the remaining values.

```
>>> from sklearn.preprocessing import OneHotEncoder
>>> encoder=OneHotEncoder()
>>> encoder.fit([[0,1,6,2],
[1,5,3,5],
[2,4,2,7],
[1,0,4,2]
])
```

Label Encoding

Sometimes labels can be words or numbers. If you want to label the training data, you need to use words to increase its readability. Label encoding changes word labels into numbers to allow algorithms operate on them. Here's an example:

```
>>> from sklearn.preprocessing import LabelEncoder
>>> label_encoder=LabelEncoder()
>>> input_classes=['Havells','Philips','Syska','Eveready','Lloyd']
>>> label_encoder.fit(input_classes)
```

Plotting using Python Funtions

NumPy and pandas are essential tools for data wrangling. Their user-friendly interfaces and performant implementation make data handling easy. Even though they only provide a little insight into our datasets, they are absolutely valuable for wrangling, augmenting, and cleaning our datasets. Mastering these skills will improve the quality of your visualizations.

Even though the statistical concepts covered are very basic, they are necessary to enrich our visualizations with information that, in most cases, is not directly provided in our datasets. This hands-on experience will help you implement exercises and activities in the following chapters.

This will give you theoretical knowledge so that you know when to use a specific chart type and why. It will also lay down the fundamentals of the chapters, which will heavily focus on teaching you how to use Matplotlib and seaborn to create the plots discussed. After we have covered basic visualization techniques with Matplotlib and seaborn, we will dive deeper and explore the possibilities of interactive and

animated charts, which will introduce the element of storytelling into our visualizations.

What you should know about plots

Identify the best plot type for a given dataset and scenario

Explain the design practices of certain plots

Design outstanding, tangible visualizations

We will describe visualizations in detail and give practical examples, such as comparing different stocks over time or comparing the ratings for different movies. Starting with comparison plots, which are great for comparing multiple variables over time, we will look at their types, such as line charts, bar charts, and radar charts. Relation plots are handy to show relationships among variables. We will cover scatter plots for showing the relationship between two variables, bubble plots for three variables, correlograms for variable pairs, and, finally, heatmaps.

Composition plots, which are used to visualize variables that are part of a whole, as well as pie charts, stacked bar charts, stacked area charts, and Venn diagrams are going to be explained. To get a deeper insight into the distribution of variables, distribution plots are used. As a part of distribution plots, histograms, density plots, box plots, and violin plots will be covered. Finally, we will talk about dot maps, connection maps, and choropleth maps, which can be categorized into geo plots.

Comparison Plots

Comparison plots include charts that are well-suited for comparing multiple variables or variables over time. For a comparison among items, bar charts (also called column charts) are the best way to go. Line charts are great for visualizing variables over time. For a certain time period (say, less than ten time points), vertical bar charts can be used as well. Radar charts or spider plots are great for visualizing multiple variables for multiple groups.

Line Chart

Line charts are used to display quantitative values over a continuous time period and show information as a series. A line chart is ideal for a time series, which is connected by straight-line segments.

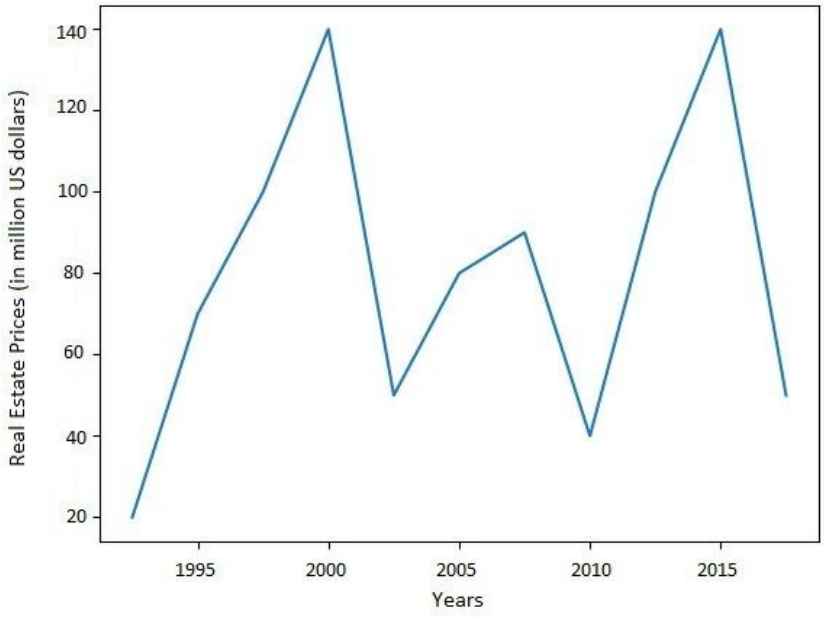
The value is placed on the y-axis, while the x-axis is the timescale.

Uses:

Line charts are great for comparing multiple variables and visualizing trends for both single as well as multiple variables, especially if your dataset has many time periods (roughly more than ten).

For smaller time periods, vertical bar charts might be the better choice.

The following diagram shows a trend of real-estate prices (in million US dollars) for two decades. Line charts are well-suited for showing data trends:



Line chart for a single variable

Example:

The following diagram is a multiple variable line chart that compares the stock-closing prices for Google, Facebook, Apple, Amazon, and Microsoft. A line chart is great for comparing values and visualizing

the trend of the stock. As we can see, Amazon shows the highest growth:

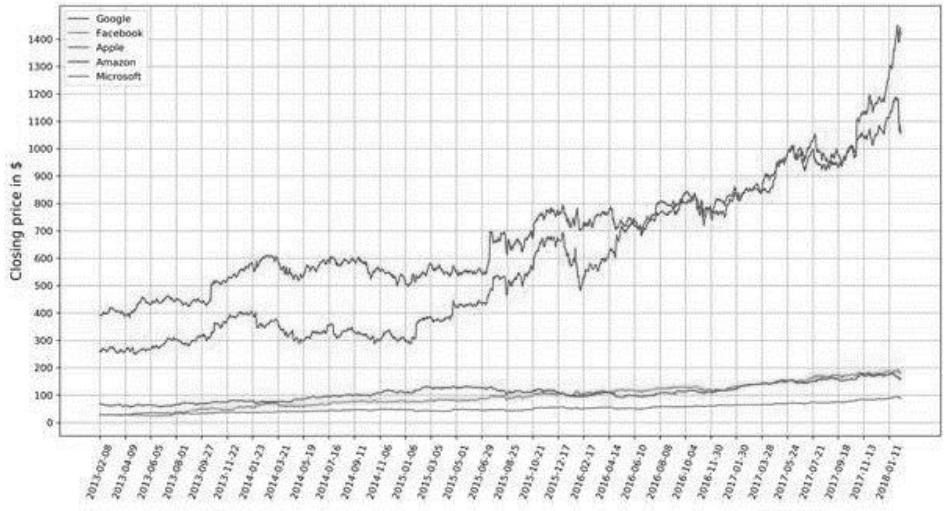


Figure: Line chart showing stock trends for the fivecompanies

Design practices:

Avoid too many lines per chart

Adjust your scale so that the trend is clearly visible

Note

Design practices for plots with multiple variables. A legend should be available to describe each variable.

Bar Chart

The bar length encodes the value. There are two variants of bar charts: vertical bar charts and horizontal bar charts.

Uses:

While they are both used to compare numerical values across categories, vertical bar charts are sometimes used to show a single variable over time.

The do's and the don'ts of bar charts:

Don't confuse vertical bar charts with histograms. Bar charts compare different variables or categories, while histograms show the

distribution for a single variable. Histograms will be discussed later in this chapter.

Another common mistake is to use bar charts to show central tendencies among groups or categories. Use box plots or violin plots to show statistical measures or distributions in these cases.

Examples:

The following diagram shows a vertical bar chart. Each bar shows the marks out of 100 that five students obtained in a test:

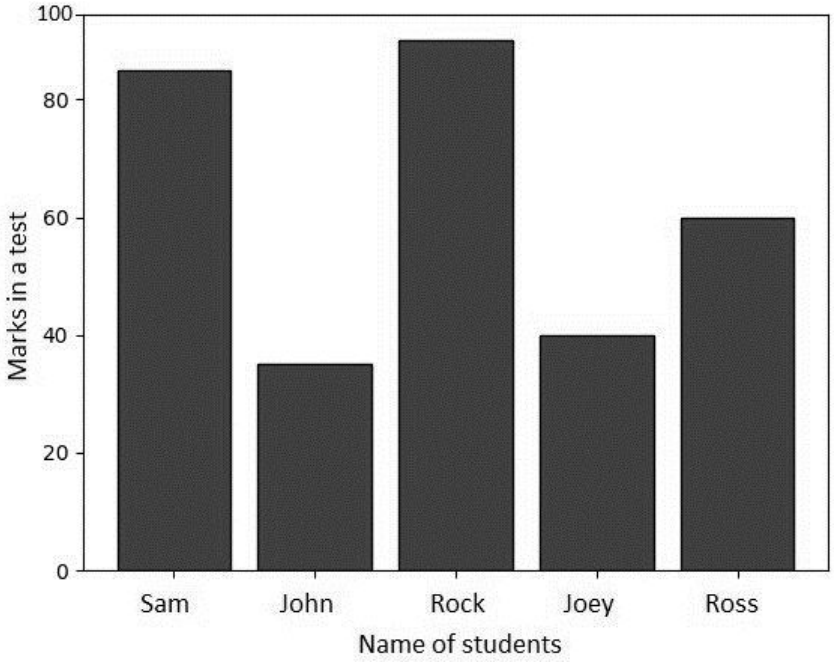


Figure: Vertical bar chart using student test data

The following diagram shows a horizontal bar chart. Each bar shows the marks out of 100 that five students obtained in a test:

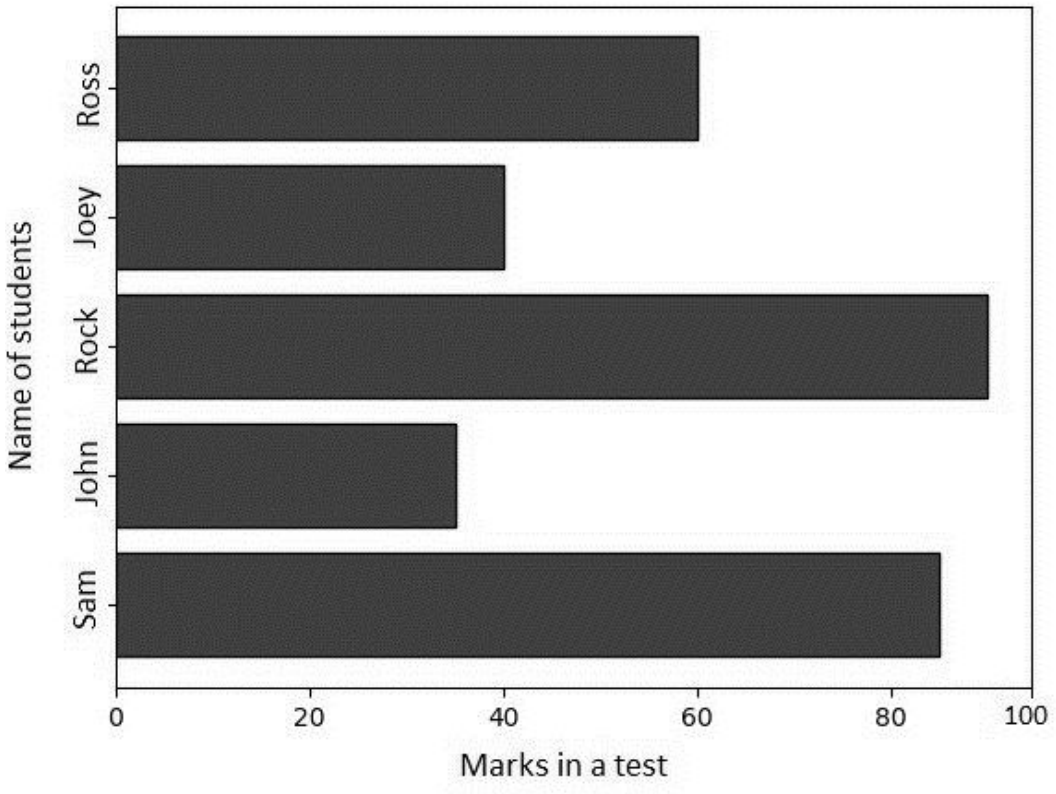
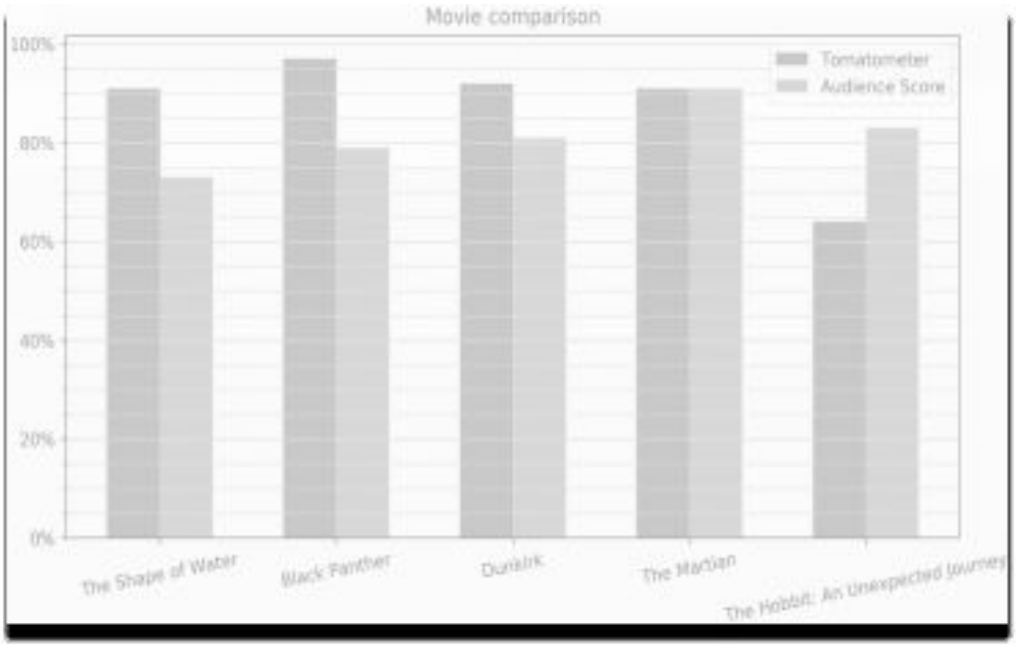


Figure: Horizontal bar chart using student test data

The following diagram compares movie ratings, giving two different scores. The Tomatometer is the percentage of approved critics who have given a positive review for the movie. The Audience Score is the percentage of users who have given a score of 3.5 or higher out of 5. As we can see, **The Martian** is the only movie with both a high Tomatometer score and Audience Score. **The Hobbit: An Unexpected Journey** has a relatively high Audience Score compared to the Tomatometer score, which might be due to a huge fan base:



Comparative bar chart

Design practices:

The axis corresponding to the numerical variable should start at zero. Starting with another value might be misleading, as it makes a small value difference look like a big one.

Use horizontal labels, that is, as long as the number of bars is small and the chart doesn't look too cluttered.

Radar Chart

Radar charts, also known as **spider** or **web charts**, visualize multiple variables with each variable plotted on its own axis, resulting in a polygon. All axes are arranged radially, starting at the center with equal distances between one another and have the same scale.

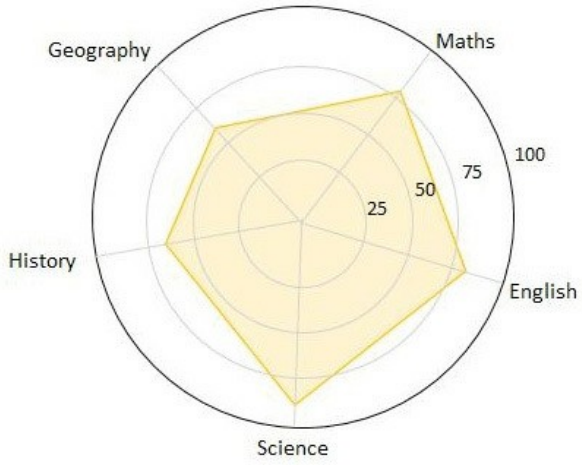
Uses:

Radar charts are great for comparing multiple quantitative variables for a single group or multiple groups.

They are also useful to show which variables score high or low within a dataset, making them ideal to visualize performance

Examples:

The following diagram shows a radar chart for a single variable. This chart displays data about a student scoring marks in different subjects:



Conclusion

Thank you for making it to the last chapter of the book *Python Programming for beginners.* I hope that you found it informative and helpful. Every measure was taken into consideration to ensure that all the chapters give you detailed and easy to understand information. I intentionally used simple language throughout the book to make sure that you get empowered after reading. The book has deliberately avoided sophisticated theories and stuck to simple Explanations that you can use at your convenience when studying.

The moment you understand programming basics using Python language, it becomes easier to learn advanced concepts such as Artificial Intelligence and Machine Learning. Artificial intelligence is important in everyday life. This book has taken you through many concepts of Python Programming such as Data Analysis, Polymorphism, Inheritance, Lists, Classes, Loops, Objects, Variables, Methods, and many more. There is no one specific thing that you can do to learn object-oriented programming overnight. However, if you follow the right steps with commitment and dedication, you will get the results you desire. Make it your routine to combine a number of practical sessions to improve your python programming skills. If you are working with an experienced programmer, follow all the instructions provided to you and ask questions where you do not understand.

The next step is to stop reading and start applying the lessons you have learned in real life. Do whatever you have identified as necessary to improve applications of programming in real life. You will realize that the majority of those who seem to have it all together lack the basic Python programming skills. Try to engage them and teach them a thing or two you have learned herein. You may even recommend or gift this book to them.

Finally, if you found this book useful in any way, a good review on Amazon will be highly appreciated!

THE ULTIMATE BEGINNERS GUIDE TO LEARN SQL PROGRAMMING:

A Smart Step By Step Guide To Learn SQL Database And Server. How To Building An Advanced And Elite Level In SQL.

Introduction

Congratulations on downloading The ultimate beginners guide to learn SQL programming: a smart step by step guide to learn SQL database and server. How to building an advanced and elite level in SQL and thank you for doing so.

The following chapters will discuss the fundamental concepts of the “structured query language (SQL)” to help the beginners looking to learn and master this analytical language for the “relational database management systems”. This book will provide you everything you need to know to efficiently and effectively use “SQL” for retrieving and updating information on SQL databases and servers, primarily focusing on the MySQL server, which is one of the highly touted interface for relational database management.

In the first chapter of this book titled “Introduction to SQL and the data definition language”, you will start with an overview of the database management systems along with the different types of database management systems and their advantages. You will be introduced to the SQL language and the five fundamental types of SQL queries namely, “Data definition language (DDL)”, “Data manipulation language (DML)”, “Data control language (DCL)”, “Data query language (DQL)” and “Transaction control language (TCL)”. This chapter will also provide you with the thumb rules required to build SQL syntax or query, which pertains to the chord structure that can be processed by the database server. You will also learn the most commonly used data types, which are integral to their learning of SQL. We will kick start the actual programming language learning with the “SQL CREATE” statements and continue to build up on these concepts. The vital concept of SQL constraints used with the “SQL ALTER” statements is also explained in detail with multiple examples and hands-on exercises.

In the chapter 2 titled, “Creating a database using SQL and maintaining data integrity”, you will be introduced to “MySQL”, which is a “free and open source relational database management system”. There are a variety of user interfaces available with “MySQL” servers including “MySQL workbench”, “phpMyAdmin”, “Adminer”, “ClusterControl”, “Database workbench”, “DBeaver”, “DBEdit”, “HeidiSQL”. “Sequel Pro”, “Toad” among others; which are used by anyone looking to execute SQL queries on the MySQL database server. You will be walked through the process of installing MySQL on your operating system(s). Since, this is an open and free program you can easily download and install it on your system, so you can make the best use of this book and included hands-on exercises and example to effectively learn the SQL programming language. You will be learning how to generate a whole new database and subsequently create tables and insert data into those tables on the “MySQL” server. You will also learn the concept of temporary tables, derived tables and how you can generate a new table from a table already present in the database.

In the chapter 3 titled, “SQL Joins and Union commands”, you will be introduced to the “SQL SELECT” statement along with the various Data manipulation clauses including “ORDER BY”, “WHERE”. The key concept of SQL Joins is presented in detail including different “SQL JOIN” functions such as “INNER JOIN”, “LEFT JOIN”, “RIGHT JOIN”, “CROSS JOIN” and “SELF JOIN”. The syntaxes of these clauses are explained in exquisite detail the with examples and pictures of the result set that you can expect to obtain while performing hands-on execution of the examples on your own MySQL instance. This effort has been made to bolster your understanding and allow hands on practice of the SQL programming language. In this chapter, you will also learn about the “SQL union” function that is used to collate a number of outputs into a single comprehensive result set. The “MySQL UNION” and “MySQL UNION ALL” statements are presented in detail along with the distinction between MySQL join and union functions.

In the chapter 4 titled, “SQL Views and Transactions”, you will learn the concept of “Database View” in SQL, which is simply a virtual table defined using the “SQL SELECT” statements with the “SQL JOIN” clause(s). There are a wide variety of advantages of using the “Database View” or “SQL View” as explained in this chapter. The “CREATE VIEW” statement is explained along with the underlying processing algorithms used in MySQL such as, "MERGE", "TEMPTABLE" and "UNDEFINED". You will also learn how to create a view from another existing view which may or may not include a sub query. The concept of “Updatable SQL Views” is also explained with examples and hands on exercises, so you can learn how to modify SQL views using “ALTER VIEW” and “CREATE OR REPLACE VIEW” statements. In SQL, transaction is defined as "a unit of work that is performed against a database". The different properties of SQL transactions and various SQL transaction statements with controlling clauses such as, “START TRANSACTION”, “COMMIT”, “ROLLBACK” among others are included in this chapter. Another important concept of database recovery models is explained here along with different types of recovery models and “SQL BACKUP” statements. The process of restoring database in case of a failure can be crucial to maintain the valuable data, as you will learn in this book.

The last chapter titled, “Database Security and Administration”, will primarily focus on the user access privileges to manage and secure the data on a MySQL server. You will be given a step-by-step walk-through on how to create new user accounts, update the user password as needed, grant and revoke access privileges to ensure that only permitted users have authorized and required access to the database.

There are plenty of books on this subject on the market, thanks again for choosing this one! Every effort was made to ensure it is full of as much useful information as possible, please enjoy

Chapter 1:
Introduction to SQL and Data Definition Language

A database could be defined as “an organized collection of information or data, which can be stored electronically on a computer system and accessed as needed”. A Database Management System (DBMS) is defined as “a software that interacts with the end users, applications and the database itself to capture and analyze the data”. “DBMS” is an integrated computer software package enabling users to communicate with a number of databases and offering access to the information stored on the database. The DBMS offers multiple features which enable big volumes of data to be entered, stored and retrieved as well as offers methods for managing how the data can be organized. Considering this close association, the term “database” is frequently used to refer to both databases as well as “DBMS”.

Existing DBMSs provide multiple features to manage and classify a database and its information into four primary functional groups as follows:

Data definition – Definitions that represent the organization of data are created, modified and eliminated.

Update – Inserting, modifying and deleting the actual data.

Retrieval - Provision of data in a form that can be used directly or processed further by other applications. Data collected may be accessible in the same form as the one in the database or in a different form, acquired through alteration or combination of data sets from various databases.

Management – User registration and tracking, data security enforcement, performance tracking, data integrity, competence control, and the recovery of data damaged by a certain event, for example, an accidental glitch in the system.

A database and its DBMS are in accordance with a given model database. "Database system" can be defined as a collection of the database, DBMS and database model.

Database servers are physically connected computers which only run DBMS and associated software, holding real databases. Multiprocessor machines with extensive memory and “RAID disk arrays” used for long term storage of the data are called as "database servers". If one of the disks fails RAID can be used to recover the data. In a heavy volume transaction processing setting, “hardware database accelerators” linked to single or multiple servers via the high-speed channel are being utilized. Most database applications have DBMS at their core. DBMS may be created with built-in network assistance around a custom multi-tasking kernel, but contemporary DBMSs typically use a normal operating system to offer the same functionalities.

As DBMS represents an important market, DBMS requirements are often taken into consideration by computers and storage providers in their own production plans.

“Databases” and “DBMSs” can be grouped by on the basis of following parameters:

The supported database models, for example, relational or XML databases.

The variety of computers on they can operate on, for example, “server cluster to a mobile device”.

The query language in use, for example, “SQL” or “XQuery”.

The “internal engineering” which drives the efficiency, scalability, resiliency, and safety of the system.

We will be primarily focusing on relational databases in this book, which pertains to the SQL language. In the beginning of the 1970s, "IBM" began to work on a prototype model based on the ideas of the computer scientist named Edgar Codd as “System R". The first version was introduced in 1974 and 1975 and followed by work on multi-table systems that allowed splitting of the data so as to

avoid storage of all the data as a single big slice for individual record. Customers tested subsequent multi-user versions in 1978 and 1979, after a standardized query language called "SQL", was added to the database. Codd's concepts forced “IBM” to create a real version of “System R”, known as “SQL / DS” and subsequently “Database 2 (DB2)”.

The "Oracle database” developed by Larry Ellison, began from a distinct chain on the basis of the work published by “IBM” on “System R”. Whilst the implementation of "Oracle V1" was finished in 1978, but it was with the release of "Oracle Version 2" in 1979 that Ellison succeeded over “IBM” in entering the market.

For the further development of a new database, "Postgres", now referred to as "PostgreSQL", Stonebraker had used the "INGRES classes". “PostgreSQL" is widely used for worldwide mission critical apps. The registrations of ‘.org’ and ‘.info’ domain names are primarily using "PostgreSQL" for data storage and similarly done by various big businesses and financial institutions. The computer programmers at the “Uppsala University in Sweden”, were also inspired by Codd's work and developed "Mimer SQL" in the mid-1970s. In 1984, this project was transitioned into an independent company.

In 1976, the "entity relationship model" was developed and became highly popular for designing databases as it produced a relatively better known description compared to the preceding “relational model”. Subsequently, the structure of the entity-relationship was rebuilt as “a data modeling construct for the relational model”, and the distinction between the two became insignificant.

“Relational databases” consist of a collection of tables that can be matched to a predetermined category. “Each table has at least one data category in a column, and each row has a certain data instance for the categories which are defined in the columns”. Relational databases have multiple names such as "Relational Database Management Systems (RDBMS)" or "SQL databases". The "Microsoft SQL Server", "Oracle Database", "MySQL" and "IBM

DB2" have historically been the most popular of Relational databases. The "Relational databases" are mainly used in big corporate settings with the exception of "MySQL", which can also be used in web-based data storage.

All relation databases can be utilized for management of “transaction oriented applications (OLTP)”. On the other hand, most non-relational databases in the classifications of “Document Stores” and “Column Store”, may also be used for OLTP, thus causing confusion between the two. OLTP databases can be considered "operational" databases, distinguished by regular quick transactions, including data updates, small volumes of data, and simultaneously processing hundreds and thousands of transactions, such as, online reservations and banking apps.

Advantages of Database Management System

The database management system (DBMS) is described as “software system that enables users to identify, develop, maintain and regulate access to the database”. DBMS allows end customers to generate information in the database as well as to read, edit and delete desired data. It can be viewed as a layer between the information and the programs utilizing that data.

DBMS offers several benefits in comparison to the "file-based data management system". Some of these advantages are listed below:

Reduction in the redundancy of data - The “file based DBMS” contain a number of files stored in a variety of location on a system and even across multiple systems. Due to this, several copies of the same file can often result in data redundancy. This can be easily avoided in a database since there is only a single database holding all the data and any modifications made to the database are immediately reflected across the entire system. Hence, there is no possibility that duplicate information will be found in the database.

Seamless data sharing - Users can share all existing data with each other found within a database. Different levels of authorizations exist within the database for selective access to the information.

Therefore, it is not possible to share the information without following the proper authorization protocols. Multiple remote users can concurrently access the database and share desired information as needed.

Data integrity - The integrity of data implies that the information in the database is reliable and accurate. The integrity of data is very crucial as a DBMS contains a variety of databases. The information contained in all of these databases is available to all the users across the board. It is therefore essential to make sure that all the databases and customers have correct and coherent data available to them at all times.

Data security - Data security is a vital in creation and maintenance of a database. Only authorized users are allowed to access the database by authenticating their identity using a valid username and password. Unauthorized users can not, under any conditions, be permitted to access the database, as it infringes upon the data integrity rules.

Privacy - The "privacy rule" in a database dictates that only authorized users are allowed to access the database on the basis of the predefined privacy constraints of the database. There are multiple database access levels and only permitted data can be viewed by the user. For example, various access limitations on the social networking sites for accounts that a user may want to access.

Backup and Recovery - DBMS is capable of generating automated backup and recovery of the database. As a result, the users are not required to backup data regularly since the DBMS can efficiently handles this. In addition, it will also restore the database to its preceding state, if a technical error or system failure occurs.

Data Consistency - In a database, the consistency of data is guaranteed, due to lack of any redundant data. Entire database contains all data consistently, and all the users accessing the database receive the same data. In addition, modifications to the

database are instantly reported to all the users to avoid any inconsistency of the existing data.

Structured Query Language

The "Structured Query Language (SQL)" in context of relational databases is considered a standard user and application program interface. In 1986, “SQL” became a standard of the "American National Standards Institute (ANSI)", and in 1987, it was added to the “International Organization for Standardization (ISO)”. Since then, the standards have been frequently improved and are endorsed by all mainstream commercial relational DBMSs (with different extents of compliance).

For the relational model, SQL was one of the initial commercial languages, even though it is distinct from the relational structure in certain aspects, according to Codd. For instance, SQL allows organization of data rows and columns.
The relational databases are highly extensible. After the initial database has been created, a new data category can be easily introduced without needing to alter any current apps.

Types of SQL Queries

Database languages are defined as “special-purpose languages”, that enable execution of one or more of the tasks listed below and often called as “sublanguages”:

“Data definition language (DDL)” – It is used to define data types including their creation, modification, or elimination as well as the relationships among them. (As you go through this chapter, you will learn more about this topic)

“Data manipulation language (DML)” –Only after the database is created and tables have been built using DDL commands, the DML commands can be used to manipulate the data within those tables and databases. The convenience of using DML commands is that they can readily be changed and rolled back if any incorrect modifications to the data or its values have been made. The DML commands used to perform specific tasks are:

“Insert” – For insertion of new rows in the table.

“Update” – For modification of the data values contained in the rows of the table.

“Deletion” – For deletion of selected rows or complete table within the database.

“Lock” – To define the user access to either “read only” or “read and write” privilege.

“Merge” – To merge couple of rows within a table.

“Data control language (DCL)” – As the name indicates, the DCL commands pertain to data control problems in a database. DCL commands provide users with unique database access permissions and are also used to define user roles as applicable. There are two DCL commands that are frequently used:

“Grant” – To give access permissions to the users.

“Revoke” – To remove the access permission given to the users.

“Data query language (DQL)” – DQL comprises of a single command that drives data selection in SQL. In conjunction with other SQL clauses, the "SELECT" command is used for collection and retrieval of data from databases or tables based on select user-applied criteria. A **"SELECT"** statement is used to search for data and computing derived information from table and/or database.

“Transaction control language (TCL)” – As indicated by its name, TCL administers transaction related issues and problems in a database. They are used to restore modifications made to the original database or confirm them.

Roll back implies the modifications "Undo," and Commit means the modifications "Apply." The 3 main TCL commands available are:

“Rollback” – Used to “cancel or undo” any updates made in the table or database.

“Commit” – Used to “deploy or apply or save” any updates made in the table or database.

“Savepoint” – Used to temporarily “save” the data in the table or database.

“Database languages” are limited to a specific data model. Some of the examples are:

“SQL” – It offers functions such as data definition, data manipulation and query as a unified language.

"OQL” - It is an “object model language standard developed by the Object Data Management Group". This language inspired the development of a few of the modern query languages such as "JDOQL" and "EJBQL".

"XQuery" – It is a “standard XML query language”, which was introduced by "XML database technologies like MarkLogic and eXist, relational databases with XML capabilities like Oracle and DB2, as well as in memory XML processors like Saxon".

“SQL / XML" – It is a combination of "SQL" and "XQuery".

SQL SYNATX

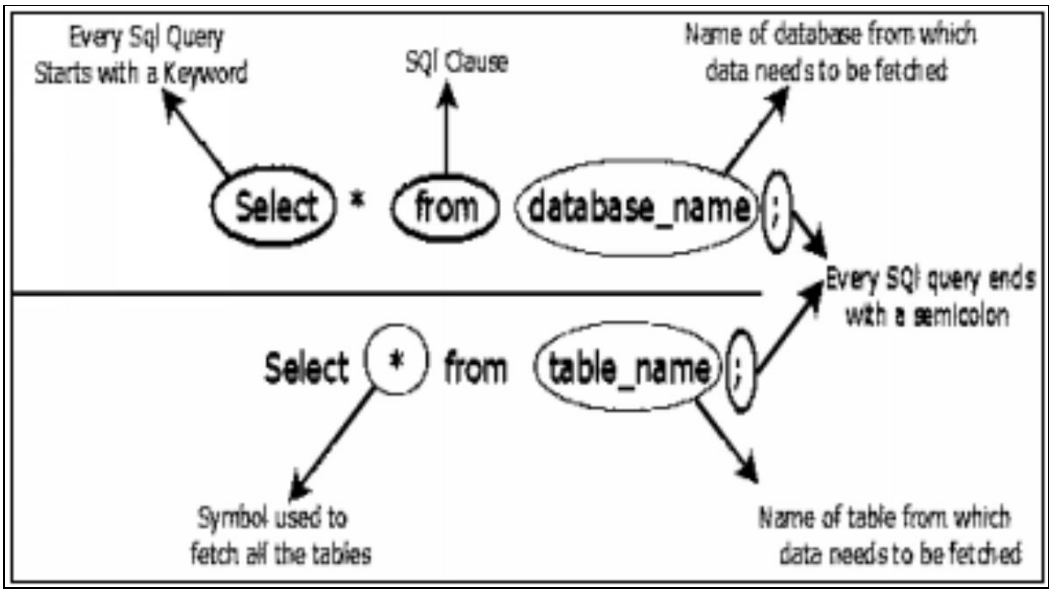
The picture below depicts the most common syntax (code structure that can be understood by the server) of a “SQL query”. Before we jump into the actual SQL commands that you can execute for a hands-on learning experience. Here are some thumb rules that you must memorize first:

A “semicolon” must be used at the end of each command.

A “keyword” must be used at the beginning of the command. You will learn all the keywords as you progress through this book.

The case structure of the alphabets is irrelevant to the commands.

You must remember the name or title of the tables and databases, used at the time of creation and make sure you always use those specific names for all your commands.



SQL Data Types

Another important concept in learning SQL, is the various data types that can be used to "define the type of data values that can be entered in specific columns". The commonly used data types are listed in the table below:

DATA TYPE	DESCRIPTION
"char(size)"	This is used to define a "fixed length character string". The size of the data value can be specified using desired number in the parenthesis as shown. Maximum allowed size is "255 characters".
"varchar(size)"	This is used to define a "variable length character string". The size of the data value can be specified using desired number in the parenthesis as shown.

	Maximum allowed size is “255 characters”.
“number(size)”	This is used to define a “numerical value” with a specified in parenthesis. The maximum number of digits in the column can be specified using desired number in the parenthesis as shown.
“date”	This is used to define a “date value” for pertinent column.
“number(size, d)”	This is used to define a “numerical value” with a specified in parenthesis. The maximum number of digits in the column can be specified using desired number in the parenthesis, along with the number of digits to the right of the decimal “d” as shown.

Data definition language (DDL)

Along with the introduction of the “Codasyl database model”, the term “data definition language” was coined. The concept of DDL required the database scheme to be developed in a syntax that described the “records, fields and sets of the user data model”. Subsequently, DDL was considered a subset of “SQL” used to declare tables, columns, data types and constraints. The third major revision of SQL called “SQL – 92”, laid the foundation for “schema manipulation language and schema information tables to query schemas”. In “SQL – 2003” the data tables were described as “SQL/Schemata”.

DDL can be used to make or perform modifications to the physical structure of a desired table in given database. All such commands can be inherently auto committed upon execution, and all modifications to the table are immediately reflected and stored across the database. Here are the most known and widely used DDL commands:

“Create” – For creation of new table(s) or an entire database.

“Alter” – To modify the data values of rows that already exist in the table.

“Rename” – To rename the table(s) or the database.

“Drop” – To delete the table(s) from a database.

“Truncate” – To delete an entire table from the database.

Now, we will explore various uses of these DDL commands to execute required operations. For ease of learning all the fixed codes or keywords will be written in CAPITAL LETTERS, so you can easily differentiate between the dummy data and command syntax.

SQL “CREATE” command

Usage – As the name suggests, “CREATE” command is primarily used to build database(s) and table(s). This always tends to be the first step in learning SQL.

Syntax:

“CREATE DATABASE database_name;”

“CREATE TABLE table_name;”

“CREATE TABLE table_name (column1 datatype, column2 datatype, column3 datatype,….,);”

Examples:

Let’s assume you would like to create a database named as “database_football”, which contains one table that stores details of

the teams called “team_details” and another table that stores details of the players called “player_details”.

The first step here would be to build the desired database, with the use of the command below:

“CREATE DATABASE database_football;”

Now, that you have your database in the system, you can create the desired tables, using the command below:

“CREATE TABLE team_details;”

“CREATE TABLE player_details;”

If you wanted to create the table “player_details” with some columns like “player_firstname” with column size as 15, “player_lastname” with column size as 25 and “player_age”. You will use the command below:

*“CREATE TABLE player_details (*player_firstname varchar (15), player_lastname varchar (25), player_age int*);”*

Remember the size of the “varchar” data type can be mentioned in the statement with the maximum size as “255 characters” and in case the column size has not been specified, the system will “default” you to the maximum size

Hands-on Exercise

For exercise purposes, imagine that you are a brand new business owner and looking to hire some employees. Now, you would first have to create a database for your company, let’s name your company “Fit Life” and create a database named “database_FitLife”. You can then add a table titled “New_Employees” with columns to store employees’ “First Name”, “Last Name”, “Job Title”, “Age” and “Salary”.

*****Use your discretion and write your SQL statements first!*****

Now you can verify your statement against the right command as given below:

“CREATE DATABASE database_FitLife;”

*“CREATE TABLE New_Employees (*employee_firstname varchar, employee_lastname varchar, employee_jobtitle varchar, employee_age int, employee_salary int*);”*

SQL “ALTER” command

Usage – This command is primarily used to “add a column, remove a column, add different data constraints and remove predefined data constraints”.

Data Constraints – In terms of SQL, “constraints” are nothing but specific rules applied to the data in the table. They can be applied at a “column level”, which is only applicable to the selected column or at a “table level”, which is applicable to all the columns in the table. They “limit” the type of data values that are allowed to be added to the table, to ensure the data in the table is accurate and reliable. Violating these constraints will abort the command that you are trying to execute.

The data constraints can be added while creating the table (using the “CREATE” command) or once the table is generated (utilizing the “ALTER” command).

The most widely used SQL constraints are:

“NOT NULL” – It is utilized to make sure that the column always contains a value and does not hold a “NULL value”. Meaning, a new record cannot be added or a record that already exists cannot be modified, if there are no values added to this column.

“UNIQUE” - This constraint is used to make sure that every single data value in a column is distinct from one another.

“PRIMARY KEY” - This constraint is essentially a “combination of a NOT NULL and UNIQUE” constraints. It serves as a unique identification for each record in the table, which is required to contain a data value. Only one “primary key” can be indicated for individual tables.

“FOREIGN KEY” - This constraint serves as a unique identification for a record/row existing in a different table that has been linked to

the table you are working on. This helps in aborting commands that will break the link between the two tables i.e. invalid data cannot be added to the “foreign key” column as it has to be a value as contained in the table that is holding that specific column.

“CHECK” - This constraint is used to make sure that all the data values in a column meet a specific condition and are limited to the predefined value range. Applying “Check” constraint on a column will allow only select values to be added to that column. However, applying “Check” constraint on a table will put restrictions on the values that can be added in specific columns, on the basis of the values defined in other columns of a row.

“DEFAULT” – It is utilized to set a “default data value” for specific column, in case no other value has been specified for that column.

“INDEX” - This constraint is utilized to “create and retrieve” data values from the database at a quick pace, using indexes which are not visible to the users. This should only be used for columns that will be searched more frequently than the others.

Syntax:

Adding Column

“ALTER TABLE table_name

ADD column_name datatype;”

Dropping Column

“ALTER TABLE table_name

DROP column_name;”

Adding “Not Null” Constraint

“ALTER TABLE table_name

MODIFY column_name datatype NOT NULL;”

Adding “Unique” Constraint

“ALTER TABLE table_name

ADD UNIQUE column_name;”

Adding “Primary Key” Constraint

“ALTER TABLE table_name

ADD PRIMARY KEY column_name;”

Adding “Foreign Key” Constraint

“ALTER TABLE table_name

ADD FOREIGN KEY (column_name) REFERENCES table_name (column_name);”

Adding “Check” Constraint

“ALTER TABLE table_name

ADD CHECK (column_name <= ‘numeric value’);”

Adding “Default” Constraint

“ALTER TABLE table_name

ADD CONSTRAINT (column_name) DEFAULT ‘data_value’ FOR (column_name);”

Adding “Index” Constraint

“CREATE INDEX index_name

ON table_name (column1, column2,…,);”

Dropping “Not Null” Constraint

“ALTER TABLE table_name

DROP CONSTRAINT column_name datatype;”

Dropping “Unique” Constraint

“ALTER TABLE table_name

DROP CONSTRAINT column_name datatype;”

Drop “Primary Key” Constraint

“ALTER TABLE table_name

DROP PRIMARY KEY;”

Dropping “Foreign Key” Constraint

“ALTER TABLE table_name

DROP FOREIGN KEY (column_name);”

Dropping “Check” Constraint

“ALTER TABLE table_name

DROP CHECK (column_name);”

Dropping “Default” Constraint

“ALTER TABLE table_name

ALTER COLUMN (column_name) DROP DEFAULT;”

Dropping “Index” Constraint

“DROP INDEX table_name.index_name;”

Examples:

Now, assume that you would like to alter the “player_details” table, that was generated in the section above. To add a column titled “player_ID” and then drop the “player_age” column, you could execute the commands below:

“ALTER TABLE player_details

ADD player_ID *number;”*

“ALTER TABLE player_details

DROP player_age*;”*

Let’s now look at how various “constraints” can be added with the “player_details” table!

You can specify that the column “player_firstname” has the “Not Null” constraint. The column “player_lastname” is unique. The column “player_ID” can be defined as the primary key and can be utilized as the foreign key in our other table titled “team_details”. We can define the “check constraint” on the column titled “player_age” to make sure all the values entered are above or equal to 20. We can also use the default value for “player_age” column as “20”. Since, we will be

frequently using the “player_lastname” column, let’s add an index to it.

