PROGRAMMING
COMPUTER PROGRAMMING FOR BEGINNERS
Learn the Basics of Java, SQL & C++
SECOND EDITION
JOSEPH CONNOR
Programming:

Computer Programming for Beginners

Learn the Basics of Java, SQL & C++

Joseph Connor
© Copyright 2015 - All rights reserved.

In no way is it legal to reproduce, duplicate, or transmit any part of this document in either electronic means or in printed format. Recording of this publication is strictly prohibited and any storage of this document is not allowed unless with written permission from the publisher. All rights reserved.

The information provided herein is stated to be truthful and consistent, in that any liability, in terms of inattention or otherwise, by any usage or abuse of any policies, processes, or directions contained within is the solitary and utter responsibility of the recipient reader. Under no circumstances will any legal responsibility or blame be held against the publisher for any reparation, damages, or monetary loss due to the information herein, either directly or indirectly.

Respective authors own all copyrights not held by the publisher.

Legal Notice:
This book is copyright protected. This is only for personal use. You cannot amend, distribute, sell, use, quote or paraphrase any part or the content within this book without the consent of the author or copyright owner. Legal action will be pursued if this is breached.

Disclaimer Notice:
Please note the information contained within this document is for educational and entertainment purposes only. Every attempt has been made to provide accurate, up to date and reliable complete information. No warranties of any kind are expressed or implied. Readers acknowledge that the author is not engaging in the rendering of legal, financial, medical or professional advice.

By reading this document, the reader agrees that under no circumstances are we responsible for any losses, direct or indirect, which are incurred as a result of the use of information contained within this document, including, but not limited to, —errors, omissions, or inaccuracies.
Contents

Introduction

Chapter 1 - What is a Computer Program?
  What Can a Computer Program do?

Chapter 2 - The Elements of a Computer Program
  How to Set Up Your Environment
  What is a Text Editor?
  What is a Compiler?
  What is the Interpreter?
  Reserved Keywords
    C Language Reserved Keywords
    Java Programming Reserved Keywords
    SQL Reserved Keywords

Chapter 3 - “Hello, World!”
  SQL
    To enable CLR integrations:
    Building “Hello World!”
    Sample Code
  Java
    Creating the Java Program
    Compiling the Java Program
    Executing the Java Program
    Errors
    Input and Output
  C++
    Structure of a C++ Program

Chapter 4 - What You Need to Know
  Data Types
  C and Java Data types
  SQL Data Types
  Variables
    Creating Variables
    Store Values in Variables
    Access Stored Values in Variables
    Variables in Java
    Variables in SQL
Parameters or Arguments
Operators
Arithmetic Operators
Relational Operators
Logical Operators
Operators in Java
Operators in SQL
  SQL Arithmetic Operators:
  SQL Comparison Operators:
  SQL Logical Operators:
  Braces
  Functions
  Include Files
Basic Syntax
Java and C++
  Java Identifiers
  Java Modifiers
  Java Variables
  Java Arrays
  Java Enums
  Java Comments
  Using Blank Lines in Java
  Java Inheritance
  Java Interfaces
  C++ Semicolons and Blocks
  C++ Identifiers
  C++ Trigraphs
  C++ Whitespace
SQL
  SQL Loops
  Labeling a Loop
  Loop Control Statements
Java
  Loop Control Statements:
    Java Enhanced For Loop
C++
  Loop Control Statements:
    The Infinite Loop:
Java Arrays
- Declaring Array Variables
- Creating Arrays:
- Processing Arrays:
- The foreach Loops:
- Passing Arrays to Methods:
- Returning an Array from a Method:
- The Arrays Class:

SQL
- Creating a VARRAY Type

C++
- Declaring Arrays:
- Initializing Arrays:
- Accessing Array Elements:

Strings
Java
- Creating a String
- String Length:
- Concatenating Strings:
- Creating Format Strings:
- String Methods:

C++
- The C-Style Character String:
- The String Class in C++:

SQL
- Declaring String Variables
- SQL String Functions and Operators

Glossary of Common Programming Terms
- Compiler
- Database
- Algorithm
- Object Oriented
- Platform

Conclusion

Check Out My Other Books
Introduction

Welcome to the world of computer programming, or the act of writing a program to tell your computer what to do. Programs are written using a series of instructions in a particular language, three of which I will be talking about here – Java, SQL and C++.

Computer programming is not as difficult as it first looks and it can be a lot of fun, so long as you do it properly. I have prepared a basic “Hello, World!” tutorial for each of the three languages, just to give you an idea of how it all works.

This is aimed at people with little to no experience at programming or for those who have jumped in and are struggling to make anything work. I am confident that this book will give you an idea of how to go about learning the language of computer programming and you will be able to have a good at programming in your chosen language, moving on to more advanced coding.

Without further ado, let’s get started!
Chapter 1 - What is a Computer Program?

As I said earlier, a computer program is nothing more than a sequence of instructions that have been written with a computer programming language, telling a computer what to do. Let’s take a closer look at two important terms that I used here:

- Sequence of instructions
- Computer Programming Language

In order to best understand these terms, you need to put yourself into an imaginary situation. Someone has asked you how to get to the nearest KFC restaurant. What would you tell that person?