Step 1:

“ALTER TABLE player_details

MODIFY player_firstname *varchar (15) NOT NULL;”*

Step 2:

“ALTER TABLE player_details

ADD UNIQUE player_lastname*;”*

Step 3:

“ALTER TABLE player_details

ADD PRIMARY KEY player_ID*;”*

Step 4:

“ALTER TABLE team_details

*ADD FOREIGN KEY (*player_ID*) REFERENCES* player_details *(*player_ID*);”*

Step 5:

“ALTER TABLE player_details

ADD CHECK (player_age <= “20”);”

Step 6:

“ALTER TABLE player_details

ADD CONSTRAINT (player_age) DEFAULT ‘20’ FOR (player_age);”

Step 7:

“CREATE INDEX idx_name

ON player_details *(*player_lastname*,* player_firstname*);”*

Now, lets drop all these constraints we just defined above, in the same order.

Step 1:

“ALTER TABLE player_details

DROP CONSTRAINT player_firstname *varchar (15) NOT NULL;”*

Step 2:

“ALTER TABLE player_details

DROP CONSTRAINT player_lastname*;”*

Step 3:

“ALTER TABLE player_details

DROP PRIMARY KEY;”

Step 4:

“ALTER TABLE team_details

*DROP FOREIGN KEY (*player_ID*);”*

Step 5:

“ALTER TABLE player_details

DROP CHECK (player_age);”

Step 6:

“ALTER TABLE player_details

ALTER COLUMN (player_age) DROP DEFAULT;”

Step 7:

“DROP INDEX player_details*.idx_name;”*

Hands-on Exercise

For exercise purposes, we will continue to use the “database_FitLife” that we created in the previous section, which contains a table titled “New_Employees” with columns to store “First Name”, “Last Name”, “Job Title”, “Age” and “Salary” of the employees.

Now, let’s add another table to this database titled “Current_Employees”. Then add a new column for employee ID, which will be the “primary key” for the “New_Employees” table and

“foreign key” for “Current_Employees” table. The salary column can be dropped for this exercise.

Make the column for the first name as “Not Null” and last name as “Unique”. Add a “check” constraint on the employee age column to be “greater than or equal to 21”. Then set the default value for the same column as “21”. And lastly, add an index for the last name column to the “New_Employees” table.

*****Use your discretion and prep your SQL statements first!*****

Now you can verify your statement against the right command as given below:

Step 1:

*“CREATE TABLE Current_Employees (*employee_firstname varchar, employee_lastname varchar, employee_jobtitle varchar, employee_age int, employee_salary int*);”*

Step 2:

“ALTER TABLE New_Employees

ADD employee_ID *number;”*

Step 3:

“ALTER TABLE New_Employees

DROP employee_salary*;”*

Step 4:

“ALTER TABLE New_Employees

ADD PRIMARY KEY employee_ID*;”*

Step 5:

“ALTER TABLE Current_Employees

*ADD FOREIGN KEY (*employee_ID*) REFERENCES* New_Employees *(*employee_ID*);”*

Step 6:

“ALTER TABLE New_Employees

MODIFY employee_firstname *varchar NOT NULL;"*

Step 7:

"ALTER TABLE New_Employees

ADD UNIQUE employee _lastname*;"*

Step 6:

"ALTER TABLE New_Employees

*ADD CHECK (*employee *_age <= "21");"*

Step 7:

"ALTER TABLE New_Employees

*ADD CONSTRAINT (*employee *_age) DEFAULT '21' FOR (*employee *_age);"*

Step 8:

"CREATE INDEX idx_name

ON New_Employees *(*employee_lastname*);"*

Viola! Now you are tasked to drop all these constraints and then add the salary column back to the "New_Employees" table.

*****Use your discretion and write your SQL statements first!*****

Now you can verify your statement against the right command as given below:

Step 1:

"ALTER TABLE New_Employees

DROP PRIMARY KEY;"

Step 2:

"ALTER TABLE Current_Employees

*DROP FOREIGN KEY (*employee_ID*);"*

Step 1:

"ALTER TABLE New_Employees

*DROP CONSTRAINT employee*_firstname *varchar NOT NULL;"*

Step 2:

"ALTER TABLE New_Employees

*DROP CONSTRAINT employee*_lastname*;"*

Step 5:

"ALTER TABLE New_Employees

DROP CHECK (employee _age);"

Step 6:

"ALTER TABLE New_Employees

ALTER COLUMN (employee_age) DROP DEFAULT;"

Step 7:

"DROP INDEX New_Employees*.idx_name;"*

Step 8:

"ALTER TABLE New_Employees

ADD employee_salary *number;"*

SQL "DROP DATABASE" command

Usage – This command is primarily used to delete an already existing SQL database.

Syntax:

"DROP DATABASE database_name;"

Examples:

Let's assume you would like to delete the database "database_football" for any business reason like irrelevant data in the database or duplicate databases etc.

You can simply use the command below and the entire database will be wiped out from the system.

"DROP DATABASE database_football;"

Hands-on Exercise

For exercise purposes, imagine that your company “Fit Life” has gone under or renamed and you want to discard all the data you have available related to the company. What would you do?

*****Use your discretion and write your SQL statements first!*****

Now you can verify your statement against the right command as given below:

“DROP DATABASE database_FitLife;”

SQL “CREATE VIEW” command

Usage – This command is primarily used to generate a “virtual view of the tables” in the database on the basis of the “result-set” of a SQL command. The view resembles an actual database table, in that it also has columns and rows but the data values can be called from a single or multiple tables in the database using SQL functions “WHERE” and “JOIN”. “SQL Views” are used to enhance the structure of the data making it more user friendly.

In SQL, the “WHERE” function is used to selectively filter the data or records on the basis of specific conditions. It is a highly versatile function and can also be used with other SQL statements including “UPDATE” and “DELETE”.

Only latest and greatest data will be used for generation of a view!

Syntax:

“CREATE VIEW view_name AS

SELECT column1, column2, ...

FROM table_name

WHERE condition;”

Examples:

Let’s go back to the database “database_football” and assume you would like to view details of the players who are 25 years old.

You can simply use the command below and view desired information.

“CREATE VIEW young_players AS

SELECT player_firstname, player_lastname

FROM player_details

WHERE player_age = ‘25’;”

Hands-on Exercise

For exercise purposes, imagine that your company “Fit Life” is undergoing an organizational change and you would like to view a list of all the employees with the job title as “managers”.

Use your discretion and write your SQL statements first!

Now you can verify your statement against the right command as given below:

“CREATE VIEW employee_managers AS

SELECT employee_firstname, employee_lastname

FROM Current_Employees

WHERE employee_jobtitle = ‘manager’;”

Chapter 2:
Creating a Database using SQL and maintaining Data integrity

“MySQL” has always been a popular “free and open source SQL based Relational Database Management System (RDMS)”. In 1995, a Swedish company called "MySQL AB" originally developed, marketed and licensed the MySQL data management system, that was eventually acquired by “Sun Microsystems” (now called “Oracle Co.”). MySQL is a "LAMP software stack web application component, an acronym for Linux, Apache, MySQL, Perl / PHP / Python". Several database-controlled web applications, including “Drupal”, “Joomla”, “PhpBB”, and “WordPress”, are using “MySQL”. It is one of the best applications for RDBMS to create various online software applications and has been used in development of multiple renowned websites including “Facebook”, “YouTube”, “Twitter” and “Flickr”.

MySQL has been used with both "C" and "C++". MySQL can be operated on several systems, including "AIX, BSDi, FreeBSD, HP-Ux, eComStation, i5/OS, IRIX, Linux, macOS, Microsoft Windows, NetBSD, Novell NetWare, Open Solaris, OS/2 Warp, QNX, Oracle Solaris, Symbian, SunOS, SCO Open Server, SCO UnixWare, Sanos and Tru64". Its SQL parser is developed in "yacc" and it utilizes an in house developed "lexical analyzer". A MySQL port for "OpenVMS" has also been developed. Dual-license distribution is utilized for the "MySQL server" and its "client libraries", which are available under "version 2 of the GPL" or a proprietary license. Additional free assistance is accessible on various IRC blogs and channels. With its "MySQL Enterprise goods", Oracle is also providing paid support for their software. The range of services and prices vary. A number of third party service providers, including "MariaDB" and "Percona", offer assistance and facilities as well.

MySQL has been given highly favorable feedback from the developer community and reviewers have noticed that it does work exceptionally well on most cases. The developer interfaces exist and there's a plethora of excellent supporting documentation, as well as feedback from web locations in the real world. “MySQL” has also been successfully passed the test of a "fast, stable and true multi-user, multi-threaded SQL database server".

“MySQL” can be found in two different editions: the "MySQL Community Server" (open source) and the "Enterprise Server", which is proprietary. MySQL Enterprise Server" differentiates itself on the basis of a series of proprietary extensions that can be installed as "server plugins", but nonetheless share the numbering scheme of versions and are developed using the same basic code.

Here are some of the key characteristics provided by “MySQL v5.6”:

“A broad subset of ANSI SQL 99, as well as extensions.”

“Cross-platform support.”

“Stored procedures, using a procedural language that closely adheres to SQL/PSM.”

“Triggers”; “Cursors”; “Updatable views”; “SSL support”; “Query caching”; “Sub-SELECTs (i.e. nested SELECTs)”; “Built-in replication support”; “Information schema.”

“Online Data Definition Language (DDL) when using the InnoDB Storage Engine.”

“Performance Schema that collects and aggregates statistics about server execution and query performance for monitoring purposes.”

“A set of SQL Mode options to control runtime behavior, including a strict mode to better adhere to SQL standards.”

“X/Open XA distributed transaction processing (DTP) support; two phase commit as part of this, using the default InnoDB storage engine.”

“Transactions with savepoints when using the default InnoDB Storage Engine.” “The NDB Cluster Storage Engine also supports

transactions.”

“ACID compliance when using InnoDB and NDB Cluster Storage Engines.”

“Asynchronous replication: master-slave from one master to many slaves or many masters to one slave.”

“Semi synchronous replication: Master to slave replication where the master waits on replication.”

“Synchronous replication: Multi-master replication is provided in MySQL Cluster.”

“Virtual Synchronous: Self managed groups of MySQL servers with multi master support can be done using: Galera Cluster or the built in Group Replication plugin.”

“Full-text indexing and searching”; “Embedded database library”; “Unicode support”.

“Partitioned tables with pruning of partitions in optimizer.”

“Shared-nothing clustering through MySQL Cluster.”

“Multiple storage engines, allowing one to choose the one that is most effective for each table in the application.”

“Native storage engines InnoDB, MyISAM, Merge, Memory (heap), Federated, Archive, CSV, Blackhole, NDB Cluster.”

“Commit grouping, gathering multiple transactions from multiple connections together to increase the number of commits per second.”

User Interfaces for MySQL

A graphical user interface (GUI) can be defined as “a type of interface that allows users to interact with electronic devices or programs through graphical icons and visual indicators such as secondary notation, as opposed to text-based interfaces, typed command labels or text navigation”. “GUIs” tend to be easier to learn in comparison to the “command-line interfaces (CLIs)”, where

commands must be entered on your keyboard. “MySQL” offers a variety of interfaces that you can leverage to best meet your need. Some of the widely popular “MySQL interfaces” are:

MySQL Workbench

MySQL Workbench is the “official built-in environment for MySQL database”. It has been created by "MySQL AB" and allows the user to supervise MySQL databases graphically and to design the databases visually. It has substituted "MySQL GUI Tools", which was the preceding software suite. “MySQL Workbench” enables users to perform database designing & model creation, “SQL implementation” (replacing "MySQL Query Browser") and “Database administration” (replacing "MySQL Administrator"). These functionalities are also available from other third-party applications but “MySQL Workbench” is still regarded as the official MySQL front end. “MySQL Workbench” can be downloaded from the official MySQL website, in two separate versions: a regular "free and open source community" version, and a "proprietary standard" version, which continues to expand and enhance the community version feature set.

phpMyAdmin

PhpMyAdmin is “a free and open source application developed in PHP programming language, that allows use of a web browser to manage MySQL administration”. It allows undertaking of multiple functions such as generation, nodificatiob, or deletion of database, table, field, or row by running SQL queries as well as management of users and permissions. phpMyAdmin is easily accessible in whopping 78 different languages and managed by "The phpMyAdmin Project". IThe data can be imported from "CSV" and "SQL" and used with a set of predefined features to convert stored data into a desirable format, such as "representing BLOB data as pictures or download links".

Adminer

Adminer (previously referred to as "phpMinAdmin") is another “free and open source MySQL front end for maintaining information in the MySQL databases”. The release of version 2, the "Adminer" can also be used on "PostgreSQL", "SQLite" and "Oracle" databases. The “Adminer” can be used to manage several databases with multiple "CSS skins" that are easily accessible. It is supplied under the "Apache License" or "GPL v2" as a single “PHP file”. Jakub Vrána, began developing "Adminer", in July 2007, as a light weight option to the original "phpMyAdmin" application.

ClusterControl

ClusterControl is defined as “an end-to-end MySQL management system or GUI for managing, monitoring, scaling and deploying versions of MySQL from a single interface”. It has been engineered by "Severalnines". The "ClusterControl community version” can be accessed for free and allow deployment and tracking of the MySQL instances, directly by the user. Advanced characteristics such as load balance, backup and restoration, among other are additional features that can be paid for.

Database Workbench

“Database Workbench" application was created by "Upscene Productions", for the creation and management of various relational databases that are interoperable within separate database systems, using SQL. “Databases Workbench" can support several database systems and offers a cross-database tool for computer programmers with a similar interface and production environment, for a variety of databases. The relational databases that can be supported by “Database Workbench” are: "Oracle Database, Microsoft SQL Server, SQL Anywhere, Firebird, NexusDB, InterBase, MySQL and MariaDB". “Database Workbench 5” can be operated on Windows systems that are either "32-bit or 64-bit" as well as on "Linux, FreeBSD, or MacOS" operating systems by leveraging the "Wine" application.

DBeaver

DBeaver is an “open source and free software application marketed under "Apache License 2.0", used as a database administration tool and a SQL client with the source code hosted on GitHub”. DBeaver incorporates extensive assistance for the following databases: "MySQL and MariaDB, PostgreSQL, Oracle, DB2 (LUW), Exasol, SQL Server, Sybase, Firebird, Teradata, Vertica, Apache Phoenix, Netezza, Informix, H2, SQLite and any other JDBC or ODBC driver database".

DBEdit

DBEdit is a "database editor, which can connect to an Oracle, DB2, MySQL and any database that provides a JDBC driver, and its source code is hosted on SourceForge ". It is also defined as “a free and open source software, which is distributed under the GNU General Public License”. It is available on "Windows, Linux and Solaris".

HeidiSQL

HeidiSQL, earlier referred to as "MySQL-Front", is another free and open source application and serves as a front end for MySQL, operable with "MariaDB", "Percona Server", "Microsoft SQL Server" and "PostgreSQL". German computer scientist Ansgar Becker and several other contributors at "Delphi", developed the "HeidiSQL" interface. To operate “HeidiSQL” databases, customers need to log in and create a session to a MySQL server locally or remotely with acceptable credentials. This session enables the users to connect and manage “MySQL Databases” on the “MySQL server” and to disconnect from the server once they have completed their task. “HeidiSQL” function set is suitable for most popular and sophisticated activities pertaining to the database, table and data records, but it continues to actively grow towards the complete feature desired in a MySQL user interface.

LibreOffice Base

LibreOffice Base enables databases to be created and managed, forms to be prepared and reports providing simple access to

information for the end users. It also serves as an interface for different database systems like "Microsoft Access", including "Access databases (JET), ODBC information sources, and MySQL or PostgreSQL".

Navicat

Navicat is a "series of graphical database management and development software for MySQL, MariaDB, Oracle, SQLite, PostgreSQL and Microsoft SQL Server manufactured by PremiumSoft CyberTech Ltd". It has a graphical user interface similar to the "Microsoft Internet Explorer" and supports various local and remote database connections. It has been designed to satisfy the demands of a range of customers, from database admins and developers to various corporations that are serving the public and share data with their partner companies. “Navicat” is a cross-platform software that can be operated on systems such as "Microsoft Windows, OS X and Linux". Once the software has been purchased, the user can select one of the available eight languages to work with their software, which are: "English, French, German, Japanese, Polish, Simplified Chinese, and Traditional Chinese".

Sequel Pro

Sequel Pro is also free and open source "Macintosh" operating system software that can be operated locally or remotely with "MySQL databases” and hosted on “Sourceforge". It utilizes the "freemium model", which effectively provides "Gratis users" with most of the fundamental features. These applications need to be managed by a SQL Table itself. For newer unicode, it can manage the latest "fun" UTF-8 functions as well as having various GB tables with little to no difficulty.

SQLBuddy

SQLBuddy is a "web-based open-source software published in PHP that manages MySQL and SQLite administration using a web

browser". The objective of this design is easy software set-up and an enhanced and convenient interface for he users.

SQLyog

SQLyog is another “MySQL GUI” application which is available for free but also offers paid software versions of their platform. There is a spreadsheet resembling interface that allows data manipulation (e.g. insert, update and delete) to be accomplished easily.

Its editor offers multiple choices for automatic formatting such as "syntax highlighting". A query can be used to manipulate both raw table data and outcome set. Its search function utilizes "Google-like search syntax", which can be translated into SQL for users transparently. It is provided with a backup utility to execute unmonitored backups. Backups can be compressed and stored, if needed, as "a file-per-table as well as identified with a timestamp".

Toad

Toad for MySQL was developed by Dell Software, as a computer application utilized by database programmers, database admins as well as data analysts, using SQL to operate on both “relational and non-relational databases”. Toad can be used with numerous databases and environments. It can be easily installed on all "32-bit/64-bit Windows platforms, including Microsoft Windows Server, Windows XP, Windows Vista, Windows 7 and 8(32-Bit or 64-Bit)". A Toad Mac Edition has also been recently published by the "Dell Software" and is offered as business version and trial / freeware version. The freeware version of Toad can be accessed from the "ToadWorld.com" community.

Webmin

Webmin was originally developed as “a web-based system configuration tool for Unix-like systems", but the newer versions are available for installation and can be operated on "Windows operating system" as well. It allows customization of internal “operating system configurations such as users, disk quotas, utilities or configuration

files” as well as “open source applications such as Apache HTTP Server, PHP or MySQL" can be modified and controlled.

“Webmin” is primarily built on "Perl", and one that runs as its own internet server and process. For communication, it would default to the "TCP port 10000" and configured to utilize "SSL" (when “OpenSSL” has already been installed on the system with all “Perl” modules that are needed for execution). It is developed around modules that have an interface with the "Webmin server" and the "configuration files". This makes adding latest features much more convenient. Because of the modular design of “Webmin”, custom plugins can be created for desktop setup for those who are keen. “Webmin” also enables control on many devices on the same subnet or LAN via one comprehensive user interface, or seamless connection to other “Webmin” hosts.

Installing MySQL on Linux/UNIX

MySQL should be installed using "RPM" on the "Linux system". The following RPMs are accessible on the "MySQL AB" official site for download:

MySQL – “The MySQL database server manages the databases and tables, controls user access and processes the SQL queries”.

MySQL client – “MySQL client programs, which make it possible to connect to and interact with the server”.

MySQL devel – “Libraries and header files that come in handy when compiling other programs that use MySQL”.

MySQL shared – “Shared libraries for the MySQL client”.

MySQL bench – “Benchmark and performance testing tools for the MySQL database server”.

In order to continue with your setup, you must follow the instructions below:

Use the root user to log into the system.

Change to the RPM-containing directory.

Execute the command below to install the MySQL database server. Keep in mind to substitute “the filename in italics with a desired file name of your RPM”.

“[root@host] # rpm -i MySQL-5.0.9-0.i386.rpm”

This function is responsible for the installation of the “MySQL server”, the creation of a “MySQL” user, the required setup and automated startup of the “MySQL server”.

You will be able to locate all “MySQL” associated binaries in "/usr / bin" and "/usr / sbin" directories. Every database and table would be generated in the "/var / lib / mysql" directory.

The commands below are optional and can be used to install the other RPMs using the same approach, but it is highly recommended to install all these RPMs on your system:

“[root@host]# rpm -i MySQL-client-5.0.9-0.i386.rpm”

“[root@host]# rpm -i MySQL-devel-5.0.9-0.i386.rpm”

“[root@host]# rpm -i MySQL-shared-5.0.9-0.i386.rpm”

“[root@host]# rpm -i MySQL-bench-5.0.9-0.i386.rpm”

Installing MySQL on Windows Operating System

The standard installation of MySQL on any Windows version has been made much easier than in the past, as MySQL is now offered as an installation package. All you have to do is “download the installer package, unzip and run the setup file”.

Installation processes in the default setup.exe are insignificant and all of them are installed under “C:\mysql” by default.

To test the server for the first time simply open user interface using the command prompt. Go to the MySQL server location that is likely going to be "C:\mysql\bin" and type “mysqld.exe –console”.

After successful installation some startup and "InnoDB" messages will be displayed on your screen. A failed installation may be related to "system permissions" issue. It is important to ensure that the

directory holding the data is available to every user (likely MySQL) under which the procedures of the database are run.

MySQL can not be added to the start menu with installation and no user friendly way is offered to exit the server. So you should consider stopping the process manually with the use of "mysql admin", "task list", "task manager", or other “Windows-specific” methods, instead of double-clicking on the "mysqld executable" to launch the server.

Installing MySQL on Macintosh Operating System

To install “MySQL” on “Mac OS X”, follow the step below:

Download the "disk image file (.dmg)" containing the "MySQL package installer". For mounting the “disk image” and viewing the contents, click twice on the ".dmg" file.

Now, click twice on the "MySQL installer package". The name of the "OS X MySQL installer package version" would be in accordance with the "MySQL" and the "OS X" version you are working on. For instance, "if you have downloaded the package for MySQL 5.6.46 and OS X 10.8, double-click mysql-5.6.46-osx-10.8-x86 64.pkg".

The opening installer message will be displayed on the screen. To start installation, click “Continue”.

A copy of the accompanying "GNU General Public License" will be displayed for all downloads of the "MySQL community version". Click "Continue" and then "Agree" to proceed.

You can select "Install" to run the installation wizard from the "Installation Type" page with all default settings. Then click "Customize" to select the components you desire to install ("MySQL server", "Preference Pane" or “Launchd Support", all of which are activated by default).

To start the installation process, click "Install".

After successful installation has been done, an "Install Succeeded" message with a brief overview will be displayed on the screen. You can exit the installation assistant at this point and start using the "MySQL server".

MySQL has been configured now, but can not be launched automatically. You can either select “launchctl” from the command line or click on "Start" on the MySQL preference pane to fire up the MySQL server.

Installations done using the package installer will configure the files within "/usr / local" directory, that matches the name of the installation version and operating system being used. For instance, “the installer file mysql-5.6.46-osx10.8-x86_64.dmg installs MySQL into /usr/local/mysql-5.6.46-osx10.8-x86_64/”. A symbolic connection to a particular directory depending on the version or platform will be generated during the setup phase from "/usr / local / mysql", which is automatically updated while you are installing the package. The table below illustrates the installation directory layouts.

Directory	**Contents of Directory**
bin, scripts	“mysqld server, client and utility programs”
data	“Log files, databases”
docs	“Helper documents, like the Release Notes and build information”
include	“Include (header) files”
lib	“Libraries”
man	“Unix manual pages”
mysql-test	“MySQL test suite”
share	“Miscellaneous support files, including error messages, sample configuration files, SQL for database installation”
sql-bench	“Benchmarks”

Directory	Contents of Directory
support-files	“Scripts and sample configuration files”
/tmp/mysql.sock	“Location of the MySQL Unix socket”

Creating a new database using “MySQL server”

In “MySQL server”, databases are implemented as directories that contain all the files corresponding to the tables in a particular database.

Now that you have installed the “MySQL Server”, you can follow the instructions below to generate new databases with the “CREATE” command, using the syntax below:

“CREATE DATABASE [IF NOT EXISTS] database_name”

“[CHARACTER SET charset_name]”

“[COLLATE collation_name]”

In the syntax above, the first step is specifying the “database_name” right after the “CREATE DATABASE” command. In the “MySQL server” instance the name of the new database must be unique. When you're trying to build a database with an existing name, the server will abort the action. Next you can indicate the choice "IF NOT EXISTS" to prevent an infringement, if you erroneously generate a new database that shares its name with a database that already exists on the server. In this scenario, MySQL will not generate an issue but will instead terminate the "CREATE DATABASE" statement. Lastly, when the new database is created, you are able to indicate the character set and combination requirements. If the clauses "CHARACTER SET" and "COLLATE" are omitted, then “MySQL” utilizes the default settings for the new database.

Alternatively, you can build a new database using **“MySQL Workbench”** and following the steps below:

Open “MySQL Workbench” and click on the “setup new connection” button on the screen.

Enter desired name for your connection and click on the "Test Connection” button. A window pane requiring the "root user" password would be displayed in "MySQL Workbench". You must enter the "root user password", check the "Save password in vault" and select “OK”.

To connect to the “MySQL server”, click twice on the "connection name Local". MySQL Workbench opens the window containing four different components namely: "Navigator, Query, Information, and Output".

Click the "create a new schema in the connected server" button from the toolbar. The "schema" is synonymous with the "database" in MySQL. Therefore, developing a new schema just implies that you are generating a new database.

In the subsequent window you must enter the “schema name” and modify the “character set and collation” as required and click on the “Apply” button.

“MySQL Workbench” will open up a new window displaying the SQL script that needs to be executed. Keep in mind that the "CREATE SCHEMA" query will lead to the same result as the "CREATE DATABASE" query. If all goes well, the new database will be generated and displayed in the “schemas tab of the Navigator section”.

To select the "testdb2 database", simply right click on the database name and select "Set as Default Schema".

The “testdb2 node” would be open so you can run queries on "testdb2" using the MySQL Workbench.

Managing a database using “MySQL server”

Once all the databases containing your desired data have been created, you can selectively display and use the content needed using the instructions below.

Display Databases

The "SHOW DATABASES" statement is utilized to list all the databases currently existing on the “MySQL server”. You can inspect only your database or every database on the server through the use of a "SHOW DATABASES" statement before building a new database. For example, if you assume that the databases we have created so far are all on the “MySQL Server” and the *“SHOW DATABASES;”* statement is executed, you will see the result as “information_schema; *database_football; database_FitLife; mysql”.*

Now you know that there are four different databases on your MySQL server. The “information_schema” and “mysql” are default databases that are created upon installation of the MySQL server and the other two databases, namely “*database_football”* and “*database_FitLife”* were created by us.

Selection of a database

Before operating with a specific database, you are required to first inform MySQL, which database you would like to operate by utilizing the "USE" command as below:

“USE database_name;”

To select the “*database_FitLife”, you can write the command as below:*

“USE database_FitLife;”

Creating a new database table using “MySQL server”

You will be using the “MySQL CREATE TABLE" statement to build a new table in a database. The "CREATE TABLE" command is considered as one of the most complicated statements in SQL. A simple “CREATE TABLE” syntax can be written as below:

“CREATE TABLE [IF NOT EXISTS] table_name(

column_list

) ENGINE=storage_engine”

Start by indicating the table name after the "CREATE TABLE" clause, that you would like to create. The name of the new table must be unique. The "IF NOT EXISTS" is an auxiliary clause for verification as to whether the table you are trying to build already happens to be in that database. If there is a duplication in the table name, the entire statement will be ignored by MySQL and the new table will not be generated. You are advised to use the "IF NOT EXISTS" clause in every "CREATE TABLE" statement, in order to prevent an accidental creation of a new table name that can be found on the server.

Next, in the "column list" section, a list of columns for the table can be specified, and each column name can be distinguished with the use of commas.

Lastly, in the "ENGINE" clause, you have the optiona to indicate the table storage engine. Any storage engine like "InnoDB" and "MyISAM" could be used. If the storage engine is not specifically declared, MySQL will be using the default "InnoDB" engine.

The following syntax can be utilized to describe a table column in the "CREATE TABLE" statement:

“column_name data_type(length) [NOT NULL] [DEFAULT value] [AUTO_INCREMENT]”

In the syntax above, the column name has been specified by the “column_name” clause. Each column contains a particular type and the desired length of the data, for example, “VARCHAR (255)”. The "NOT NULL" clause specifies that NULL value is not permitted by this column.

The value "DEFAULT" can be utilized to indicate the default value for specific column. The "AUTO_INCREMENT" shows that when a new row is added to a table, the column value will be automatically created by the column. The “AUTO_INCREMENT” columns in each table are unique. For instance, you can create a table named “tasks”, with the command below:

“CREATE TABLE IF NOT EXISTS tasks (

task_id INT AUTO_INCREMENT,

title VARCHAR(255) NOT NULL,

start_date DATE,

due_date DATE,

status TINYINT NOT NULL,

priority TINYINT NOT NULL,

description TEXT,

PRIMARY KEY (task_id)

) ENGINE=INNODB;”

In the syntax above, the "task Id" is an “auto increment column”. If the "INSERT" query is used to add a new record to the table, with the "task Id" column value not being specified, the "task Id" column will be given an “auto-generated” integer value starting with '1'. The "task Id" column has been specified as the “primary key” for the table.

The "title" column has been given a "variable character string" data type with maximum allowed length of “255 characters”. This implies that a string with a length larger than “255 characters” can not be inserted in this column. The "NOT NULL" suggests that a value is required for the column. I.e. when inserting or updating this column, a value must be provided.

The "start date" and "due date" are date columns that will allow “NULL” values.

The "status" and "priority" are the "TINYINT" columns and will not permit “NULL” values.

The "description" column is a "TEXT" column that will allow NULL values.

Inserting data into a table on the “MySQL server”

To add one or more records into the table, the “INSERT” statement is used, as shown in the syntax below:

“INSERT INTO table (c1, c2,...)

VALUES (v1, v2,...);”

In the statement above, the table name must be specified followed by a list of desired columns separated by “commas” written within “parentheses”, after the "INSERT INTO" clause. After which, a list of values separated by “commas” corresponding to the columns within the parentheses must be written, after the "VALUES" function. Make sure there is same quantity of columns and corresponding column values. Furthermore, the column positions must correspond with the position of the column values.

Use the syntax below to add a number of rows to a table, using only one "INSERT" statement:

“INSERT INTO table(c1,c2,...)

VALUES

(v11,v12,...),

(v21,v22,...),

...

(vnn,vn2,...);”

The example below will generate three rows or records in the table "tutorials tbl" and display the confirmation that the “row has been affected”:

“root@host# mysql -u root -p password;

*Enter password:*******

mysql> use TUTORIALS;

Database changed

mysql> INSERT INTO tutorials_tbl

->(tutorial_title, tutorial_author, submission_date)

->VALUES

->(‘Learn PHP’, ‘John Poul’, NOW());

Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO tutorials_tbl

->(tutorial_title, tutorial_author, submission_date)

->VALUES

->(‘Learn MySQL’, ‘Abdul S’, NOW());

Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO tutorials_tbl

->(tutorial_title, tutorial_author, submission_date)

->VALUES

->(‘JAVA Tutorial’, ‘Samer’, '2007-05-06');

Query OK, 1 row affected (0.01 sec)

mysql>”

Here is another example, the query below can be utilized to add a new record to the “tasks” table that was created earlier:

"INSERT INTO

tasks (title, priority)

VALUES

('Learn MySQL INSERT Statement',1);

1 row(s) affected;"

The output should look like the image below:

	task_id	title	start_date	due_date	priority	description
▶	1	Learn MySQL INSERT Statement	NULL	NULL	1	NULL

We indicated the values, in the example above, only for the "title" and "priority" columns. MySQL will use the "default values" for the other non-specified columns.

The "task_id" column has been specified as an auto increment column. This implies that upon addition of a new record to the table, MySQL will generate a subsequent integer.

The “start_date”, “due_date”, and “description” columns have been specified to hold NULL as the default value, so if no values are specified in the "INSERT" statement, MySQL will insert 'NULL' into those columns.

To “Insert a Default Value” into a specific column, either simply disregard the column name and its corresponding values or indicate the column name in the “INSERT INTO” function and type “DEFAULT” in the “VALUES” clause. For example, in the syntax below the “priority” column has been specified with the “DEFAULT” keyword.

“INSERT INTO

tasks (title, priority)

VALUES

('Understanding DEFAULT keyword in INSERT statement', DEFAULT);”

You can insert dates into the table using the “YYYY-MM-DD” format, which represents year, month and date in the mentioned order. For example, you can add the “start date” and “due date” values to the “tasks” table using the command below:

“INSERT INTO tasks (title, start_date, due_date)

VALUES ('Insert date into table','2018-01-09','2018-09-15'):”

The “VALUES” clause also supports expressions as shown in the example below, where the syntax will add a new task utilizing the “current date for start date and due date columns” using the "CURRENT_DATE()" function. Remember the "CURRENT_DATE()" function will always return the current date in the system.

“INSERT INTO tasks (title, start_date, due_date)

VALUES

('Use current date for the task', CURRENT_DATE (), CURRENT_DATE ());”

To insert multiple rows or records in the, take a look at the example below of the “tasks” table, where each record is indicated as values listed in the “VALUES” clause:

“INSERT INTO tasks (title, priority)

VALUES

('My first task', 1),

('It is the second task', 2),

('This is the third task of the week', 3);”

The resulting output should be: *“3 row(s) affected Records: 3 Duplicates: 0 Warnings: 0*”, which implies that the records have been added to the table and there are no “duplicates or warnings” in the table.

Creating a “Temporary table” in the database using “MySQL server”

A "temporary table" in MySQL is a peculiar kind of table that enables storage of a temporary outcome set for reuse over multiple times over the same session.

A "temporary table" is extremely useful when querying information that needs a single "SELECT" statement with the "JOIN" clauses which is not feasible or costly. In such scenario, the temporary table can be used as a storage for the immediate result which can be processed further using a different query.

A "temporary table" in MySQL may have the characteristics listed below:

A "temporary table" is generated using the "CREATE TEMPORARY TABLE" statement. Note that between "CREATE" and "TABLE" functions, the "TEMPORARY" keyword must be added.

When the session is completed or the connection is ended, MySQL server will automatically remove the temporary table from its memory. Another thing to remember is that once you are done with

the temporary table, it can be explicitly deleted utilizing the “DROP TABLE” command.

A "temporary table" will only be available to and accessed by the user that generates it. Different users can potentially generate multiple "temporary tables" with the same name and still not cause any error, since they can only be seen by the user who produced that specific temporary table. Two or more "temporary tables" can not, however, share the same name over the same session.

In a database, a "temporary table" may be given the same name as a standard table. The existing "employee table", for instance, cannot be accessed if you create a "temporary table" called "employees", within the sample database. All queries you make against the "employees table" will now be referred to the "temporary employees table". When you delete the "employees temporary table", the permanent "employees table" would still be in the system and can be accessed again.

While a "temporary table" can be given the same name as a “permanent table” without triggering any error, it is advised to have a different name for the "temporary table". As the same name can lead to confusion and may result in an unexpected loss of information. For example, you can not distinguish between the "temporary table" and the "permanent table", if the connection to the database server is dismissed for some reason and then automatically reconnected to the server. Then, rather than the "temporary table", you could enter a "DROP TABLE" statement to delete the "permanent table" unexpectedly.

Now, a “temporary table” can be created in MySQL with the use of the “TEMPORARY” keyword in the “CREATE TABLE” statement. For instance, the command below will create a “temporary table” storing the top ten clients by revenue with the given table name as “top10customers”:

"CREATE TEMPORARY TABLE top10customers

SELECT p.customerNumber,

c.customerName,

ROUND(SUM(p.amount),2) sales

FROM payments p

INNER JOIN customers c ON c.customerNumber = p.customerNumber

GROUP BY p.customerNumber

ORDER BY sales DESC

LIMIT 10;"

Using this “temporary table”, you can further run queries on this table like you would on a standard or permanent table in the database.

"SELECT

customerNumber,

customerName,

sales

FROM

top10customers

ORDER BY sales;"

“Creating a new table from an existing table in the database using MySQL server”

You may also generate a copy of a table that already exists in the table using the "CREATE TABLE" command, as shown in the syntax below:

"CREATE TABLE new_table_name AS

SELECT column1, column2,...

FROM existing_table_name

WHERE ....;"

The new table will have the same specifications for the columns as the parent table. It is possible to select all columns or a particular

column. If a new table is generated using a table that already exists in the database, then the current values from the parent table will be automatically loaded into the new table. For example, in the syntax below a table called “TestTables” will be created as a replica of the parent “Customers” table:

"CREATE TABLE TestTable AS

SELECT customername, contactname

FROM customers;"

You might encounter a scenario, when you require a precise copy or "clone" of a table and "CREATE TABLE... SELECT" does not meet the requirement, since you would like the the clone to contain the identical indexes, default values, etc as the parent table.

This scenario can be addressed by executing the measures provided below:

Use "SHOW CREATE TABLE" to get a "CREATE TABLE" query specifying the structure, indexes and other feature of the source or parent table.

Adjust the query to alter the table name to be same as the "clone table" and run the query. You would be able to get the precise clone of the table with this method.

Optionally, if you would also like to copy the table contents, you can execute a "INSERT INTO... SELECT" statement.

For example, you can generate a “clone table” for “tutorials_tbl” using the syntax below:

FIRST, copy the entire schema of the table:

"mysql> SHOW CREATE TABLE tutorials_tbl \G;

************************** 1. row ***************************

Table: tutorials_tbl

Create Table: CREATE TABLE `tutorials_tbl` (

`tutorial_id` int (11) NOT NULL auto_increment,

`tutorial_title` varchar (100) NOT NULL default ' ',

`tutorial_author` varchar (40) NOT NULL default ' ',

`submission_date` date default NULL,

PRIMARY KEY (`tutorial_id`),

UNIQUE KEY `AUTHOR_INDEX` (`tutorial_author`)

) TYPE = MyISAM

1 row in set (0.00 sec)

ERROR:

No query specified"

SECOND, rename the parent table to generate a new “clone table”:

"mysql> CREATE TABLE clone_tbl (

-> tutorial_id int (11) NOT NULL auto_increment,

-> tutorial_title varchar (100) NOT NULL default ' ',

-> tutorial_author varchar (40) NOT NULL default ' ',

-> submission_date date default NULL,

-> PRIMARY KEY (tutorial_id),

-> UNIQUE KEY AUTHOR_INDEX (tutorial_author)

->) TYPE = MyISAM;

Query OK, 0 rows affected (1.80 sec)"

LASTLY, if you would like to replicate the data from the source table to the clone table, you can use the command below:

"mysql> INSERT INTO clone_tbl (tutorial_id,

-> tutorial_title,

-> tutorial_author,

-> submission_date)

-> SELECT tutorial_id, tutorial_title,

-> tutorial_author, submission_date

-> FROM tutorials_tbl;

Query OK, 3 rows affected (0.07 sec)

Records: 3 Duplicates: 0 Warnings: 0"

DERVIVED TABLES in MySQL Server

A "derived table" can be defined as "a virtual table returned upon execution of a SELECT statement". A "derived table" seems to be comparable to a "temporary table", however, in the "SELECT" statement it is much easier and quicker to use a "derived table" than a "temporary table" as it would not entail additional steps to create the "temporary table".

There is often interchangeable use of the term "derived table" and "subquery." When the "FROM" clause of a "SELECT" query uses a stand-alone "subquery", it is called as a "derived table". For example, the syntax below can be used to create a "derived table":

"SELECT

column_list

FROM

(SELECT

column_list

FROM

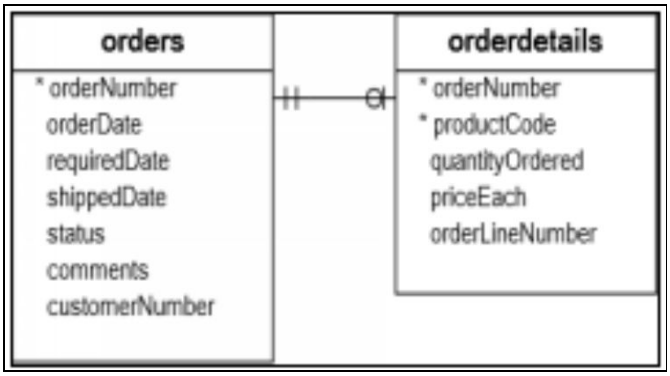
table_1) derived_table_name;

WHERE derived_table_name.c1 > 0;"

A "derived table" is required to have an alias, which can be referenced in future queries, however, there is no such requirement for a subquery. If there is no alias defined for a "derived table", the error message below will be displayed by MySQL:

"Every derived table must have its own alias."

Review the example below of a query that will generate “top five products by sales revenue in 2003” from the “orders” and “orderdetails” tables in the MySQL server “sample database”:

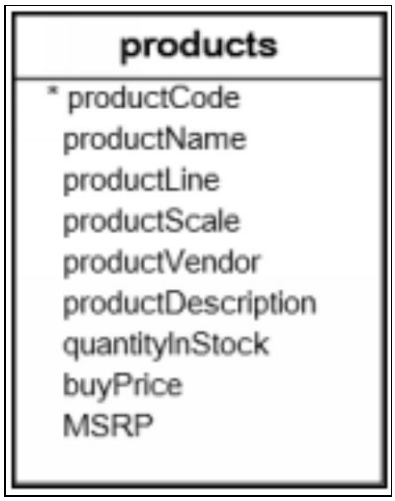


"SELECT
productCode,
*ROUND (SUM (quantityOrdered * priceEach)) sales*
FROM
orderdetails
INNER JOIN
orders USING (orderNumber)
WHERE
YEAR(shippedDate) = 2003
GROUP BY productCode
ORDER BY sales DESC
LIMIT 5;"

The resulting table would be as shown in the picture below:

	productCode	sales
▶	S18_3232	103480
	S10_1949	67985
	S12_1108	59852
	S12_3891	57403
	S12_1099	56462

The result shown in the picture above can be used as a “derived table” and joined with the “products” table using the syntax below:




```
"SELECT
   productName, sales
FROM
   (SELECT
      productCode,
      ROUND (SUM (quantityOrdered * priceEach)) sales
   FROM
      orderdetails
   INNER JOIN orders USING (orderNumber)
   WHERE
```

YEAR(shippedDate) = 2003

GROUP BY productCode

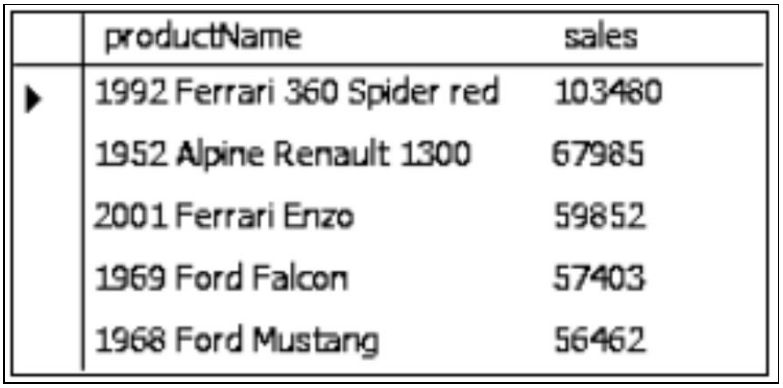
ORDER BY sales DESC

LIMIT 5) top5products2003

INNER JOIN

products USING (productCode);"

The query above will result in the output as shown in the picture below:



	productName	sales
▶	1992 Ferrari 360 Spider red	103480
	1952 Alpine Renault 1300	67985
	2001 Ferrari Enzo	59852
	1969 Ford Falcon	57403
	1968 Ford Mustang	56462

To drive this concept home, review a relatively complicated example of “derived tables” below:

Assume that the clients from the year 2009 have to be classified into three groups: "platinum, gold and silver". Moreover, taking into consideration the criteria below, you would like to calculate the number of clients within each group:

Platinum clients with orders larger than 100K in quantity.

Gold clients with quantity orders ranging from 10K to 100K.

Silver clients with orders less than 10K in quantity.

To generate a query for this analysis, you must first classify each client into appropriate group using “CASE” clause and “GROUP BY” function as shown in the syntax below:

"SELECT

customerNumber,

```
    ROUND (SUM(quantityOrdered * priceEach)) sales,
    (CASE
        WHEN SUM (quantityOrdered * priceEach) < 10000 THEN 'Silver'
        WHEN SUM (quantityOrdered * priceEach) BETWEEN 10000 AND 100000 THEN 'Gold'
        WHEN SUM (quantityOrdered * priceEach) > 100000 THEN 'Platinum'
    END) customerGroup
FROM
    orderdetails
        INNER JOIN
    orders USING (orderNumber)
WHERE
    YEAR(shippedDate) = 2003
GROUP BY customerNumber;"
```

The query output is shown in the picture below:

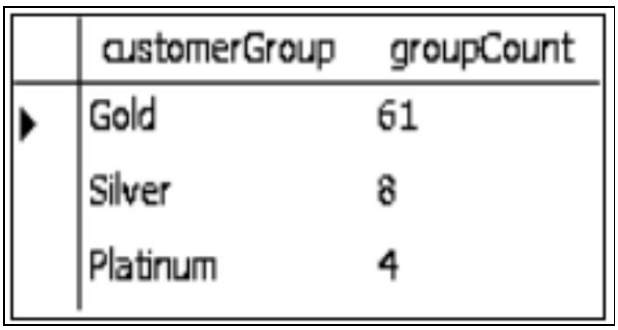
	customerNumber	sales	customerGroup
▶	103	14571	Gold
	112	32642	Gold
	114	53429	Gold
	121	51710	Gold
	124	167783	Platinum
	128	34651	Gold
	129	40462	Gold
	131	22293	Gold
	141	189840	Platinum

Now, you can execute the code below to generate a “derived table” and group the clients as needed:

```
"SELECT
```

```
    customerGroup,
    COUNT(cg.customerGroup) AS groupCount
FROM
    (SELECT
        customerNumber,
            ROUND (SUM (quantityOrdered * priceEach)) sales,
            (CASE
                WHEN SUM (quantityOrdered * priceEach) < 10000 THEN
'Silver'
                    WHEN SUM (quantityOrdered * priceEach) BETWEEN
10000 AND 100000 THEN 'Gold'
                     WHEN SUM (quantityOrdered * priceEach) > 100000
THEN 'Platinum'
            END) customerGroup
    FROM
        orderdetails
    INNER JOIN orders USING (orderNumber)
    WHERE
        YEAR(shippedDate) = 2003
    GROUP BY customerNumber) cg
GROUP BY cg.customerGroup; "
```

The query output is shown in the picture below:



	customerGroup	groupCount
▶	Gold	61
	Silver	8
	Platinum	4

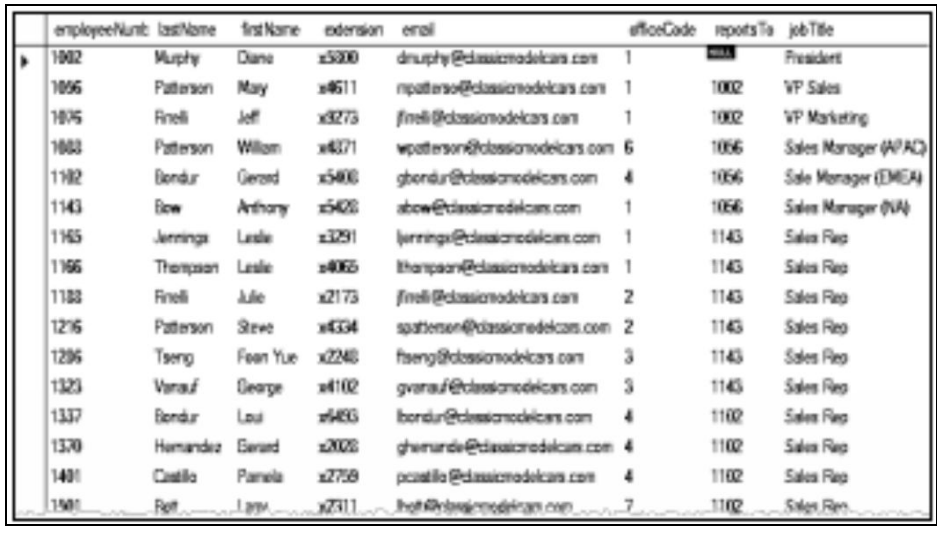
Chapter 3:
SQL Joins and Union commands

The most common command used in SQL is the “SELECT” command, which we have already used in the exercises mentioned earlier in this book. So, now let me give you a deep dive into the “SQL SELECT” command and various clauses that can be used with it.

MySQL SELECT

You can selectively fetch desired data from tables or views, using the "SELECT" statement. As you already know, similar to a spreadsheet, a table comprises of rows and columns. You are highly likely to view a subset of columns, a sub-set of rows, or a combination of the two. The outcome of the “SELECT” query is known as a "result set", which is a list of records comprising the same number of columns per record.

In the picture shown below of the "employees table" from the "MySQL sample database", the table has 8 different columns, namely: "employee number", "last name", "first name", "extension", "email", "office code", "reports to", "job title" and several rows or records.



employeeNumb	lastName	firstName	extension	email	officeCode	reportsTo	jobTitle
1002	Murphy	Diane	x5800	dmurphy@classicmodelcars.com	1	NULL	President
1056	Patterson	Mary	x4611	mpatterso@classicmodelcars.com	1	1002	VP Sales
1076	Firrelli	Jeff	x9273	jfirrelli@classicmodelcars.com	1	1002	VP Marketing
1088	Patterson	William	x4871	wpatterson@classicmodelcars.com	6	1056	Sales Manager (APAC)
1102	Bondur	Gerard	x5408	gbondur@classicmodelcars.com	4	1056	Sale Manager (EMEA)
1143	Bow	Anthony	x5428	abow@classicmodelcars.com	1	1056	Sales Manager (NA)
1165	Jennings	Leslie	x3291	ljennings@classicmodelcars.com	1	1143	Sales Rep
1166	Thompson	Leslie	x4065	lthompson@classicmodelcars.com	1	1143	Sales Rep
1188	Firrelli	Julie	x2173	jfirrelli@classicmodelcars.com	2	1143	Sales Rep
1216	Patterson	Steve	x4334	spatterson@classicmodelcars.com	2	1143	Sales Rep
1286	Tseng	Foon Yue	x2248	ftseng@classicmodelcars.com	3	1143	Sales Rep
1323	Vanauf	George	x4102	gvanauf@classicmodelcars.com	3	1143	Sales Rep
1337	Bondur	Loui	x6493	lbondur@classicmodelcars.com	4	1102	Sales Rep
1370	Hernandez	Gerard	x2028	ghernande@classicmodelcars.com	4	1102	Sales Rep
1401	Castillo	Pamela	x2759	pcastillo@classicmodelcars.com	4	1102	Sales Rep
1501	Bott	Larry	x2311	lbott@classicmodelcars.com	7	1102	Sales Rep

The "SELECT" query dictates the columns and records that can be fetched from the table. For instance, if you would like to display just the "first name", "last name", and "job title" of all the employees or you selectively desire to see the information pertaining to employees with "job title" as "sales rep", then you will be able to utilize the "SELECT" query to achieve this.

Here is a standard syntax for “SELECT” statement:

"SELECT

column_1, column_2, ...

FROM

table_1

[INNER | LEFT |RIGHT] JOIN table_2 ON conditions

WHERE

conditions

GROUP BY column_1

HAVING group_conditions

ORDER BY column_1

LIMIT offset, length;"

The "SELECT" query above contains various provisions, as described below:

The "SELECT" clause followed by a list of columns isolated by “comma” or an “asterisk (*)” suggests that all columns are to be returned.

The "FROM" clause defines the table or view from which the data should be queried.

The "JOIN" clause receives associated data from other tables based on the specified join criteria.

In the "result set", the "WHERE" clause is utilized for selectively filtering the rows or records.

The "GROUP BY" clause is utilized to selectively group a set of records and apply “aggregate functions” to each group.

The "HAVING" clause is used to filter a group based on the "GROUP BY" clause specified groups.

The "ORDER BY" clause can be utilized for specifying a list of columns to be sorted.

The "LIMIT" clause can be utilized for restricting the number of rows displayed upon execution of the command.

Note that only “SELECT” and “FROM” clauses are necessary to execute the command and all other clauses can be used on ad-hoc basis.

For instance, if you are interested in displaying only the "first name", "last name", and "job title" of all employees, you can utilize the syntax below:

"SELECT

lastname, firstname, jobtitle

FROM

*employees;"*But if you would like to view all the columns in the “employees” table, use the syntax below:

*"SELECT * FROM employees;"*

In practice, it is recommended to list the columns you would like to view instead of using the asterisk (*) command, as the asterisk (*) will return all column data, some of which you may not be allowed to use. It will create “non-essential I/O disk and network traffic between the MySQL database server and the application”. If the columns are defined explicitly, then the "result set" becomes simpler to predict and handle. Suppose you are using the asterisk (*) and some other user modified the table and generated additional columns, you would receive a "result set" containing different columns than needed. Moreover, the use of asterisk (*) can potentially reveal sensitive data to unauthorized users.

MySQL SELECT DISTINCT

You can receive duplicate rows when searching for information from a table. You could utilizee the "DISTINCT" clause in the "SELECT" query to get rid of these redundant rows.

The "DISTINCT" clause syntax is given below:

"SELECT DISTINCT

columns

FROM

table_name

WHERE

where_conditions;"

EXAMPLE

The syntax below presents an example for the "DISTINCT" clause, used to selectively view “unique last names” of the employees from the "employees" table.

First, you need to display the “last names” of all employees from the “employees” tables using the syntax below:

"SELECT

last name

FROM

employees

ORDER BY last name;"

Now to get rid of the repeated last names, use the syntax below:

"SELECT DISTINCT

last name

FROM

employees

ORDER BY last name;"

Please note, if a column has been indicated to hold “NULL” values and the “DISTINCT” clause is used. Only one “NULL” value will be retained by the sever since the “DISTINCT” clause will consider all “NULL” values as identical.

Using “DISTINCT” clause on multiple columns

To accomplish this “MySQL server” will use a combination of all values in selected columns to identify unique records in the "result set". For example, you can view singular combination of “city” and “state” columns from the “customers” table using the syntax below:

"SELECT DISTINCT

state, city

FROM

customers

WHERE

state IS NOT NULL

ORDER BY state, city;"

MySQL “ORDER BY”

When using the "SELECT" statement to query the data, the "result set" will not be organized in a particular format. That is where the "ORDER BY" clause can be utilized to organize the "result set" as desired. The “ORDER BY” clause will allow sorting of the “result set” on the basis of one or more columns as well as sorting a number of columns in ascending or descending order.

Here is the "ORDER BY" clause syntax:

"SELECT column1, column2,...

FROM tbl

ORDER BY column1 [ASC|DESC], column2 [ASC|DESC],...;"

As you would expect the "ASC" means ascending and the "DESC" means descending. By default, if "ASC" or "DESC" has not been

specified explicitly, the "ORDER BY" clause will sort the "result set" in ascending order.

EXAMPLE

The syntax below presents an example for the "ORDER BY" clause, used to selectively view contacts from the "customers" table and sort them in ascending order of the “last name”.

"SELECT

contact Last name,

contact First name

FROM

customers

ORDER BY

contact Last name;"

The ascending “result set” is shown in the picture below:

	contactLastname	contactFirstname
▶	Accorti	Paolo
	Altagar,G M	Raanan
	Andersen	Mel
	Anton	Carmen
	Ashworth	Rachel
	Barajas	Miguel
	Benitez	Violeta
	Bennett	Helen
	Berglund	Christina

Now, if you wanted to view the last name in descending order, you will use the syntax below:

"SELECT

contact Last name,

contact First name

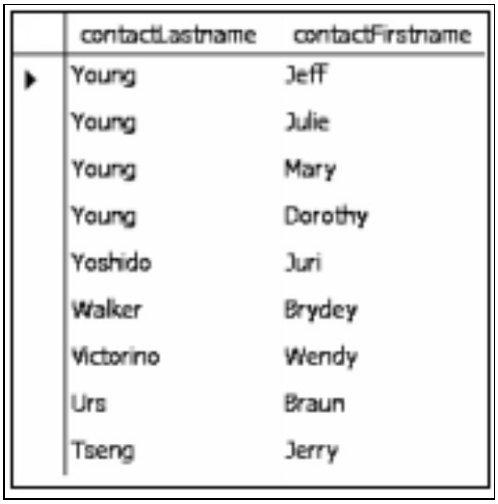
FROM

customers

ORDER BY

contact Last name DESC;"

The descending “result set” is shown in the picture below:



	contactLastname	contactFirstname
▶	Young	Jeff
	Young	Julie
	Young	Mary
	Young	Dorothy
	Yoshido	Juri
	Walker	Brydey
	Victorino	Wendy
	Urs	Braun
	Tseng	Jerry

Or, if you would like to arrange the “last name” in descending order and the “first name” in the ascending order, you can utilize the “DESC” and “ASC” clause in the same query but with the relevant column as shown in the syntax below:

"SELECT

contact Last name,

contact First name

FROM

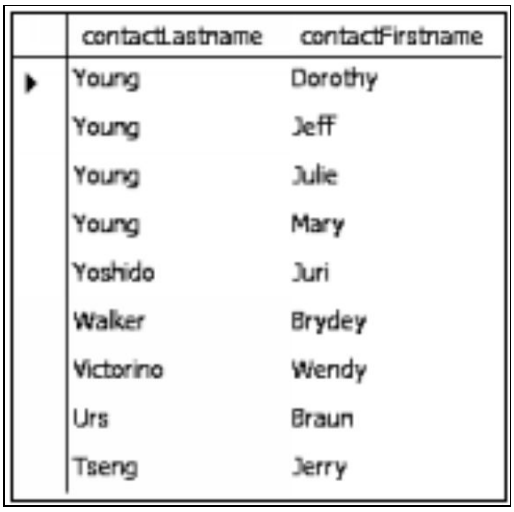
customers

ORDER BY

contact Last name DESC,

contact First name ASC;"

The “result set” is shown in the picture below, where the “ORDER BY” clause will first sort the “last name” in descending order and then subsequently sort the “first name” in the ascending order:



	contactLastname	contactFirstname
▶	Young	Dorothy
	Young	Jeff
	Young	Julie
	Young	Mary
	Yoshido	Juri
	Walker	Brydey
	Victorino	Wendy
	Urs	Braun
	Tseng	Jerry

MySQL WHERE

The "WHERE" clause enables the search criteria to be specified for the records displayed after running the query.

The "WHERE" clause syntax is given below:

"SELECT

select_list

FROM

table_name

WHERE

search_condition;"

The “search_condition” is a composite of one or more "predicates" utilizing the “logical operators” like "AND", "OR" and "NOT", as shown in the table below. A "predicate" in SQL, is defined as “an expression that assesses true, false, or unknown”.

Operator	Description
=	“Equal to. You can use it with almost any data types.”
<> or !=	“Not equal to”

<	“Less than. You typically use it with numeric and date/time data types.”
>	“Greater than”
<=	“Less than or equal to”
>=	“Greater than or equal to”

The final "result set" includes any record from the "table_name" that makes the "search_condition" to be evaluated as valid.

In addition to the "SELECT" statement, you may indicate the rows that need to be updated and deleted, utilizing the "WHERE" clause in the "UPDATE" and "DELETE" query.

EXAMPLE

The syntax below presents an example for the "WHERE" clause, used to selectively view employees with job title as “Sales Rep” from the “employees” table in the “MySQL sample database”:

"SELECT

lastname,

firstname,

jobtitle

FROM

employees

WHERE

jobtitle = 'Sales Rep';"

Although the "WHERE" clause is defined at the end of the query, “MySQL” first selects the corresponding rows after evaluating the phrase in the “WHERE” clause. It selects the rows with a job title as "Sales Rep" and then chooses the columns in the "SELECT" clause

from the “select list”. The highlighted rows in the picture below, include the final result set columns and rows.

employeeNumber	lastName	firstName	extension	email	officeCode	reportsTo	jobTitle
1002	Murphy	Diane	x5800	dmurphy@classicmodelcars.com	1		President
1056	Patterson	Mary	x4611	mpatterso@classicmodelcars.com	1	1002	VP Sales
1076	Firrelli	Jeff	x9273	jfirrelli@classicmodelcars.com	1	1002	VP Marketing
1088	Patterson	William	x4871	wpatterson@classicmodelcars.com	6	1056	Sales Manager (APAC)
1102	Bondur	Gerard	x5408	gbondur@classicmodelcars.com	4	1056	Sale Manager (EMEA)
1143	Bow	Anthony	x5428	abow@classicmodelcars.com	1	1056	Sales Manager (NA)
1165	Jennings	Leslie	x3291	ljennings@classicmodelcars.com	1	1143	Sales Rep
1166	Thompson	Leslie	x4065	lthompson@classicmodelcars.com	1	1143	Sales Rep
1188	Firrelli	Julie	x2173	jfirrelli@classicmodelcars.com	2	1143	Sales Rep
1216	Patterson	Steve	x4334	spatterson@classicmodelcars.com	2	1143	Sales Rep
1286	Tseng	Foon Yue	x2248	ftseng@classicmodelcars.com	3	1143	Sales Rep
1323	Vanauf	George	x4102	gvanauf@classicmodelcars.com	3	1143	Sales Rep
1337	Bondur	Loui	x6493	lbondur@classicmodelcars.com	4	1102	Sales Rep

MySQL JOIN Statements

The MySQL “JOIN" function is a way to link information between one or more tables on the basis of common attributes or column values between the selected tables. A relational database comprises of various tables that have been linked by a shared or common column, known as "foreign key". As a result, information in each individual table can be deemed incomplete from the business perspective. For instance, there are two separate tables in the "MySQL sample database" called "orders" and "orderdetails" that have been connected to each other using a column called "orderNumber". You will have to search for information in both the "orders" and the "orderdetails" table to obtain complete order information.

There are 6 different SQL “JOINS” functions, namely, “INNER JOIN”, “LEFT JOIN”, “RIGHT JOIN”, “CROSS JOIN” and “SELF JOIN”.

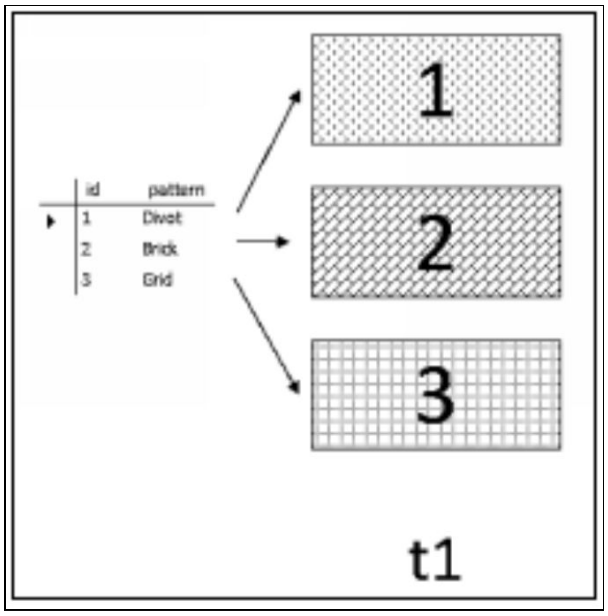
We will be using tables named "t1" and "t2" as described in the syntax below, in order to facilitate your understanding of each type of "JOIN". The "pattern" column in both "t1" and "t2" tables serves as the common column between them.

```
"CREATE TABLE t1 (
    id INT PRIMARY KEY,
    pattern VARCHAR (50) NOT NULL
);
CREATE TABLE t2 (
    id VARCHAR (50) PRIMARY KEY,
    pattern VARCHAR (50) NOT NULL
);"
```

We can insert some data into the two tables using “INSERT” function as shown in the syntax below:

```
"INSERT INTO t1(id, pattern)
VALUES(1,'Divot'),
      (2,'Brick'),
      (3,'Grid');
INSERT INTO t2(id, pattern)
VALUES('A','Brick'),
      ('B','Grid'),
      ('C','Diamond');"
```

The result of the syntax above is shown in the picture below:



Now, let us look at every type of “JOIN” clause in detail!

“INNER JOIN”

The “MySQL INNER JOIN” is used to align “rows in one table with rows in other tables”, so that rows containing columns from the two tables can be queried.

The "SELECT" statement is not required to contain the "INNER JOIN" clause for execution, which appears right next to the "FROM" clause.

You must indicate the following requirements before using the "INNER JOIN" clause:

Start with the primary table to be included in the "FROM" clause.

Then, choose the table you would like to connect to the primary table included in the "INNER JOIN" clause, in step 1. Remember theoretically you can join one table to multiple tables at the same time. However, it is recommended to restrict the number of tables to be joined, to achieve higher performance and efficiency.

Lastly, the "join condition" or "join predicate" must be defined. Following the initial "ON" keyword for the "INNER JOIN", the “join condition” can be indicated. The join condition is the rule that

dictates the matching of a row in the “primary table” with the records from the another table.

The "INNER JOIN" clause syntax is given below:

"SELECT column_list

FROM t1

INNER JOIN t2 ON join_condition1

INNER JOIN t3 ON join_condition2

...

WHERE where_conditions;"

Now, for further simplification of the query above, just assume that you are interested in linking only tables “t1” and “t2” using the syntax below:

"SELECT column_list

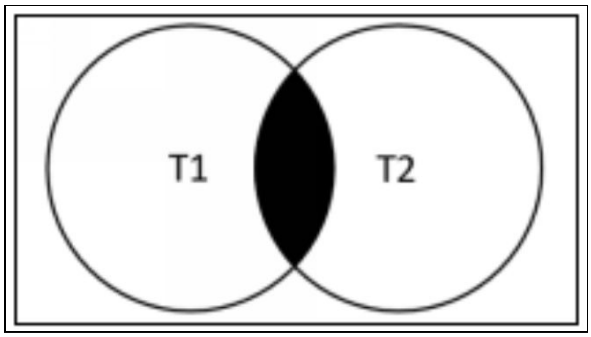
FROM t1

INNER JOIN t2 ON join_condition;"

The "INNER JOIN" clause will compare each row of the "t1" table with every single record of the "t2" table to determine whether both of them fulfill the defined “join condition”. Once the “join condition” has been satisfied, the "INNER JOIN" then returns a new record, consisting of columns from both "t1" and "t2".

Note that the records in both "t1" and "t2" need to be merged according to the “join condition”. If no two records from the two tables are identified meeting the “join condition”, an empty "result set" is returned by the query. The same logic applies when more than two tables are joined.

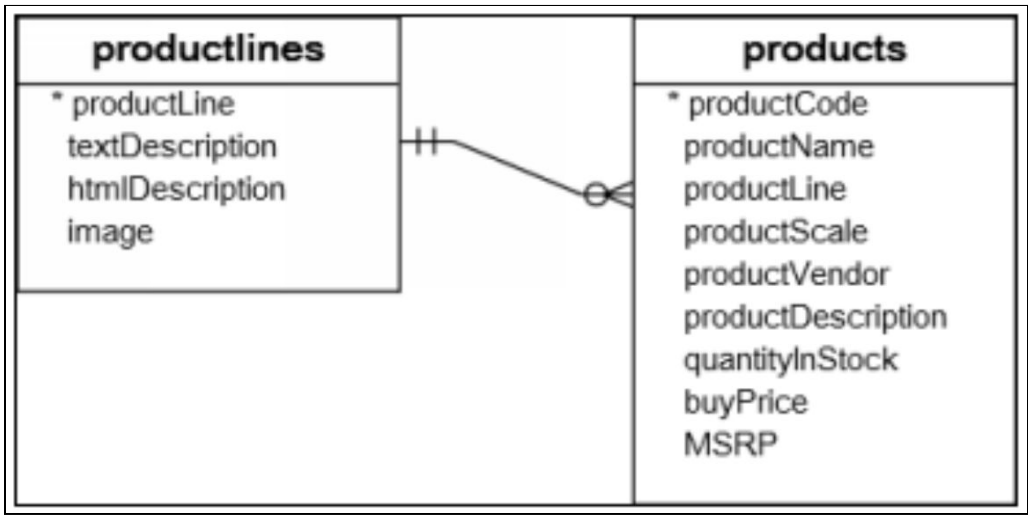
The Venn diagram in the picture below shows the working of the "INNER JOIN" clause. The records in the "result set" are required to be present in both "t1" and "t2" as represented by the section overlapping the 2 circles, depicting each table respectively.



Example

Consider the “products” and “productlines” table in the “MySQL sample database”, shown in the picture below. The “products” table is connected to the “productlines” table by referencing the “productline” column. Therefore, the “productline” column serves as “foreign key” in the “products” table.

Conventionally, tables that have “foreign key” relationships are queried using the “JOIN” clause.



Let’s assume that you would like to view the “productCode” and “productName” columns from the “products” table as well as “textDescription” of the product lined from the “productlines” table. You can accomplish this by using the syntax below and matching the rows of both the tables on the basis of “productline” columns in the two tables.

```
"SELECT
    productCode,
```

productName,

textDescription

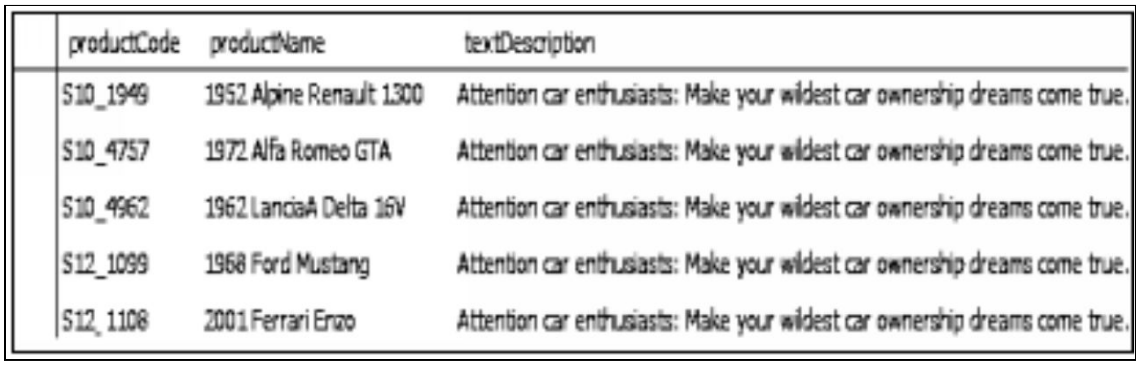
FROM

products t1

INNER JOIN

productlines t2 ON t1.productline = t2.productline;"

The “result set” is shown in the picture below:



productCode	productName	textDescription
S10_1949	1952 Alpine Renault 1300	Attention car enthusiasts: Make your wildest car ownership dreams come true.
S10_4757	1972 Alfa Romeo GTA	Attention car enthusiasts: Make your wildest car ownership dreams come true.
S10_4962	1962 LanciaA Delta 16V	Attention car enthusiasts: Make your wildest car ownership dreams come true.
S12_1099	1968 Ford Mustang	Attention car enthusiasts: Make your wildest car ownership dreams come true.
S12_1108	2001 Ferrari Enzo	Attention car enthusiasts: Make your wildest car ownership dreams come true.

Here’s a tip!

Since both the tables contain identical column called “productline”, you could use the simpler query below (without the need of using the table aliases) and obtain the same “result set” as shown in the picture above:

"SELECT

productCode,

productName,

textDescription

FROM

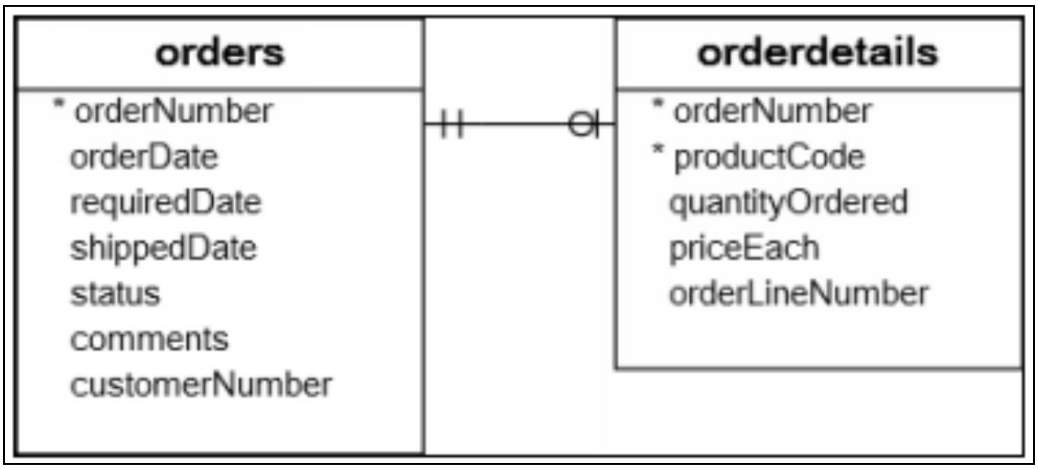
products

INNER JOIN

productlines USING (productline);"

“INNER JOIN” with “GROUP BY”

The picture below contains the “orders” and “orderdetails” tables:



You could utilize "INNER JOIN" with "GROUP BY" clause to obtain desired information from the "orders" and "orderdetails" tables as shown in the two syntax below:

```
"SELECT
    T1.orderNumber,
    status,
    SUM(quantityOrdered * priceEach) total
FROM
    orders AS T1
        INNER JOIN
    orderdetails AS T2 ON T1.orderNumber = T2.orderNumber
GROUP BY orderNumber;"
"SELECT
    orderNumber,
    status,
    SUM(quantityOrdered * priceEach) total
FROM
    orders
        INNER JOIN
    orderdetails USING (orderNumber)
```

GROUP BY orderNumber;"

The “result set” is shown in the picture below:

	orderNumber	status	total
	10100	Shipped	10223.83
	10101	Shipped	10549.01
	10102	Shipped	5494.78
	10103	Shipped	50218.95
	10104	Shipped	40206.20

MySQL “INNER JOIN” with different operators

We have only explored the join condition using the "equal operator (=)" to match the rows. But we can also use other operators with the join predicate including "greater than (>)", 'less than (<)", and "not-equal (< >)" operators.

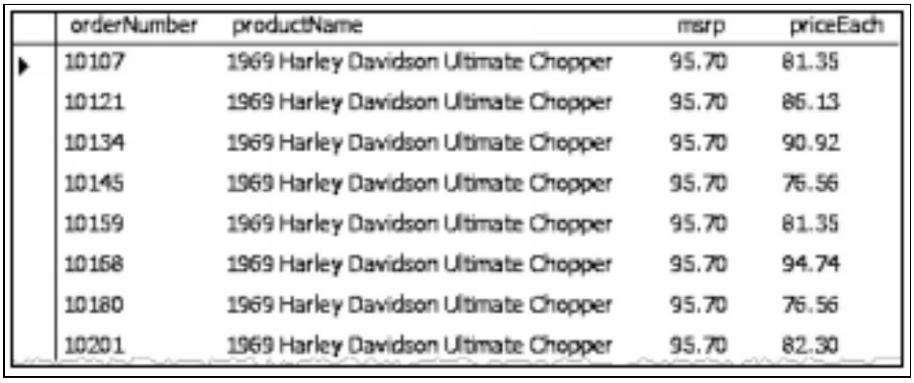
The syntax below utilizes a "less than (<)" operator to identify the sales price for the item with the code "S10 1678", which are less than the “manufacturer's suggested retail price (MSRP)”.

```
"SELECT
    orderNumber,
    productName,
    msrp,
    priceEach
FROM
    products p
        INNER JOIN
    orderdetails o ON p.productcode = o.productcode
        AND p.msrp > o.priceEach
```

WHERE

p.productcode = 'S10_1678';"

The “result set” is shown in the picture below:



orderNumber	productName	msrp	priceEach
10107	1969 Harley Davidson Ultimate Chopper	95.70	81.35
10121	1969 Harley Davidson Ultimate Chopper	95.70	86.13
10134	1969 Harley Davidson Ultimate Chopper	95.70	90.92
10145	1969 Harley Davidson Ultimate Chopper	95.70	76.56
10159	1969 Harley Davidson Ultimate Chopper	95.70	81.35
10168	1969 Harley Davidson Ultimate Chopper	95.70	94.74
10180	1969 Harley Davidson Ultimate Chopper	95.70	76.56
10201	1969 Harley Davidson Ultimate Chopper	95.70	82.30

“LEFT JOIN”

The “MySQL LEFT JOIN” enables querying of data from 2 or multiple tables on a database. The "SELECT" statement is not required to contain the "LEFT JOIN" clause for execution, which appears right next to the "FROM" clause. The principles of "left table" and "right table" are implemented when the 2 tables are linked, using the "LEFT JOIN" clause.

Unlike an "INNER JOIN", the "LEFT JOIN" or “LEFT OUTER JOIN” will return all records in the left table, the records meeting the “join condition” and even the records that don’t. “NULL" value appears in the "result set" of the “columns from the right table for rows that do not match the join condition”.

Now, to further simplify this query we will assume that we are interested in linking only tables “t1” and “t2” with the "LEFT JOIN" per the syntax below:

"SELECT

t1.c1, t1.c2, t2.c1, t2.c2

FROM

t1

LEFT JOIN

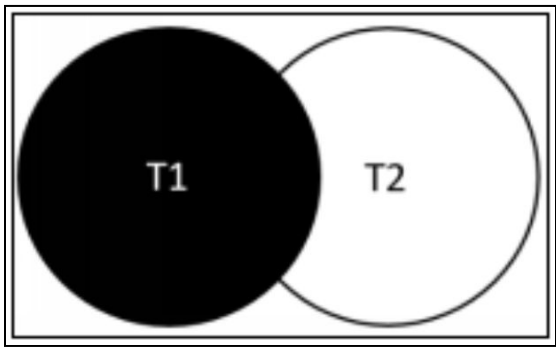
t2 ON t1.c1 = t2.c1;"

In the syntax above, you are using the "LEFT JOIN" clause to connect the "t1" to the "t2", on the basis of a record from the "left table t1" aligning a record from the "right table t2" as defined by the “join predicate (t1.c1= t2.c1)", which are then included in the "result set".

If the record in the “left table” has no matching record that aligns with it in the “right table”, then the record in the “left table” will still be selected and joined with a "virtual record” containing "NULL" values from the “right table”.

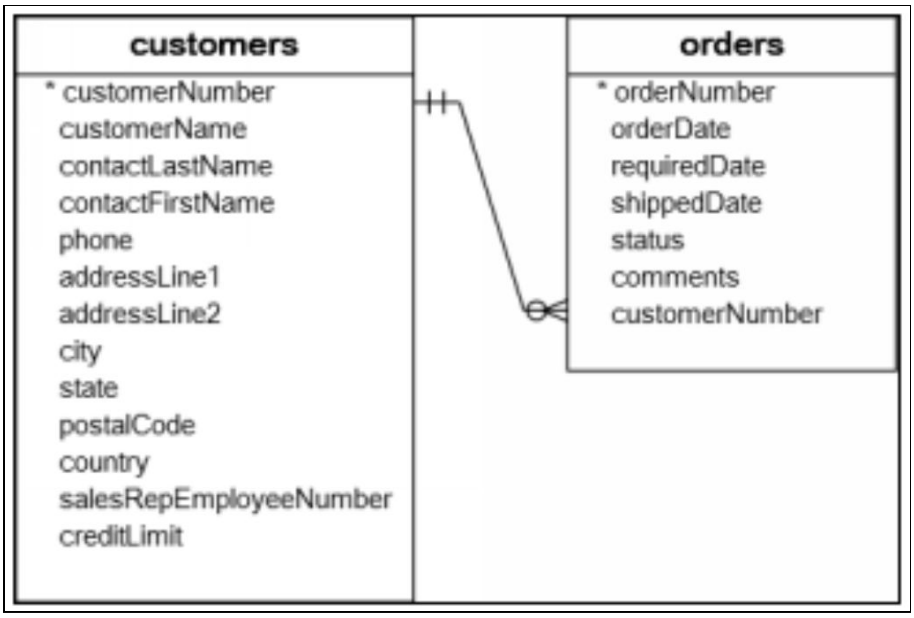
To put it simply, the "LEFT JOIN" clause enables selection of matching records from both the “left and right tables”, as well as all records from the "left table (t1)", even without aligning records from the "right table (t2)".

The “Venn diagram” shown in the picture below will help you understand how the working of the "LEFT JOIN". The overlapping section of the two circles includes records that matched from the two tables, and the rest of the portion of the “left circle” includes records in the "t1" table with “no corresponding record in the t2 table”. Therefore, the "result set" will include all records in the “left table”.



Example

Consider the “customers” and “orders” table in the “MySQL sample database”, shown in the picture below, wherein every “order value” in the “orders” table corresponds to a “customer value” in the “customers” table but every customer value in the “customers” table may not have a corresponding order value in the “orders” table.



All the “orders corresponding to a customer” can be queried using the “LEFT JOIN” clause using the syntax below:

"SELECT

c.customerNumber,

c.customerName,

orderNumber,

o.status

FROM

customers c

LEFT JOIN orders o ON c.customerNumber = o.customerNumber;"

The “result set” is shown in the picture below, where the left table refers to the "customers" table, so all the rows with value in the "customers" table will be added to the "result set"; although it contains rows with customer data that have no order data such as "168" and "169". For such rows, the order data value is reflected as "NULL" implying that there is no order in the "orders" table for those customers.

customerNumber	customerName	orderNumber	status
166	Handji Gifts& Co	10288	Shipped
166	Handji Gifts& Co	10409	Shipped
167	Herkku Gifts	10181	Shipped
167	Herkku Gifts	10188	Shipped
167	Herkku Gifts	10289	Shipped
168	American Souvenirs Inc	NULL	NULL
169	Porto Imports Co.	NULL	NULL
171	Daedalus Designs Imports	10180	Shipped
171	Daedalus Designs Imports	10224	Shipped
172	La Corne D'abondance, ...	10114	Shipped

Here's a tip!

Since both the tables share the column name "customersNumber", you could utilize the simpler query below (without the need of using the table aliases) and obtain the same "result set" as shown in the picture above:

"SELECT

c.customerNumber,

customerName,

orderNumber,

status

FROM

customers c

LEFT JOIN orders USING (customerNumber);"

MySQL "LEFT JOIN" with "WHERE" clause

Consider the syntax below with "WHERE" clause is used after the "LEFT JOIN" to retrieve desired information from the "orders" and "orderDetails" tables.

"SELECT

o.orderNumber,

customerNumber,

productCode

FROM

orders o

LEFT JOIN

orderDetails USING (orderNumber)

WHERE

orderNumber = 10123;"

The “result set” is shown in the picture below for all the order number listed as “10123”:

	orderNumber	customerNumber	productCode
▶	10123	103	S18_1589
	10123	103	S18_2870
	10123	103	S18_3685
	10123	103	S24_1628

Note that if the “join condition” was changed from “WHERE” clause to the “ON” clause as shown in the syntax below, then the query will produce a different “result set” containing all orders but only the details associated with the order number “10123” will be displayed, as shown in the picture below:

"SELECT

o.orderNumber,

customerNumber,

productCode

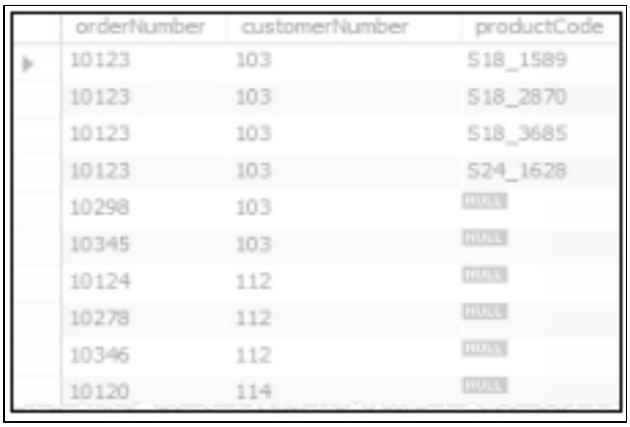
FROM

orders o

LEFT JOIN

orderDetails d ON o.orderNumber = d.orderNumber

AND o.orderNumber = 10123;"



orderNumber	customerNumber	productCode
10123	103	S18_1589
10123	103	S18_2870
10123	103	S18_3685
10123	103	S24_1628
10298	103	NULL
10345	103	NULL
10124	112	NULL
10278	112	NULL
10346	112	NULL
10120	114	NULL

“RIGHT JOIN”

The “RIGHT JOIN" or “RIGHT OUTER JOING” is similar to the "LEFT JOIN" with the only difference being the treatment of the tables, which has been reversed from left to right. Each record from the "right table (t2)" appears in the "result set" with a "RIGHT JOIN". The “NULL" value will be displayed for columns in the "left table (t1)" for the records in the “right table without any corresponding rows in the left table (t1)".

You could utilize the "RIGHT JOIN" clause to link tables “t1” and “t2” as illustrated in the syntax below:

"SELECT

*

FROM t1

RIGHT JOIN t2 ON join_predicate;"

In the syntax above, the right table is “t2” and the left table is “t1” and “join_predicate” indicated the matching criteria with which records from “t1” will be aligned with records from “t2”.

With the execution of the query above, all the rows from the “right table t2” will be displayed in the “result set” and on the basis of the join condition, if no match are found for “t2” with the rows of “t1” then “NULL” value will be added to those columns from “t1” table.

Example

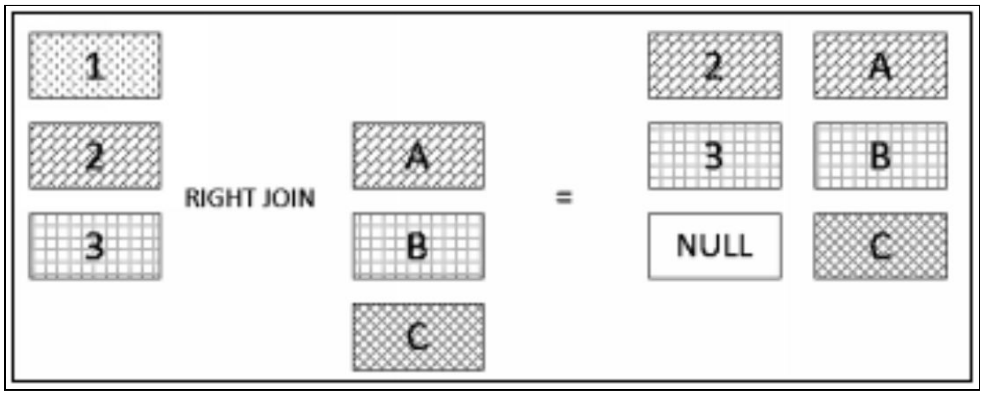
Let’s consider the tables “t1” and “t2” as described in the syntax below:

```
"CREATE TABLE t1 (
    id INT PRIMARY KEY,
    pattern VARCHAR(50) NOT NULL
);
CREATE TABLE t2 (
    id VARCHAR(50) PRIMARY KEY,
    pattern VARCHAR(50) NOT NULL
);
INSERT INTO t1(id, pattern)
VALUES(1,'Divot'),
      (2,'Brick'),
      (3,'Grid');
INSERT INTO t2(id, pattern)
VALUES('A','Brick'),
      ('B','Grid'),
      ('C','Diamond');"
```

The tables “t1” and “t2” can be joined with the “pattern” columns, as illustrated in the query below:

```
"SELECT
    t1.id, t2.id
FROM
    t1
        RIGHT JOIN t2 USING (pattern)
ORDER BY t2.id;"
```

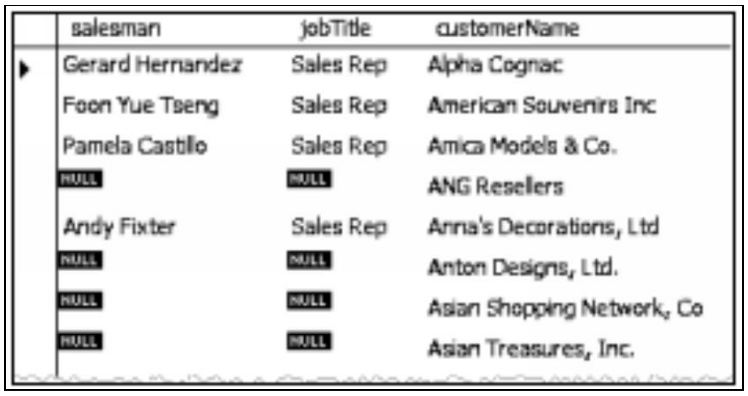
The “result set” is shown in the illustrated below:



Or look at the syntax below to view the “sales rep” and their “customers” from the “employees” and “customers” table in the “MySQL sample database”:

```
"SELECT
    concat(e.firstName,' ', e.lastName) salesman,
    e.jobTitle,
    customerName
FROM
    employees e
        RIGHT JOIN
                customers c ON e.employeeNumber = c.salesRepEmployeeNumber
        AND e.jobTitle = 'Sales Rep'
ORDER BY customerName;"
```

The “result set” is shown in the picture below:



salesman	jobTitle	customerName
Gerard Hernandez	Sales Rep	Alpha Cognac
Foon Yue Tseng	Sales Rep	American Souvenirs Inc
Pamela Castillo	Sales Rep	Amica Models & Co.
NULL	NULL	ANG Resellers
Andy Fixter	Sales Rep	Anna's Decorations, Ltd
NULL	NULL	Anton Designs, Ltd.
NULL	NULL	Asian Shopping Network, Co
NULL	NULL	Asian Treasures, Inc.