Using human language, you would say something like “go straight for ½ a kilometer, turn left at the traffic lights and drive for one kilometer. KFC is on the right”.

So, that is instructions given in English language and if followed in sequence, the person will reach KFC:

- Go straight
- Drive for ½ a kilometer
- Go left at the lights
- Drive for one kilometer
- Look for KFC on the right

Now, let’s try to imagine that same situation with a computer program. What the above sequence is, is a human program that instructs a person how to reach KFC. The same could have been done in any language; we chose English. The same can also be done with the computer programming language; we just use different words.

A computer program, sometimes called software, can range from just a couple of lines right up to millions of lines, each giving instructions. Those instructions are also called source code, while computer programming is known as program coding. Without a program, a computer is nothing more than a dumping box – the program is what brings the computer to life when you switch it on.

Like the human race speaks in a variety of different languages, computer scientists have come up with a number of different ones for the purpose of writing computer
programming code, such as

- Java
- C
- C++
- SQL
- Python
- PHP
- Perl
- Ruby

We will be talking about three of those later on in this book.
What Can a Computer Program do?

These days, computer programs are used virtually everywhere, in every household, in the agricultural industry, medical, defense, communications, just about anything you can think of. Examples of computer programs include:

- MS Office Suite
- Adobe Photoshop
- Any internet browser
- Any piece of software you use on your computer
- In special effects for movies
- In hospitals, in scanners, x-ray machines, ultrasound, etc.
- In your mobile phone or tablet
Chapter 2 - The Elements of a Computer Program

The English language is made of many different elements, including redefined grammar that has to be understood if you are to write English statements correctly. It also consists of other elements, like nouns, verbs, adjectives, conjunctions, prepositions, adverbs, and so on. In the same way, computer programming languages also consist of a number of different elements, including:

- Programming Environment
- Basic Syntax
- Data Types
- Variables
- Keywords
- Basic Operators
- Decision Making
- Loops
- Numbers
- Characters
- Arrays
- Strings
- Functions
- File I/O
How to Set Up Your Environment

Although this is not an element of actual computer programming, it is a vital part of the entire setup and is the first thing you should do. When I talk about environment, I am talking about the base from which you intend to carry out the programming. For this, you will need:

- A PC
- A working internet connection that is a good speed
- A web browser – Internet Explorer, Chrome, Mozilla, etc.

You will need to download one of the following programs, depending on which language you are intending to program with:

- SQL
- Java
- C++

You will also need:

- A text editor to write your code in – Notepad or Notepad++ are good ones – NOT Microsoft Word as it can’t be read by the programming languages
- A compiler to turn your written code into binary so it can be understood
- An interpreter to directly execute your program

The latter two are usually included with the programming language download. Full instructions for setting up the programs for first use can be found on the internet.
What is a Text Editor?

A text editor is a piece of software in which you write your computer programming code. Windows computers have Notepad installed as standard and this is perfectly OK to use. To locate and launch Notepad, click on Start, All Programs, Accessories, Notepad. Double click to open it.

Once you have typed your computer program into Notepad, you can save it to any location, as long as you remember where it is. If you use a Mac, you will have software called TextEdit installed.
What is a Compiler?

When you have written your program using Notepad or TextEdit, or whichever text editor you choose, you will need a compiler to translate it. This is because the computer does not recognize your code in text format so it must be converted to binary – the computer is easily able to read binary.

The compiler is responsible for this conversion – without it, you will not be able to execute your program and you will not see it running on your computer.

So, make sure you have the correct compiler for your chosen programming language before you start.
What is the Interpreter?

Following the compiler, we have the interpreter, which is responsible for executing the binary code that has just been produced from your written text program. Not all compute programming languages need a compiler; they can go straight to the interpreter, which can then read the code you have written one line at a time and execute it without any further need for any work.

As with the compiler, make sure you have the correct interpreter installed for the language you are intending to use.

You are now ready to begin programming. For each of the three languages – C++, Java and SQL, I am going to show you how to write and produce the famous “Hello, World!” sample
Reserved Keywords

All computer programming languages have a list of reserved keywords, words that are built in to the language for use by the language, and cannot be used as any form of identifier, such as a variable. The following lists are the reserved keywords for C language, Java and SQL – get to know them because using them out of place can result in serious errors:

C Language Reserved Keywords

- auto
- break
- case
- char
- const
- continue
- default
- do
- double
- else
- enum
- extern
- float
- for
- goto
- if
- int
- long
- register
- return
- short
- signed
- sizeof
- static
- struct
- switch
- typedef
- union
- unsigned
- void
- volatile
- while
Java Programming Reserved Keywords

- abstract
- assert
- boolean
- break
- byte
- case
- catch
- char
- class
- const
- continue
- default
- do
- double
- else
- enum
- extends
- final
- finally
- float
- for
- goto
- if
- implements
- import
- instanceof
- int
- interface
- long
- native
- new
- package
- private
- protected
- public
- return
- short
- _Packed
• static
• strictfp
• super
• switch
• synchronized
• this
• throw
• throws
• transient
• try
• void
• volatile
• while