“CROSS JOIN”

The “CROSS JOIN” clause is used to obtain a “Cartesian product”, which is defined as a “join of every row of one table to every row of another table”. When the “CROSS JOIN” clause is utilized, the "result set" contains all rows from the two tables, which are a composite of the record from the “first table” and record from the second table. This case exists only when a “common link or column between the two tables” that are joined cannot be found.

The biggest drawback is sheer volume of data, assume each table contains 1,000 rows, then the "result set" will contain 1,000x 1,000= 1,000,000 rows.

You could utilize the "CROSS JOIN" to join “t1” and “t2” as displayed in the syntax below:

```
"SELECT
    *
FROM
    T1
        CROSS JOIN
    T2;"
```

Unlike the “RIGHT JOIN” and “LEFT JOIN”, the “CROSS JOIN” can be operated without using a “join predicate”. Moreover, if the “WHERE” clause is added to the “CROSS JOIN” and the two tables “t1” and “t2” have a connection then the “CROSS JOIN” will operate as “INNER JOIN”, as displayed in the syntax below:

```
"SELECT
    *
FROM
    T1
        CROSS JOIN
    T2
WHERE
    T1.id = T2.id;"
```

Example

Let's use "testdb" database as described in the syntax below to better understand how the "CROSS JOIN" operates:

```
"CREATE DATABASE IF NOT EXISTS testdb;
USE testdb;
CREATE TABLE products (
    id INT PRIMARY KEY AUTO_INCREMENT,
    product_name VARCHAR (100),
    price DECIMAL (13 , 2 )
);
CREATE TABLE stores (
    id INT PRIMARY KEY AUTO_INCREMENT,
    store_name VARCHAR (100)
);
CREATE TABLE sales (
    product_id INT,
    store_id INT,
    quantity DECIMAL (13 , 2 ) NOT NULL,
    sales_date DATE NOT NULL,
```

```
    PRIMARY KEY (product_id , store_id),
    FOREIGN KEY (product_id)
        REFERENCES products (id)
        ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (store_id)
        REFERENCES stores (id)
        ON DELETE CASCADE ON UPDATE CASCADE
);"
```

In the syntax above, we have 3 tables:

The “products” table, which holds the fundamental data of the products including “product id”, “product name”, and “sales price”.

The “stores” table, which holds the data pertaining to the stores where the products are available for purchase.

The “sales” table, which holds the data pertaining to the products sold in specific stores by date and quantity.

Let’s assume there are 3 items “iPhone”, “iPad” and “MacBook Pro” available for sell in 2 stores named “North” and “South”. We can populate the tables to hold this data using the syntax below:

```
"INSERT INTO products(product_name, price)
VALUES('iPhone', 699),
    ('iPad',599),
    ('Macbook Pro',1299);
INSERT INTO stores(store_name)
VALUES('North'),
    ('South');
INSERT INTO sales(store_id,product_id,quantity,sales_date)
VALUES(1,1,20,'2017-01-02'),
    (1,2,15,'2017-01-05'),
```

(1,3,25,'2017-01-05'),

(2,1,30,'2017-01-02'),

(2,2,35,'2017-01-05');"

Now, to view the “total sales for every store and for every product”, the sales must be calculated first and then “grouped” by the store and product using the syntax below:

```
"SELECT
   store_name,
   product_name,
   SUM (quantity * price) AS revenue
FROM
   sales
      INNER JOIN
   products ON products.id = sales.product_id
      INNER JOIN
   stores ON stores.id = sales.store_id
GROUP BY store_name , product_name;"
```

The “result set” is shown in the picture below:

	store_name	product_name	revenue
▶	North	iPad	8985.0000
	North	iPhone	13980.0000
	North	Macbook Pro	32475.0000
	South	iPad	20965.0000
	South	iPhone	20970.0000

Now, if you wanted to view the stores that had zero sale for a particular product, the syntax above will not provide you that information. This is where you can utilize the “CROSS JOIN” to first

view a composite of all products and stores, as displayed in the query below:

"SELECT

store_name, product_name

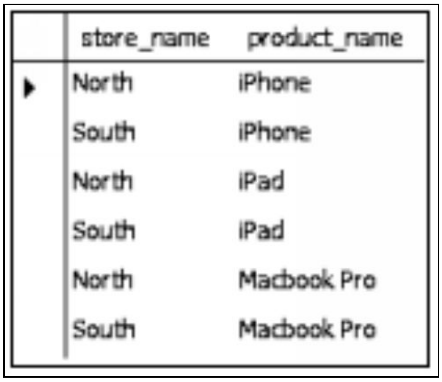
FROM

stores AS a

CROSS JOIN

products AS b;"

The “result set” is shown in the picture below:



store_name	product_name
North	iPhone
South	iPhone
North	iPad
South	iPad
North	Macbook Pro
South	Macbook Pro

And then, you can join the query result received earlier with this query that will result in the “total of the sales by store and product”, using the syntax below:

"SELECT

b.store_name,

a.product_name,

IFNULL (c.revenue, 0) AS revenue

FROM

products AS a

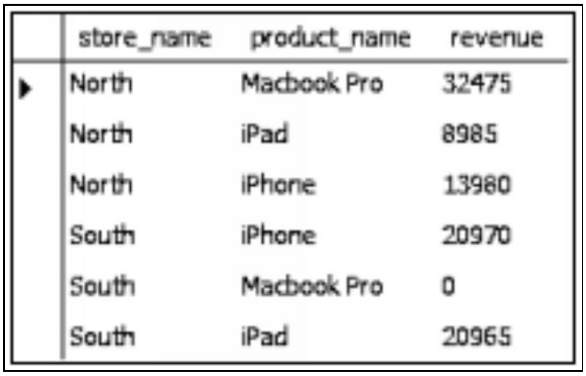
CROSS JOIN

stores AS b

LEFT JOIN

```
(SELECT
    stores.id AS store_id,
    products.id AS product_id,
    store_name,
        product_name,
        ROUND (SUM(quantity * price), 0) AS revenue
FROM
    sales
INNER JOIN products ON products.id = sales.product_id
INNER JOIN stores ON stores.id = sales.store_id
 GROUP BY store_name , product_name) AS c ON c.store_id =
b.id
    AND c.product_id= a.id
ORDER BY b.store_name;"
```

The “result set” is shown in the picture below:



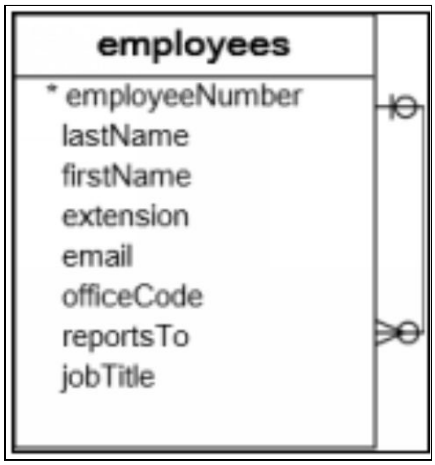
	store_name	product_name	revenue
▶	North	Macbook Pro	32475
	North	iPad	8985
	North	iPhone	13980
	South	iPhone	20970
	South	Macbook Pro	0
	South	iPad	20965

The “IFNULL” function was used in the syntax above to return “0” in case the revenue was reported as “NULL”.

“SELF JOIN”

As the name indicates, the “SELF JOIN” statements are utilized to link rows of a table to rows within the same table instead of another table. This requires you to use a “table alias” to aid the server in differentiating between the “left table” from the “right table” within the

same query. For instance, take the “employees” table in the “MySQL sample database” where structural data for the organization is stored along with the employee data, as shown in the picture below. The “reportsTo” column can be utilized to view the manager Id of any employee.



If you would like to view the complete “organization structure”, you could self-join the “employees” table using “employeeNumber” and “reportsTo” columns, using the syntax below. The 2 roles in the “employees” table are “Manager” and “Direct Reports”.

```
"SELECT
    CONCAT (m.lastname, ', ', m.firstname) AS 'Manager',
    CONCAT (e.lastname, ', ', e.firstname) AS 'Direct report'
FROM
    employees e
        INNER JOIN
    employees m ON m.employeeNumber = e.reportsto
ORDER BY manager;"
```

The “result set” is shown in the picture below:

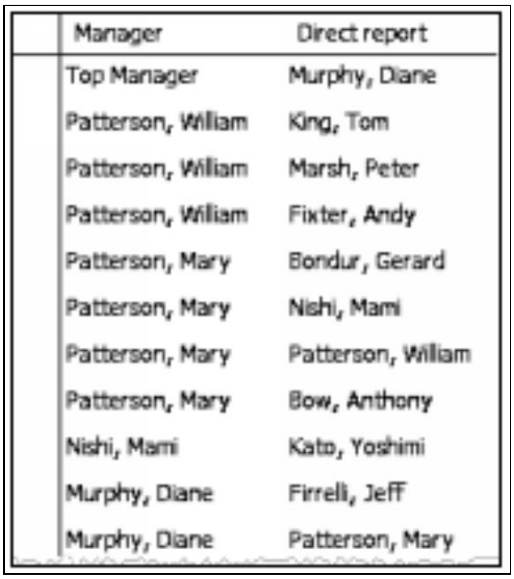
Manager	Direct report
Bondur, Gerard	Jones, Barry
Bondur, Gerard	Bott, Larry
Bondur, Gerard	Castillo, Pamela
Bondur, Gerard	Hernandez, Gerard
Bondur, Gerard	Bondur, Loui
Bondur, Gerard	Gerard, Martin
Bow, Anthony	Tseng, Foon Yue
Bow, Anthony	Patterson, Steve
Bow, Anthony	Firrelli, Julie
Bow, Anthony	Thompson, Leslie
Bow, Anthony	Jennings, Leslie
Bow, Anthony	Vanauf, George

Only employees who are reporting to a manager can be seen in the "result set" above. But the "top manager" is not visible as his name has been filtered out with the "INNER JOIN" clause. The "top manager" is the manager who is not reporting to any other manager or their “manager number” is holding a “NULL” value.

We can alter the "INNER JOIN" clause in the syntax above to the "LEFT JOIN" clause so it would include the "top manager" in the "result set". If the name of the "manager" is holding a “NULL” value, you can utilize the "IFNULL" clause to include the "top manager" in the output, as shown in the query below:

```
"SELECT
   IFNULL (CONCAT (m.lastname, ', ', m.firstname),
        'Top Manager') AS 'Manager',
   CONCAT (e.lastname, ', ', e.firstname) AS 'Direct report'
FROM
   employees e
      LEFT JOIN
   employees m ON m.employeeNumber = e.reportsto
ORDER BY manager DESC;"
```

The “result set” is shown in the picture below:



Manager	Direct report
Top Manager	Murphy, Diane
Patterson, William	King, Tom
Patterson, William	Marsh, Peter
Patterson, William	Fixter, Andy
Patterson, Mary	Bondur, Gerard
Patterson, Mary	Nishi, Mami
Patterson, Mary	Patterson, William
Patterson, Mary	Bow, Anthony
Nishi, Mami	Kato, Yoshimi
Murphy, Diane	Firrelli, Jeff
Murphy, Diane	Patterson, Mary

You will be able to show a “list of customers that are in the same location” by linking the "customers" table to itself with the use of the MySQL self join, as shown in the query below:

```
"SELECT
    c1.city, c1.customerName, c2.customerName
FROM
    customers c1
        INNER JOIN
    customers c2 ON c1.city = c2.city
        AND c1.customername > c2.customerName
ORDER BY c1.city;"
```

The “result set” is shown in the picture below:

city	customerName	customerName
Auckland	Kelly's Gift Shop	Down Under Souveniers, Inc
Auckland	GiftsForHim.com	Down Under Souveniers, Inc
Auckland	Kelly's Gift Shop	GiftsForHim.com
Boston	Gifts4AllAges.com	Diecast Collectables
Brickhaven	Online Mini Collectables	Auto-Moto Classics Inc.
Brickhaven	Collectables For Less Inc.	Auto-Moto Classics Inc.
Brickhaven	Online Mini Collectables	Collectables For Less Inc.
Cambridge	Marta's Replicas Co.	Cambridge Collectables Co.
Frankfurt	Messner Shopping Network	Blauer See Auto, Co.
Glendale	Gift Ideas Corp.	Boards & Toys Co.
Lisboa	Porto Imports Co.	Lisboa Souveniers, Inc
London	Stylish Desk Decors, Co.	Double Decker Gift Stores, Ltd

MySQL UNION

The “UNION” function in MySQL is used to merge 2 or more “result sets” into one comprehensive “result set”. The syntax for the “UNION” operator is given below:

"SELECT column_list

UNION [DISTINCT | ALL]

SELECT column_list

UNION [DISTINCT | ALL]

SELECT column_list

..."

The fundamental rules to follow with the use of the “UNION” operator are:

It is very critical that the number and the sequence of columns included in the “SELECT” statement is the same.

The “data types” of the columns are required to be identical or “convertible” to the same.

The redundant or duplicate rows will be removed without explicitly using the “DISTINT” function in the query.

Example

Consider the sample tables “t1” and “t2” as described in the syntax below:

"DROP TABLE IF EXISTS t1;

DROP TABLE IF EXISTS t2;

CREATE TABLE t1 (

id INT PRIMARY KEY

);

CREATE TABLE t2 (

id INT PRIMARY KEY

);

INSERT INTO t1 VALUES (1),(2),(3);

INSERT INTO t2 VALUES (2),(3),(4);"

The query below can be used to combine the “result sets” from “t1” and “t2” tables using the “UNION” operator:

"SELECT id

FROM t1

UNION

SELECT id

FROM t2;"

The combined “result set” generated will contain varying values from both the “result sets”, as shown below:

"+----+

| *id* |

+----+

| *1* |

| *2* |

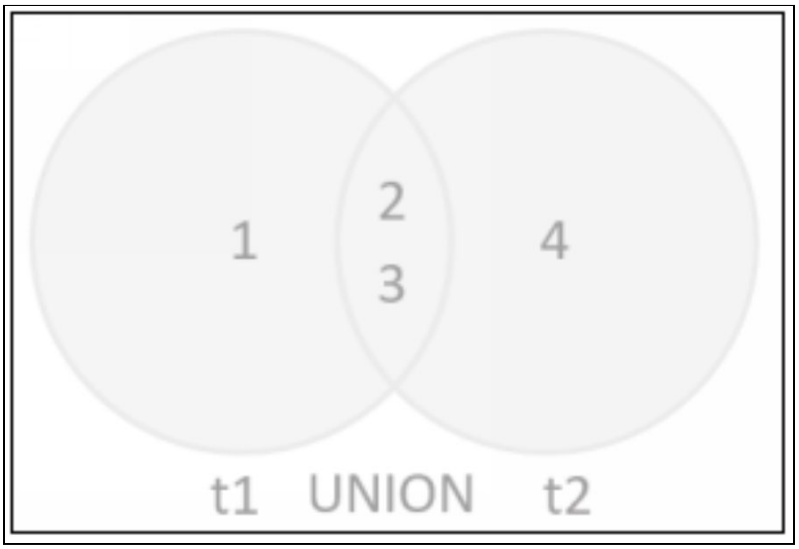
| *3* |

| *4* |

+----+

4 rows in set (0.00 sec)"

One can notice that the rows containing values “2” and “3” are redundant, so the “UNION” function dropped the duplicate and retain the distinct values only. The picture below of the Venn diagram, represents the combination of the “result sets” from “t1” and “t2” tables:



MySQL UNION ALL

The “UNION ALL” operator is utilized when you want to retain the duplicate rows (if any) in the “final result set”. The speed with which the query is executed is much higher for the “UNION ALL” operator than the “UNION” or “UNION DISTINCT” operator, as it does not need to deal with the redundancy of the data. The syntax for the “UNION ALL” operator is shown in the query below:

"SELECT id

FROM t1

UNION ALL

SELECT id

FROM t2;"

The “result set” is given below, which contains duplicate rows:

"+----+

| *id* |

+----+

| *1* |

| *2* |

| *3* |

| *2* |

| *3* |

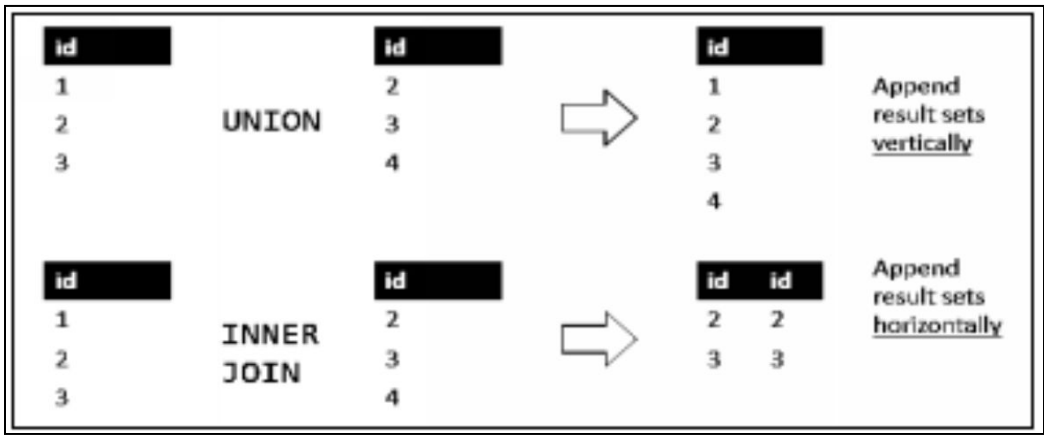
| *4* |

+----+

6 rows in set (0.00 sec)"

MySQL JOIN vs UNION

The “JOIN” clause is utilized to merge the “result sets” horizontally or on the basis of the rows or records. On the other hand, the “UNION” clause is utilized to merge the “result sets” vertically or on the basis of the columns of the tables. The picture below will help you understand the distinction between “UNION” and “JOIN” operators:



MySQL UNION and ORDER BY

The “ORDER BY” clause can be utilized to sort the results of a “UNION” operator as shown the query below:

"SELECT

concat (firstName,' ', lastName) fullname

FROM

employees

UNION SELECT

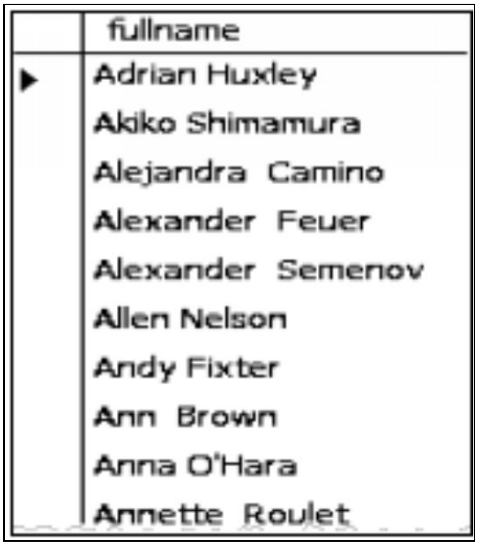
concat (contactFirstName,' ', contactLastName)

FROM

customers

ORDER BY fullname;"

The “result set” is shown in the picture below:



	fullname
▶	Adrian Huxley
	Akiko Shimamura
	Alejandra Camino
	Alexander Feuer
	Alexander Semenov
	Allen Nelson
	Andy Fixter
	Ann Brown
	Anna O'Hara
	Annette Roulet

If you would like to sort the result on the basis of a column position, you could utilize the “ORDER BY” as shown in the syntax below:

"SELECT

concat (firstName,' ', lastName) fullname

FROM

employees

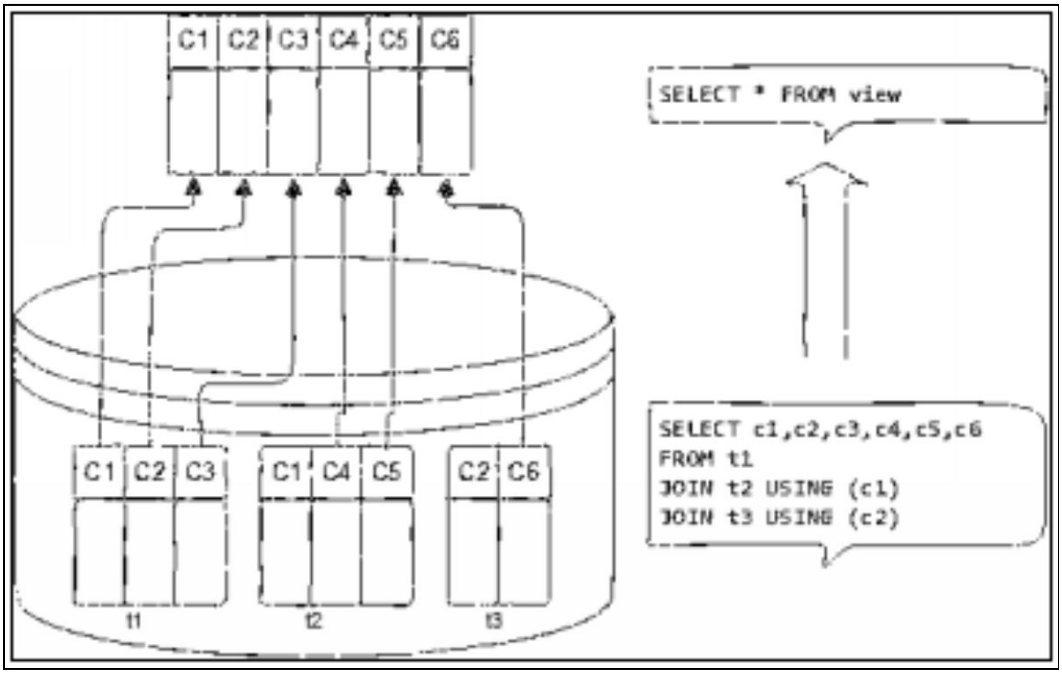
UNION SELECT

```
    concat (contactFirstName,' ', contactLastName)
FROM
    customers
ORDER BY 1;"
```

Chapter 4:
SQL Views and Transactions

A "Database View" in SQL can be defined as a “virtual table” or “logical table” described as "SQL SELECT" statement containing join function. As a "Database View" is like a table in the database consisting of rows and columns, you will be able to easily run queries on it. Many DBMSs, including “MySQL”, enable users to modify information in the existing tables using "Database View" by meeting certain prerequisites, as shown in the picture below.

A "SQL database View" can be deemed dynamic since there is no connection between the "SQL View" to the physical system. The database system will store "SQL Views" in the form on "SELECT" statements using "JOIN" clause. When the information in the table is modified, the "SQL View" will also reflect that modification.



Pros of using SQL View

A "SQL View" enables simplification of complicated queries: a "SQL View" is characterized by a SQL statement which is associated with

multiple tables. To conceal the complexity of the underlying tables from the end users and external apps, "SQL View" is extremely helpful. You only need to use straightforward "SQL" statements instead of complicated ones with multiple "JOIN" clauses using the "SQL View".

A "SQL View" enables restricted access to information depending on the user requirements. Perhaps you would not like all users to be able to query a subset of confidential information. In such cases "SQL View" can be used to selectively expose non-sensitive information to a targeted set of users.

The "SQL View" offers an additional layer of safety. Security is a key component of every "relational database management system". The "SQL View" ensures “extra security” for the DBMS. The "SQL View" enables generation of a “read-only” view to display “read-only” information for targeted users. In "read-only view", users are able to only retrieve data and are not allowed to update any information.

The "SQL View" is used to enable computed columns. The table in the database is not capable of containing computed columns but a "SQL View" can easily contain computed column. Assume in the "OrderDetails" table there is "quantityOrder" column for the amount of products ordered and "priceEach" column for the price of each item. But the "orderDetails" table cannot contain a calculated column storing total sales for every product from the order. If it could, the database schema may have never been a good design. In such a situation, to display the calculated outcome, you could generate a computed column called "total", which would be a product of "quantityOrder" and "priceEach" columns. When querying information from the "SQL View", the calculated column information will be calculated on the fly.

A "SQL View" allows for backward compatibility. Assume that we have a central database that is being used by multiple applications. Out of the blue you have been tasked to redesign the database accommodating the new business needs. As you delete some tables and create new ones, you would not want

other applications to be affected by these modifications. You could generate "SQL Views" in such situations, using the identical schematic of the “legacy tables” that you are planning to delete.

Cons of using SQL View In addition to the pros listed above, the use of "SQL View" may have certain disadvantages such as:

Performance: Executing queries against "SQL View" could be slow, particularly if it is generated from another "SQL View".

Table dependencies: Since a "SQL View" is created from the underlying tables of the database. Anytime the tables structure connected with "SQL View" is modified, you also need to modify the "SQL View".

Views in MySQL server

As of the release of MySQL version 5+, it has been supporting database views. In MySQL, nearly all characteristics of a "View" conform to the "standard SQL: 2003".

MySQL can run queries against the views in couple of ways:

MySQL can produce a “temporary table” on the basis of the “view definition statement” and then execute all following queried on this "temporary table".

MySQL can combine the new queries with the query that defines the “SQL View” into a single comprehensive query and then this merged query can be executed.

MySQL offers versioning capability for all "SQL Views". Whenever a “SQL View” is modified or substituted, a clone of the view is backed up in the "arc (archive) directory" residing in a particular database folder. The “backup file” is named as "view name.frm-00001". If you modify your view again, then MySQL will generate a new “backup file” called "view name.frm-00002."

You can also generate a view based on other views through MySQL, by creating references for other views in the "SELECT" statement defining the target "SQL View".

“CREATE VIEW” in MySQL

The “CREATE VIEW” query can be utilized to generate new “SQL Views” in MySQL, as shown in the syntax below:

"CREATE

[ALGORITHM = {MERGE | TEMPTABLE | UNDEFINED}]

VIEW view_name [(column_list)]

AS

select-statement;"

“View processing algorithms”

The attribute of the algorithm enables you to regulate which mechanism is utilized by MySQL to create the "SQL View". Three algorithms are provided by MySQL, namely: "MERGE", "TEMPTABLE" and "UNDEFINED".

To use the "MERGE" algorithm, server will start by combining the incoming query with the "SELECT" statement, that will define the "view" into a merged query. Subsequently, the merged query is executed by MySQL to retrieve the "result set". The "MERGE" algorithm cannot be used if the "SELECT" statement consists of “aggregate functions” such as "MIN, MAX, SUM, COUNT, AVG or DISTINCT, GROUP BY, HAVING, LIMIT, UNION, UNION ALL, subquery". If the "SELECT" statement does not refer to a table, then the “MERGE” algorithm cannot be utilized. In the cases, where the "MERGE" algorithm is not permitted, server will instead use the "UNDEFINED" algorithm. Please remember that the view resolution is defined as "the combination of input query and the query in the view definition into one query".

Using the "TEMPTABLE" algorithm, server will start with production of a "temporary table" that describes the "SQL View" on the basis of the "SELECT" statement. Subsequently any incoming query will be executed against this "temporary table". The "TEMPTABLE" algorithm is lower efficiency than the "MERGE" algorithm because MySQL requires generation of a "temporary table" to store the "result set" and will transfer the data

from the “permanent tables” to the "temporary table". Moreover, a "SQL View" using the "TEMPTABLE" algorithm can not be updated.

If you generate a "view" without explicitly stating an algorithm that needs to be used, then the "UNDEFINED" algorithm will be used by default. The "UNDEFINED" algorithm allows MySQL to choose from the MERGE or TEMPTABLE algorithm to be used. Since the "MERGE" algorithm is much more effective, MySQL prefers "MERGE" algorithm over "TEMPTABLE" algorithm.

View Name

Views and tables are stored in the same space within the database, so it is not possible to give the same name to a view and a table. Furthermore, the name of a view has to be in accordance with the naming conventions of the table.

“SELECT” statement

It is easy to query desired data from any table or view contained within a database by utilizing the "SELECT" statement. The "SELECT" statement is required to meet various guidelines, such as:

It may comprise a “subquery” in the "WHERE" clause but cannot have one in the "FROM" clause.

No variables, including "local variables", "user variables", and "session variables", can be referred to in the "SELECT" statement.

The "SELECT" statement can also not have a reference to the parameters of prepared statements and is not required to refer to any table in the database.

Example

To create a “SQL View” of the “orderDetails” table that contains the total “sales per order”, you can use the query below:

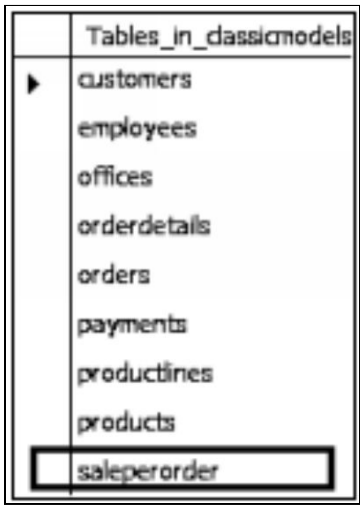
```
"CREATE VIEW SalePerOrder AS
   SELECT
      orderNumber, SUM (quantityOrdered * priceEach) total
   FROM
```

orderDetails

GROUP by orderNumber

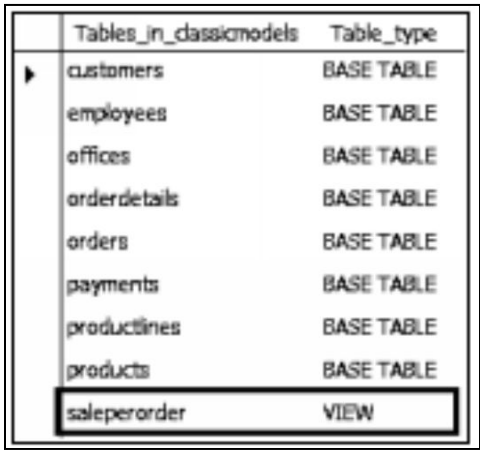
ORDER BY total DESC;"

By using the "SHOW TABLE" command you can check all the tables in the "classicmodels" database, it is easily visible that the "SalesPerOrder" view is on the displayed list (shown in the picture below):



Tables_in_classicmodels
customers
employees
offices
orderdetails
orders
payments
productlines
products
saleperorder

This proves that the view names and table names are stored in the same space within the database. So, now if you would like to know which object in the picture above is a “view” and which one is a “table”, use the “SHOW FULL TABLE” query and the “result set” will be displayed as shown in the picture below:



Tables_in_classicmodels	Table_type
customers	BASE TABLE
employees	BASE TABLE
offices	BASE TABLE
orderdetails	BASE TABLE
orders	BASE TABLE
payments	BASE TABLE
productlines	BASE TABLE
products	BASE TABLE
saleperorder	VIEW

You can check the “table_type” column in the picture above to confirm which object is a view or a table.

Now, you can simply use the “SELECT” statement below to view “total sales for each sales order”:

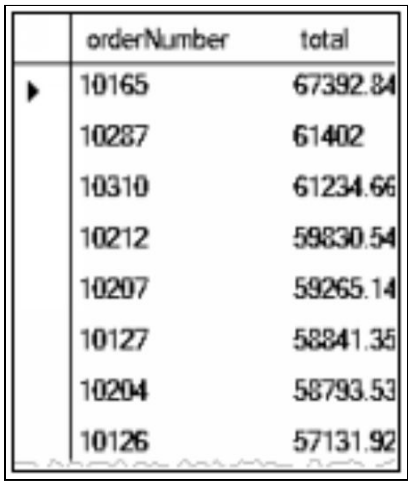
"SELECT

***

FROM

salePerOrder;"

The “result set” is displayed in the picture below:



	orderNumber	total
▶	10165	67392.84
	10287	61402
	10310	61234.66
	10212	59830.54
	10207	59265.14
	10127	58841.35
	10204	58793.53
	10126	57131.92

Creating a “SQL View” from another “SQL view”

The MySQL server permits creation of a new view on the basis of another view. For instance, you could produce a view named "BigSalesOrder" based on the "SalesPerOrder" view, that we created earlier to show every sales order for which the total adds up to more than 60,000 using the syntax below:

"CREATE VIEW BigSalesOrder AS

SELECT

orderNumber, ROUND(total,2) as total

FROM

saleperorder

WHERE

total > 60000;"

Now, you could easily retrieve desired data from the "BigSalesOrder" view as shown below:

"SELECT

orderNumber, total

FROM

BigSalesOrder;"

	orderNumber	total
▶	10165	67392.85
	10287	61402.00
	10310	61234.67

Create “SQL View” with “JOIN” clause

The MySQL database allows you to create “SQL View” using “JOIN” clause. For instance, the query below with the “INNER JOIN” clause can be used to create a view containing "order number", "customer name", and "total sales per order":

"CREATE VIEW customerOrders AS

SELECT

d.orderNumber,

customerName,

*SUM (quantityOrdered * priceEach) total*

FROM

orderDetails d

INNER JOIN

orders o ON o.orderNumber = d.orderNumber

INNER JOIN

customers c ON c.customerNumber = c.customerNumber

GROUP BY d.orderNumber

ORDER BY total DESC;"

You can use the syntax below to retrieve desired data from the “customerOrders” view:

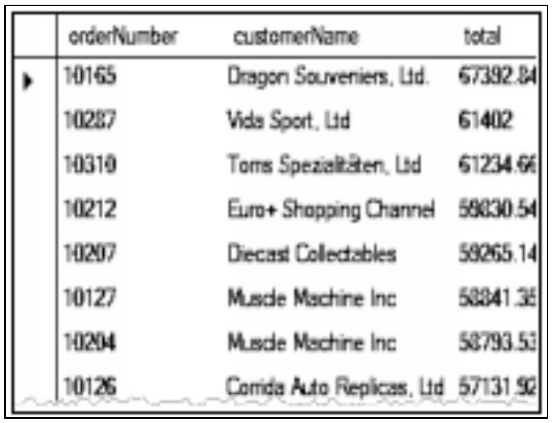
"SELECT

*

FROM

customerOrders;"

The “result set” is displayed in the picture below:



	orderNumber	customerName	total
▶	10165	Dragon Souveniers, Ltd.	67392.84
	10287	Vida Sport, Ltd	61402
	10310	Toms Spezialitäten, Ltd	61234.66
	10212	Euro+ Shopping Channel	59830.54
	10207	Diecast Collectables	59265.14
	10127	Muscle Machine Inc	58841.35
	10264	Muscle Machine Inc	58793.53
	10126	Corrida Auto Replicas, Ltd	57131.92

Create “SQL View” with a subquery

The query below can be used to generate a “SQL View” with a “subquery”, containing products with purchase prices greater than the average product prices.

"CREATE VIEW aboveAvgProducts AS

SELECT

productCode, productName, buyPrice

FROM

products

WHERE

buyPrice >

```
(SELECT
        AVG(buyPrice)
    FROM
        products)
ORDER BY buyPrice DESC;"
```

The query to extract data from the “aboveAvgProducts” view is even more straightforward, as shown in the picture below:

```
"SELECT
    *
FROM
    aboveAvgProducts;"
```

The “result set” is displayed in the picture below:

	productCode	productName	buyPrice
▶	S10_4962	1962 LanciaA Delta 16V	103.42
	S18_2238	1998 Chrysler Plymouth Prowler	101.51
	S10_1949	1952 Alpine Renault 1300	98.58
	S24_3856	1956 Porsche 356A Coupe	98.3
	S12_1108	2001 Ferrari Enzo	95.59
	S12_1099	1968 Ford Mustang	95.34
	S18_1984	1995 Honda Civic	93.89
	S18_4027	1970 Triumph Spitfire	91.92

Updatable SQL Views

The MySQL server offers the capability to not only query the view but also to update them. The “INSERT” or “UPDATE” statements can be used add and edit the records of the underlying table through the updatable “MySQL View”. Moreover, the “DELETE” statement can be used to drop records from the underlying table through the updatable “MySQL View”.

Furthermore, the "SELECT" statement used to generate and define an updatable view can not include any of the functions listed below:

“Aggregate functions” such as “MIN, MAX, SUM, AVG, and COUNT”.

“DISTINCT”.

“GROUP BY”.

“HAVING”.

“UNION” or “UNION ALL”.

“LEFT JOIN” or “OUTER JOIN”.

“Subquery in the SELECT statement or in the WHERE clause referring to the table indicated in the FROM clause”.

“Reference to non-updatable view in the FROM clause”.

“Reference only to literal values”.

“Multiple references to any column of the underlying table”.

Remember, the “TEMPTABLE” algorithm cannot be used to generate an updatable “MySQL View”.

Example

You can use the syntax below to generate a view called “officeInfo” on the basis of the “offices” table in the “MySQL sample database”. This “updatable view” will refer to 3 columns of the "offices" table: "officeCode", "phone", and "city".

```
"CREATE VIEW officeInfo
 AS
   SELECT officeCode, phone, city
   FROM offices;"
```

Now, if you would like to query the data from the “officeInfo” view, you can easily do that by using the syntax below:

```
"SELECT
    *
```

FROM

officeInfo;"

The “result set” is displayed in the picture below:

	officeCode	phone	city
▶	1	+1 650 219 4782	San Francisco
	2	+1 215 837 0825	Boston
	3	+1 212 555 3000	NYC
	4	+33 14 723 4404	Paris
	5	+81 33 224 5000	Tokyo
	6	+61 2 9264 2451	Sydney
	7	+44 20 7877 2041	London

You could also update the office contact number with "officeCode" 4 using the "UPDATE" statement as given below, on the "officeInfo" view:

"UPDATE officeInfo

SET

phone = '+33 14 723 5555'

WHERE

officeCode = 4;"

Lastly, you can confirm this modification, using the syntax below to retrieve desired data from the "officeInfo" view:

"SELECT

*

FROM

officeInfo

WHERE

officeCode = 4;"

The “result set” is displayed in the picture below:

	officeCode	phone	city
▶	4	+33 14 723 5555	Paris

Check if an existing view is updatable

By running a query against the "is_updatable" column from the view in the "information_schema" database, you should verify whether a view in the database is updatable or not.

You can use the query below to display all the views from the "classicmodels" database and check which of them are updatable:

"SELECT

table_name,

is_updatable

FROM

information_schema.views

WHERE

table_schema = 'classicmodels';"

The “result set” is displayed in the picture below:

	table_name	is_updatable
▶	aboveavgproducts	YES
	customerorders	NO
	officeinfo	YES
	saleperorder	NO

Dropping rows using “SQL View”

To understand this concept, execute the syntax below to first create a table called “items’, use the “INSERT” statement to add records into this table and then use the “CREATE” clause to generate a view containing items with prices higher than “700”.

"-- create a new table named items

```
CREATE TABLE items (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    price DECIMAL(11 , 2 ) NOT NULL
);
-- insert data into the items table
INSERT INTO items(name,price)
VALUES('Laptop',700.56),('Desktop',699.99),('iPad',700.50) ;
-- create a view based on items table
CREATE VIEW LuxuryItems AS
    SELECT
        *
    FROM
        items
    WHERE
        price > 700;
-- query data from the LuxuryItems view
SELECT
    *
FROM
    LuxuryItems;"
```

The “result set” is displayed in the picture below:

	id	name	price
▶	1	Laptop	700.56
	3	iPad	700.50

Now, use the “DELETE” clause to drop a record with “id value 3”.

"DELETE FROM LuxuryItems

WHERE

id = 3;"

After you run the query above, you will receive a message stating “1 row(s) affected”.

Now, you can verify the data with the view, using the query below:

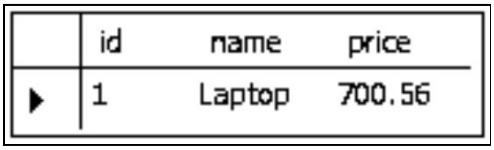
"SELECT

*

FROM

LuxuryItems;"

The “result set” is displayed in the picture below:



	id	name	price
▶	1	Laptop	700.56

Finally, use the syntax below to retrieve desired data from the underlying table “items” to confirm if the “DELET” statement in fact removed the record:

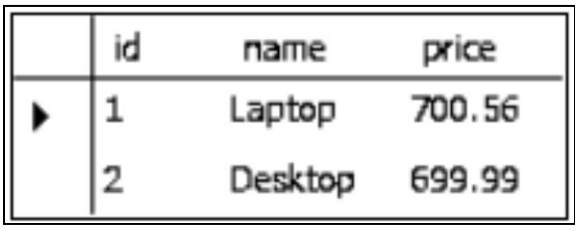
"SELECT

*

FROM

items;"

The “result set” is displayed in the picture below, which confirms that the row with “id value 3” was deleted from the “items” table:



	id	name	price
▶	1	Laptop	700.56
	2	Desktop	699.99

Modification of “SQL View”

In MySQL, you can use “ALTER VIEW” and “CREATE OR REPLACE VIEW” statements to make changes to an existing view.

Using “ALTER VIEW” statement

The syntax for “ALTER VIEW” is very similar to the “CREATE VIEW” statement that you learned earlier, the only difference being that the “ALTER” keyword is used instead of the “CREATE” keyword, as shown below:

```
"ALTER
 [ALGORITHM =  {MERGE | TEMPTABLE | UNDEFINED}]
  VIEW [database_name].  [view_name]
   AS
 [SELECT  statement]"
```

The query below will change the “organization” view by incorporating the “email” column in the table:

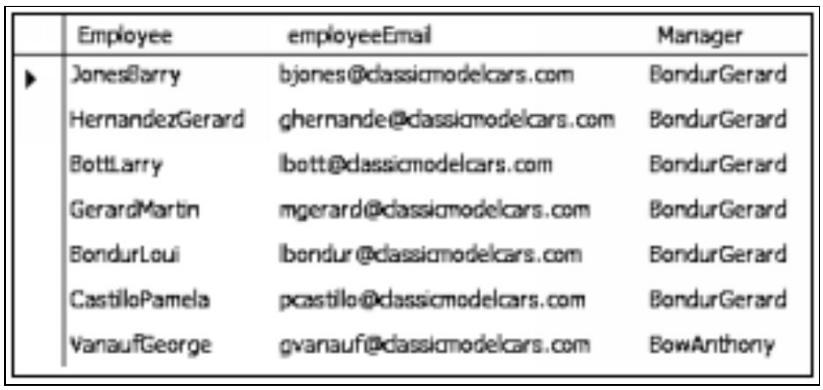
```
"ALTER VIEW organization
  AS
  SELECT CONCAT(E.lastname,E.firstname) AS Employee,
        E.email AS  employeeEmail,
        CONCAT(M.lastname,M.firstname) AS Manager
  FROM employees AS E
  INNER JOIN employees AS M
    ON M.employeeNumber = E.ReportsTo
  ORDER BY Manager;"
```

You can run the query below against the “organization” view to verify the modification:

```
"SELECT
    *
FROM
```

Organization;"

The “result set” is displayed in the picture below:



	Employee	employeeEmail	Manager
▶	JonesBarry	bjones@classicmodelcars.com	BondurGerard
	HernandezGerard	ghernande@classicmodelcars.com	BondurGerard
	BottLarry	lbott@classicmodelcars.com	BondurGerard
	GerardMartin	mgerard@classicmodelcars.com	BondurGerard
	BondurLoui	lbondur@classicmodelcars.com	BondurGerard
	CastilloPamela	pcastillo@classicmodelcars.com	BondurGerard
	VanaufGeorge	gvanauf@classicmodelcars.com	BowAnthony

Using “CREATE OR REPLACE VIEW” statement

The “CREATE OR REPLACE VIEW” can be used to replace or generate a “SQL View” that already exists in the database. For all existing views, MySQL will easily modify the view but if the view is non-existent, it will create a new view based on the query.

The syntax below can be used to generate the “contacts view” on the basis of the “employees” table:

"CREATE OR REPLACE VIEW contacts AS

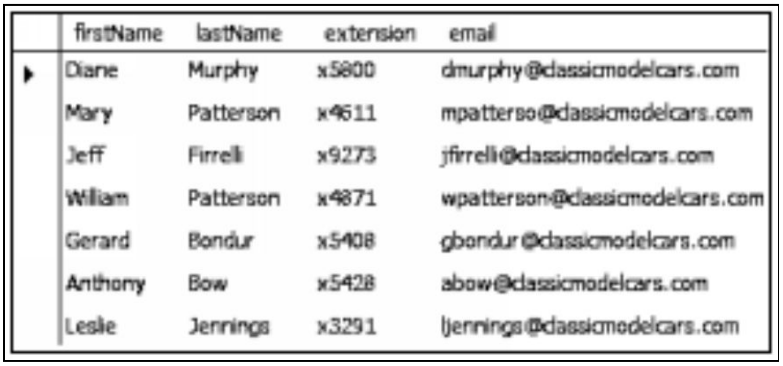
SELECT

firstName, lastName, extension, email

FROM

employees;"

The “result set” is displayed in the picture below:



	firstName	lastName	extension	email
▶	Diane	Murphy	x5800	dmurphy@classicmodelcars.com
	Mary	Patterson	x4611	mpatterso@classicmodelcars.com
	Jeff	Firrelli	x9273	jfirrelli@classicmodelcars.com
	William	Patterson	x4871	wpatterson@classicmodelcars.com
	Gerard	Bondur	x5408	gbondur@classicmodelcars.com
	Anthony	Bow	x5428	abow@classicmodelcars.com
	Leslie	Jennings	x3291	ljennings@classicmodelcars.com

Now, assume that you would like to insert the “jobtitle” column to the “contacts view”. You can accomplish this with the syntax below:

"CREATE OR REPLACE VIEW contacts AS

SELECT

firstName,

lastName,

extension,

email,

jobtitle

FROM

employees;"

The “result set” is displayed in the picture below:

firstName	lastName	extension	email	jobtitle
Diane	Murphy	x5800	dmurphy@classicmodelcars.com	President
Mary	Patterson	x4611	mpatterso@classicmodelcars.com	VP Sales
Jeff	Firrelli	x9273	jfirrelli@classicmodelcars.com	VP Marketing
William	Patterson	x4871	wpatterson@classicmodelcars.com	Sales Manager (APAC)
Gerard	Bondur	x5408	gbondur@classicmodelcars.com	Sale Manager (EMEA)
Anthony	Bow	x5428	abow@classicmodelcars.com	Sales Manager (NA)
Leslie	Jennings	x3291	ljennings@classicmodelcars.com	Sales Rep

Dropping a “SQL View”

The “DROP VIEW” statement can be utilized to delete an existing view from the database, using the syntax below:

"DROP VIEW [IF EXISTS] [database_name].[view_name]"

The "IF EXISTS" clause is not mandatory in the statement above and is used to determine if the view already exists in the database. It prevents you from mistakenly removing a view that does not exists in the database.

You may, for instance, use the "DROP VIEW" statement as shown in the syntax below to delete the "organization" view:

"DROP VIEW IF EXISTS organization;"

SQL TRANSACTIONS

A transaction can be defined as "a unit of work that is performed against a database". They are “units or work sequences” that are performed in a “logical order”, either manually by a user or automatically using by the database program.

Or simply put, they are “the spread of one or more database modifications”. For instance, if you create a row, update a row, or delete a row from the table, that means you are executing a transaction on that table. To maintain data integrity and address database errors, it is essential to regulate these transactions.

Basically, to execute a transaction, you must group several SQL queries and run them at the same time.

Properties of Transactions

The fundamental properties of a transaction can be defined using the acronym “**ACID**” for the properties listed below:

Atomicity − guarantees successful completion of all operations grouped in the work unit. Or else, at the point of failure, the transaction will be aborted and all prior operations will be rolled back to their original state.

Consistency − makes sure that when a transaction is properly committed, the database states are also correctly updated.

Isolation − allows independent and transparent execution of the transactions.

Durability − makes sure that in case of a system malfunction, the outcome or impact of a committed transaction continues to exist.

To explain this concept in greater detail, consider the steps below for addition of a new sales order in the “MySQL sample database”:

Start by querying the most recent “sales order number” from the "orders" table and utilize the next “sales order number” as the new “order number”.

Then use the "INSERT" clause to add a new “sales order” into the "orders" table.

Next, retrieve the “sales order number” that was inserted in the previous step.

Now, "INSERT" the new “sales order items” into the "orderdetails" table containing the “sales order numbers”.

At last, to verify the modifications, select data from both "orders" and "orderdetails" tables.

Try to think, how would the sales order data be modified, if even a single steps listed here were to fail, for whatever reason. For instance, if the step for inserting items of an order to the "orderdetails" table failed, it will result in an “empty sales order”.

This is where the "transaction processing" is used as a safety measure. You can perform "MySQL transactions" to run a set of operations making sure that the database will not be able to contain any partial operations. When working with multiple operations concurrently, if even one of the operation fails, a "rollback" can be triggered. If there is no failure, all the statements will be “committed” to the database.

“MySQL Transaction” statements

MySQL offers statements listed below for controlling the transactions:

For initiating a transaction, utilize the "START TRANSACTION" statement. The "BEGIN" or "BEGIN WORK" statement are same as the "START TRANSACTION".

For committing the “current transaction” and making the modifications permanent, utilize the "COMMIT" statement.

By using the "ROLLBACK" declaration, you can simply undo the current transaction and void its modifications.

By using the "SET auto-commit" statement, you can deactivate or activate the auto-commit mode for the current transaction.

By default, MySQL is designed to commit the modifications to the database permanently. By using the statement below, you can force MySQL not to commit the modifications by default:

"SET autocommit = 0;

Or

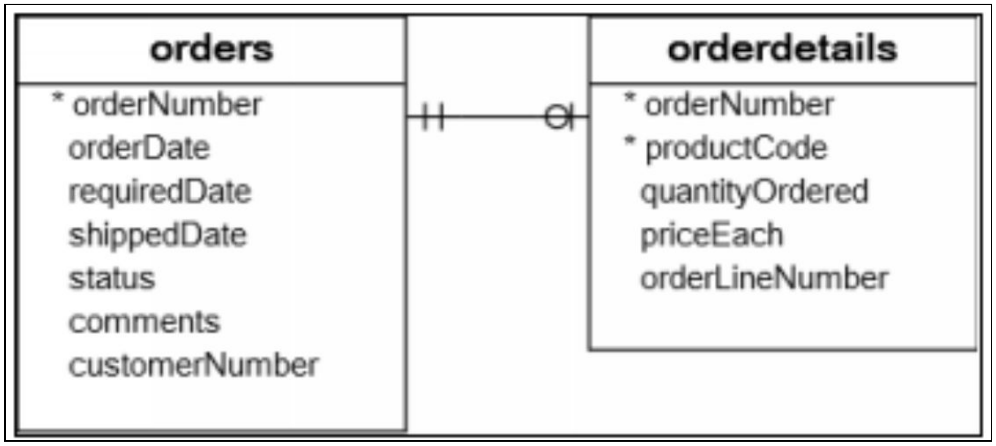
SET autocommit = OFF;"

To reactivate the default mode for auto-commit, you can use the syntax below:

"SET autocommit = ON;"

Example

Let’s utilize the “orders” and “orderDetails” tables, shown in the picture below, from the “MySQL sample database” to understand this concept further.



“COMMIT” transaction

You must first split the SQL statement into logical parts to effectively use a transaction and assess when the transaction needs to be committed or rolled back.

The steps below show how to generate a new “sales order”:

Utilize the "START TRANSACTION" statement to begin a transaction.

Select the most recent “sales order number” from the "orders" table and utilize the subsequent “order number” as the new

“order number”.

“Insert" a new sales order in the "orders" table.

“Insert" sales order items into the "orderdetails" table.

Lastly, use the "COMMIT" statement to commit the transaction.

You could potentially use data from “orders” and “orderdetails” table to verify the new “sales order”, as shown in the syntax below:

```
“ start a new transaction
START TRANSACTION;
Get the latest order number
SELECT
   @orderNumber:=MAX(orderNUmber)+1
FROM
   orders;
insert a new order for customer 145
INSERT INTO orders (orderNumber,
                orderDate,
                requiredDate,
                shippedDate,
                status,
                customerNumber)
VALUES (@orderNumber,
      '2005-05-31',
      '2005-06-10',
      '2005-06-11',
      'In Process',
       145);
```

Insert order line items

INSERT INTO orderdetails(orderNumber,

productCode,

quantityOrdered,

priceEach,

orderLineNumber)

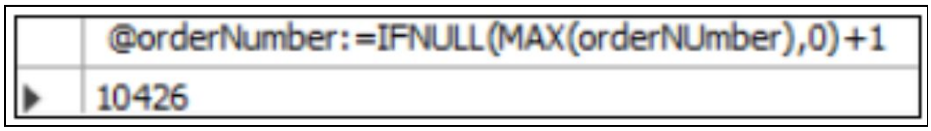
VALUES (@orderNumber,'S18_1749', 30, '136', 1),

(@orderNumber,'S18_2248', 50, '55.09', 2);

commit changes

COMMIT;"

The “result set” is displayed in the picture below:



	@orderNumber:=IFNULL(MAX(orderNUmber),0)+1
▶	10426

Using the query below, you can retrieve the new sales order that you just created:

"SELECT

a.orderNumber,

orderDate,

requiredDate,

shippedDate,

status,

comments,

customerNumber,

orderLineNumber,

productCode,

quantityOrdered,

priceEach

FROM

orders a

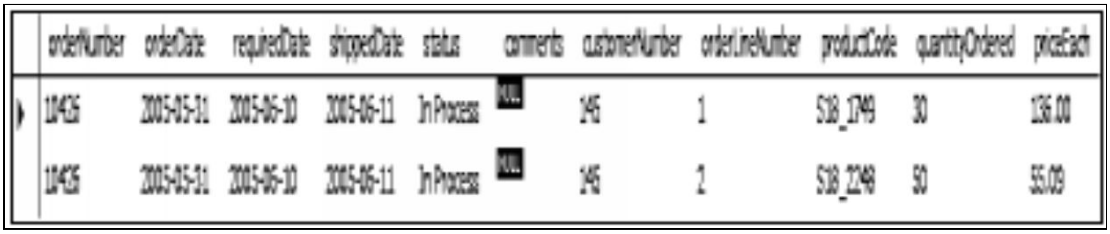
INNER JOIN

orderdetails b USING (orderNumber)

WHERE

a.ordernumber = 10426;"

The “result set” is displayed in the picture below:



orderNumber	orderDate	requiredDate	shippedDate	status	comments	customerNumber	orderLineNumber	productCode	quantityOrdered	priceEach
10426	2005-05-31	2005-06-10	2005-06-11	In Process	NULL	145	1	S18_1749	30	136.00
10426	2005-05-31	2005-06-10	2005-06-11	In Process	NULL	145	2	S18_2248	50	55.09

“ROLLBACK” transaction

The first step here is deletion of the data in the “orders” table, using the statement below:

"mysql> START TRANSACTION;

Query OK, 0 rows affected (0.00 sec)

mysql> DELETE FROM orders;

Query OK, 327 rows affected (0.03 sec)"

It's obvious from from the output of the queries above that all records from the “orders” table have been removed.

Now, you need to login to the “MySQL database server” in a distinct session and run the query below against the "orders" table.

"mysql> SELECT COUNT(*) FROM orders;

+----------+

| COUNT(*) |

+----------+

| *327* |

+----------+

1 row in set (0.00 sec)"

The data from the “orders” table can still be viewed from the second session.

The changes made in the first session are not permanent so you need to either commit them or roll them back. Let’s assume you would like to undo the changes from the first session, utilize the query below:

"mysql> ROLLBACK;

Query OK, 0 rows affected (0.04 sec)"

The data in the “orders” table from the first session can be verified, using the syntax below:

"mysql> SELECT COUNT(*) FROM orders;

+----------+

| *COUNT(*)* |

+----------+

| *327* |

+----------+

1 row in set (0.00 sec)"

It can be seen in the “result set” above that the modifications have been rolled back.

“SAVEPOINT” Transaction

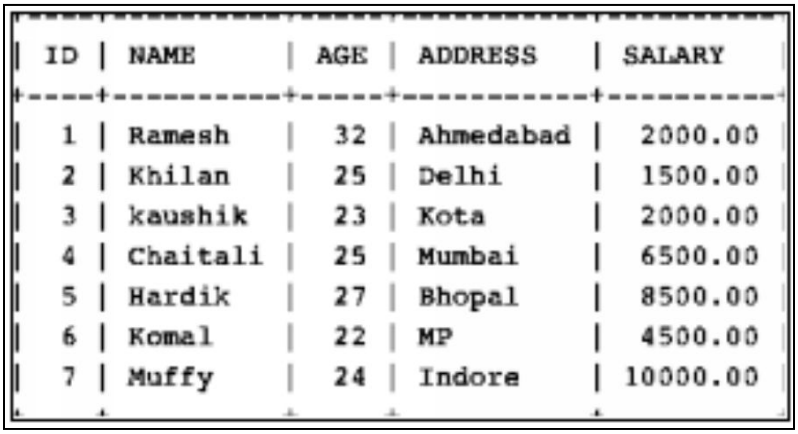
The “SAVEPOINT” is defined as “a point in a transaction when you can roll the transaction back to a certain point without rolling back the entire transaction”, as shown in the syntax below:

*"SAVEPOINT SAVEPOINT_NAME;"*The query above is only used to create a “SAVEPOINT” within all the transaction statements and then

the “ROLLBACK” query can be utilized to retract desired transactions, as shown in the query below:

"ROLLBACK TO SAVEPOINT_NAME;"

For example, assume you have a “customers” table as shown in the picture below and want to delete 3 rows and create a “SAVEPOINT” prior to every deletion and then subsequent “ROLLBACK” the change to the preceding state as needed.



ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

You can accomplish this by running the queries below:

"SQL> SAVEPOINT SP1;

Savepoint created.

SQL> DELETE FROM CUSTOMERS WHERE ID=1;

1 row deleted.

SQL> SAVEPOINT SP2;

Savepoint created.

SQL> DELETE FROM CUSTOMERS WHERE ID=2;

1 row deleted.

SQL> SAVEPOINT SP3;

Savepoint created.

SQL> DELETE FROM CUSTOMERS WHERE ID=3;

1 row deleted."

With the query above, you have successfully deleted 3 records and are now ready to use the query below to “ROLLBACK” the change to the “SAVEPOINT” named “SP2”, which was generated post first deletion so the last 2 deletions will be rolled back:

"SQL> ROLLBACK TO SP2;

Rollback complete."

The “result set” is displayed in the picture below:

ID	NAME	AGE	ADDRESS	SALARY
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

“RELEASE SAVEPOINT” Transaction

The “RELEASE SAVEPOINT” can be utilized to delete a “SAVEPOINT” that might have been generated earlier, as shown in the syntax below:

"RELEASE SAVEPOINT SAVEPOINT_NAME;"

Remember, once you run the query above there is no function undo the transactions that might have been executed after the last “SAVEPOINT”.

“SET TRANSACTION”

The “SET TRANSACTION” can be utilized to start a transaction by specifying features of the subsequent transactions. For instance, using the syntax below, a transaction can be made “read and write only”:

"SET TRANSACTION [READ WRITE | READ ONLY];"

Database Recovery Models

A recovery model can be defined as "a database property that controls how transactions are logged, whether the transaction log requires and allows backing up, and what kinds of restore operations are available". It is a database configuration option that specifies the type of backup that can be performed and enables the data to be restored or recovered in case of a failure. The 3 types of “SQL Server recovery models” are:

SIMPLE

This model has been deemed as the easiest of all the recovery models that exist. It offers "full, differential, and file level" backups. However, backup of the transaction log is not supported. The log space would be reused whenever the checkpoint operation of the SQL Server background process takes place. The log file's inactive part is deleted and made accessible for reuse.

There is no support for point-in-time and page restoration, only the secondary read-only files can be restored.

FULL

All types of transactions "(DDL (Data Definition Language) as well as the DML (Data Manipulation Language))" in the transaction log file can be fully registered in the "FULL" recovery model. The sequence of logs remains untouched and maintained for restoration activities of the database. Unlike the "Simple Recovery Model", during the "CHECKPOINT" transaction, the “transaction log” file would not be truncated automatically.

It supports all restoration activities, such as "point-in-time restore, page restore and file restore".

Bulk Logged

The "Bulk Logged recovery model" is a unique database configuration option, which produces similar outcome as the "FULL recovery model" with the exception of some “bulk operations” that can be logged only minimally. The “transaction log” file utilizes a method called as "minimal logging for bulk operations". The caveat being that specific point in time data can not be restored.

“SQL BACKUP DATABASE” Statement

The “BACKUP DATABASE” statement can be utilized to generate a full back up of a database that already exists on the SQL server, as displayed in the syntax below:

"BACKUP DATABASE databasename

TO DISK = 'filepath';"

"SQL BACKUP WITH DIFFERENTIAL" Statement

A "differential backup" is used to selectively back up the sections of the database which were altered after the last “full backup of the database”, as displayed in the syntax below:

"BACKUP DATABASE databasename

TO DISK = 'filepath'

WITH DIFFERENTIAL;"

Example

Consider that you have a database called “testDB” and you would like to create a full back up of it. To accomplish this, you can use the query below:

"BACKUP DATABASE testDB

TO DISK = 'D:\backups\testDB.bak';"

Now, if you made modifications to the database after running the query above. You can use the query below to create a back up of those modifications:

"BACKUP DATABASE testDB

TO DISK = 'D:\backups\testDB.bak'

WITH DIFFERENTIAL;"

RESTORING Database

The Restoration can be defined as the method by which “data is copied from a backup and logged transactions are applied to that data”. The backup file generated (as discussed earlier) is

used to return database to a former state. There are couple of methods to accomplish this:

T-SQL

The syntax for this method is given below:

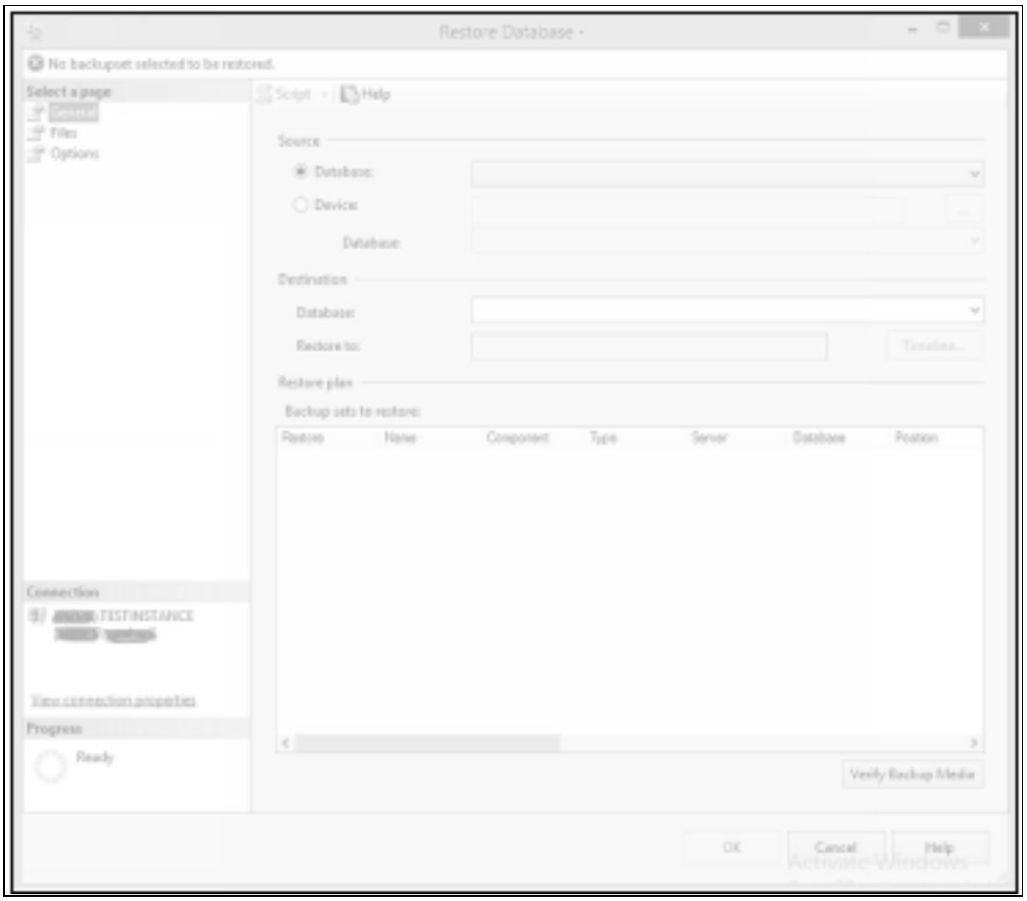
"Restore database <Your database name> from disk = '<Backup file location + file name>';"

For instance, the query below can be utilized to restore a database named “TestDB” using the back up file named “TestDB_Full.bak”, located in “D:\:

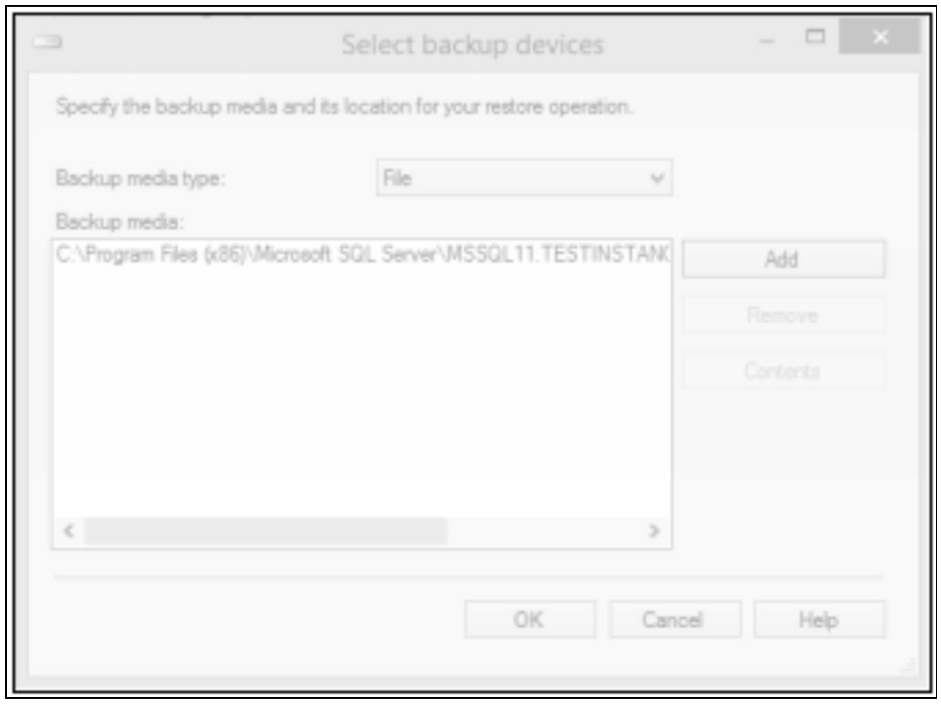
"Restore database TestDB from disk = ' D:\TestDB_Full.bak' with replace;"

“SSMS (SQL SERVER Management Studio)”

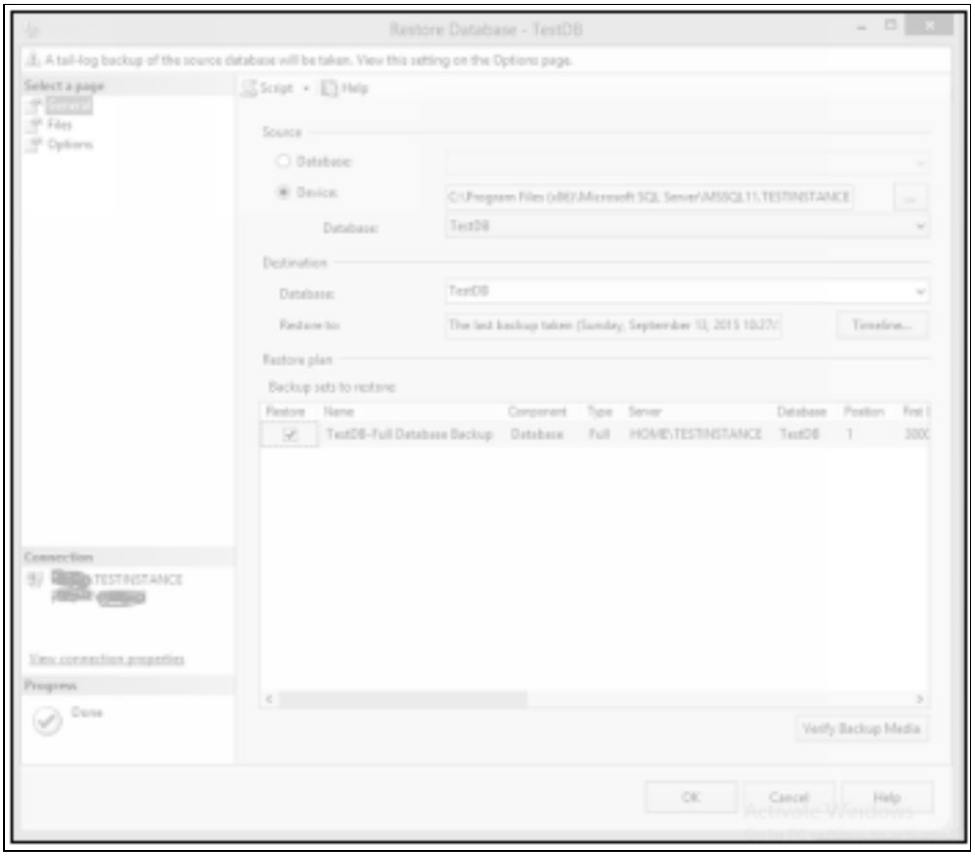
First connect to the "TESTINSTANCE" database then right-click on databases folder and click on "Restore database", as shown in the picture below.



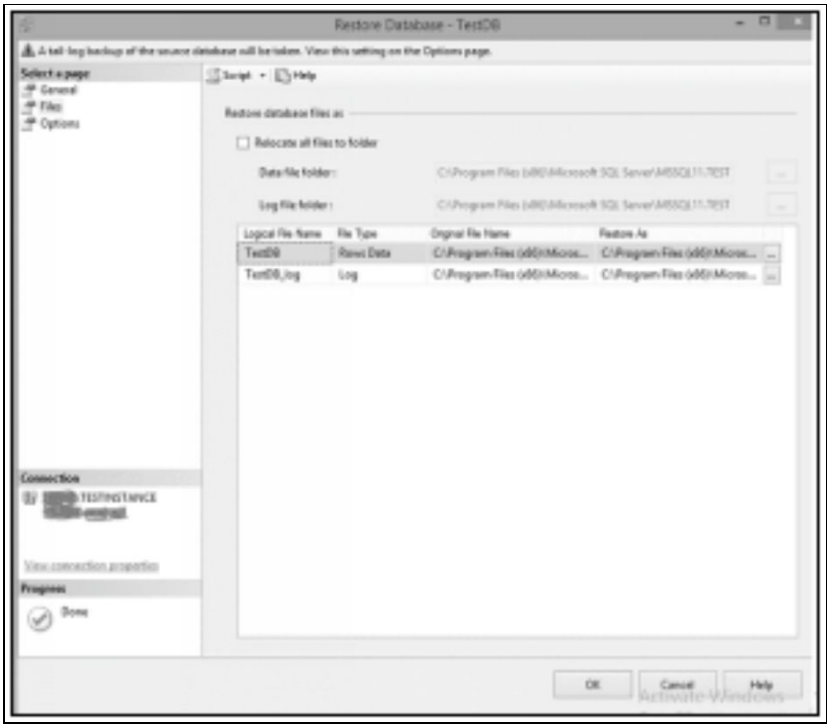
Now, to select the backup file as shown in the picture below, “select the device radio button and click on the ellipse”.



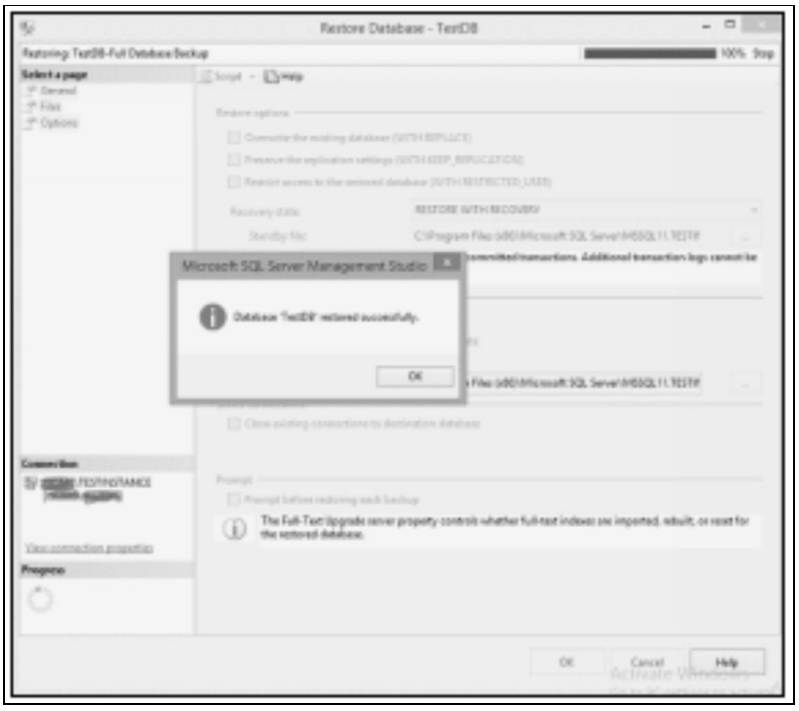
Next, click Ok and the window shown below will be displayed to you.



Now, from the “top-left corner of the screen select the Files option”, as shown in the picture below:



Lastly, select “Options” from the “top-left corner of your screen” and click “OK” to restore the “TestDB” database, as depicted in the picture below:


Restore Database - TestDB
Restoring TestDB-Full Database Backup
100%
Stop
Select a page
General
Files
Options
Script
Help
Restore options
Recovery state:
RESTORE WITH RECOVERY
Standby file:
Microsoft SQL Server Management Studio
Database 'TestDB' restored successfully.
OK
Close existing connections to destination database
Prompt
Prompt before restoring each backup
The Full-Text Upgrade server property controls whether full-text indexes are imported, rebuilt, or reset for the restored database.
Connection
View connection properties
Progress
OK
Cancel
Help

Chapter 5:

Database Security and Administration

MySQL has an integrated advanced access control and privilege system that enables generation of extensive access guidelines for user activities and efficiently prevent unauthorized users from getting access to the database system.

There are 2 phases in MySQL access control system for when a user is connected to the server:

Connection verification: Every user must have a valid username and password, that is connected to the “MySQL database server”. Moreover, the host used by the user to connect must be the same as the host in the "MySQL grant table".

Request verification: After a link has been effectively created for every query executed by the user, MySQL will verify if the user has required privileges to run that specific query. MySQL is capable of checking user privileges at database, table, and field level.

The MySQL installer will automatically generate a database called "mysql". The "mysql database" is comprised of 5 main grant tables. Using "GRANT" and "REVOKE" statements like, these tables can be indirectly manipulated.

User: includes columns for "user accounts" and "global privileges". MySQL either accepts or rejects a connection from the host using these user tables. A privilege given under the "user table" is applicable to all the databases on the “MySQL server”.

Database: comprises “database level” privileges. MySQL utilizes the “db_table” to assess the database that can be used by a user to access and to host the connection. A "database level" privilege in the “db_table” is applicable to the particular database and all the object available in that database, such as “tables, triggers, views, stored procedures”, and many more.

“Table_priv” and “columns_priv”: includes the "table level" and "column level" privileges. A privilege given in the "table priv" table is applicable to that particular table and its columns, on the other hand a privilege given in the "columns priv" table is applicable only to a particular column of the table.

“Procs_priv”: includes privileges for "stored functions" and "stored processes".

MySQL uses the tables listed above to regulate MySQL database server privileges. It is extremely essential to understand these tables before implementing your own dynamic access control system.

Creating User Accounts

You are able to indicate in MySQL not just which user is allowed to connect to the database server, but also which host the user can use to build that connection. As a result, for every user account a "username" and a "host name" in MySQL is generated and divided by the "@" character.

For instance, if an "admin user" is connected from a "localhost" to the “MySQL database server”, the user account will be named as “admin@localhost”. However, the “admin_user” is only allowed to connect to the “MySQL database server” using a "localhost" or a remote host like "mysqltutorial.org", which ensures that the MySQL database server has higher security.

Moreover, by merging the "username" and "host", various accounts with the same name can be configured and still possess the ability to connect from distinct hosts while beng given distinct priviliages, as needed.

In the "mysql database" all the user accounts are stored in the "user grant" table.

Using “MySQL CREATE USER” statement

The “CREATE USER” is utilized with the MySQL server to setup new user accounts, as shown in the syntax below:

"CREATE USER user_account IDENTIFIED BY password;"

In the syntax above, the "CREATE USER" clause is accompanied by the name of the user account in “username @ hostname” format.

In the "IDENTIFIED BY" clause, the user password would be indicated. It is important that the password is specified in plain text. Prior to the user account being saved in the "user" table, MySQL is required to encrypt the "user password".

For instance, the "CREATE USER" statement can be utilized as given below, for creating a new "user dbadmin", that is connected to the “MySQL database server” from "localhost" using user password as “SECRET”.

"CREATE USER dbadmin@localhost

IDENTIFIED BY 'SECRET';"

If you would like to check the “privileges of a user account”, you can run the syntax below:

"SHOW GRANTS FOR dbadmin@localhost;"

"+--+

| *Grants for dbadmin@localhost* |

+--+

| *GRANT USAGE ON *.* TO `dbadmin`@`localhost`* |

+--+

1 row in set (0.00 sec)"

The *. * in the "result set" above indicates that the "dbadmin user" account is allowed to log into the server only and does not have any other access privilege.

Bear in mind that the portion prior to the "dot (.)" is representing the database and the portion after the "dot (.)" is representing the table, for example, "database.table".

The "percentage (%)" wildcard can be used as shown in the syntax below for allowing a user to create a connection from any host:

"CREATE USER superadmin@'%'

IDENTIFIED BY 'secret';"

The "percentage (%)" wildcard will lead to identical result as when included in the “LIKE” operator, for example, to enable "mysqladmin user" account to link to the server from any “subdomain” of the "mysqltutorial.org" host, the "percentage (%)" wildcard can be used as shown in the syntax below:

"CREATE USER mysqladmin@'%.mysqltutorial.org'

IDENTIFIED by 'secret';"

It should also be noted here, that another wildcard “underscore (_)” can be used in the “CREATE USER” statement.

If the host name portion can be omitted from the user account, server will recognize it and enable the user to get connected from any host. For instance, the syntax below will generate a new remote user account that allows creation of a connection to the server from any host:

"CREATE USER remote;"

To view the privileges given to the “remote” account, you can use the syntax below:

"SHOW GRANTS FOR remote;"

"+----------------------------------+

| *Grants for remote@%* |

+----------------------------------+

| *GRANT USAGE ON *.* TO `remote`@`%`* |

+----------------------------------+

1 row in set (0.00 sec)"

It is necessary to remember that the single quote (' ') in the syntax above is particularly significant, if the user account has special characters like "underscore" or "percentage".

If you inadvertently cite the user account as "username@hostname", the server will create a user with the "username@hostname" name and enables the user to connect from any host that might not be as expected.

The syntax below, for instance, can be used to generate a new account "api@localhost" that can be connected to the “MySQL database server” from any host.

"CREATE USER 'api@localhost';"

"SHOW GRANTS FOR 'api@localhost';"

"+--+

| Grants for api@localhost@% |

+--+

*| GRANT USAGE ON *.* TO `api@localhost`@`%` |*

+--+

1 row in set (0.00 sec)"

If you accidentally generate a user that already exists in the database, then an error will be issued by MySQL. For instance, the syntax below can be used to generate a new user account called "remote":

"CREATE USER remote;"

The error message below will be displayed on your screen:

"ERROR 1396 (HY000): Operation CREATE USER failed for 'remote'@'%'"

It can be noted that the "CREATE USER" statement will only create a new user without any privileges. The "GRANT" statement can be utilized to give access privileges to the users.

Updating “USER PASSWORD”

Prior to altering a MySQL user account password, the concerns listed below should be taken into consideration:

The user account whose password you would like to be modified.

The applications that are being used with the user account for which you would like to modify the password. If the password is changed without altering the application connection string being used with that user account, then it would be not feasible for those applications to get connected with the database server.

MySQL offers a variety of statements that can be used to alter a user's password, such as "UPDATE", "SET PASSWORD", and "GRANT USAGE" statements.

Let’s explore some of these syntaxes!

Using “UPDATE” Statement

The “UPDATE” can be utilized to make updates to the “user” table of the “mysql” database. You must also run the "FLUSH PRIVILEGES" statement to refresh privileges from the "grant" table in the "mysql" database, by executing the "UPDATE" statement.

Assume that you would like to modify the password for the "dbadmin" user, which links from the "localhost" to the "dolphin". You can accomplish this by executing the query below:

"USE mysql;

UPDATE user

SET password = PASSWORD('dolphin')

WHERE user = 'dbadmin' AND

host = 'localhost';

FLUSH PRIVILEGES;"

Using “SET PASSWORD” Statement

For updating the user password, the "user@host" format is utilized. INow, imagine that you would like to modify the password for some other user's account, then you will be required to have the "UPDATE" privilege on your user account.

With the use of the "SET PASSOWORD" statement, the "FLUSH PRIVILEGES" statement is not required to be executed, in order to reload privileges to the "mysql" database from the "grant" tables.

The synatx below could be utilized to alter the "dbadmin" user account password, by executing the "SET PASSWORD" statement:

"SET PASSWORD FOR 'dbadmin'@'localhost' = PASSWORD('bigshark');"

Using “ALTER USER” Statement

Another method to alter the password for a user is by using the “ALTER USER” statement with the “IDENTIFIED BY” clause. For instance, the query below can be executed to change the password of the “dbadmin” user to “littlewhale”.

"ALTER USER dbadmin@localhost IDENTIFIED BY 'littlewhale';"

USEFUL TIP

If you need to change the password of the MySQL “root account”, then you would need to force quit the “MySQL database server” and restart the server without triggering the “grant table validation”.

Granting User Privileges

As a new user account is created, there are no access privileges afforded to the user by default. The "GRANT" statement must be used in order “to grant privileges to the user account”. The "GRANT" statement syntax is shown below:

"GRANT privilege, [privilege],.. ON privilege_level

TO user [IDENTIFIED BY password]

[REQUIRE tsl_option]

[WITH [GRANT_OPTION | resource_option]];"

In the syntax above, we start by specifying one or more privileges following the "GRANT" clause. Every privilege being granted to the

user account must be isolated using a "comma", in case you would like to give the user account more than one privilege at the same time. (The list of potential privileges that can be “granted to a user account” is given in the table below).

After that you must indicate the "privilege_level" that will determine the level at which the privileges are applied. The privilege level supported by MySQL are "global (*. *)", "database (database. *)", "table (database.table)" and "column" level. If you are using the "column" privilege level, single or multiple (a list of comma separated columns after each privilege) must be indicated.

Next you need to indicate the user that needs to be granted the privileges. If the indicated user can be found on the server, then the "GRANT" statement will modify its privilege. Or else, a new user account will be created by the "GRANT" statement. The "IDENTIFIED BY" clause is not mandatory and enables creation of a new password for the user.

Thereafter, it's indicated if the user needs to start a connection to the database via a “secure connection” such as "SSL", "X059", etc.

At last, the "WITH GRANT OPTION" clause is added which is not mandatory but enables granting and revoking the privileges of other user, that have been “granted to your own user account”. Moreover, the "WITH" clause can also be used to assign the resources from the MySQL database server, for example, putting a limit on the number of links or statements that can be used by the user per hour. In shared environments like "MySQL shared hosting", the "WITH" clause is extremely useful.

Note that the "GRANT OPTION" privilege and the privileges you are looking to grant to other users must already be configured to your user account, so that you are able to use the "GRANT" statement. If the read only system variable has been allowed, then execution of the "GRANT" statement requires the "SUPER" privilege.