**SQL Reserved Keywords** – all in capital letters

• ADD
• ALL
• ALTER
• AND
• ANY
• AS
• ASC
• AUTHORIZATION
• BACKUP
• BEGIN
• BETWEEN
• BREAK
• BROWSE
• BULK
• BY
• CASCADE
• CASE
• CHECK
• CHECKPOINT
• CLOSE
• CLUSTERED
• COALESCE
• COLLATE
• COLUMN
• COMMIT
• COMPUTE
• CONSTRAINT
• CONTAINS
• CONTAINSTABLE
• CONTINUE
• CONVERT
• CREATE
• CROSS
• CURRENT
• CURRENT_DATE
• CURRENT_TIME
• CURRENT_TIMESTAMP
• CURRENT_USER
• CURSOR
• DATABASE
• DBCC
• DEALLOCATE
• DECLARE
• DEFAULT
• DELETE
• DENY
• DESC
• DISK
• DISTINCT
• DISTRIBUTED
• DOUBLE
• DROP
• DUMMY
• DUMP
• ELSE
• END
• ERRlvl
• ESCAPE
• EXCEPT
• EXEC
• EXECUTE
• EXISTS
• EXIT
• FETCH
• FILE
• FILLFACTOR
• FOR
• FOREIGN
• FREETEXT
• FREETEXTTABLE
OUTER
OVER
PERCENT
PLAN
PRECISION
PRIMARY
PRINT
PROC
PROCEDURE
PUBLIC
RAISERROR
READ
READTEXT
RECONFIGURE
REFERENCES
REPLICATION
RESTORE
RESTRICT
RETURN
REVOKE
RIGHT
ROLLBACK
ROWCOUNT
ROWGUIDCOL
RULE
SAVE
SCHEMA
SELECT
SESSION_USER
SET
SETUSER
SHUTDOWN
SOME
STATISTICS
SYSTEM_USER
TABLE
TEXTSIZE
THEN
TO
TOP
TRAN
TRANSACTION
TRIGGER
TRUNCATE
- TSEQUAL
- UNION
- UNIQUE
- UPDATE
- UPDATETEXT
- USE
- USER
- VALUES
- VARYING
- VIEW
- WAITFOR
- WHEN
- WHERE
- WHILE
- WITH
- WRITETEXT
Chapter 3 - “Hello, World!”

The Hello World sample that we are all so used to seeing now is the perfect demonstration for the basic operations that are needed to create a simple CLR (common language runtime) program, to deploy it and test it. The following brief tutorials show you how to recreate the “Hello, World!” program in SQL, Java and C++. 
In order to create this project on SQL, you will need the following software:

- SQL Server or SQL Express. This can be got free from [here](#).
- AdventureWorks Database which can be downloaded from the [SQL Developer Website](#).
- .NET Framework SDK 2.0 or higher, or you can use Visual Studio 2005 or higher. .NET Framework is free to download.
- The Server Instance that you are running must have CLR integration enabled.
To enable CLR integrations:

You need to have ALTER SETTINGS server level permissions. These are implicitly held by serveradmin and sysadmin fixed server roles

Execute the following command at C:

- `sp_configure 'clr enabled', 1`
- `GO`
- `RECONFIGURE`
- `GO`
Building “Hello World!”

- Open up the .NET Framework or Visual Studio command prompt
- Create a new directory if necessary, or just use C:/MySample
- Inside of C:/MySample (or whatever you have called it), create a new file – for Visual Basic, it should be called HelloWorld.vb and for C# it should be HelloWorld.cs
- From the code samples below, copy the right one into the right file:
  - vbc C:HelloWorld.vb /target:library
  - csc /target:library HelloWorld.cs

You must now compile the sample code

- Compile the code, starting at the command line prompt, by using one of the samples under the sample code heading.
- Copy the installation code from Transact-SQL into a new file and save it in the C:\MySample directory as Install.sql
- Execute the following code to deploy both the assembly and the stored procedure:
  - sqlcmd -E -I -i install.sql -v root = “C:\MySample”
- Copy the test command script from Copy Transact-SQL to a new file and save it in the C:\MySample directory as test.sql
- Use the following commend to execute the test script
  - sqlcmd -E -I -i test.sql
- Copy the Cleanup script from Transact-SQL to a new file and save it in the C:\MySample directory as cleanup.sql
- Use the following command to execute the cleanup script
  - sqlcmd -E -I -i cleanup.sql