PRIVILEGE	DESCRIPTION	LEVEL	LEVEL	LEVEL	LEVEL	LEVEL	LEVEL

		Global	Database	Table	Column	Procedure	Proxy
“ALL”	“Grant all privileges at specified access level except GRANT OPTION”						
“ALTER”	“Allow user to use of ALTER TABLE statement”	**Y**	**Y**	**Y**			
“ALTER ROUTINE”	“Allow user to alter or drop stored routine”	**Y**	**Y**			**Y**	
“CREATE”	“Allow user to create database and table”	**Y**	**Y**	**Y**			
“CREATE ROUTINE”	“Allow user to create stored routine”	**Y**	**Y**				
“CREATE TABLESPACE”	“Allow user to create, alter or drop table spaces and log file groups”	**Y**					
“CREATE TEMPORARY TABLES”	“Allow user to create temporary table by using CREATE TEMPORARY TABLE”	**Y**	**Y**				
“CREATE USER”	“Allow user to use the CREATE USER, DROP USER, RENAME USER, and REVOKE ALL PRIVILEGES statements”	**Y**					
“CREATE VIEW”	“Allow user to create or modify view”	**Y**	**Y**	**Y**			
“DELETE”	“Allow user to use DELETE”	**Y**	**Y**	**Y**			
“DROP”	“Allow user to drop database, table and view”	**Y**	**Y**	**Y**			
“EVENT”	“Enable use of events for the	**Y**	**Y**				

	Event Scheduler”						
“EXECUTE”	“Allow user to execute stored routines”	**Y**	**Y**	**Y**			
“FILE”	“Allow user to read any file in the database directory.”	**Y**					
“GRANT OPTION”	“Allow user to have privileges to grant or revoke privileges from other accounts.”	**Y**	**Y**	**Y**		**Y**	**Y**
“INDEX”	“Allow user to create or remove indexes.”	**Y**	**Y**	**Y**			
“INSERT”	“Allow user to use INSERT statement”	**Y**	**Y**	**Y**	**Y**		
“LOCK TABLES”	Allow user to use LOCK TABLES on tables for which you have the SELECT privilege”	**Y**	**Y**				
“PROCESS”	“Allow user to see all processes with SHOW PROCESSLIST statement”	**Y**					
“PROXY”	“Enable user proxying”						
REFERENCES	“Allow user to create foreign key”	**Y**	**Y**	**Y**	**Y**		
“RELOAD”	“Allow user to use FLUSH operations”	**Y**					
“REPLICATION CLIENT”	“Allow user to query to see where master or slave servers are”	**Y**					
“REPLICATION SAVE”	“Allow the user to use replicate	**Y**					

	slaves to read binary log events from the master"						
"SELECT"	"Allow user to use SELECT statement"	**Y**	**Y**	**Y**	**Y**		
"SHOW DATABASES"	"Allow user to show all databases"	**Y**					
"SHOW VIEW"	"Allow user to use SHOW CREATE VIEW statement"	**Y**	**Y**	**Y**			
"SHUTDOWN"	"Allow user to use mysqladmin shutdown command"	**Y**					
"SUPER"	"Allow user to use other administrative operations such as CHANGE MASTER TO, KILL, PURGE BINARY LOGS, SET GLOBAL, and mysqladmin command"	**Y**					
"TRIGGER"	"Allow user to use TRIGGER operations"	**Y**	**Y**	**Y**			
"UPDATE"	"Allow user to use UPDATE statement"	**Y**	**Y**	**Y**	**Y**		
"USAGE"	"Equivalent to no privileges"						

Example

More often than not, the "CREATE USER" statement will be used to first create a new user account and then the "GRANT" statement is used to assign the user privileges.

For instance, a new "super user" account can be created by the executing the "CREATE USER" statement given below:

"CREATE USER super@localhost IDENTIFIED BY 'dolphin';"

In order to check the privileges granted to the “super@localhost” user, the query below with “SHOW GRANTS” statement can be used.

"SHOW GRANTS FOR super@localhost;"

"+--+

| *Grants for super@localhost* |

+--+

| *GRANT USAGE ON *.* TO `super`@`localhost`* |

+--+

1 row in set (0.00 sec)"

Now, if you wanted to assign all privileges to the “super@localhost” user, the query below with “GRANT ALL” statement can be used.

*"GRANT ALL ON *.* TO 'super'@'localhost' WITH GRANT OPTION;"*

The "ON*. *" clause refers to all databases and items within those databases. The "WITH GRANT OPTION" enables "super@localhost" to assign privileges to other user accounts.

If the "SHOW GRANTS" statement is used again at this point then it can be seen that privileges of the "super@localhost's" user have been modified, as shown in the syntax and the “result set” below:

"SHOW GRANTS FOR super@localhost;"

"+--+

| *Grants for super@localhost* |

+--+

| *GRANT ALL PRIVILEGES ON *.* TO `super`@`localhost` WITH GRANT OPTION* |

+--+

1 row in set (0.00 sec)"

Now, assume that you want to create a new user account with all the server privileges in the “classicmodels sample database”. You can accomplish this by using the query below:

"CREATE USER auditor@localhost IDENTIFIED BY 'whale';

GRANT ALL ON classicmodels. TO auditor@localhost;"*

Using only one "GRANT" statement, various privileges can be granted to a user account. For instance, to generate a user account with the privilege of executing "SELECT", "INSERT" and "UPDATE" statements against the database "classicmodels", the query below can be used.

"CREATE USER rfc IDENTIFIED BY 'shark';

GRANT SELECT, UPDATE, DELETE ON classicmodels. TO rfc;"*

Revoking User Privileges

You will be using the "MySQL REVOKE" statement to revoke the privileges of any user account(s). MySQL enables withdrawal of one or more privileges or even all of the previously granted privileges of a user account.

The query below can be used to revoke particular privileges from a user account:

"REVOKE privilege_type [(column_list)]

[, priv_type [(column_list)]]...

ON [object_type] privilege_level

FROM user [, user]..."

In the syntax above, we start by specifying a list of privileges that need to be revoked from a user account next to the "REVOKE" keyword. YAs you might recall when listing multiple privileges in a statement they must be separated by commas. Then we indicate the "privilege level" at which the "ON" clause will be revoking these privileges. Lastly we indicate the user account whose privileges will be revoked in the "FROM" clause.

Bear in mind that your own user account must have the "GRANT OPTION" privilege as well as the privileges that you want to revoke from other user accounts.

You will be using the "REVOKE" statement as shown in the syntax below, if you are looking to withdraw all privileges of a user account:

"REVOKE ALL PRIVILEGES, GRANT OPTION FROM user [, user] …"

It is important to remember that you are required to have the "CREATE USER" or the "UPDATE" privilege at "global level" for the mysql database, to be able to execute the "REVOKE ALL" statement.

You will be using the "REVOKE PROXY" clause as shown in the query below, in order to revoke proxy users:

"REVOKE PROXY ON user FROM user [, user]..."

A proxy user can be defined as "a valid user in MySQL who can impersonate another user". As a result, the proxy user is able to attain all the privileges granted to the user that it is impersonating.

The best practice dictates that you should first check what privileges have been assigned to the user by executing the syntax below with "SHOW GRANTS" statement, prior to withdrawing the user's privileges:

"SHOW GRANTS FOR user;"

Example

Assume that there is a user named "rfc" with "SELECT", "UPDATE" and "DELETE" privileges on the "classicmodels sample database" and you would like to revoke the "UPDATE" and "DELETE" privileges from the "rfc" user. To accomplish this, you can execute the queries below.

To start with we will check the user privileges using the "SHOW GRANTS" statement below:

"SHOW GRANTS FOR rfc;"

"GRANT SELECT, UPDATE, DELETE ON 'classicmodels'. TO 'rfc'@'%'"*

At this point, the “UPDATE” and “REVOKE” privileges can be revoked from the “rfc” user, using the query below:

"REVOKE UPDATE, DELETE ON classicmodels.* FROM rfc;"

Next, the privileges of the “rfc” user can be checked with the use of “SHOW GRANTS” statement.

"SHOW GRANTS FOR 'rfc'@'localhost';"

"GRANT SELECT ON 'classicmodels'.* TO 'rfc'@'%'"

Now, if you wanted to revoke all the privileges from the "rfc" user, you can use the query below:

"REVOKE ALL PRIVILEGES, GRANT OPTION FROM rfc;"

To verify that all the privileges from the "rfc" user have been revoked, you will need to use the query below:

"SHOW GRANTS FOR rfc;"

*"GRANT USAGE ON *.* TO 'rfc'@'%'"*

Remember, as mentioned in the privileges description table earlier in this book, the “USAGE” privilege simply means that the user has no privileges in the server.

Resulting impact of the “REVOKE” query

The impact of "MySQL REVOKE" statement relies primarily on the level of privilege granted to the user account, as explained below:

The modifications made to the "global privileges" will only take effect once the user has connected to the MySQL server in a successive session, post the successful execution of the "REVOKE" query. The modifications will not be applicable to all other users connected to the server, while the "REVOKE" statement is being executed.

The modifications made to the database privileges are only applicable once a "USE" statement has been executed after the execution of the "REVOKE" query.

The "table and column privilege" modifications will be applicable to all the queries executed, after the modifications have been rendered with the "REVOKE" statement.

Conclusion

Thank you for making it through to the end of ***The ultimate beginners guide to learn SQL programming:*** *a smart step by step guide to learn SQL database and server. How to building an advanced and elite level in SQl*, let's hope it was informative and able to provide you with all of the tools you need to achieve your goals whatever they may be.

The next step is to make the best use of your new found wisdom and learning of the SQL programming language, that can position you to enter the world of systems and data analysis. It is very important that you follow the instructions in this book and install the free and open MySQL user interface on your operating system, so you can get hands on practice and be able to create not only correct but efficient SQL queries to succeed at work or during job interviews. As is the case with most of the learnings in life repeated practice is the key to achieve perfection. So let this book be your guiding beacon through the journey of learning a programming language for relational database management systems. To make the best use of this book, as you read and understand all the concepts and SQL query structures, try to come up with your own names and labels to be used within the presented examples and verify the result set up obtained with the one given in this book.

Finally, if you found this book useful in any way, a good review on Amazon is always appreciated!

Learn Java Programming

A Definitive Crash Course for Beginners to Learn Java Fast. Secrets, Tips, and Tricks to Programming with Java Code and the Fundamentals to Creating your First Program


Java™

Introduction To Java

“Write once, run everywhere.”

Java is a language. Like any language, it requires time to be fluent in. While learning a language like spanish or french allows us to interact with people from that country, programming languages allow us to interact with computers by giving us the ability to write instructions that computers can understand and execute.

Released by Sun Microsystems in 1995, Java is famous for its portability, security, and robustness. It remains one of the most favorable programming languages over two decades later.

The *Java Virtual Machine (JVM)* is one of the main virtues that is responsible for this. The JVM is what allows Java code to be run on most any operating system (OS). The “Write once, run everywhere” slogan was used by Sun when Java was released and by the end of this book, we hope you’ll understand why.

Besides being a programming language, Java also acts as a **Platform**: a hardware or software environments that executes programs. Windows, Linux, Solaris, and Mac OS are platforms that you’re most likely already familiar with. Platforms are essentially a mixture of an operating system and the underlying hardware. The Java platform is unique in the act that it is software only. this is how Java achieves its versatility. Because Java is software-only it’s able to be run over any other platform that is hardware based.

The Java platform has two components: The Java Application Programming Interface and the Java Virtual Machine. The API is made up of packages. Packages are libraries containing classes and interfaces. These packages are collected in the API and provide many useful abilities.

Chapter 1:
Getting Ready For Java

What is JDK?

The **Java Development Kit (JDK)** is used along with the **Java Virtual Machine (JVM)** and the **Java Runtime Environment (JRE)** to develop Java programs. These technologies are distinct yet completely reliant upon each other and it's important to understand the difference between the three. The Java Virtual Machine is what executes programs. The Java Runtime Environment is responsible for creating the Java Virtual Machine. Finally, the Java Development Kit is what programmers actually interact with in order to develop code that is executed by the JVM and JRE.

What is NetBeans?

NetBeans is an Integrated Development Environment (IDE). IDEs allow programmers the ease of consolidating the various aspects of developing code. IDEs are useful in the ways that they increase productivity through consolidating the actions of editing source code, developing executable, and debugging.

Installing JDK and NetBeans

In order to start writing code you'll need to do two things: download the Java Development Kit and the NetBeans IDE. Both of these can be found using a quick google search. Download the most recent versions available (at the time of writing it should be 12.0.2 for the JDK and 8.0.2 for NetBeans.)

Chapter 2:
HelloWorld.java

We begin with the universal introduction to all programming languages, the **Hello World program**. Speak with any person even remotely involved in computer programming and chances are they will be all too familiar with the “Hello World.” The purpose of this program is to introduce you to the general structure of a Java program, illustrate the basic syntax of a programming language, and will also begin our discussion on **class** and **method**.

HelloWorld.java

```
public class HelloWorld {
  public static void main(String[] args) {
    System.out.println("Hello World!");
  }
}
```

Take a moment to look over this code. The purpose of this code is simple: When ran, our code will print the words “Hello World!” to our screen. The syntax may seem arbitrary and alien to you right now, but trust that every word, bracket, parenthesis, period, and semicolon has a purpose. In no time at all you’ll have the confidence and ability to fully understand this program.

You’ll see that the program filename ends with the **.java** extension. This is the case for all program files written in Java. Within the HelloWorld.java file we have saved our code for the HelloWorld program.

Let’s take a look at the first line of our code.

```
public class HelloWorld {
  public static void main(String[] args) {
    System.out.println("Hello World!");
```

```
  }
}
```

There are a few things to notice within this first line. First of all, we have established a **class**, a template used to create objects and define methods (do not worry too much about these terms right now, they will be explored further in later chapters). The name of our class is **HelloWorld** (pay special attention to the fact that each word is capitalized). Each file we make will have a primary class that has the same name as the file. In this case, our file is **HelloWorld.java**, so we've made our primary class **HelloWorld** as well. Secondly, we have defined the domain of this class using curly braces {}. Now, all syntax that we put within these braces will become part the HelloWorld class.

Our next line of code lies within the HelloWorld class.

```
public class HelloWorld {
  public static void main(String[] args) {
    System.out.println("Hello World!");
  }
}
```

The idea here is the establishment of the **main() method**. In Java, methods contain a collection of statements that perform specific tasks and return responses when that method is invoked. The main() method is the point at which java programs begin execution and because of this every Java program you write will contain a main() method. Similarly to the HelloWorld class, we have used curly braces {} to define the domain of the main() method so any syntax we put within these braces will belong to it. Inside the method we will finally provide statements that will be executed by the computer in order to perform tasks. The terms *public, static*, and *void* are syntax that will be discussed later on, ignore them for now. *String[] args* is simply a placeholder for information and will also be discussed later.

These terms are necessary for the program to run but beyond our ability to understand at this point.

As was stated earlier, the entire point of this program is to print something to our screen. So far we've set up our code to accept instructions and now that we have established a class and a method within that class we can begin to specify the tasks we want our computer to execute. With this in mind, we proceed to the final line.

```
public class HelloWorld {
  public static void main(String[] args) {
    System.out.println("Hello World!");
  }
}
```

This line is the executable statement that tells our computer to perform the action we want. The **println** term is short for "print line", and System.out.println is what we'll use whenever we want our program to print something (in this case Hello World!) to the screen.

Comments

Obviously, writing code is our main purpose while programming. The code is what instructs the computer to perform tasks which is our ultimate goal. However, in any useful code there will be **comments** in addition to the syntax of our code that are instrumental to describing the function of our code and clarifying any aspects of a program that aren't immediately obvious to someone who's seeing it for the first time. It may seem counterintuitive, but programming is a highly cooperative pursuit, and effective commenting is crucial to making code useful to other programmers.

Thankfully, commenting code is easy. When comments are short we use the single line syntax: **//**, followed by what you want to say. When comments are longer, we use the multi-line syntax: **/* */**.

```
    /*
```

This is a multiline comment
You can say as much as you want
Within the multiline syntax.
***/**

Chapter 3:
The World of Variables

We've discussed the basic needs of a Java program, classes and methods. We know methods contain tasks that are executed by a program in order to tell our computers what to do, but what kind of tasks can we execute? We now dive into the kind of things you'll be putting in your methods.

Let's say we're asked to develop a program that will help users find new jobs. In order to do this we will at some point need to know some basic facts about the person trying to find a job. We'll need the users name, their age, salary, date of birth, employment status, etc. All of these pieces of information will need to be stored somewhere within our program. This is the role of **variables**. In Java, variables are named locations in memory, essentially containers that programmers use to store values so that these values can be called upon later in the program.

Variables give context and meaning to the data that is being stored. Without a unit associated with it, the number 42 on its own could refer to infinitely many things. It could be someone's age, a weight in pounds or kilograms, it could be the number or orders placed, who knows. Without a specified variable we'll never know why the number 42 is important. Examples of names for variables could either be *age*, *weightInPounds*, or *numOrdersPlaced*. When naming variables it's common practice to leave no spaces between words and to leave the first word lowercase while all following words are capitalized (this is referred to as *camelCase*). You can technically name your variables almost anything you want, however, the more obvious of a name you choose the easier it will be to remember what the variable stores and its purpose.

In Java, we need to specify the type of information we're storing. **Primitive data types** are built-in to the Java system and are used to tell the computer what type of information is being held by a variable.

Before a variable can be referenced in our code we need to declare the variable by specifying its name and data type.

```
// datatype variableName
```

int age;

double salaryRequirement;

boolean isEmployed;

Listed above are three examples of data types: int, double, and boolean**.** The names of these variables are *age, salaryRequirement,* and *isEmployed.* In this state, these variables do not have any associated value. They're more or less empty containers waiting to be filled. To assign a variable to a value, or fill the container so to speak, we use the assignment operator "=":

int age = 42;

It's common to assign the value in the same line that we declare a variable. In this line, we've essentially created a variable (container) of type *int*, named *age*, and filled the variable with the value 42. Now, whenever we refer to this variable *age* in our program, it will return the value 42.

Primitive Data Types

At this point, we know that variables are used to store information and are declared by specifying data type and name. Naming a variable is relatively straightforward, but what about specifying data type? What kind of data types are there and how do we know which one to choose? In Java, primitive data types are made of 8 different types: int, double, boolean, char, byte, short, long, and float.

int

The first type of data we will store is the whole number. Whole numbers are very common in programming. You often see them used to store ages, or maximum size, or the number of times a code has been run. In Java, whole numbers are stored in the **int** primitive data type. Ints hold positive numbers, negative numbers, and zero. They do not store fractions or decimals. The int data type allows

values between -2,147,483,648 and 2,147,483,647. To declare a variable of type int, we use the int keyword:

int sizeOfHousehold = 5;

double

Whole numbers do not accomplish what we need for every program. What if we wanted to store the price of something? We would need a decimal point. What if we wanted to store the world's population? That number would be larger than the int type can hold. The **double** primitive data type can help. Double can hold decimals as well as very large and very small numbers. The maximum value is 1.797,693,134,862,315,7 E+308, which is approximately 17 followed by 307 zeros. The minimum value is 4.9 E-324, which is 324 decimal places!

To declare a variable of type double, we use the double keyword in the declaration:

double price = 7.39;*boolean*

Often our programs face questions that can only be answered with yes or no. Is the oven on? Is the light green? Did I eat breakfast? These questions are answered with a **boolean**, a type that contains one of two values: true or false.

We declare boolean variables by using the keyword boolean before the variable name:

boolean isItColdOutside = true;

Later on we will see how booleans are incredibly important for navigating through programs and decision making.

char

How do we answer questions like: What grade did you get on the test? What letter does your name start with? The **char** data type can hold any character, like a letter, space, or punctuation mark. It must be surrounded by single quotes.

char testGrade = 'B';

byte

A **Byte** has a minimum value of -128 and a maximum of 127. Bytes are helpful in big arrays to save time or in circumstances where memory saving counts. Instead of int, bytes can be used where boundaries help explain a code. Variable variety can be used as a paperwork type.

short

A **Short** has a minimum of -32,768 and a max of 32.767. Shorts are similar to bytes in the way that they can be used to save data in situations where memory matters.

long

A **Long** has a min of-2 up to the 63rd and a minus of 2 up to the 63rd. Alternatively, to depict values with a minimum of 0 and a maximum of 2 raised to 64th minus 1, you can use longs. Also the Long category has helpful techniques such as comparingUnsigned, dividingUnsigned, etc. These courses promote lengthy activities of arithmetic.

float

A **Float** has a range of values that goes past the scope of this book. use floats if you need to save memory (similar to byte and short).

Reference Data Types

Aside from primitive data types, there are *reference data types*. We will dive more into these later one but, suffice to say ,they can also be used to declare variables. One important thing is that primitive data types, like the one discussed above, store and refer to primitive values (numbers, decimals, etc). Reference data types, on the other hand, merely refer to an address where data is stored. Don’t think too deeply on this now, like was said, this concept will be explained later on.

Chapter 4:
Strings and Arrays

Strings

So far, we have learned primitive data types, which are the simplest types of data with no built-in behavior. Our programs will also use **Strings**, which are sequences of characters defined as *objects* instead of primitives. Objects have built-in behavior. Strings are widely used in Java programming. We declare strings in the same way we declare any variable; with a type and a name.

```
String example = “Hello World”;
System.out.println(example);
```

Here, we have declared a string with the name “greeting”, and assigned it the value Hello World. Now instead of typing Hello World everytime you want to print it, you can simply reference the greeting string. When printed using System.out.println(), this string will print Hello World to our screens.

String Methods

Because of how useful strings are for storing information, Java has dedicated an entire class to them. This class is crucial for Java programming as it provides an extensive library of useful methods to help us perform operations on strings that we will use to manipulate data. We’ll discuss seven of the most useful string methods available: length(), concat(), equals(), indexof(), substring(), toUpperCase(), and toLowerCase().

length()

In Java, the **length()** string method returns, as you may have guessed, the length of a string. If we have a string named *example*, then running str.length () would return a range of values indicating the amount of characters included within the *example* string.

Length() returns an int value. Examine the following program file, StrCounter.java:

StrCounter.java

```
private class static StrCounter {
    private static void main(String[] args) {
        String example = "this is a string.";
        int exampleLength = example.length();
        System.out.println(exampleLength);
}
}
```

If we were to run this program, our code would return a value of 17 to our terminal screen. Notice that spaces and punctuation are considered when using this method. We declared an int variable name strLength and within that variable we placed the str.length() method, which returns an int when executed. We then used System.out.println() to print the resulting integer value contained within our variable strLength. For those familiar with the social media app Twitter, length() is how they determine if a tweet falls within the 280 character limit that has been set for each tweet.

concat()

Concatenation is the act of joining two strings together. The **concat()** method does just this. Say we have two strings, str1 and str2. If we ran str1.concat (str2), our program would return the two strings joined together. Concat() returns another string value. See the following example:

StrConcat.java

```
private class StrConcat{
    private static void main(String[] args){
    String ex1 = "this is ";
    String ex2 = "a string.";
```

```
        String ex3 = ex1.concat(ex2);
        System.out.println(ex3);
        }
}
```

Because concat() returns a string value, we initialized a new string named str3 in order to contain the ex1.concat(ex2) return value. By printing ex3 using System.out.println() and running the program, the resulting statement “this is a string.” is printed to our screen.

equals()

In some cases it can be useful to compare the values of two strings. The **equals()** method gives us the ability to do this. The equals() method uses the memory address of two strings to compare the two strings. Equals() returns a boolean value. If the characters of those addresses do not match, it returns false. If all characters are matched, it returns true.

StrEqual.java

```
private class StrEquals{
        private static void main(String[] args){
        String flavor1 = "mango";
        String flavor2 = "peach";
        booleancompareFlavor = flavor1.equals(flavor2);
        System.out.println(compareFlavor);
        }
}
```

In this case, flavor1.equals(flavor2) will return a boolean value. Because the statement “mango” which is held in flavor1 obviously does not match the statement “peach” held in flavor2, flavor1.equals(flavor2) will return false. Since this result is now stored in the boolean compareFlavor, when we use System.out.println(compareFlavor) the result false will be printed to our terminal.

On a side note, there's also an *equalsIgnoreCase()* method that will also compare two strings. However, capitalization is not taken into consideration when this method is run.

indexOf()

We can use the **indexOf()** technique on a string if we want to understand the index of the first event of a element in a string. IndexOf() returns an int value, the position within the string of the character we are looking for.

strIndex.java

```
private class strIndex{
private static void main(String[] args){
String ex1 = "ABCDEFGHIJKLMNOP";
int ex1Count = ex1.indexOf("B");
System.out.println(ex1Count);
{
{
```

Here we established a String named *ex1* containing every letter in the alphabet up to P. We then established an int variable names ex1Count containing the indexOf() method. We do this because indexOf() returns an int value. The int this code would return be an output of 1. This may seem counterintuitive, but it's important to remember that *indices in Java begin at 0, not 1.* So while 'B' is technically the second letter of the alphabet, it will returned as 1 because in this case 'A' is located at position 0. In this way, B is located at position 1, C is located at position 2, D is located at position 3, E is located at position 4, and so on.

Suppose we want to understand the index of a whole substring's first incidence (a multi-character aspect of string we've already defined). If the entire alphabet constitutes a string, the letters "ABC" would constitute a substring of the alphabet string). The indexOf() instance method can also return the position where a substring begins (the

position of the first character in the substring). Lets alter one of the lines from the previous block of code:

```
String ex1 = “ABCDEFGHIJKLMNOP”;
int ex1Count = ex1.indexOf(“BCD”);
System.out.println(ex1Count);
```

The return value of this code would still be 1. What happened? When we run our program it won’t return the unique location of each character we specify, it’ll simply return the value of the first character in the substring BCD, which is B. As we know, B is in the 1 location. If indexOf() doesn’t find what it’s looking for it will simply return -1.

Consider the following lines:

```
String ex1 = “ABCDEFGHIJKLMNOP”;
int ex1Count1 = ex1.indexOf(“BCD”);
int ex1Count2 = ex1.indexOf(“BDF”);
```

It’s worth mentioning that if you’re trying to index a substring the characters contained within must be consecutive. One of these statements will return -1. Can you tell which? The top line will run how we expect and return the value 1. The bottom line is incorrect. Even though the bottom line contains letters that are present in the alphabet, it will return -1 because there’s no substring made up of *BDF* present in that order within the alphabet string.

charAt()

The **charAt()** technique yields a character, specifically the character with a sequence at a particular index.. It could be argued charAt() is the antithesis of indexOf(). IndexOf() takes in a char as an argument and returns an int value equal to location that character. charAt() on the other hand takes an int as an argument and returns the char present at that location. The charAt() method returns a char value.

strChar.java

```
private static class strChar{
private static void main(String[] args){
```

```
String ex1 = “apple”;
char ex1Char = ex1.indexOf(“3”);
System.out.println(ex1Count);
{
{
```

Executing this block of code would return the char value “l”. Remember, indices start at 0, meaning if we had entered the number 5 as our argument in str.indexOf an error would’ve been returned as that would be out of range (a=0 p=1 p=2 l=3 e=4).

substring()

Think back to when we searched for a group of letters within our alphabet string during the discussion on char(). For any number of reasons we may want to access a certain portion of a previously defined string. We do this by utilizing the **substring()** method. This method allows us to access full sections of other strings that have been established in our code instead of just one character at a time. See below:

strSubstring.java

```
private class yep{
private static void main(String[] args){
String ex1 = “Alexander the Great was great.”;
String ex1Substring = ex1.substring(3);
System.out.println(ex1Substring);
{
{
```

Upon printing, the output for this code would be “xander the Great was great.”, as x is the third character in the string *ex1.* Substring() will find character located at the indicate location then output this character plus any others that follow until it hits the end of line. Obviously if you’re just looking for a specific character the charAt()

would probably be an easier method to achieve this. However, substring() is special in that you cut a choose entire subsections of a string at will. Look at the changes made in the above code:

```
private class exampleSubstring{
    private static void main(String[] args){
    String ex1 = “Alexander the Great was great.”;
    String ex1Substring = ex1.substring(3,7);
    System.out.println(ex1Substring);
    {
    {
```

By adding a comma followed by another number, we can isolate a single aspect of a string. Now instead of starting at the third character and outputting everything that follows after it, we've made it so substring() will only output the character that exist between the third and seventh position. So by running this updated code System.out.println(ex1Substring) will now output “xand”. Very useful for recycling code.

toUpperCase/toLower Case

Say we have two separate strings.

String upper = “UPPER”

String lower = “upper”

Now let us say, for whatever reason, we need to compare both of these string using the equals() method and we need the result to be true. While the two terms say the same word, one is fully uppercase while the other is written completely in lowercase. Obviously, in this state upper.equal(lower) is going to return false. For the more intrepid of you, you'd probably consider using the equalsIgnoreCase() method that was mentioned during our discussion on the equals() method. While this would be effective, there's a second method that may serve our needs just as well.

There are two methods we'll discuss here: **toUpperCase()** and **toLowerCase()**. The purpose of these methods is more or less intuitive. The toUpperCase() method will accept a statement and return the same statement in all uppercase. See below:

```
private class ex1ChangeCase{
    private static void main(String[] args){
    String example1 = "up";
        String upperCase = example1.toUpperCase();
    System.out.println(upperCase);
    {
    {
```

In this example we stored the word "up" in a string named *example1*. We then used *example1* as the argument in the toUpperCase() method that we stored in a separate string we named *upperCase*. When we print *upperCase* using System.out.println() the return value should be the word "UP" printed in all caps.

It should be no surprise then that if *toUpperCase()* changes all the characters in a statement to uppercase letters, the *toLowerCase()* will do the opposite.

```
private class strChangeCase{
    private static void main(String[] args){
String example2 = "DOWN";
String lowerCase = example2.toLowerCase()
    System.out.println(lowerCase);
    {
    {
```

As you might expect, the concept illustrated by this example is the same as the one used to explain the *toUpperCase()* method. We stored the word "DOWN" (all uppercase) in the string we name

example2. We then created another string named *lowerCase*. We used the *toLowerCase()* method on the *example2* string and stored the result within the string *lowerCase*. We then used System.out.println() to print the value stored within *lowerCase* which is the word “down” printed in all lowercase.

Arrays

In the context of Java, an **array** is an extremely important object. Arrays serve as containers used to hold a constant number of values of the same data type. When we declare an array, memory space is allocated for a specified number of values of a specified type. At the time of creation, the length of an array must be declared and cannot be changed.

Imagine that we’re using a program to keep track of the prices of different clothing items we want to buy. We would want a list of the prices and a list of the items they correspond to. Establishing an array is similar to establishing a typical variable.

double priceOfClothes;

double[] priceOfClothes;

Take a look at the two lines above. The first line declares a variable of type double with the name *priceOfClothes*. We’ve declared variables of this type before in chapter 3. The next line, on the other hand, is slightly different. Notice the pair of brackets [] next to the data type double. The presence of these brackets declares an array, meaning that we have declared an array named *pricesOfClothes* that will contain values of the double data type.

Now that our array is declared we can do two things: assign values to it directly or leave it empty to be filled at a later time. How do we go about doing this? To fill your array directly is easy, simply list the values you want your array to contain.

double[] pricesOfClothes = {32.29, 15.78, 9.01, 100.28};

Our *pricesOfClothes* array now contains four elements. One of the benefits of arrays is that any element within that array, as long as

your know its name the position of the element you're looking for, can be recalled and used elsewhere in your code. (Note: Arrays are similar to indexes in the way that they begin at 0. Meaning that while there are four elements in the *pricesOfClothes* array, the final element "100.28" is technically at position three.) Now that we know what values are in our array and their location we can recall any of these values whenever we want.

If you know you'll need an array at some point later on but do not currently have or know the values you need to fill the array, you can simply declare an empty array. If you do this, you still need to establish the number of elements your array can hold. Otherwise, your computer won't be able to allocate the necessary memory needed by the array to contain information and when it comes time to fill the array there'll be nowhere for the information to go. Not good.

Let's keep going with the items of clothing example. At some point the customer will want to purchase some items of clothing, and those items will have prices we'll want to enter into our array. Until we know what items of clothing the customer wants, however, we'll have no idea what price values to enter into our array. For the sake of this example, we're going to assume we know the customer is planning on buying four items. Here is how we'd establish our empty array:

double[] pricesOfClothes = new double[4];

Now we have an array that we know will contain the data type double and will contain four elements to be assigned at another time. How do we assign elements? Simply type the name of the array, the element number you want to assign within brackets, and use the assignment operator to indicate what you want to put in that position.

double[] pricesOfClothes = new double[4];

pricesOfClothes[0] = 32.29;

pricesOfClothes[1] = 15.78;

pricesOfClothes[2] = 9.01;

pricesOfClothes[3] = 100.28;

Now we have assigned values the four open spots in our array. Pay close attention to the fact that the first value assigned was in position [0] and the final value assigned was in position [3]. Despite the fact that the final value is at position [3] all together there are still *four* elements, which is why we designated four positions when we declared our empty array.

It's important to mention that not only can we declare arrays using primitive types, but also strings. The method for doing this is the same:

String[] strArr = {"I love programming", "It's the best"};

Now that we have an array declared and initialized, we want to be able to access the values within. We use square brackets, [ ], and the array name to access data at a certain index:

int[] evenNumbers = {0, 2, 4, 6, 8};

System.out.println(evenNumbers[1]);

Here, we've used System.out.println() to print the value located at index position one within the *evenNumbers* array. In this case, the number "2" would be printed to the screen (remember: Array indexes start at 0).

Printing Arrays

In all the previous examples concerning arrays we never actually listed the set of code needed to print an array. While we can recall specific elements, printing entire arrays is more difficult. Unlike with strings and primitive variables, we need an additional method to convert an array to a string that can be printed using the System.out.println() method. In order to get a more descriptive print out of an array we'll need the **toString() method** that is provided within the Array *package* in Java. Without going to much into detail, packages in Java contain useful classes, interfaces, and subpackages that provide tools that are crucial to writing code. Let's look at the array we established earlier in this chapter:

double[] pricesOfClothes = {32.29, 15.78, 9.01, 100.28};

If we simply try to print the contents of this array using System.out.println(pricesOfClothes) we'll get gibberish printed to our screen as a result. System.out.println() is unable to perform all the tasks needed to interpret and return what we're asking for. In order to give it the ability to do so we need to import the toString() method. In order to do that we'll need to utilize one of Java's built-in packages. In this case, we'll be using **java.util.Arrays.** The first term" *java*" is the top level package, "*util*" is the subpackage, and "*Arrays*" is a class that is present within the util subpackage. We'll import this package before we even establish a class or main() method in our code. When we import a package we make all the classes and other properties held within available to us.

```
import java.util.Arrays
private class prinArr {
  private static void main(String[] args){
      double[] pricesOfClothes = {32.29, 15.78, 9.01, 100.28};
      String printOut = Arrays.toString(pricesOfClothes);
          System.out.println(printOut);
  }
}
```

So what have we done here? We used the term *import* to open the java.util.Arrays package. Once again, we import packages before we write any code or establish any methods or classes. This way, everything we've imported will be available throughout our entire code. Why did we need to import this package? So we could use the **Arrays.toString()** method. We initialized and defined our array named *pricesOfClothes* but in order to change it to a string that we can print using System.out.println(), we need to create a separate string (the string we've named *printOut*). We then filled *printOut* with the return value we get from putting the *pricesOfClothes* array into the Arrays.ToString() method, which obviously returns a string. The

printOut string contains all the same information as the *pricesOfClothes* array, but is now in a form that can be printed using System.out.println(). When we print the *printOut* string to our screen it should look like [32.29, 15.78, 9.01, 100.28].

Finding Array Length

What if we have an array storing all the usernames for our program, and we want to quickly see how many users we have? In this case, we will use the **.length() method**. To get the length of an array, we can access the length field of the array object:

```
String[] menuItems = new String[5];
int menuItemsLength = menuItems.length;
System.out.println(menuItemsLength);
```

This command would print “5”. We defined an empty array with five slots that can hold the string type. We then defined a new variable of type int and named it *menuItemsLength*. We filled this variable with the output from the .length method that we used on the *menuItems* array, which returns an int. We then printed this value using System.out.println(). Notice that despite the fact that we never filled the *menuItems* array with any values, we can still find the length because even empty arrays must be assigned a length when they’re defined.

Copying Arrays

When you want to copy data from one array into another use the **arraycopy() method**.

```
private static void arraycopy(
Object source, int sourcePosition,
Object destination, int destinationPosition, int length
)
```

Here we have two objects that coordinates which array is copied from and which array is copied to. The three ints are used to specify

the index within the source array, the start index within the destination array, and the number of array elements to be copied.

The following program, copyDemo, declares an array of chars, spelling the word "deoxygenated." A subsequence of array components is copied using the arraycopy method:

```
class copyDemo {
   private static void main(String[] args) {
      char[] copyFrom = { 'd', 'e', 'o', 'x', 'y', 'g', 'e',
            'n', 'a', 't', 'e', 'd'};
      char[] copyTo = new char[9];
      System.arraycopy(copyFrom, 2, copyTo, 0, 9);
      System.out.println(new String(copyTo));
   }
}
```

The output from this program is “oxygenated”.

Array Manipulations

Manipulating arrays is one of the most useful and powerful aspects of Java programming. Many Java methods are used for doing just this. An example is the arraycopy method that is held within the System class. This method is useful in the way that it runs through elements of source arrays and places each element within the destination array. Java makes it easy to perform this.

Other convenient methods provided by Java that manipulate arrays are stored within the java.util.Arrays class. Within this class are common tasks like copying, sorting and searching arrays. The **copyOfRange** method (found within the java.util.Arrays class) can be used to modify the previous example. In this case, using the copyOfRange method doesn’t require creating the destination array prior to calling the method. This is because the destination is already going to be returned by the method:

```
class copyDemo {
   private static void main(String[] args) {
      char[] copyFrom = { 'd', 'e', 'o', 'x', 'y', 'g', 'e',
            'n', 'a', 't', 'e', 'd'};
      char[] copyTo = java.util.Arrays.copyOfRange(copyFrom, 2,
11);
      System.out.println(new String(copyTo));
   }
}
```

We use fewer lines of code to get the same output (oxygenated). The fact that the second parameter of the copyOfRange method is the initial index of the array to copied is to be noted. That, as well as the fact that the third parameter is the final index of the array to be copied.

The java.util.Arrays class provides many useful operations. The following are a couple examples:

Using the binarySearch() method will return a specific value that is the index at which the value is positioned.

Using the equals() method to compare two arrays.

Using the fill() method to place a certain value at each position.

Using the sort() method or parallelSort() method, we can sort an array into ascending order.

Multidimensional Arrays

By using various bracket sets, you can also define an array of variables (also recognized as a multidimensional array). A specified amount of attributes will then access each component. Multidimensional arrays in Java are arrays whose elements are arrays as well. As a consequence, the duration of the lines can differ, as shown in the Demo program below:

```
class Demo {
```

```
    private static void main(String[] args) {
        String[][] e1 = {
            {"0 ", "1 ", "2"},
            {“Yup", "Nope"}
        };
        // Prints Yup 0
        System.out.println(names[1][0] + names[0][0]);
        // Prints Nope 2
        System.out.println(names[1][1] + names[0][2]);
    }
}
```

The output from this program is Yup 0 and Nope 2.

ArrayLists

Arrays are limited by the fact that once they’re created they have a fixed size. What does this mean? It means they can’t have elements added or removed. Problem is, we have things like assignment lists, to-do lists, and other structures that would easily be represented using arrays. These structures are what we would call “dynamic lists”. To represent these things we’ll use something referred to as **ArrayLists**. There are many aspects of Java’s ArrayLists package that’re useful. Google can tell you all of these aspects. It’s important to remember that you need to use boxed types like Integer, Character, Boolean etc. These are discussed in Chapter 11 under the topic “Generics”.

Remember how we had to import java.util.Arrays in order to use additional array methods? To use an ArrayList at all, we need to import them from Java’s util package using *import java.util.ArrayList;*

We need to declare the data type of the objects an ArrayList will hold, just like arrays:

ArrayList<String> exampleName;

We use angle brackets, <>, to declare the type of the ArrayList. These symbols are used for “generics”. As was said before, generics will be discussed in more detail later. For now, accept that generics are a concept that allows us to define classes and objects that can be used by ArrayList. Primitive types (int, char, double) can’t be used in an ArrayList as a result. Instead, we use <Integer> <Double>, and <Char> to declare ArrayLists. The String type can still be used as well.

We can initialize an empty ArrayList using the *new* keyword:

ArrayList<Integer> ages = new ArrayList<Integer>();

How do we get an empty ArrayList to store values? Thankfully ArrayList is equipped with an **add() method** that enables us to add arguments to an ArrayList.

```
Private class Demo{
        Private static void main(String[] args) {
                // An ArrayList is created
                List<String> demo = new ArrayList<>();
                // Elements are added
                demo.add("1")
demo.add("2");
                demo.add("3");
                demo.add("4");
                System.out.println(demo);
                //An element is added at a specific index
                demo.add(2, " two");
                System.out.println(demo);
    }
}
```

The output of this code is:

1, 2, 3, 4

1, 2, two, 3, 4

ArrayListSize

Let's say we have an ArrayList that stores tools in a person's tool bag. As the person adds items, their bag gets larger and larger. If we wanted to display the number of tools in the bag, we could find the size of it using the **size() method**:

```
ArrayList<String> toolBag = new ArrayList<String>();
toolBag.add("Trench Coat");
System.out.println(toolBag.size());
// 1 is printed
toolBag.add("Tweed Houndstooth Hat");
System.out.println(toolBag.size());
// 2 is printed
toolBag.add("Magnifying Glass");
System.out.println(toolBag.size());
// 3 is printed
```

Here, we established an empty ArrayList named *toolBag* then began to fill it using the add() method. We used the size() method after adding an item in order to determine the total number of items in the array. In dynamic objects like ArrayLists, it's important to know how to access the amount of objects we have stored.

Accessing an Index

With arrays, we can use bracket notation to access a value at a particular index:

```
double[] movieRatings = {3.2, 2.5, 1.7};
System.out.println(movieRatings[1]);
```

This code prints 2.5, the value at index 1 of the array.

Bracket notation won't work for ArrayLists. In this case, we use the **get() method** to access an index:

```
ArrayList<String> toolBag = new ArrayList<toolBag>();
toolBag.add("0");
toolBag.add("1");
toolBag.add("2");
System.out.println(toolBag.get(2));
```

This code prints "2", which is the value at index 2 of the ArrayList.

Changing A Value

Recall that we can rewrite array entries using bracket notation:

```
String[] toolBag = {"0", "1", "2"};
toolBag[0] = "zero";
// toolBag now holds ["zero", "1", "2"]
```

Array lists utilize the **set() method** in order to achieve this:

```
ArrayList<String> toolBag = new ArrayList<toolBag>();
toolBag.add("0");
toolBag.add("1");
toolBag.add("2");
toolBag.set(0, "zero");
// toolBag now holds ["zero", "1", "2"]
```

Primitive Types vs. Reference Types

We now have the capacity to discuss an idea that was brought up in the last chapter: **Reference Types**. If you recall, after discussing primitive data types we touched upon the concept of reference types. Reference types are another classification of data type. Reference types are used to declare variables and hold information. Strings and arrays are reference types. Intuitively, you can probably tell how strings and arrays differ from ints, doubles, and the like. While

primitives create containers that store primitive values, reference types merely create locations in memory that store valuable information.

When a primitive variable is used, it simply delivers the int or double or whatever value that is held within that variable. When a reference type is called, it refers to an address where information lives. In many cases, this really means nothing to us. You print a primitive, it gives you a value. You print a reference, it gives you a statement. However, it is useful to recognize the differences between primitives and references when they're being assigned or compared.

When you assign a variable to a primitive, it's straightforward: the content of the primitive is copied to the other variable. When a reference is assigned, the address, not the content, is copied. In the same way, when we compare two primitives, the actual content of the variable is being compared. When two strings are compared, it's really the addresses these strings refer to that are being compared, not the content. This brings us to the idea of *immutability*.

Immutability

Immutability means that an object's internal state can not be altered. More generally, once an immutable object has been created, it will remain the same throughout the life of that object. Strings, for instance, are immutable. Once a string is declared, it's content cannot be altered. Even methods contained within the String class such as replace(), which one assumes would change a string, must be initialized under a new string.

String example1 = "Strings are immutable.";

String example2 = name.replace("immutable", "absolutely immutable");

System.out.println(example1);

System.out.println(example2);

System.out.println(example1.equals(example2)).

Here, the first two println statements will print different strings to the screen. Even though we altered example1, the altered version is merely an altered copy and exists at a different location (example2). We can replace any part of example1 we want, but the example1 string will always read “Strings are immutable.” when printed. More to the point, when we use the equals() method to compare the two, we get the value false returned to us.

While immutability is a quality that is built into strings, it’s not for other types. Consider the humble int.

int a = 0;

System.out.println(a);

a = 1;

system.out.println(a);

Here, we’ve simply changed the value of a from “0” to “1”. As far as our program is concerned, a was never assigned the value of “0”. That value is gone now. Obviously, ints aren’t immutable by nature. It’s possible to make an int or other data types immutable but the methods that allow us to do that won’t be discussed in this book. However, we can discuss the benefits of immutability and why we would even want to do that in the first place. Because an immutable object's inner state stays continuous in moment, we can securely communicate it between various threads. We can also use it openly, and there will be no distinction in any of the items it refers to. We can tell, in other words, that immutable objects are safe from side effects.

Chapter 5:
The World of Operators

An **operator** in Java, is a specific symbol that acts upon specific variables and return results. Depending on the operator, there may be up to three variables that are acted upon at a time. Java provides an expansive selection of its own operators, many of which share the same qualities as operators in other languages. Classification of an operator is a result of two main properties: The number of variables that can be affected by the operator and the type of operation performed by the operator.

Java operators are classified as either unary, binary, or ternary operators. A unary operator operates on one variable, a binary operates on two, and a ternary operator operates on three. The assignment operator is an example of unary operator. Arithmetic operators are an example of the binary class of operators. In Java, there's only one ternary operator and it is used in very specific situations.

The second method of classifying operators is concerned with the type of operation an operator performs. These are defined as follows:

Assignment Operators

Arithmetic Operators

Increment/Decrement Operators

Bitwise Operators

Equality/Relation Operators

Logical/Conditional Operators

Miscellaneous Operators

The Assignment Operator

The basic Assignment Operator in Java is a single equals-to sign (=). This operator assigns a value on the right-hand side to a

variable on the left. The variable on the left must be a variable of a data type capable of holding the assigned value. The assignment operator operates with the following syntax on primitive and reference types:

dataType variableName = expression;

Note that variable sort must be consistent with expression type. Using an assignment chain, you can also assign a single value to various variables. Using syntax similar to the following, an assignment chain is accomplished:

// set variables a, b, and c to 1

int a, b, c;

a = b = c = 1;

The software establishes the variables a, b, and c to 1 using a single statement. Remember, the assignment operator is an element that gives the right-hand expression value. The value of int a is equal to 1 because int b is equal to 1 because int c has been allocated a value of 1. Using an assignment chain is an easy way to set the common value of a group of variables.

Arithmetic Operators

There are 5 **Arithmetic Operators +**, **-**, *******, **/**, and **%**. These operators conduct addition, subtraction, multiplication, division, and modulo operations, respectively. Any numeric type can be used with arithmetic operators.Java also provides plus (+) and minus (-) for making numeric values positive or negative. By default, numeric values are positive if they are not given any sign.

Addition and String Concatenation Operator (+)

The **Addition/Concatenation Operator (+)** adds two numbers. Besides performing addition, it is also used to concatenate two strings. Therefore, if the addition operator is given to two numerical amounts, arithmetic addition is performed, and if applied to two strings, they are concatenated together. If either variable is a string then the other one is also transformed to a string, a very significant

notice about the addition operator is. When combining addition with concatenation, be sure to use parentheses.

The following part of code shows the addition operator's string concatenation procedure:

```
public class addDemo{
   public static void main(String[] args){
      int x = 1, y = 2;
System.out.println("You may expect 3 but the result is: " + x + y);

   }
}
```

System.out.println will send the following to our screen: “You may expect 3 but the result is: 12.” Here, the addition operator simply pushed the two terms together instead of adding the two numbers. To add 1 and 2, x and y must be enclosed in brackets.

```
System.out.println("To get 3, put x and y in brackets, like this: " + (x + y));
```

The first print statement concatenates variables x and y with the string "You may expect 3 but the result is: " then prints "You may expect 3 but the result is: 12" as an output. In the second print statement, x + y are enclosed in brackets and thus, the addition of x and y will take place first. This can be attributed to the fact that the addition operator has a lower precedence than brackets.

Subtraction Operator (-)

The **Minus Operator** (-) is a binary and unary operator. As a binary operator, the minus operator simply performs subtraction on two variables. The right hand variable will be subtracted from the left. Consider 11 - 5. This will evaluate to 6. As a unary operator, the minus operator simply makes a value negative.

Multiplication Operator (*)

The **Multiplication Operator** (*) is more or less intuitive. This operator performs multiplication. For example, 4*11 evaluates to 44.

Division Operator (/)

The **Division Operator** (/) divides the variable on the left by the variable on the right. Unlike the more obvious arithmetic operators, the division operator comes with some complications. If an integer is divided by an integer, the result will also be an integer. The caveat here is that any remainder will be left off. If one variable is a floating point type, the result will also be a floating point type. Dividing by zero will either result in an *ArithmeticException* for integers or an infinite or NaN (Not a Number) for floating point values. See below:

11/5 *// 2*

11/5.0 *// 2.2*

11/0 *// ArithmeticException*

11/0.0 *// positive infinity*

0.0/0.0 *// NaN*

Modulo Operator (%)

The **Modulo Operator** (%) is a strange one. This operator will return whatever remains after performing division. This operator also works from left to right. The result of 11%5 will be 1. The result will be positive or negative depending on the status of the right hand variable. In Java, floating point values and integers are treated equally by the modulo operator. Consider the following: performing the modulo operation on 11 and 5 will evaluate to 1 while performing the modulo operator on 11.5 and 5 will evaluate to 1.5. Similar to the division operator, performing modulation will zero will throw an AritmeticException for integers and NaN for floating point values. NaN will also be the result if modulation is used on positive or negative infinity.

Consider the following:

public class modOpDemo{

```
    public static void main(String[] args){
        int a = 11;
        double b = 11.5;
              double c = Double.POSITIVE_INFINITY;
        System.out.println("a modulus 5 is: " + (a % 5));
        System.out.println("b modulus 5 is: " + (b % 5));
        System.out.println("c modulus 5 is: " + (c % 5));
    }
}
```

The output of the first print out will be "x modulus 5 is: 1". The second print out will read "y modulus 5 is: 1.5". Finally, the third print statement will ready "c modulus 5 is: NaN".

Modulo can be a confusing concept. It can be useful to remember this application of it. Imagine we need to know whether a number is even or odd. An even number is divisible by 2. Modulo can help. Dividing an even number by 2 will have a remainder of 0. Dividing an odd number by 2 will have a remainder of 1.

Increment/Decrement Operators

Java provides two **Increment** (++) and **Decrement** (--) **Operators**. Increment and decrement operators are used to increase or decrease a variable's value by one.

The carriers of increment and decrease increase or decrease the amount of an int variable by 1 or a floating point value (float, double) by 1.0. In the Unicode sorting series, the unary increment and decrement operators can also be introduced to char variables to move one pixel place forward or backward. As they are subjected to a single variable, these operators are regarded as unary operators. The operators of increment and decrement are used as prefixes or postfixes. If the user is put in front of the variable, increment and decrement mode is called prefix.

The prefix mode of increment and decrement first increases or decreases the value of the variable, then the updated value of the variable is used in assignment during assignment of one variable to another. On the contrary, the first variable is used in assignment in the postfix increment and decrement mode, then the variable is increased or decremented. (Note that prefix and postfix method of activities do not create any distinction if they are used in a separate declaration where only the price is increased or decreased but no task is produced.) Now let us explain in detail increment and decrement operators.

Increment Operator (++)

The **Increment Operator (+ +)** increases by one a single variable. The conduct of the increment operator relies on its location close to the variable (prefix or postfix) during an assignment operation. Used in prefix mode, the variable rises and evaluates the incremented value of that variable. It raises the variable when used in postfix mode, but before incrementing it evaluates the meaning of that variable.

Let's take an instance to see Java's increment operator's prefix and postfix conduct.

```
int a = 10;
int b;
/*
The following code shows increment of the prefix. First, an increment will be allocated to an updated value of a b.
*/
b = ++a;
System.out.println("b : " + b); //will print b : 11
System.out.println("a : " + a); //will print a : 11
/*
```

The following code shows increment of postfix. A will be allocated the first value of a to b then incremented.

/

```
b = a++;
System.out.println("b : " + b); //will print b : 11
System.out.println("a : " + a); //will print a : 12
/*
The following code displays postfix increment. A is assigned the first value of a to b incremented afterwards.
*/
++a;
System.out.println("a : " + a); //will print a : 13
a++;
System.out.println("a : " + a); //will print a : 14
```

Decrement Operator (--)

The Operator of Decrement(—) decreases by one its single variable. The conduct of the decrement operator during an assignment operation relies on whether it is used in prefix or postfix mode in relation to the variable. It decreases the variable when used in prefix mode and evaluates the decreased importance of that variable. It decreases its variable when used in postfix mode, but assesses the value of that variable before it is decremented.

Let's take an instance to see Java's decrement operator's prefix and postfix conduct.

```
int a = 10;
int b;
/*
The following code shows decrement of the prefix. First, a decrement will then be assigned an updated value of a to b.
```

```
*/
b = --a;
System.out.println("a : " + a); //will print a : 9
System.out.println("b : " + b); //will print b : 9
/*
The following code shows decrement of postfix. A first value will be assigned to b and then a decrement will be applied.
*/
y = a--;
System.out.println("a : " + a); //will print a : 8
System.out.println("b : " + b); //will print b : 9
/*
When a decrement is produced in an independent statement, there is no distinction between prefix and postfix methods.
*/
--a;
System.out.println("a : " + a); //will print a : 7
a--;
System.out.println("a : " + a); //will print a : 6
```

Strange Behavior of Java Postfix Operators

Sometimes you might see a strange behavior of the postfix form of increment or decrement operator. Take the following piece of software for an instance:

```
int a = 1;
a = a++;
System.out.println("a : " + a);
```

You might have thought after studying the code section above that a would have been 2. In this case, our code will return a value of 1. It's

not a Java bug, and it has a valid cause. Before describing this cause, it is suggested that if you find"a = a++;" you should alternate it with "a++" in any software you are using instantly. Now, let's explore: why is it oddly conducting?

The postfix increment or decrement operator, by definition, first returns the variable's initial value and then increases the variable. The postfix protocol in Java has a greater priority than the assignment operator, so the a++ returns the initial value of a rather than the incremented one. In the meantime, a increases and becomes 2. However, the initial value received by a++ is allocated to a, which was 1.

Bitwise Operators

Low-level operators are Java **Bitwise Operators**. This implies that they function at a bit level and are used to create a bit pattern in individual pieces. Only integer kinds, i.e. byte, short, int, long, and char, can be used by bitwise operators. For checking and storing individual parts in a value, bitwise operators are most frequently used. If one of the statements is a lengthy one for a bitwise operator, the outcome is a lengthy one. Otherwise, an int is the consequence. Bitwise operators of Java are ' bitually complement or not operator, ' ' bitwise and ' ' bitwise or ' and ' bitwise xor or strange operator. ' Floating point, boolean, array, or object variables can not be used by bitwise operators.

Bitwise Complement (~)

The **Bitwise Complement** (~) or **Bitwise NOT Operato**r It is a single agent that inverts each piece of its single variable and converts them into zeros and zeros.

Bitwise AND (&)

The **Bitwise AND Operator** (&)Is a binary operator operating on two integer factors by performing a Boolean AND operation on their single pieces. The outcome is only 1 if in both variables the respective parts are 1. The bitwise AND distributor is a perfect way to check if an integer piece is a 1 or a 0.

Bitwise OR (|)

The **Bitwise OR Operator** (|) is a binary operator that performs a Boolean OR function on its respective pieces on two integer variables. The outcome is 1 if a 1 in either or both of the factors is the respective number. Only when the two respective variable parts are null has it a null point.

Bitwise XOR (^) or Exclusive OR

The **Bitwise XOR Operator** (^) is a binary operator that works on two integer factors by performing on their respective pieces a Boolean XOR (Exclusive OR) function. The result is 1 if the two variables differ in the respective pieces. If both or both zeros are the respective variable numbers, the outcome bit is a zero.

Binary Shift Operators

Binary Shift Operators shift all the bits of the input value either to the left or right based on the shift operator. There is the left shift operator, right shift operator, and the unsigned right shift operator. The syntax for all shift operators is as follows:

value <operator> <number_of_shifts>

The left side of the phrase is the shifting integer, and the right side of the phrase denotes the number of times it has to be shifted.

The **Left Shift Operator** (< <) moves the parts to the top according to the amount of moments indicated by the variable's correct hand. The vacant room on the right is filled with 0 after the left shift.

Another significant point to remember is that moving an amount by one is equal to multiplying it by 2 or, generally speaking, moving an amount by n locations is equal to multiplying by 2^n.

We have an example code below.

```
public void
shiftLeft{
      int value = 12;
int leftShift = value <<2;
```

```
assertEquals(48, leftShift); //assertEquals will check if the value of leftShift equals 48 after value has been left shifted twice
}
```

Here, we've taken the number twelve and shifted it two places to the left (12 <<2). The binary equivalent of 12 is 00001100. After shifting to the left by 2 places, the result is 00110000, which is equivalent to 48 in decimal, so the assertEquals method will return true.

The **Right Shift Operator** (>>) shifts all the bits to the right. The empty space on the left side is filled depending on the input number. When an input number is negative, where the leftmost bit is 1, then the empty spaces will be filled with 1. When an input number is positive, where the leftmost bit is 0, then the empty spaces will be filled with 0. Let's continue the example using 12 as input.

```
public void
shiftRight{
int value = 12;
int rightShift = value >>2;
assertEquals(3, rightShift);
}
```

In this case we took twelves and shifted it 2 places to the right (12 >>2). The input number is positive, so after shifting to the right by 2 places, the result is 0011, which is 3 in decimal. The assertEquals method will return true.

The **Unsigned Right Shift Operator** (>>>) is very similar to the signed right shift operator. The only difference is that the empty spaces in the left are filled with 0 regardless of whether the number is positive or negative. Therefore, the result will always be a positive integer.

Applications of Java Bitwise Operators

Bitwise operators are of excellent assistance when operating on individual pieces instead of a full amount. Masking is a technique for

removing a particular piece. For example, if you want to check if a given integer's 5th bit is 0 or 1 or not, you can do it using the following:

int x = 34;

int sixthBit = (x & 16) / 16; ***// the sixth bit is 0, as the binary form of 34 is 100010***

System.out.println("The sixth bit of " + x +" is " + sixthBit);

This will return “The fifth bit of 34 is 0”.

We can also test if a specified integer is strange or even strange without using the procedure of the module or division. If the yield (x & 1) is 1, it's odd. Otherwise, even it is. Bitwise operators can also compute a number's sign. We understand that if an integer's leftmost digit (also known as the Most Significant Bit or MSB) is null, the amount will be positive. The value is negative otherwise. If an integer is a power of 2, we can also identify it as follows:

int num = 32;

int minusOne = num - 1;

boolean isPofT = (num & minusOne) == 0;

System.out.println(“Is ” + num + " a power of two? " + isPofT);

This will return “Is 32 a power of two? True”.

Equality and Relational Operators

Comparisons are made using equality and relational operators in Java. There are two carriers of equal treatment: equal(= =) and not equal! (=). These operators are testing whether or not two values are equal. Relational operators are used to experiment more than and less than interactions with arranged kinds. All operators of equality and relationships are binary operators and return a true or false boolean outcome.

Equality Operators

Java's two equality providers are the Equals-To Operator(==) and Not Equals-To Operator! (=). The agent returns real if its two factors are equivalent and otherwise yields incorrect. When introduced to natural factors, the equals-to-operator checks whether or not the variable numbers are identical. However, it checks whether or not the variables refer to the same item or set for reference type data. In other words, the equals-to-operator compares two memory locations while comparing two objects and, obviously, if two references point to a single object, the location of the memory will be the same. You can't check two separate strings for equality with this operator.

The non-equals-to-operator, on the contrary, is exactly the opposite of the operator-equals. It returns accurate if there are distinct values for the two natural factors being analyzed or if its two reference variables refer to distinct sets or items. Otherwise, it's judging to be wrong.

Note that it is possible to use equality operators with boolean values, objects, or arrays, but relational operators can not. It is only possible to use relational operators with numbers and symbols.

Relational Operators

Relational operators are operating on the kinds ordered. There are four relational procedures: the **Greater Than Operator** (>), the **Greater Than Or Equal To Operator** (> =), the **Less Than Operator** (<), and the **Less than Or Equal To Operator** (< =). These operators, more or less, speak for themselves. If you want to know if a value is higher than another value, just use the greater than operator. It is important to remember that even if only one aspect of the argument is true, the greater than or equal to and the less than or equal to operators will return true. See below

```
int a = 100;
int b = 200;
boolean result1 = a>b;
System.out.println(result1);
```

```
boolean result2 = a>=b;
        System.out.println(result2);
boolean result3 = a<b;
        System.out.println(result3);
boolean result4 = a<=b;
        System.out.println(result4);
```

Running this code will return the value false for results one and two, as 100 is not greater than nor is is equal to 200, and will return true for results three and four, because is less than 20. Note, that while 10 isn't equal to 20, the less than or equal to operator still returned true because at least one aspect of that argument was true.

Logical and Conditional Operators

When a decision-making declaration involves two or more relational phrases, they are paired by logical operators. There are two types of logical operators: short circuit and non-short circuit. Both of them work on boolean values and back to boolean values. Short-circuit operators are called logical operators while boolean operators are called non-short-circuit operators.

The operators of both short and non-short circuits conduct logical operations on boolean expressions. The primary distinction is in their working modes: only if needed, short circuit logical operators assess subsequent phrases. For instance, if you want to perform a logical AND operation between expression 1 and expression 2 and expression 1 then the result will be false irrespective of what the second expression returns. In that situation, if left un-executed, expression 2, will have no effect on the logical operation's eventual results. The advantage of this is that the procedure is made quicker. On the other hand, not-short circuit logical will execute all steps available when running a logical operation. Short circuit operations are efficient and safe, so the alternative is rarely used.

Logical AND (&&)

The Logical AND Operator (&&) performs on its operands a Boolean AND procedure. It assesses accurate if and only if both of the logical AND's operands are valid. It evaluates to incorrect if either or both operands are incorrect. For example:

//Evaluates to true, if both comparisons are true

if ((x < 20) && (y > 5)) …

The logical AND or conditional AND operator is a short-circuit operator because only if necessary it evaluates the second operand. If the first operand evaluates true, the phrase valuation is incorrect, irrespective of the second operand's significance. The programmer can therefore use a shortcut and ignore the second operand to boost effectiveness. Since there is no guarantee that the second operand will be assessed, you must be careful when using this carrier.

Logical OR (||)

The Logical OR Operator performs a boolean OR operation on two boolean operands. It evaluates to actual if either or both of its operands are true. If both operands are inaccurate, it evaluates to real. Like the logical AND operator, the logical OR operator does not always evaluate the second operand. If the first operand is valid, the valuation of the phrase will be valid, regardless of the significance of the second operand. The user therefore simply skips that second operand in that situation.

Boolean AND (&)

When used with boolean expressions, the AND procedure functions like the logical AND procedure, except that it always evaluates both operands regardless of the significance of the first operand. This tool is almost always used as a bitwise operator with integer operands, however, and most programmers would question its use with boolean operands.

Boolean OR (|)

This person operates a boolean OR feature on two boolean operands. The logical OR operator is similar except that both operands are always evaluated, even if the first is true. Like the

boolean AND operator, the boolean OR operator is almost always used as a bitwise operator and very rarely performed on boolean expressions.

Boolean XOR (^)

When used with boolean operands, this agent calculates its operands ' Exclusive OR (XOR). If one of the two operands is valid, it will evaluate to real. Essentially, it will be assessed to false if both operands are incorrect or if both operands are accurate. Unlike the logical AND and the logical OR operators, this one must always evaluate both operands. The exclusive OR distributor is much more commonly used as a bitwise operator. It's the same user as the!Operator of Boolean expressions used.

Boolean NOT (!)

The operator of Boolean NOT(!) is a single agent that changes its operand's boolean value. It evaluates to false when contrasted with a true value and evaluates to real when compared with a fake value. Since the boolean NOT operator is a one-time operator, it has a high priority and often has to be used with parentheses.

if (!f) ... **// f is a boolean variable declared somewhere**

while (!isE()) ... **// isE() returns a boolean value**

if (!(a > b && b > c))

Miscellaneous Operators

In addition to the basic assignment operator, Java also defines 12 shorthand assignment operators that combine tasks with 5 arithmetic operators (+=, -=,*=,/=, percent =) and 6 bitwise and change operators (&=,,^=, < <=, > >=, > =). For example, the+= operator reads the left variable value, adds the right variable value to it, then stores the sum back into the left variable as a side effect and returns the sum as the expression value. So the sentence a += 2 is almost the same as the sentence a= a+ 2.

The distinction between these two terms is that the remaining variable is only assessed once when we use the += operator. This

makes a distinction when there is a side effect in that variable. Consider the following two terms: x[j++] += 4 and x[j++] = x[j++]+4. These are not equivalent.

```
public class Demo1
{
  public static void main(String []args)
  {
    int x[] = new int[6];
    int j = 1;
    x[j++] += 2; // i is incremented only once
    System.out.println(i);
    x[j++] = x[j++] + 2; // i is incremented twice
    System.out.println(i);
  }
}
```

The output of this code would be one and four.

Type Casting

Type casting allows us to transform primitive values, typically from larger to lower, from one type to another. Casting can be either explicit or implicit. Conversion occurs automatically in an implicit cast, which means we don't have to write any code. When we transform to a wider sort, we typically have an implicit cast. For instance, if we put a smaller thing (say, a byte) in a larger container (such as an int), it will be an implicit cast. A transition from a big container to a tiny container is called a conversion that is widening and will involve an explicit cast. We notify the compiler, in an explicit cast, that we recognize the risk and assume complete accountability for dealing with any subsequent issues.

It is very essential to remember that the shorthand assignment operators allow us to conduct addition, subtraction, multiplication or

division without specific casting. In fact, +=, -=, *=, and /= will all be put in an implicit cast. Below is an example:

```
byte y = 5;
y += 5;
// No problem. This will add 5 to y (result is 10) and is the same as:
byte z = 5;
z = (byte) (z + 5);
// Without the cast this will compile because z+5 has an int result
```

Precedence and Associativity

In Java, there may be more than one operators engaged in an expression when an object is assessed. Java has made it apparent which operator should be assessed first when more than one operator has to be assessed in an action. Java creates this choice based on the operators precedence and associativity.

Precedence is the operator's priority order, if there are two or more operators in an expression, then the highest priority operator will be executed first, then the next highest, and so on. For example, the multiplication(*) operator will be processed first and then added in the expression 1 + 2* 5. This is because the rate or precedence of multiplication is greater than comparison.

Alternatively, if an operand is shared by two operators (2 in the above example is shared by the addition and multiplication operators), then the higher priority operator first gets the shared operand to be processed. The above instance can be intuitively grasped by anyone acquainted with the fundamental mathematical notion of "order of activities." In more subtle cases, however, the scenario may not be as obvious.

What do we do if the same priority is given to every provider in an expression? In this situation, it becomes easily evident the second

property connected with Java operators. This property is called associativity. Associativity informs the operator whether or not the implementation path is left to right or right to left. For instance, the assignment operator is executed from right to left in the expression a= b= c= 10. This implies c is allocated by 10, then b is allocated by c, and lastly a is allocated by b.

Note that by enclosing the lower order priority operator in parentheses, you can change the priority of a Java operator (but not associativity). In the phrase (1+ 2)* 3, it will be added first because the parenthesis procedure (addition) has a greater precedence than the excluded procedure (the multiplication operator).

Here is a list from highest to lowest precedence:

Postfix (expr++, expr--)

Unary (++expr, --expr, +expr, -expr, !, ~)

Multiplicative (*, /, %)

Additive (+, =)

Shift (<<, >>, >>>)

Relational (>, <, >=, <=, instanceof)

Equality (==, !=)

Bitwise AND (&)

Bitwise Exclusive OR (^)

Bitwise Inclusive OR (|)

Logical AND (&&)

Logical OR (||)

Ternary (?:)

Assignment (=, +=, -=, *=, /=, %=, &=, ^=, |=, <<=, >>=, >>>=)

Chapter 6:
Control Flow Statements

Imagine we're writing a program that enrolls students in courses. If a student has completed the prerequisites, then they can enroll in a course. Else, they need to take the prerequisite courses. They can't take Physics II without finishing Physics I.

We represent this kind of decision-making in our program using conditional or Control Flow Statements. Up to this point, our code ran line-by-line from top to bottom. However, with control flow statements we can split the execution stream by creating decisions, looping and branching, allowing our program to perform specific pieces of code on a conditional basis. Conditional statements check a boolean condition and run a block of code depending on the condition. Curly braces mark the scope of a conditional block similar to a method or class.

This section describes if-then, if-then-else, and switch statements (conditional, decision making statements), (for, while, do-while), and break, continue, and return statements (branching, iterative, looping statements) supported by the Java programming language. We will also touch upon exceptions, handling exceptions, and other unexpected events that disrupt the flow of a program.

Decision Making Statements

There are relational and logical operators to shape a conditional expression that returns either true or incorrect in order to promote conditional control flow in Java. Words true and false look like keywords, but they are actually boolean literals and can't be used in Java programs as identifiers. Java program determines the route of execution on the grounds of conditional expressions true or false.

If-Then Statements

An if-then statement is the most basic control flow statement available. It tests whether or not an expression is true or false and

executes a block of code based on it.

```
if (value == 1) {
  //Then
  System.out.println(“Yes");
}
```

The if keyword marks the beginning of the conditional statement, followed by parentheses that contain a boolean datatype as a condition. For the condition in parentheses we can also use variables that reference a boolean, or comparisons that evaluate to a boolean. The boolean condition is followed by opening and closing curly braces that define a block of code. This block of code runs if, and only if, the boolean is true.

```
boolean value1 = true;
if (value1) {
  System.out.println("Value is true.");
}
// Prints "Value is true."
int value2 = 9;
if (value2 > 12) {
  System.out.println("Yes!");
}
// Nothing prints.
```

By default, the keyword if controls only one statement. Therefore, if you’re only referring to one statement using if you do not need to enclose that statement within curly braces. However, it’s good practice to use braces every time as it increases readability of the program. If there are two or more statements to be controlled by an if statement, then curly braces must be used for the program to work. On a final note, observe that control flow statements do not end with a semicolon.

Else Statement

We've seen how to conditionally execute one block of code, but what if there are two possible blocks of code we'd like to execute? Let's say if a student has the required prerequisite, then they enroll in the selected course, else they're enrolled in the prerequisite course instead.

We create an alternate conditional branch with the else keyword:

```
int x;
int y;
boolean value1 = x>y;
if (value1) {
  //This code executes.
} else {
  // This alternative code executes.
}
```

This conditional statement ensures that exactly one code block will be run. If the condition, value1, is false, the block after else runs. There are now two separate code blocks in our conditional statement. The first block runs if the condition evaluates to true, the second block runs if the condition evaluates to false.

Else-if Statement

The conditional structure we've learned can be chained together to check as many conditions as are required by our program. Imagine our program is now selecting the correct color of car that someone wants to buy. We'll check their submission to find the correct car to purchase. The conditional statement now has multiple conditions that are evaluated from the top down:

```
String carColor = "Purple";
if (carColor.equals("Blue")) {
  // Select Blue car
```

```
} else if (carColor.equals("Red")) {
  // Select Red car
} else if (carColor.equals("Purple")) {
  // Select Purple car
} else {
  System.out.println("Vehicle not found!");
}
```

The first condition to evaluate to true will have that code block run. Here's an example demonstrating the order:

```
int value = 80;
if (value >= 90) {
  System.out.println("Result1");
} else if (value >= 80) {
  System.out.println("Result2");
} else if (value>= 70) {
  System.out.println("Result3");
} else if (value >= 60) {
  System.out.println("Result4");
} else {
  System.out.println("Result5");
}
// prints: Result2
```

This chained conditional statement has two conditions that evaluate true. Since value >= 70 comes before value >= 60, only the earlier code block is run. It's important to recognize that only one block of code can run.

Switch Statements

An alternative to chaining if-then-else conditions together is to use the **Switch Statement**. This operation will check a given value against a myriad of conditions and run the block of code associated with a true case.

Here's an example of our car selection code as a switch statement instead:

```
String carColor = "Purple";
switch (carColor) {
  case "Blue":
    // Select Blue car
    break;
  case "Red":
    // Select Red car
    break;
  case "Purple":
    // Select Purple car
    break;
  case "Green":
    // Select Green car
    break;
  default:
    System.out.println("Vehicle Not Found.");
```

This example selects the purple car contained in the parentheses, carColor, against each of the case labels. If the value after the case label matches the value within the parentheses, the switch block is run.

In the above example, carColor references the string "Purple", which matches case "Purple".

When no value matches, the default block runs. Think of this as the else equivalent.

Switch blocks are different than other code blocks because they are not marked by curly braces and we use the break keyword to exit the switch statement.

Without break, code below the matching case label is run, including code under other case labels, which is rarely the desired behavior.

```
String carColor = "Purple";
switch (carColor) {
  case "Blue":
    // Select Blue car
  case "Red":
    // Select Red car
  case "Purple":
    // Select Purple car
  case "Green":
    // Select Green car
  default:
    System.out.println("Vehicle Not Found.");
}
```

In this case, because there was no break statement this code will select the purple car as well as the blue, red, and green cars.

Looping Statements

In the programming world, we hate repeating ourselves. Writing the same code over and over again is time-consuming. Also, having less code means there's less to debug. However, we often need to do the same task more than once. Fortunately, computers are really good at doing repetitive tasks quickly and effectively. In Java, we can get our computer to do this using loops.

A loop is a programming tool that allows developers to repeat the same block of code until some condition is met. We employ loops to easily scale programs, saving time and minimizing mistakes.

There are three types of loops you can expect to see everywhere: While loops, For Loops, and For-Each Loops.

While Loops

A **While Loop** is an introductory looping statement that takes the following form:

```
while (boolean){
    //Code Block
}
```

In while loops, the compute will evaluate the boolean expression contained within the parentheses first. If it finds the expression to be true, it will execute the block of code contained within the curly braces.The boolean expression will be evaluated again, and if it fails to be valid, the code block will operate again. As soon as the boolean expression is valid, it will proceed to replay this block. The process will be performed until either the boolean expression yields incorrect, or the circuit ends with a break statement.

We can use loops to discover the popular sequence of Fibonacci:

```
public class FibonacciDemo
{
   public static void main(String[] args)
   {
      int a = 0;
            int b = 1;
      while (a < 50)
      {
         System.out.print(a);
```

```
            a = a + b;
            b = a - b;
        }
    }
}
```

The output of this program will print the first ten numbers of the Fibonacci Sequence (0, 1, 1, 2, 3, 5, 8, 13, 21, 34). After the 11th iteration, our variable “a” will equal 55, which will cause the boolean expression “a < 50” to evaluate to false and cancel our while loop. Take notice of the fact that we used System.out.println() before our equations. Remember, by design, our program runs top to bottom. If we place the print line statement after our equations, the code will still run. However, the final step before the while loop is evaluated will be println(). Meaning that the 11th value will be printed, which is a value larger than 50, before the computer realizes that this value is outside of the range of values we want.

Another beneficial task is utilizing incrementation (++, --) within a while loop. A counter (also known as an iterator) is a variable used within the loop and (usually) incremented in value during each runthrough of the code.

For example:

```
// counter is initialized
int a= 0;
// conditional logic uses counter
while (a < 4) {
  System.out.println("Success");
  // counter is incremented
  a++;
}
```

In the above example, the counter (int a) is initialized with a value of 0 before the loop. Everytime the loop is executed, the word “Success” is printed and the counter is incremented by one. “Success” will continue to be printed and the counter incremented until it reaches four. “Success” will be printed four times before the while loop is terminated. In a similar way, you can initialize your counter at a high number, and decrement down until the counter reaches zero. Imagine a clock counting down to midnight on New Years Eve.

In the case of while loops, and really any loop in Java, be weary of creating infinite loops:

```
int a = 4;
While (a<6){
System.out.println(Success);
}
```

This code will cause our computer to crash. Since there is no way for our conditional variable to be more than six, our computer will keep printing “Success” infinitely. Obviously, this is not ideal...

Do While

With Java's while statement you saw that before joining the body of the loop, the booleanExpression is checked for truth. In Java's **Do While Loop**, on the contrary, when entering the loop, the boolean expression is screened for truth. If it returns false, then the command won’t be performed again on the body of the loop.

Loops can therefore be classified as entry-controlled and exit-controlled depending on where the boolean expression is assessed (when entering the switches body or leaving the loop's body). Java's do while loop is an exit controlled circuit. From the above declaration, you can also say that the exit-controlled panel body will be conducted at least once, even if it returns true the first time the sample is delivered.

Java's fundamental syntax while statements are as follows:

```
do
  //Statement {
  }
while (booleanExpression);
```

Note that there is a semicolon (;) after the while (boolean expression) in do while.

For Loops

Incrementing while loops is so useful that Java has included syntax that specifically performs this function. This syntax is referred to as a **For Loop**. In one line, a for loop initializes a counter variable, defines a condition of the loop, and finally increments the counter.

The syntax of a for loop is as follows:

```
//(initialization; booleanExpression; updateCounter)
for (int j= 0; j< 5; j++){
      //block of code
}
```

Here, our counter, “j”, is initialized to 0 (int j = 0). A boolean expression that must be satisfied by j is given (j <5). Finally, our counter will increment at the end of each loop (j++). The code will run through the loop a total of five times. You’ll also hear the term “iteration” in reference to loops. When you iterate, it just means that you are repeating the same block of code.

Here is the success example from above using a for loop:

```
for (int j = 0; j <4; j++){
System.out.println(“Success”);
}
```

This code will initialize our counter variable to zero (int j = 0), sets a conditional statement (j < 4), and increments the counter variable (j++). “Success” will once again be printed four times.

For loops are also very useful for looping through lists of data and effecting change in each item. In Java, an example of such a list could be something like an array or ArrayList. How would this work? Remember that for loops involve a counter variable. We can use that counter to move through the positions of a list of data and track each element. Because the first position in an array or ArrayList is zero, the counter would begin at zero and increment until the end of the list. In this way, we can move through an array or ArrayList using its indices.

For example, if we wanted to add one to every item in an array, we could do this:

```
for (int j = 0; j < example.length; j++) {
  example[j] += 1;
}
```

Notice that our condition in this example is j < example.length. Because arrays start at 0, the length of example is 1 larger than its final index. Therefore, the loop should stop when it is less than BUT NOT equal to the length value.

If you wanted to do this with an ArrayList, this code would look like:

```
for (int j = 1; j< example.size(); j++) {
  int num = example.get(j);
  example.set(j, num + 1);
}
```

For Each (Enhanced For) Loops

Sometimes we need access to the elements' indices or we only want to iterate through a portion of a list. If that's the case, a regular for loop is a great choice. But sometimes we couldn't care less about the indices; we only care about the element itself. At times like these, **for-each loops** or **Enhanced For Loops** come in handy.

For-each loops allow you to directly loop through each item in a list of items (like an array or ArrayList) and perform some action with

each item. The syntax looks like this:

```
for (String inventoryItem : inventoryItems) {
  System.out.println(inventoryItem);
}
```

We can read the : as “in” like this: for each inventoryItem (which should be a String) in inventoryItems, print inventoryItem.

Note that we can name the individual item whatever we want; using the singular of a plural is just a convention. You may also encounter conventions like String word : sentence.

Branching Statements

Java offers break, continue and return statements. The break and continue in Java are two keywords that beginners will need to familiarize themselves with while using loops (for loop, while looping and doing while looping). The Java break statement is used to break the circuit and transfer power to the code immediately outside the loop while the continue statement is used to evade present execution (iteration) and transfer power back to the loop start. Both break and continue statements allow the programmer to generate advanced constructs of algorithms and loops.

In this novel we will see instances of break and continue declarations in Java as well as some significant points linked to maintaining the circuit using break and continue declaration. The break keyword can also be used to enter the present decision within the switch statement and, if not used, can trigger a break on the switch. It is possible to unlabel or label both the break statement and the continue statement. It’s much more common to use break and continue unlabeled.

Break Statements

Java's **Break Statements** do just that: they break the execution stream. It conducts three activities in Java: ends change statements

execution, exits loops, and functions as a goto. It is possible to label or unlabel these remarks. Unlabeled breaks perform the first two listed actions while a goto is used as an unlabeled break.

Java's Unlabelled break Statement

Let's first glance at the unlabeled break statement in Java. The loop is terminated when a break statement is found inside a loop and the program control resumes after the loop at the next statement. It is essential to remember that it only exits the circuit in which it is performed when a break is used within nested loops. Here is just one easy instance:

```
// Exiting a loop using break
class BreakDemo
{
   public static void main(String args[])
   {
      //solo loop break
      for(int j = 1; j <= 121; j++)
      {
         if(j * j == 169) break; // if j is 13 this loop will break
         System.out.print(j + " ");
      }
      System.out.println("Done.");
        System.out.println(); //This enters an extra blank line for the sake of formatting
//exiting from nested loops
      for(int j = 1; j <= 11; j++)
      {
         for(int k = 1; k <= 50; k++)
         {
```

```
            if(k > j) break; // terminate loop if k is greater than j
            System.out.print(j + " ");
          }
          System.out.println();
        }
        System.out.println("Done.");
    }
}
```

The output of this code is:

1 2 3 4 5 6 7 8 9 10 11 12

Done.

1

1 2

1 2 3

1 2 3 4

1 2 3 4 5

1 2 3 4 5 6

1 2 3 4 5 6 7

1 2 3 4 5 6 7 8

1 2 3 4 5 6 7 8 9

1 2 3 4 5 6 7 8 9 10

1 2 3 4 5 6 7 8 9 10 11

Done.

Under two conditions, either j exits the first loop in the above program, or j*j product is equal to 169. Using the break statement, the circuit is permitted to end earlier. In the second example, there are two nested loops when the pyramid is printed and the situation

when (k > j) only exits from the inner loop when it becomes real control. Note that out of a circuit or change it is not feasible to use unlabeled break.

Java's Labeled break Statement

In some respects, Java's marked break statement meets the goto's goal. Java has goto as a reserved word, but it is not being used at the time. Because goto comments are considered to be poor programming style, the mood of structured programming is reduced. Whether or not goto has always been a topic of debate in a program. A bigger cooperative programmer argues that goto's unrestricted use is prone to error, but use regularly to get out of a loop is expedient. Java designers dedicated to the programmers ' community in favour of this programming style and placed a marked break.

A labeled break marks a block of code and when performed from within that block, it takes you out of that block. The situation is that from the same block the break statement should be performed and correctly marked.

The syntax of labeled break statement is presented below:

break label;

Look at the following program where we have a block named break1, and if a specific situation happens, we want to get out of this block. In that scenario we're going to perform a break declaration within the designated break1 block. The next instance shows the use of labeled break:

```
public class Demo{
  public static void main(String[] args){
    break1:
      for(int a = 1; a < 6; a++){
        for(int b = 1; b < 6; b++){
          for(int c = 1; c < 6; c++){
            if (a >= 2){
```

```
            //break1 block is broken out of
             break break1;
           }
           System.out.print(j + " ");
         }
         System.out.println();
       }
       System.out.println();
     }
   System.out.println("break");
  }
}
```

The output of this program is:

1 1 1 1 1 1

2 2 2 2 2 2

3 3 3 3 3 3

4 4 4 4 4 4

5 5 5 5 5 5

6 6 6 6 6 6

break

You will see a break statement regulated by if (a > = 3) if you look at the nucleus of the innermost for declaration in the above program. The declaration breaks once this situation returns true. The break1 block is performed and control is transferred from the block.

Continue Statements

Only iterative (loop) remarks (while, do, and for) can place the continued statement of Java to trigger an early iteration. It means you can place a continuous statement at that stage in certain

situations if you don't want to process the entire loop body and want to transfer the command to the loop-continuation point. It is also possible to use Java's continuous declaration with or without a tag. In a Java program it has the following syntax:

continue optional-label;

Continue declarations in Java without transferring tags control over the enclosure while, during, or for configuration. A continuous statement with a label statement, on the other hand, moves order to the marked loop statement. The following program demonstrates the use of unlabeled and marked continuous statements.

```
private class Demo
{
  private static void main(String[] args)
  {
    // unlabelled
    System.out.println("Unlabeled: ");
    for (int j =1; j <= 20; j++)
    {
      if (j % 3 != 0) continue;
      System.out.format("%3d", j);
    }
    System.out.println("\n");
    System.out.println("Labeled: ");
    example1:
    for (int j =1; j <= 20; j++)
    {
      for(int k = 1; k <= 5; k++)
      {
```

```
      if (j % 3 != 0) continue example1;
      System.out.format("%3d", j*k);
    }
    System.out.println();
  }
 }
}
```

The labeled continue in the above instance requires command of the program to the exterior tag from where the outer loop begins.

Return Statements

The final statement of branching is the return statement. The return statement exits the present technique, and the stream of power returns to where the technique was invoked. There are two types of the return declaration: one that yields a value, and one that does not. Simply place the value (or an entity calculating the value) after the return keyword to return a value.

return ++count;

The return value data type must match the declared return value of the method type. Use the return form that does not return a value when a method is declared void.

return;

Exceptions

An exception is an unwanted or unexpected event that occurs during program execution, i.e. runtime, that interrupts the normal flow of instructions from the program. Recognizing the distinction between an exception and an error is essential. An exception is the "outstanding occurrence" shorthand. It's a less severe mistake that a program can fairly cope with due to absence of a stronger word. On the other side, an mistake shows a important problem that is beyond the capacity of the programs to handle.

When an exception happens within a procedure, the process produces an event and hands it over to the runtime system (the Java Virtual Machine). The item, referred to as an exception object, involves exception information when the exception occurred, including its type and the status of the program. To create an exception object and hand it over to the runtime system, it is called throwing an exception.

After a technique generates an exception, the runtime system tries to find something to handle. The collection of feasible "somethings" for dealing with the exception is the ordered list of techniques called to the procedure where the exception occurred. The call stack is called the procedure list.

The runtime system searches the call queue for a method that contains a piece of software that can handle the exception. This block of code is called an exception handler. The experiment begins with the technique where the error occurred and proceeds in the opposite order where the techniques were called through the call stack. When a appropriate server is found, the runtime system transfers the exception to the server. If the type of exception item tossed matches the type that the server can handle, an exception handler will be considered appropriate.

The exception will be taken by the chosen exception handler. If the runtime system scans all methods on the call stack exhaustively without discovering an appropriate exception handler, the runtime system will deliver the exception object to the standard exception handler method which is a JVM component. This manager registers the exception information in the format below and terminates the program abnormally.

Exception in thread "thread name"

Name of Exception :

Description

Here’s an example of a program that throws an exception:

public class exampleException{

```
  public static void main(String[] args) {
    String example1 = null;
    System.out.println(example1.length());
  }
}
```

The output of this program would be as follows:

Exception in thread "main"

java.lang.NullPointerException

at exampleException.main(exampleException.java:8)

All kinds of exceptions and errors are class Throwable subclasses. Exception leads one branch. This category is used to capture user programs for outstanding circumstances. For instance, IllegalAccessException shows that it was not possible to find a specific method, and NegativeArraySizeException suggests that a program tried to produce an adverse size range.