Sample Code

These are the code listings for the given sample

- C#
  - using System;
  - using System.Data;
  - using System.Data.SqlClient;
  - using System.Data.SqlTypes;
  - using Microsoft.SqlServer.Server;
  - public partial class StoredProcedures
```csharp
{ }

public static void HelloWorld()
{
    Microsoft.SqlServer.Server.SqlMetaData columnInfo
            SqlDbType.NVarChar, 12);
    SqlDataRecord greetingRecord
        = new SqlDataRecord(new Microsoft.SqlServer.Server.SqlMetaData[]
            { columnInfo });
    greetingRecord.SetString(0, "Hello world!");
    SqlContext.Pipe.Send(greetingRecord);
}
}

Visual Basic

Imports System
Imports System.Data
Imports System.Data.SqlClient
Imports System.Data.SqlTypes
Imports Microsoft.SqlServer.Server

Partial Public NotInheritable Class StoredProcedures
    Public Shared Sub HelloWorld()
        Dim columnInfo As New Microsoft.SqlServer.Server.SqlMetaData("Column1", _
            SqlDbType.NVarChar, 12)
        Dim greetingRecord As New SqlDataRecord(New _
            Microsoft.SqlServer.Server.SqlMetaData() {columnInfo})
        greetingRecord.SetString(0, "Hello World!")
        SqlContext.Pipe.Send(greetingRecord)
    End Sub
End Class

The installation script from Transact-SQL, which deploys the assembly and created a stored procedure within the database

USE AdventureWorks
GO
IF EXISTS (SELECT * FROM sys.procedures WHERE [name] = 'usp_HelloWorld')
DROP PROCEDURE usp_HelloWorld;
GO
IF EXISTS (SELECT * FROM sys.assemblies WHERE [name] = 'HelloWorld')
DROP ASSEMBLY HelloWorld;
GO
DECLARE @SamplesPath nvarchar(1024)
set @SamplesPath = '$(root)'
CREATE ASSEMBLY HelloWorld
FROM @SamplesPath + 'HelloWorld.dll'
WITH permission_set = Safe;
GO

CREATE PROCEDURE usp_HelloWorld
— (  
@Greeting nvarchar(12) OUTPUT  
)—  
AS EXTERNAL NAME HelloWorld.[StoredProcedures].HelloWorld;
GO

The Test SQL script which will execute the stored procedure and test out the sample

use AdventureWorks
go
execute usp_HelloWorld

USE AdventureWorks;
GO
IF EXISTS (SELECT * FROM sys.procedures WHERE [name] = 'usp_HelloWorld')
DROP PROCEDURE usp_HelloWorld;
GO

This Transact-SQL script will take the assembly and the stored procedure out of the database

USE AdventureWorks
GO
• IF EXISTS (SELECT * FROM sys.procedures WHERE [name] = 'usp_HelloWorld')
  • DROP PROCEDURE usp_HelloWorld;
  • GO

• IF EXISTS (SELECT * FROM sys.assemblies WHERE [name] = 'HelloWorld')
  • DROP ASSEMBLY HelloWorld;
  • GO
Java

The next step is to look at how to create the same sample but using a different computer programming language – this time, Java. It takes just the very simple steps to get this program up and running. Java is a collection of different applications, most of which are very similar to those that you are already used to using – your word processor, internet browser or email program, for example.

As you should with any application, you must ensure that you have Java installed properly on your computer system. You will also need a text editor and a terminal application for this to work.

Programming in Java

The process of building this program in Java can be broken down into three simple steps:

- Create the new program by inputting it into the text editor and saving it. Make sure you know where it is saved to and what it is called. You could, for example, call it MyProgram.java
- Compile the program by typing the following into your terminal window – javac MyProgram (or whatever yours was called).java
- Execute the program by typing the following into the terminal window – java MyProgram

The first step is creating the new program; the second converts it into characters, much like a sentence, or a poem if you like, that the computer can read and will save the output to a file called MyProgram.class and the third step is what runs the program on your computer.

Creating the Java Program

A program is simply a character sequence, similar to a poem, a sentence or a paragraph. To create the program, all you have to do is define the sequence of characters using your text editor, in exactly the same way as you would do for an email. “Hello World!” is a good sample program to start with so type the following into your text editor and then save it as a new file called HelloWorld.java.

    public class HelloWorld {
    public static void main(String[] args) {

System.out.println("Hello, World");

Compiling the Java Program

To start with, it may look as though the Java language you are using is easily understood by the computer, but that would be incorrect. The language is actually best designed to be understood by you, the programmer. This is where the compiler comes in – it is an application that translates the language that you have written the program in into something that the computer can understand. It looks for a text file that contains the extension .java and uses that as the input and the output is a file that has a .class extension, which is the language that your computer can read and understand. To compile the HelloWorld.java file, type the following into your terminal – please note that the percentage sign (%) is used purely to denote the command prompt – yours may be different.

% javac HelloWorld.java

Provided you type this in properly, there should not be any error messages. If there are, you need to go back and make sure the program was typed in exactly as it should have been.

Executing the Java Program

So, now that your program has been compiled, it is time to run it, or execute it. This is the most exciting bit, the part where you see the fruits of your labors. This is where the computer does what you have told it to do. To run your HelloWorld program, type the following in at your command prompt, not forgetting that the % sign should be omitted as it is only an indicator of the command prompt.

% java HelloWorld

If everything has gone, as it should, the following should be displayed on your terminal window

Hello, World
Errors

If you do get any error messages, don’t panic. Most errors are fixed very easily, simply by examining your code very carefully as you write it. This is the same as you would do with any Word document, fixing any spelling and grammar errors as you go.

- Compile-time Errors - These are generally caught by your computer system when you compile a program. They will stop the compiler from translating your program to a language the computer can read and it will kick up an error message that attempts to explain this to you
- Run-time Errors - These will be caught when you try to execute the program because an invalid operation is being performed, which will stop the program from running.
- Logical Errors – you should with any luck, catch these when you execute the program and a wrong answer is produced. Bugs are going to be the bane of your existence because they are tricky to find.

One of first skills that you, as a new programmer, will learn is how to identify errors. The next will be learning how to be careful when you code so you don’t create any errors.

Input and Output

As a rule, you will want to create input for your programs. An input is data that can be processed to produce a given result. The easiest way to provide the input is shown below. When you execute the following program, it will read the argument you typed on the command line, after the name of the program and will then print it out as a part of the message:

- % javac UseArgument.java
- % java UseArgument Alice
- Hi, Alice. How are you?
- % java UseArgument Bob
- Hi, Bob. How are you?
C++

Structure of a C++ Program

The very best way to learn a new programming language is to get right down to it and write a program. The very first program that any new programmer will write is “Hello World!” All this program does is prints out the words, “Hello, World!” onto your screen. It is a very simple program but it also contains all of the components that are fundamental to a C++ program:

- // my first program in C++
- #include <iostream>

- int main()
  {
    std::cout << “Hello World!”;
  } Hello World!

This is the code that needs to be written for this program to be executed. So let’s look at the code, one line at a time:

- Line 1: // my first program in C++

The two // at the start are an indication that the code that comes afterwards is a comment that you have put in but has absolutely no effect on the program. These tend to be included as descriptions, explanations or observations that are there for the reader’s information. The computer will not read a comment. In this case, the comment is merely a description of the program.

- Line 2: #include <iostream>

Lines that start with a # are directives that are read by the preprocessor and are then interpreted. These are special lines that will be interpreted before the program is compiled. In this example, the directive that reads #include <iostream>, is telling the preprocessor that it should include a piece of C++ code that is called header iostream. This piece of code allows standard input and output operations to be performed, such as the output of writing the program, HelloWorld to your screen.
Blank lines do not mean anything special, they are merely there to improve the readability of the stream of code.

This initiates a function declaration. A function is a group of statements, in code, that are given a name. In this example, the given name is Main. While I will not go into detail on functions here, I can tell you that the definition of a function starts with a type succession, in this case int, the name, which is main and open and closed parentheses. Within these parameters can be included but that is for more advanced programming.

Main is a special function in C++ because it is the one that is called when the program is executed.

Open braces indicate the start of the definition of Main and the closing brace, which you will see on line 7, indicates the end of the function. Everything that is written inside the braces is written to define the function of main when it is called. All functions begin and end with braces.

This particular statement has three separate parts – std::cout; which is what identifies the standard character output device, which is usually your computer screen. The second is <<, indication that the following code is inserted into std::cout. Lastly, the sentence that appears inside the quote marks, in this case “Hello World” is the standard output.
Note that a semicolon is used to end the statement. This always denotes the end of a statement in the same way that a full stop denotes the end of a sentence. All statements in C++ have to end with this semi-colon – failure to include it is the most common syntax error in C++ programming.

You may also have spotted that not every line of the code performs an action upon execution of the code. The lines that contain the comment (//), the directive (#) and the end of the statement (;). None of these actually performs any type of action on execution.

Sometimes you will see a computer program written with lots of indentations – while these tend to make it easier for you to read, the computer does not require any indentation or does it require the code to be written in separate lines, so it is your choice on how you write it. An example:

- int main ()
- {
-   std::cout << ” Hello World!”;
- }

- Could have been written as:

- int main () { std::cout << “Hello World!”; }

with all the instructions in one line. This would have exactly the same meaning as the broken down example above.

In C++, you just have to remember that each statement must be separated by a semi-colon. Now, let’s put another statement into your program:

- // my second program in C++
- #include <iostream>

- int main ()
- {
-   std::cout << “Hello World! “;
-   std::cout << “I’m a C++ program”;
- }Hello World! I’m a C++ program
In this particular case, the program has performed two insertions in two different statements, both into std::cout. Again, the separation on different lines is merely to allow you to read it easier; the computer would have been just as happy with this:

- int main () { std::cout << ” Hello World! “; std::cout << ” I’m a C++ program “; }

The code could also have been written over more lines, as such:

- int main ()
- {
- std::cout <<
- “Hello World!”;
- std::cout
- << “I’m a C++ program”;
- }

The result would be exactly the same, no matter how many lines you choose to write the code over.

The preprocessor directives, the ones that start with #, are not included in the rule because they are not statements. They are simply lines of code that are read and then processed by the pre-processor before the file is compiled. These directives must have their own line and do not need to end with a semi-colon, because they are not statements.
Chapter 4 - What You Need to Know

No matter which programming language you opt to use, there are a number of things that you need to know. I have listed some of the common concepts that will apply to virtually every computer programming language. Have a read though and you will begin to understand what things you need to know to start leaning how to program. It might seem like a lot to learn right now but please don’t worry. As you begin to program, you will learn all of the basic concepts that you need to know and each one will build on the last, until you finally have something that all comes together and makes perfect sense!
Data Types

One of the simplest yet most important concepts in computer programming languages is a concept called data types. A data type simply represents a specific type of data that can be processed through your computer program. Data types can be alphanumeric, numeric, decimal, etc. Just as an aside from the actual computer programming, let’s just step back to Math class and add up two whole numbers – a very simple process:

- 10 + 20

Nice and simple but, let’s say that we want to add up two decimal numbers. This is written like this:

- 10.50 + 20.50

So, these two examples may be pretty straightforward but let’s have a look at another example where we want to put some information for a student into a notebook. The following is the important information that should be recorded:

- Name:

- Class:

- Section:

- Age:

- Sex:

Now, as per the requirement, we will add in one student record:

Name: Zara Ali

- Class: 6th
So, in the first example, we looked at whole numbers and in the second one, we looked at decimal numbers, adding two of them together. The third example looked at a whole range of data. So to put that last example into computer language terms, it would describe as such:

- The student name – Zara Ali – this is a sequence of characters which is also known as a “string”
- The student class – 6th – this is represented by a mixture of whole numbers and two characters (more than one character together is called a string). This is all known as an alphanumeric data type
- The student section is represented by one single character, a “J”.
- The student age – 13 – is represented by a whole number
- The student sex is represented by one single character, “F”, to indicate that the student is female.

Through this, you can see that, in our daily life we are constantly dealing with different data types, such as characters, strings, whole numbers (also called integers), decimal numbers (also called floating point numbers) and alphanumeric strings.

This is similar to the way a computer program is written. In order for that program to process a range of different data types, we must ensure that the type is clearly specified otherwise the computer will not know how a range of different operations can be carried out in that specific data. Different languages will use different keywords as a way of specifying different data types. In the next section I am going to go over the use of different data types in different situations for C language, Java and SQL.