The Java run-time system uses another branch, Error, to identify severe scheme mistakes related to the run-time setting (JRE) itself. An instance of such an error is StackOverflowError. Typical programs do not deal with errors.

Handling Exceptions

Five keywords are used to manage Java exception processing: **try, catch, throw, throws,** and **finally**. Here's a summary of how they work. Statements of the program that you believe may increase exceptions are enclosed in a Try block. If there is an exception within the try block, it will be tossed. Your software can catch and manage this exception in a rational way (using a Catch block). The Java runtime system automatically throws system-generated exceptions. Use the Throw keyword to manually add an exception. Any exception that is discarded from a method must be defined by a Throws provision as such.

try {

```
// block of code used to monitor for errors - this will be a block of code that throws an exception
//block of code you want to execute
}
catch (ExceptionType1 object) {
// exception handler for type of exception you believe may occur
}
catch (ExceptionType2 object) {
// exception handler for second type of exception you believe may occur
}

finally {
// any code within these brackets will be executed after the try block regardless of outcome
}
```

It is essential to note that there may be more than one declaration in a method that could cast an exception. Therefore, bring all these statements into your own try block and provide a distinct exception handler for each of them within your own capture block. If an exception falls within the test block, the exception handler affiliated with it will handle that exception.

We need to insert a catch function after it to connect it to an exception handler. There may be more than one provider of exceptions. Each capture block is an exception handler that manages its argument's type exception.

There may be zero or more capture points for each attempt block, but at last only one block. The finally block is optional. Whether an exception happened in the try block or not, it always gets performed. If an exception happens, after the attempt to capture blocks, it will be

performed. If there is no exception then after the test block it will be performed. Ultimately, the finally block in java is used to insert significant instructions such as cleaning up code, e.g. file closure or link closure.

Now that you know the general idea, note that you have to throw an exception to catch an exception. That said, how are we going to do this?

Throwing Exceptions

Exceptions from any source code can be tossed. They can be thrown from your code or a code from someone else's written package, such as the packages that come with the Java platform, or the Java runtime environment. Whatever casts the exception, the throw declaration is always tossed at it.

As you likely noticed, there are countless exception categories on the Java platform. All categories are Throwable class offspring, and all enable programs to distinguish between the different kinds of exceptions that may arise during program execution.

You can also create your own exception classes to represent problems that may occur in the classes you write. Indeed, if you are a product designer, you may need to create your own amount of exceptional lessons to enable customers to differentiate between a error that may occur in your product and errors that occur in the Java platform or other apps.

All methods use the throw statement to generate an exception. The throwing statement requires a single argument: an object that can be tossed. Throwable products are instances of any subclass of Throwable class. Here is an example of a throwing statement.

throw someThrowableObject;

Let's look at the throw statement in context. The following example method is taken from a class that implements a common stack object. The method removes the top element from the stack and returns the object.

```
private example() {
   Object example1;
   if (s == 10) {
      throw new EmptyStackException();
   }
   example1 = objectAt(s - 1);
   setObjectAt(s - 1, null);
   s--;
   return example1;
}
```

The instance technique tests whether there are any components on the stack. The technique generates and launches a fresh EmptyStackException item (a part of java.util) if the stack is vacant (its volume is equivalent to 0). You can generate your own exception courses, but all you need to remember for now is that you can only add items from the java.lang.Throwable class.,.,.

Chapter 7:
Making Our Program Interactive

Displaying Output

You can simply deliver input to the standard output (display) using System.out.println, System.out.print, or System.out.printf. Print() marks any sequence within parentheses. Println() marks the strings inside the parentheses and then proceeds to the next line start. Printf() offers the formatting of strings. System.out.format() is equivalent to printf() and can also be used.

Escape Sequences

There are special characters in Java that are preceded by a backslash and have specific duties within the compiler. These are called **Escape Sequences.**

\t Inserts a tab at the point it's inserted in the text.

\b Inserts a backspace at the point it's inserted in the text.

\n Inserts a newline at the point it's inserted in the text.

\r Inserts a carriage return at the point it's inserted in the text.

\f Inserts a form feed at the point it's inserted in the text.

\' Inserts a single quote character at the point it's inserted in the text.

\" Inserts a double quote character at the point it's inserted in the text.

\\ Inserts a backslash character at the point it's inserted in the text.

When an escape sequence is encountered in a print statement, the compiler interprets it accordingly.

Formatting Outputs

We've discussed formatting using printf(), but there are other helpful ways to format outputs. For example, **DecimalForma**t is used to

format decimal numbers.

```
import java.text.DecimalFormat;
class JavaFormatter2
{
  public static void main(String args[])
  {
    double value1 = 765.4321;
    // prints only numeric part of a floating number
    DecimalFormat ft = new DecimalFormat("####");
    System.out.println(": value1 = " + ft.format(value1));
    // print up to 2 decimal places
    ft = new DecimalFormat("#.##");
    System.out.println("value1 = " + ft.format(value1));
    // append zero to the rightmost part of a decimal
   // use digit 0 instead of #
    ft = new DecimalFormat("#.000000");
    System.out.println("value1 = " + ft.format(value1));
    // append zero to the leftmost part of a decimal
   // use # instead of 0
    ft = new DecimalFormat("00000.00");
    System.out.println("value1 = "+ft.format(value1));
   // formatting money in dollars
    double income = 98765.432;
    ft = new DecimalFormat("$###,###.##");
    System.out.println(ft.format(income));
  }
```

```
}
```

The output of this program is:

```
value1 = 765
value1 = 765.43
value1 = 765.432100
value1 = 00765.43
$98,765.43
```

You can also format dates using a class found in the java.text package. It is known the **SimpleDateFormat** class.

```
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
class Demo
{
  public static void main(String args[]) throws ParseException
  {
    // Argument provides the format
     SimpleDateFormat example1 = new SimpleDateFormat("dd-MM-yyyy");
    String example2 = example1.format(new Date());
    System.out.println("Formatted Date : " + example2);
    //String parsing
    example2 = "04/10/2005";
    example1 = new SimpleDateFormat("MM/dd/yyyy");
    Date example3 = example1.parse(example2);
    // prints date as a parsed string
    System.out.println("Parsed Date : " + example3);
```

```
  }
}
```

The output of this program is:

Formatted Date : 09-07-2019

Parsed Date : Sat Apr 10 00:00:00 UTC 2005

Accepting User Input

There are several methods Java user input can be obtained. In this chapter, you will discover how to get input using the Scanner item. You need to import Scanner category for this purpose using:

import java.util.Scanner;

Then, we will create an object of Scanner class which will be used to get input from the user

```
import java.util.Scanner;
class Input {
  public static void main(String[] args) {

      Scanner input = new Scanner(System.in);

      System.out.print("Integer Entry: ");
      int example1 = input.nextInt();
      System.out.println("Integer entered " + example1);
  }
}
```

If this program is ran and you enter the number 40 the output of this program will be:

Integer Entry: 40

Integer entered: 40

Here, input object of Scanner class is created. Then, the nextInt() method of the Scanner class is used to get integer input from the user.This scanner will only accept integers! In order to get other types it must be specified using nextLong(), nextFloat(), nextDouble() and next() methods for longs, floats, double, and strings, respectively.

Chapter 8:
Object Oriented Programming Pt. 1

If you have never previously used an object-oriented programming language, you will need to know some fundamental ideas before you can actually start composing any software. Obviously, we were able to get this far just fine and we have already touched on many of these ideas in past sections. However, if you really want to get serious about mastering Java, it is essential to have a strong understanding of all these ideas in comparison to each other. This section will introduce you to subjects such as artifacts, courses, heritage, interfaces, and applications, or enhance them. Each debate focuses on how these ideas relate to the actual globe while offering a deeper understanding of the syntax used in the programming language of Java at the same time. We're going to consider some concepts:

Objects

An object is an associated state and conduct software package. Software objects are often used to model the objects you find in everyday life in the real world. This section describes how the state and conduct within an item is depicted, presents the notion of data encapsulation and illustrates the benefits of software design.

Classes

A class is a blueprint or prototype that creates objects from. This chapter describes a category that models a real-world object's state and conduct. It deliberately relies on the basics, demonstrating how state and conduct can be cleanly modeled even by a straightforward class.

Inheritance

Inheritance provides a powerful and natural structure for organizing and structuring your software. This section explains how classes, using the simple syntax of the Java programming language, derive

situation and behavior from their superclasses and how to acquire one class from another.

Interface

An interface is an external world-class agreement. It ensures that when a class utilizes an api, it will provide the behavior published by that implementation. This section defines a simple interface and explains the changes required for any class to execute it.

Package

A bundle is a namespace in which classes and features can be logically organized. Placing your code in packages makes large software initiatives easier to manage. This section explains why this is useful and takes you to the Application Programming Interface (API) of the Java platform.

Objects

Objects are essential to knowing technology that is oriented towards objects. Look around right now and you'll discover a lot of real-world object examples: your dog, your office, your TV set, your cycle.

Objects in the real world share two features: they all have state and conduct. Dogs have condition (name, colour, race, hunger) and conduct (barking, fetching, tail wagging). cycles also have condition and conduct (present equipment, present pedal cadence, present velocity) (altering equipment, altering pedal cadence, braking). Identifying the state and conduct of real-world items in terms of object-oriented programming is a wonderful place to start thinking.

Take a minute to see the real-world objects in your immediate area right now. Ask yourself two issues for each item you see: "What state can this object be in?" and "What conduct can this object be in?." Make sure your comments are written down. As you do, you will realize that real-world items differ in size; your bedroom, hallway, and kitchen lights may only have two feasible states (on / off) and two feasible habits (on / off), but your desktop radio may have extra states (on / off, present frequency, present location) and habits (on / off, off, frequency boost, frequency reduce, search, scan, and

mode). You may also realize that some items will in turn contain other items as well. All of these real-world findings translate into the object-oriented programming world.

Software objects are conceptually comparable to items in the real world: they also comprise of state and associated behaviour. An item shops its state in areas (variables in some parts of programming) and exposes its conduct through techniques (features in some forms of programming). Methods work on the united state of an object and act as the main system for communication between object and object. Hiding the inner condition and demanding all communication through the techniques of an object is regarded as information encapsulation— a basic concept of object-oriented programming.

Consider, for instance, a cycle. By assigning condition (present velocity, present pedal cadence, and present equipment) and offering techniques to change that state, the item stays in command of how it can be used by the outside world. For example, if the cycle has only 6 gears, any value less than 1 or greater than 6 could be rejected by a method of changing gears.

A range of advantages include bundling code into individual software objects:

Information-hiding: The details of its inner execution stay concealed from the outside world by interacting only with the techniques of an object.

Modularity: An object's source code can be written and maintained for other objects regardless of the source code. Once created, an object within the system can be easily passed around.

Pluggability and debugging ease: If an item is difficult, you can merely extract it from your database and insert it into another item as its substitute. This is similar to solving real-world mechanical problems. You substitute it if a bolt falls, not the whole device.

Code re-use: You can use it in your program if an object already exists (maybe developed by another software developer). This

allows specialists to run complex task-specific objects in your own software to implement/test/debug.

Class

Many objects of the same kind are often found in the real world. There may be thousands of other cycles, all of the same form and pattern. Built from the same blueprint set, the same sections are included in each bicycle. We argue that your cycle is an instance in object-oriented terms of the class of artifacts identified as cycles. A class is the blueprint where private products are created.

One feasible cycle application is the following cycle category:

```
class cycle {
   int c = 0;
   int s= 0;
   int g = 1;
   void C(int number) {
       c = number;
   }
   void G(int number) {
       g = number;
   }
   void Speed(int number) {
       s++;
   }
   void Brakes(int number) {
       s--;
   }
   void printStates() {
       System.out.println("c:" +
```

```
            c + " s:" +
            s + " g:" + g);
    }
}
```

This class ' layout is focused on the prior cycle artifacts debate. Fields c, s, and g depict the condition of the object, and techniques (C, G, Speed, etc.) describe its external environment relationship.

You might have observed that there is no primary technique in the cycle category. That's because it's not a full implementation; it's just the cycle blueprint that could be used in an implementation. In your implementation, the obligation to create and use fresh cycle items goes to some other category.

Here is a cycleDemo category that produces two cycle items separately and invokes their methods:

```
class Demo {
    public static void main(String[] args) {
      // Create two unique Cycle objects
       cycle b1 = new Cycle();
      cycle b2 = new Cycle();
      // Invokes methods on those cycles
       b1.C(23);
       b1.Speed(3);
       b1.G(6);
       b1.printStates();
       b2.C(23);
       b2.Speed(3);
       b2.G(6);
       b2.C(40);
```

```
        b2.Speed(14);
        b2.G(4);
        b2.printStates();
    }
}
```

Declaring Classes

You've seen classes defined in the following way:

```
class exampleClass {
   // Methods, fields, and constructors are contained here
}
```

That's a class declaration. The class body (the area between the braces) contains all the code that provides for the life cycle of the class-created objects: builders to initialize new objects, statements for the fields that provide the state of the class and its objects, and methods to implement the class's behavior and its objects.

The prior class declaration is a minimum declaration. It involves only those components of a class declaration that are essential. At the beginning of the class statement, you can provide more data about the class, such as its superclass title, whether it uses any features, etc. For example:

```
class exampleClass extends exampleSuperClass implements exampleInterface {
   //More methods, fields, and constructors
}
```

This implies exampleClass is a SuperClass subclass and uses the Interface user example.

At the very beginning, you can also insert modifiers such as government or private— so you can see that a class declaration's starting line can become quite complex. In this class, the government and personal modifiers that determine what other

courses can access exampleClass will be discussed subsequently. The interface and inheritance lesson will clarify how and why you'd use the extensions in a class statement and implement keywords. You don't have to care about these additional problems for the moment.

In general, class declarations can include these components, in order:

Public, private, and other modifiers. Modifiers will be discussed later on.

Class names are required. Typically, the initial letter is capitalized for ease of recognition.

Extends the parent (superclass) name of the class, if any, preceded by the keyword. Only one parent can be extended by a class (subclass).

A comma-separated list of class-implemented interfaces, if any, preceded by keyword features. A class can use more than one device.

The class body, surrounded by braces {}.

Fields

As you have seen, you can often provide an initial value for a field in its declaration:

```
public class Example {
   // we set to 20
   public static int example1 = 20;
  // We set to true
   private boolean example2 = true;
}
```

If the initialization price is accessible and the initialization can be placed on one row, this operates well. However, due to its ease, this type of initialization has constraints. If initialization needs some logic

(e.g., error handling or a loop to complete a complicated range), it is insufficient to simply assign. Instance variables can be initialized in constructors where it is possible to use error handling or other logic. The Java programming language involves static initialization blocks to provide the same capacity for class variables. At the start of the class definition, areas need not be declared, although this is the most popular practice.

Before they are used, they only need to be specified and initialized. A static initialization block is a standard code block containing { } in braces, followed by the static keyword. Here is an instance of this:

```
static {
    //code used to initialize goes here
}
```

A class may have a number of static initialization blocks, and they may appear in the class body anywhere. The runtime system ensures that the order in which they appear in the source code will call static initialization blocks.

There's an option to static blocks— a private static method can be written:

```
class Example {
    public static Type example1 = initializeExampleVariable();
    private static Type initializeExampleVariable() {
        //code used to initialize goes here
    }
}
```

Private static methods have the advantage that they can be reused later if the class variable needs to be reinitialized.

You would normally placed code in a constructor to initialize an instance variable. To initialize example variables, there are two solutions to using a constructor: initializer blocks and initial techniques.

For example, initializer blocks appear like static initializer blocks, but without the linear keyword:

```
{
  //code used to initialize goes here
}
```

The Java compiler copies parts of the initializer into each builder. This strategy can therefore be used to share a code block between various builders. A final method in a subclass can not be overridden. Here is an example of how to initialize an instance variable using a final technique:

```
class Example {
   private type example1 = initializeExampleVariable();
   protected final type initializeExampleVariable() {
      //code used to initialize goes here
   }
}
```

This is particularly helpful if the initialization technique may be reused by subclasses. The technique is final because it can trigger issues when calling non-final techniques during initialization of instances.

Methods

Here is an example of a typical method declaration:

```
public double example(double var1, int var2,
                      double var3, double var4) {
   //calculation are performed here
}
```

A method declaration's only required elements are the return type of the method, the name, a pair of parentheses and a body between braces,{}. Method declarations have six parts in order more usually; modifications— such as government, personal, and others you'll hear about later. The end type— the value received by the process sort of information, or null if a value is not received by the procedure. The name of the method— the domain boundaries laws also apply to the names of the technique, but the protocol is a bit distinct. The list of parameters in parenthesis— a comma-delimited list of input parameters preceded by their information kinds, including brackets().If there are no parameters, blank parentheses must be used. A list of exceptions. The body of the procedure, wrapped between braces — the documentation of the procedure, including local variables statement, runs here.

Two of the components of a method declaration comprise the method signature—the method's name and the parameter types.

The signature of the method declared above is:

examle(double, int, double, double)

Naming Methods

While any legal identifier may be a name for a method, code conventions restrict the names of methods. By convention, process names should be a lowercase verb or multi-word name that starts with a lowercase verb, accompanied by adjectives, nouns, etc. The first element of each of the second and subsequent phrases should be capitalized in multi-word titles. A technique typically has a distinctive name in its category. However, owing to overloading technique, a technique may have the same name as other techniques.

Method Overloading

The Java programming language promotes techniques of overloading, and with distinct method signatures, Java can differentiate between techniques. This implies that techniques can

have the same name within a class if they have distinct lists of parameters.

Suppose you have a class that can use calligraphy to write different kinds of information (strings, integer, etc.) and contain a technique to write each sort of information. Using a fresh name for each method is cumbersome— such as drawString, drawInteger, drawFloat, etc. You can use the same name for all drafting techniques in the Java programming language, but you can transfer a distinct argument list to each technique. Thus, the category of information drawing could designate four techniques termed draw, each having a distinct list of parameters.

```
public class artistArtist {
    ...
    public void makeArt(String s) {
        ...
    }
    public void makeArt(int i) {
    }
    public void makeArt(double f) {
        ...
    }
    public void makeArt(int i, double f) {
        ...
    }
}
```

The amount and sort of statements carried into the technique differentiate overloaded techniques. Draw(s) and draw(int I) are separate and unique techniques in the software sample because they involve different kinds of arguments.

With the same name and the same number and type of arguments, you can not declare more than one method as the compiler can not distinguish them.

The compiler does not consider return type when differentiating techniques, so even if they have a different return type, two methods with the same signature can not be created.

Constructors

A class includes builders that are invoked from the class blueprint to generate items. Builder statements sound like statements of method — except that they use the class name and have no sort of exchange. cycle, for instance, has one builder:

```
public cycle(int C, int S, int G) {
   g = G;
   c = C;
   s = S;
}
```

To create a new cycle object called example1, a constructor is called by the new operator:

```
Cycle example1 = new Cycle(5, 10, 4);
```

The “new Cycle(5, 10, 4)” Create object memory space and initialize the areas of the object. While Cycle has only one builder, it may have others, including a no-argument builder:

```
public Cycle() {
   g = 3;
   c = 15;
   s = 20;
}
```

Cylce example1 = new Cycle(); invokes the no-argument constructor to create a new Cycle object called example1.

Because they have distinct reasoning sheets, both constructors could have been proclaimed in cycle. As with techniques, on the grounds of the amount of alternatives in the list and their kinds, the Java platform differentiates constructors. You can't write two constructors with the same number and type of arguments for the same class because they couldn't be told apart by the platform. Doing so creates an mistake in compiling time.

You don't have to provide your class with any builders, but when doing this you have to be cautious. For any class without constructors, the compiler automatically offers a no-argument, default constructor. This standard builder will call the superclass's no-argument constructor. In this situation, if the superclass does not have a no-argument constructor, the compiler will complain, so you have to check that it does. If your class doesn't have an explicit superclass, it has an implicit Object superclass, which has a no-argument builder.

You can use yourself a super-class builder. At the start of this lesson, the Mountaincycle school did just that. In a constructor's statement, you can use access modifiers to control which other classes can call the constructor.

Chapter 9:
Object Oriented Programming Pt. 2

Inheritance

Different types of artifacts are often associated with a certain quantity.For example, all share cycle characteristics (current speed, current pedal rhythm, current facilities) are mountain cycles, road bicycles and tandem bicycles. But each also recognizes additional features that make them special: tandem bicycles have two seats and two handlebar sets; road cycles have loose handlebars; some mountain cycles have an additional chain ring that provides them a decreased gear ratio. Object-oriented programming allows courses to inherit the frequently used state and behavior from other classes.

Cycle now becomes Mountaincycle, Roadcycle, and Tandemcycle's superclass in this instance. Each class can have one direct superclass in the Java programming language, and each superclass has the potential for an unlimited number of subclasses.

The syntax is easy to create a subclass. Use the expands keyword at the start of your class statement, followed by the class title to inherit from:

```
class MB extends B {
   //fields and methods are written here
}
```

This provides MB the same areas and techniques as B, but enables its software to concentrate solely on the distinctive characteristics. This makes it simple to read code for your subclasses. However, you must be careful to correctly record the state and conduct defined by each superclass, as that code will not appear in each subclass's source file. Below is significant inheritance understanding terminology:

Super Class: The class whose features are inherited is known as super class(or a base class or a parent class).

Sub Class: The class inheriting the other class is referred to as a sub class (or a derived class, extended class, or child class). Besides the superclass areas and techniques, the subclass can add its own areas and techniques.

Reusability: Inheritance promotes the notion of "reusability," i.e. we can draw our fresh class from the current class if we want to produce a fresh class and there is already a class that contains some of the software we want. By doing this, we reuse current class areas and techniques.

Using Inheritance in Java

The keyword used for inheritance is extends. an example of the syntax is below:

```
class exampleClass1 extends exampleClassOne
{
   //code for fields and methods, etc
}
```

By using the object of the subclass we can also access the members of a superclass. Please note that during inheritance only object of subclass is created, not the superclass.

Types of Inheritance

Single Heritage: subclasses retain in a single inheritance the features of one superclass.

Multilevel Inheritance, a derivative class inherits a base class and the derivative class also acts as the base class for another class.

Hierarchical Heritage: One category in Hierarchical Heritage functions as a superclass (base category) for more than one subclass.

Multiple Inheritance (Through Interfaces): A class may have multiple inheritance features of more than one superclass and originate from all parent organizations. Note that Java does not support multiple

inheritance with courses. In java, we can achieve various inheritance only through interfaces.

Hybrid Inheritance(Through Interfaces): a combination of two or more of the above-mentioned inheritance types. Because java does not assist with classifications multiple inheritance, classes do not support hybrid inheritance either. In java, we can achieve hybrid inheritance only through interfaces.

Polymorphism

Polymorphism is an object's capacity to assume many types. Polymorphism is most commonly used in OOP when a parent class reference is used to refer to an item of child class.

It is essential to understand that a reference variable is the only way to access an item. Only one set can be a reference variable. Once declared, it is not possible to change the type of a reference variable. It is possible to reassign the reference variable to other items if it is not proclaimed final. The reference variable type would determine the techniques on the item that it can invoke.A reference variable can refer to any specified type item or any specified type subtype. It is possible to declare a reference variable as a category or sort of interface.

Abstraction

Abstraction, as per the dictionary, is the standard of coping with thoughts instead of occurrences. For example, when you consider the e-mail case, complex details such as what happens when you send an e-mail, the user's protocol is hidden from your e-mail server. Therefore, just type the material, specify the recipient's address, and press submit to submit an e-mail.

In object-oriented programming as well, abstraction is a method of hiding the user's execution information, only the user will be supplied with the features. In other words, the reader will have the data about what the item is doing rather than how it is doing it.

In Java, abstraction is achieved using Abstract classes and interfaces.

Abstract Classes

A class which contains the abstract keyword in its declaration is known as an abstract class.

Abstract classes may or may not contain abstract methods, i.e., methods without body (public void get();)

But, if a class has at least one abstract method, then the class must be declared abstract.

If a class is declared abstract, it cannot be instantiated.

To use an abstract class, you have to inherit it from another class, provide implementations to the abstract methods in it.

If you inherit an abstract class, you have to provide implementations to all the abstract methods in it.

Abstract Methods

If you want a class to contain a particular method but want children's classes to determine the actual implementation of that method, you can declare the method as an abstract in the parent class.

Abstract keyword is used to declare the method as abstract.

You have to place the abstract keyword before the method name in the method declaration.

An abstract method contains a method signature, but no method body.

Instead of curly braces, an abstract method will have a semicolon (;) at the end.

Interfaces

As you have already learned, by the techniques they disclose, artifacts describe their relationship with the outside world. Methods form the interface of the object with the outside world; for example, the buttons on the front of your TV set are the interface between you

and the electrical wiring on the other side of your plastic casing. To switch on and off the TV, click the "power" key.

An interface in its most popular type is a set of techniques linked to blank bodies. If indicated as an interface, the conduct of a Cycle may occur as follows:

```
interface Cycle {
    void C(int number);
    void G(int number);
    void Speed(int number);
    void B(int number);
}
```

To execute this interface, your class title would alter (e.g. to a specific Cycle brand such as ExampleCycle) and you would use the keyword tools in the class statement:

```
class ExampleCycle implements Cycle {
    int c = 10;
    int s = 15;
    int g = 11;
   // The compiler will now require that methods
   // C, G, Speed, and B
   // all be implemented. Compilation will fail if those
   // methods are missing from this class.
    void C(int number) {
        c= number;
    }
    void G(int number) {
        g = number;
    }
```

```
    void Speed(int number) {
        s++;
    }
    void B(int number) {
        b--;
    }
    void printStates() {
        System.out.println("c:" + c + " s:" +  s + " g:" + g);
    }
}
```

Implementing an api allows a class to become more formal about the behavior it promises. Interfaces form an arrangement between the class and the outside world, and at the time of building the compiler enforces this contract. If your class says to implement an api, all the methods indicated by that interface must occur in their source code before the class compiles efficiently. An api is a reference type of Java. It's a college like that. It's a collection of abstract methods. A class utilizes an api to inherit the abstract techniques of the interface. An interface, along with abstract procedures, may also include constants, conventional techniques, static methods, and nested shapes. There are method organs for standard techniques and static methods only.

It is comparable to creating a class to write an api. However, a category defines an object's characteristics and behaviors. And an api includes habits implemented by a class. Unless the application implementing class is abstract, all input techniques in the class must be described.

An interface is similar to a class in the following ways: any number of methods may be included in an interface. In a folder with a.java expansion, an api is published, with the button number matching the file name. An interface's byte code appears in a folder of.class.

Packages contain interfaces and their respective bytecode file must be in a directory structure that fits the title of the parcel.

In several ways, an instrument is different from a class, including: you can't install an interface. There are no builders in an interface. All the input techniques are abstract. An interface can't comprise areas of example. It is necessary to declare both static and ultimate the only areas that can occur in an interface. A class does not extend an api; a class implements it. Several applications can be extended by an api.

Declaring Interfaces

The interface keyword is used to declare an interface. Here is a simple example to declare an interface:

Example.java

```
import java.lang.*;

public interface Example {

  //Final, static fields along with abstact methods. These can be declared in any amount.

}
```

Interfaces have the following properties:

An interface is implicitly abstract. You do not need to use the abstract keyword while declaring an interface.

Each method in an interface is also implicitly abstract, so the abstract keyword is not needed.

Methods in an interface are implicitly public.

Implementing Interfaces

You can believe of the class as signing a agreement when a class uses an application, agreeing to conduct the interface's particular activities. If a class does not execute all the interface activities, the class must declare itself to be abstract. A class utilizes keyword tools to deploy an api. The keyword of implements appears in the class statement following the declaration's expands part.

```
public class ExampleInt implements Numbers {
  public void action() {
    System.out.println("action occurs");
  }
  public void action2() {
    System.out.println("action2 occurs");
  }
  public int values {
    return 0;
  }
  public static void main(String args[]) {
    ExampleInt m = new ExampleInt();
    m.action();
    m.action2();
  }
}
```

Output:

action occurs

action2 occurs

When overriding methods defined in interfaces, there are several rules to be followed:

Checked exceptions should not be declared on methods of implementation other than those declared by the method of interface or subclasses declared by the method of interface.

When overriding the methods, the interface method signature and the same type or subtype of return should be maintained.

An application category itself can be abstract and interface techniques do not need to be enforced if so.

When implementing interfaces, there are several rules:

A class can implement more than one interface at a time.

A class can extend only one class, but implement many interfaces.

An interface can extend another interface, in a similar way as a class can extend another class.

Package

A package is a namespace organizing a collection of courses and interfaces ##s to it. Conceptually, you can believe of packages on your desktop as comparable to various folders. You may keep HTML pages in one folder, images in another folder, and scripts or apps in another folder. Because Java programming language-written software can consist of hundreds or thousands of individual courses, it makes sense to maintain stuff arranged by putting associated courses and interfaces into applications.

The Java platform provides a enormous class library (a package collection) for use in your own applications. This library is called the "Application Programming Interface" or, in short, "API." For particular reasons, its components are the features most commonly associated with programming. For example, a String object involves character string status and behaviour; a File object allows a programmer to create, delete, inspect, attach, or change a file on the file system easily; a Socket object allows to create and use network sockets; various GUI products control buttons and checkboxes and anything else connected with graphical user interfaces.

There are literally thousands of classes to choose from. This enables you, the programmer, to concentrate not on the infrastructure needed to make it function, but on your application's layout.

Chapter 10:
File Handling

This chapter discusses the details of reading, writing, creating, and opening files. There are a wide array of file I/O classes and methods to choose from.

Reading a text File

Reading a text file is a crucial ability in Java and has many practical applications. **FileReader**, **BufferedReader**, and **Scanner** are useful classes for reading a plain text file in Java. Each of them possess specific qualities that make them uniquely qualified to handle certain situations.

BufferedReader

This technique reads text from a stream of character input. It buffers characters, arrays, and rows to be read efficiently. You can specify the buffer type or use the standard type. For most reasons, the default is big enough. In particular, each read application produced from a Reader leads the fundamental personality or byte stream to make a respective read application. Therefore, it is advisable to wrap a BufferedReader around any Reader whose read) (transactions, like FileReaders and InputStreamReaders, can be expensive.

For example:

BufferedReader in = new BufferedReader(Reader in, int size)

FileReader

Class of convenience to read character documents. This class ' constructors suppose the default character format and the default byte-buffer size is suitable.

Scanner

A simple text scanner that uses regular expressions to parse primitive types and strings. A Scanner uses a delimiter model to break its entry into tokens that suits white space by definition. Using

distinct next techniques, the resulting tokens can then be transformed into values of distinct kinds.

```
import java.io.File;
import java.util.Scanner;
public class ReadFromFileUsingScanner
{
  public static void main(String[] args) throws Exception
  {
    // filepath is set as a parameter now so that it can be scanned
    File example =
      new File("C:\\Users\\userName\\Desktop\\example.txt");
    Scanner example1 = new Scanner(file);
    while (example1.hasNextLine())
      System.out.println(example1.nextLine());
  }
}
```

Using Scanner class but without using loops:

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
public class example{
  public static void main(String[] args)
                    throws FileNotFoundException {
                                  File example = new
File("C:\\Users\\userName\\Desktop\\example.txt");
    Scanner example1 = new Scanner(file);
    // we will use \\Z as a delimiter
```

```
    sc.useDelimiter("\\Z");
    System.out.println(example1.next());
  }
}
```

Read a text file as String in Java

```
package io;
import java.nio.file.*;;
public class example{
  public static example(String fileName)throws Exception {
    String example1 = "";
                                          example1 = new
String(Files.readAllBytes(Paths.get(fileName)));
    return example1;
  }

  public static void main(String[] args) throws Exception
  {
                                        String example1 =
example("C:\\Users\\userName\\Desktop\\example.java");
    System.out.println(example1);
  }
}
```

Writing to a text file

You can use one of the write methods to write bytes, or lines, to a file. Write methods include:

write(Path, byte[], OpenOption...)

write(Path, Iterable< extends CharSequence>, Charset, OpenOption...)

Renaming and Deleting Files

Renaming

In Java, there's a method called renameTo(fileName) within the File class that we can use to rename a file.

Deleting Files

Files that are saved using the java program will be permanently removed without moving to the trash / recycle bin. Using **java.io.File.delete** deletes this abstract path name from the file or folder. Using **java.nio.file.files.deleteifexists** will delete a folder, if it occurs. If the folder is not open, it also deletes a folder listed in the route.

Chapter 11:
Advanced Topics In Java

Generics

In any non-trivial software project, bugs are simply a fact of life. Careful planning, programming, and testing may help diminish their omnipresence, but somehow they will always find a way to enter your system somewhere. This becomes especially apparent as new features are introduced and your code base's magnitude and complexity increases.

Fortunately, it is easier to detect some bugs than others. Compile-time bugs, for instance, can be identified soon; you can use the compiler's error codes to determine what the issue is and solve it, right then and there. Runtime bugs, however, can be much harder; they do not always occur instantly, and when they do, they may be at a point in the program far apart from the true cause of the problem. Generics help stabilize your software by creating it feasible at compile time to identify more of your bugs.

Generics, in a nutshell, allow parameters for kinds (classes and objects) when defining courses, interfaces and techniques. Like the more familiar formal parameters used in declarations of methods, type parameters provide you with distinct outputs to re-use the same code. The distinction is that values are the inputs to formal parameters, while kinds are the answers to type parameters. Code using generics has many advantages over non-generic code. Through the use of generics, programmers can introduce generic algorithms that operate on distinct kinds of collections, can be tailored, and are secure and simpler to read form.

Generic Types

A generic form is a types-parameterized generic category or interface. It is possible to modify a easy box class to show the idea. Consider a non-generic box category that works on any sort of object. It only requires to provide two techniques: set), (adding an

item to the cabinet, and get), (retrieving it. Because their methods accept or return an object, you are free to pass in whatever you want, as long as it is not one of the primitive types. There is no way to check how the class is used at compile time.One portion of the software may put an integer in the cabinet and expect to get integers out of it, while another portion of the software may erroneously move through a string, leading in an mistake in runtime.

The segment of type parameter, delegated by angle brackets (< >), displays the title of the category. It indicates the parameters of form (also known as factors of sort) T1, T2, till T. You generate a generic type statement by altering the file "government class box" to "government class box <T >" to update the box category to use generics. This presents the type variable, T, which can be used inside the class anywhere. This replaces all Object events with T. Any non-primitive type you specify can be a type variable: any type of class, any type of interface, any type of array, or even some other type variable. It is possible to apply this same method to generic interfaces.

Type designations are single, upper case letters by convention. This contrasts sharply with the variable naming conventions you already understand about, and with excellent reason: it would be hard to say the distinction between a type variable and an normal class or object name without this convention.

The most commonly used type parameter names are:

E - Element (used extensively by the Java Collections Framework)

K - Key

N - Number

T - Type

V - Value

S,U,V etc. - 2nd, 3rd, 4th types

You'll see these names used throughout the JDK and the API.

Invoking and Instantiating a Generic Type

To mention the generic box category within your system, a generic type invocation must be performed that brings T with a certain concrete value, such as Integer:

Box<Integer> integerBox;

You may believe that an invocation of a generic sort is comparable to an normal process invocation, but instead of adding an assertion to a procedure, you transfer a type argument — Integer in this case — to the box category itself.

Many designers interchangeably use the words "type parameter" and "type statement," but not the same definitions. When coding, to generate a parameterized type, one offers sort arguments. The T in Foo < T > is therefore a type parameter and the Foo < String > f string is a type contention. In using these words, this class follows this concept.

Like any other statement of variable, this software does not generate a fresh item of the box. It merely states that integerBox will have a reference to an "Integer Box," which is how it reads Box < Integer>. A generic type invocation is usually referred to as a parameterized type.

To instantiate this class, use the new keyword, as usual, but place <Integer> between the class name and the parenthesis:

Box<Integer> integerBox = new Box<Integer>();

The Diamond

In the latest versions of Java, you can replace the type arguments needed to invoke a generic class constructor with an empty set of type arguments (< >) as long as the compiler can determine, or infer from the context, the type arguments. This angle bracket couple, < >, is called The Diamond loosely. For example, you can use the previous declaration to generate an instance of Box < Integer>:

Box<Integer> integerBox = new Box<>();

Generic Methods

Generic techniques are techniques which implement parameters of their own sort. This is similar to a generic type declaration, but the scope of the type parameter is limited to the method in which it is declared. In addition to generic class constructors, static and non-static generic techniques are permitted.

The syntax for a generic technique involves a list of parameters of type inside angle brackets that appear before the return type of the procedure. The type parameter segment must occur before the return type of the method for static generic techniques. The Util class involves, compare, a generic technique comparing two Pair items.

Pair<Integer, String> ex1 = new Pair<>(49, "string1");

Pair<Integer, String> ex2 = new Pair<>(64, "string2");

boolean comparison = Util.<Integer, String>compare(ex1, ex2);

Bounded Type Parameters

Sometimes you want to limit the kinds that can be used in a parameterized type as form statements. For instance, a technique that works on figures could only recognize Number cases or its subclasses. That's what limit parameters of the sort are for.

List the name of the type parameter to indicate a defined type parameter, accompanied by the extensions keyword, followed by the upper bound, which is Number in this instance. Note that expands is generally used in this context to mean either "extends" (as in courses) or "implements" (as in interfaces).

Multiple Bounds

Type parameters can have more than one bound. A type variable with multiple bounds is a subtype of all the types listed in the bound. If one of the bounds is a class, it must be specified first. For example:

Class X { /* ... */ }

interface Y { /* ... */ }

interface Z { /* ... */ }

class R <T extends X & Y & Z> { /* ... */ }

If bound X is not specified first, you get a compile-time error:

class R <T extends Y & X & Z> { /* ... */ } // compile-time error

Chapter 12: Collections

A **Collections Framework** is a unified architecture for representing and manipulating collections. All collections frameworks contain the following:

Interfaces: These are kinds of abstract information representing collections. Interfaces permit the manipulation of collections regardless of the information of their depiction. Usually, interfaces create a hierarchy in object-oriented languages.

Implementations: These are the collection interfaces' tangible applications. In principle, they are data structures that can be reused.

Algorithms: These are the techniques that conduct helpful computations on items that execute functions for compilation, such as searching and sorting. It is said that the algorithms are polymorphic: that is, the same technique can be used on the suitable collection interface in many separate applications. Essentially, algorithms are features that can be reused.

Java Collections Framework

The Java Collections Framework offers the following advantages: By offering helpful data structures and algorithms, the Collections Framework allows you to focus on the significant components of your program rather than the low-level "plumbing" needed to make it function. The Java Collections Framework allows you to write adapter items or transformation code to link APIs by enabling interoperability between unrelated APIs.

This Collections Framework offers helpful data structures and algorithms with high-performance, high-quality applications. Each interface's different applications are interchangeable, allowing programs to be readily customized by changing applications of

collections. Because you are freed from the drudgery of writing your own data structures, you will have more time to enhance the quality and performance of the programs.

The interfaces of the compilation are the vernacular by which APIs move back and forth records. If my Network Administration API provides a set of node numbers and your GUI toolkit assumes a set of column headings, our APIs will interoperate seamlessly, even if they have been published separately.

Naturally, many APIs bring input sets and provide them as output. Each of these APIs had a tiny sub-API dedicated to controlling their collections in the past. These ad hoc collections sub-APIs had little consistency, so you had to know each from scratch, and making errors when using them was simple. The issue disappeared with the introduction of conventional collection interfaces.

This is the prior advantage's flip side. Designers and implementers do not have to reinvent the wheel each time they produce a collection-based API; instead, they can use conventional tools for collection.

New data structures that comply with normal interfaces for compilation are reusable by nature. The same applies to fresh algorithms operating on items implementing these interfaces.

In addition to the Java Collections Framework, C++ Standard Template Library (STL) and Smalltalk's collection hierarchy are the best known instances of collection frameworks. Historically, frameworks for libraries have been quite complicated, giving them a reputation for a steep learning curve. We think that this tradition breaks with the Java Collections Framework.

Autoboxing and Unboxing

The Java compiler makes automatic conversions between primitive types and their corresponding object wrapper classes. This conversion is known as **Autoboxing.** The conversion of an int to an Integer is an example of autoboxing. The same can be performed on

doubles, converting them to Doubles. This pattern is repeated for the other primitive types.

Here is an example of autoboxing using the char primitive type:

```
Character cha = 'b';
```

The rest of the examples in this section use generics. Generics are discussed in Chapter 11.

Consider the following code:

```
List<Integer> ex1 = new ArrayList<>();
for (int j = 1; j < 50; j += 2)
   ex1.add(j);
```

Even if you add the int numbers to ex1, the software compiles as primitive types, rather than Integer items. Since ex1 is a list of Integer objects, not a list of int values, you might wonder why the Java compiler is not making a compile-time error. No mistake is generated by the compiler because it generates an Integer item from I and connects the item to ex1. The compiler therefore transforms the prior software at runtime to the following:

```
List<Integer> ex1 = new ArrayList<>();
for (int j = 1; j < 50; j += 2)
   ex1.add(Integer.valueOf(j));
```

It is called autoboxing to convert a primitive value (e.g. an int) into an object of the corresponding wrapper class (Integer). When a primitive value is: passed as a parameter to a method which expects an object of the corresponding wrapper class, the Java compiler applies autoboxing. Assigned to the appropriate wrapper class variable.

Consider the following method:

```
public static int example(List<Integer> ex1) {
   int value = 0;
   for (Integer j: ex1)
```

```
        if (j % 2 == 0)
            value += j;
        return value;
}
```

Since the remaining officers (percent) and unary plus (+ =) do not apply to Integer products, you may be wondering why the Java compiler compiles the method without creating any errors. The compiler does not make a error because it invokes the process intValue to convert an integer to an int at runtime:

```
public static int example(List<Integer> ex1) {
    int value = 0;
    for (Integer j : ex1)
        if (j.intValue() % 2 == 0)
            value += j.intValue();
        return value;
}
```

Autoboxing's reverse is Unboxing. This method transforms a wrapper type item to a primitive type like and int or double like an Integer or Double. When a wrapper class object is: passed to a method that requires a primitive type as a value to be used, the Java compiler applies unboxing. Assigned to a respective primitive type variable.

The following example demonstrates this property:

```
import java.util.ArrayList;
import java.util.List;
public class Demo{
    public static void main(String[] args) {
        Integer j = new Integer(15);
        //Method invocation
```

```
        int var1 = absoluteValue(j);
        System.out.println(var1);
        //Assignment Invocation
        double pie = pie.get(0);
        System.out.println(pie);
    }
    public static int absoluteValue(int j) {
        return (j < 0) ? -j : j;
    }
}
```

Autoboxing and unboxing lets developers write cleaner code, making it easier to read.

Lists

The ArrayList, LinkedList, Vector and Stack classes all share something in common. These classes all implement the **List Interface**. The list interface allows positional access/insertion of elements by preserving insertion order. Java.util.List is a child interface of Collection. It is an ordered collection of objects in which duplicate values can be stored.

Linked List

A **Linked List** is a list that contains elements that are stored in non-sequential locations. Every element in a linked list has a data section and an address section. How do we link these elements? Using addresses and pointers. An element in a linked list is referred to as a node. These structures are preferred over arrays due to ease of insertion/deletion as well as how dynamic they are. There are distinct disadvantages howevers. For one, nodes must be accessed indirectly. This is accomplished by starting at the head and following links until a node is found. Storing elements in a linked list requires using the LinkedList class which provides an abstract class, implement list, and necessary list interfaces.

The **LinkedList() method** is used to initialize and empty linked list while **LinkedList(Collection C)** is used to create a list containing all elements of a specific collection.