C and Java Data types

C and Java tend to share many of the same data types, although Java does include support for extra ones. The following are the common data types that both languages support:

<table>
<thead>
<tr>
<th>Type</th>
<th>Keyword</th>
<th>Value range that may be represented by this data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Character</td>
<td>char</td>
<td>-128 to 127 or 0 to 255</td>
</tr>
<tr>
<td>Number</td>
<td>int</td>
<td>-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647</td>
</tr>
<tr>
<td>Small Number</td>
<td>short</td>
<td>-32,768 to 32,767</td>
</tr>
<tr>
<td>Long Number</td>
<td>long</td>
<td>-2,147,483,648 to 2,147,483,647</td>
</tr>
<tr>
<td>Decimal Number</td>
<td>float</td>
<td>1.2E-38 to 3.4E+38 till 6 decimal places</td>
</tr>
</tbody>
</table>

These are all called primitive data types and they can be used in the process of building up more complex data types. The more complex ones are known as user-defined data types, for example a sequence of characters known as a string.
## SQL Data Types

In SQL, every column that is present inside a database must have a name and a data type attached to it. As the developer, it is up to you to determine what data type is going to be stored inside each of the table columns when you are creating a table in SQL. The data type is a way for SQL to understand what it should expect in terms of data inside each separate column and it also determines how SQL will interact with the data stored. These are the general data types for SQL:

<table>
<thead>
<tr>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHARACTER(n)</td>
<td>Character string. Fixed-length n</td>
</tr>
<tr>
<td>VARCHAR(n) or CHARACTER VARYING(n)</td>
<td>Character string. Variable length. Maximum length n</td>
</tr>
<tr>
<td>BINARY(n)</td>
<td>Binary string. Fixed-length n</td>
</tr>
<tr>
<td>BOOLEAN</td>
<td>Stores TRUE or FALSE values</td>
</tr>
<tr>
<td>VARBINARY(n) or BINARY VARYING(n)</td>
<td>Binary string. Variable length. Maximum length n</td>
</tr>
<tr>
<td>INTEGER(p)</td>
<td>Integer numerical (no decimal). Precision p</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>Integer numerical (no decimal). Precision 5</td>
</tr>
<tr>
<td>INTEGER</td>
<td>Integer numerical (no decimal). Precision 10</td>
</tr>
<tr>
<td>BIGINT</td>
<td>Integer numerical (no decimal). Precision 19</td>
</tr>
<tr>
<td>DECIMAL(p,s)</td>
<td>Exact numerical, precision p, scale s. Example: decimal(5,2) is a number that has 3 digits before the decimal and 2 digits after the decimal</td>
</tr>
<tr>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>NUMERIC(p,s)</td>
<td>Exact numerical, precision p, scale s. (Same as DECIMAL)</td>
</tr>
<tr>
<td>FLOAT(p)</td>
<td>Approximate numerical, mantissa precision p. A floating number in base 10 exponential notation. The size argument for this type consists of a single number specifying the minimum precision.</td>
</tr>
<tr>
<td>REAL</td>
<td>Approximate numerical, mantissa precision 7</td>
</tr>
<tr>
<td>FLOAT</td>
<td>Approximate numerical, mantissa precision 16</td>
</tr>
<tr>
<td>DOUBLE PRECISION</td>
<td>Approximate numerical, mantissa precision 16</td>
</tr>
<tr>
<td>DATE</td>
<td>Stores year, month, and day values</td>
</tr>
<tr>
<td>TIME</td>
<td>Stores hour, minute, and second values</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>Stores year, month, day, hour, minute, and second values</td>
</tr>
<tr>
<td>INTERVAL</td>
<td>Composed of a number of integer fields, representing a period of time, depending on the type of interval.</td>
</tr>
<tr>
<td>ARRAY</td>
<td>A set-length and ordered collection of elements</td>
</tr>
<tr>
<td>MULTISET</td>
<td>A variable-length and unordered collection of elements</td>
</tr>
<tr>
<td>XML</td>
<td>Stores XML data</td>
</tr>
</tbody>
</table>
Variables

In programming, a variable is similar to those variables you may (or may not) remember from Algebra. Instead of simply holding a number, variable in computer programming hold numbers, text and other things.

The most commonly used types of variable are:

- Integer - a whole number, without a decimal point
- Double or Float – A number that may or may not have a decimal point
- String – quite simply, a string of characters - numbers, letters, words – written as text.
- Boolean – this variable can be set to TRUE or FALSE and is useful in executing the program controls – once you get deeper into programming

Variables are another very important concept in computer programming. They are the names that you give the computer memory location where the values are stored inside the program. Let’s say that you wanted to store two values in your program, with a view to using them later on. Let’s have a look at how that is done, in just three easy steps:

- You create the variables using appropriate names – one for each value
- You store your values within those two variables
- You retrieve the stored values from the variables and use them

Creating Variables

The act of creating a variable is also known as declaring the value in C programming. Different languages use different ways to create or declare variables in your program. An example in C language would be:

- `#include <stdio.h>`

- `main()`
  - `{`
    - `int a;`
    - `int b;`
  - `}`
This program has declared two variables – it has reserved two locations in the memory with the names “a” and “b”. These have been created by using the “int” keyword, specifying the variable data type, which says that the values we want to store in the variables are integers. In the same way, using the right keywords, you can also store data in variables that are declared as “float”, “long”, “char” or any of the other data types. An example of this:

- /* variable to store long value */
  - long a;

- /* variable to store float value */
  - float b;

You can also declare variables that are similar in type by placing them on one line of code, each separated by a comma. For example:

- #include <stdio.h>

- main()
  - {
    - int a, b;
  - }

Here are some important things you need to keep in mind about variables:

- The name of a variable is able to hold one single value type only, i.e. if a variable called “a” has been given a data type “int”, it is only able to store integers
- C language needs a variable to be declared before it can be used in your program. You are not able to use any variable without declaring it in most languages, one exception to that rule being Python, which does allow you to do this
- You can only use a particular variable name once in your program. For example, if you define variable “a” as an “int”, to store values that are integers, you cannot then define “a” to store any other value that is not an integer, or is any other type of data.
- Some programming languages do not want you specifying any particular data type when you create a variable. This means that you can, in some languages, store
float, integer or long values without actually specifying what their data type is.

- You can name a variable anything you like, but be aware that many languages do limit you to a set number of characters per variable name. For now, stick to using the following characters in your variable names as they are accepted by all languages and begin your variable name with a letter rather than just using digits – “a-z”, “A-Z”, and “0-9”
- Virtually no programming language will allow you to begin your variable name with a digit so something like “2015” would not be acceptable, whereas “year2015” would be valid

There are lots more rules for variables, specific to each programming language and, as you get deeper into each language, you will learn more about them. These basic rules are enough for you to begin with for now so let’s move on to storing values.

**Store Values in Variables**

Now you know how to create or declare a variable, it’s time to look at how to store values in them:

```
#include<stdio.h>

main()
{
  int a;
  int b;

  a = 10;
  b = 20;
}
```

In this program, we have included a pair of additional statements. In variable “a” we are storing a value of 10 and in variable “b” we are storing a value of 20. Virtually all computer programming languages provide a similar way of doing this, where the variable name is on the left of the = (equal) sign and the value that you want to store in the variable is on the right side.

Now, we have declared two variables and we have also stored the necessary values in
those variables. One variable now has a value of 10 and the other has a value of 20. This means that, when this program is executed, the memory location for “a” will hold 10 and the memory location for “b” will hold 20.

Access Stored Values in Variables

There is little point in going to the effort to declare a variable and store a value in it if you are not going to use that value at some point. The program above contains two variables with separate values. The next step is to try to print those values that are stored in the variables. The following example is for C language and the command we use will print those values for us:

- #include <stdio.h>

- main()
  - {
    - int a;
    - int b;

    - a = 10;
    - b = 20;

    - printf( “Value of a = %d\n”, a );
    - printf( “Value of b = %d\n”, b );

  - }

When we execute this program, we get this result:

- Value of a = 10
- Value of b = 20

You will have spotted that we used the “printf” function when we looked at the “Hello, World!” example. In this particular program, we are using that function to print out the values that are stored inside the variables. As well as that, we have used “%d” which will be replaced with the values that are stored in the variables that are used in the “printf”
Both values can be printed using the “printf” statement, as follows:

```c
#include <stdio.h>

int a;
int b;

a = 10;
b = 20;

printf( "Value of a = %d and value of b = %d\n", a, b );
```

When we execute this program, we get this result:

- Value of a = 10 and value of b = 20

In C programming, if you want to make use of the float variable, you will need to use “%f” instead of “%d”. If you want to print out a value, you would use “%c”. In a similar manner, a range of different data types can be printed just by using “%” and different characters.

**Variables in Java**

The above was for C program so let’s have a look at the equivalent program using the Java language instead. We will declare two variables, “a” and “b” and we will assign the values “10” and “20” to them as well before printing the values in two different ways:

```java
public class DemoJava
{
    public static void main(String []args)
    {
```
• int a;
• int b;

• a = 10;
• b = 20;

• System.out.println("Value of a = " + a);
• System.out.println("Value of b = " + b);

• System.out.println("Value of a = " + a + " and value of b = " + b);
• }
• }
Variables in SQL

In the SQL language, a variable lets the programmer store data on a temporary basis, while the code is being executed. The syntax used to declare the variables in SQL, using the statement “DECLARE”, is:

- DECLARE @variable_name datatype [ = initial_value ],
- @variable_name datatype [ = initial_value ],
- …;

Parameters or Arguments

To break that down a little:

- The variable name is the name that you assign to that variable
- The data type is the type that is assigned to the variable, i.e. “int”, “float” etc.
- The initial_value is optional and it is the value that you first assign to the variable when you declare it

Below is an example of how we declare a variable in SQL:

- DECLARE @techonthenet VARCHAR(50);

The DECLARE statement is declaring a variable that we have called “@techonthenet”, which is a VARCHAR type of data, containing 50 characters. Next, using the SET statement, we are going to change the value of @techonthenet, like this:

- SET @techonthenet = ‘Example showing how to declare variable’;

Next, we are going to look at how an INT variable is declared in SQL:

- DECLARE @site_value INT;

If you want to assign a particular value to “@site_value”, use the SET statement, like this:
• SET @site_value = 10;

This statement is giving a value of integer 10 to the variable

What if you wanted to declare more than one variable in SQL? You would do it like this:

• DECLARE @techonthenet VARCHAR(50),
  @site_value INT;

We have declared two variables, the first being “@techofthenet”, defined as VARCHAR (50) and the second is called “site_value”, defined as INT.

Now let’s have a quick look at how to declare a variable and give it an initial value

• DECLARE @techonthenet VARCHAR(50) = 'how to declare variable’;

We use the DECLARE statement to declare the “@techonthenet” statement that has been defined as data type VARCHAR, with a character length of 50. We then set the “techonthenet” variable to declare “how to declare variable” – bear in mind, this is just an example of how to do it.

Lastly, let’s have a look at declaring an INT variable in SQL with an initial value:

• DECLARE @site_value INT = 10;

This declaration is declaring a variable that we called “@site_value”, with an INT data type. The value of that variable is then set at an integer value of 10.

To declare more than one variable with initial values, you would do something like this:

• DECLARE @techonthenet VARCHAR(50) = ‘how to declare variable’,
  @site_value INT = 10;
Here, we have declared a pair of variables and each has been assigned an initial value in the declaration:

The first one, called “@techonthenet”, has been defined as a VARCHAR data type with a 50 character length and has been assigned an initial value of “how to declare variable”.

The second, called “@site_value”, has been declared as an INT data type and has been given an integer value of 10

 Operators

Operators are used to work on the variables in your program. The most common operators are:

- + - addition
  - - subtraction
- * - multiplication
- / - division
- = - an assignment of a value, such as X=4
- ++ - increments of an integer by 1
  - subtraction from an integer by 1

In any programming language, an operator is used to tell the interpreter or the compiler to carry out particular and very specific operations, either logical, mathematical or relational, and produce an end result. We are now going to talk about some of the more important relational and operational operators that are used in C and Java languages

Arithmetic Operators

We often use computer programs to carry out mathematical calculations and we can easily write a program that will add up two numbers, or another simple equation. We can write a program that will solve a more complex equation as well. Have a look at the following mathematical examples:
2 + 3
P(x) = x^4 + 7x^3 - 5x + 9.

These are both called arithmetic expressions in the computer programming language and the + and – are called arithmetic operators. The values, such as 2, 3, and so on, are called operands. Expressions like these, in their simplest form, always produce a result that is numerical.

In a similar way, computer programming languages provide a number of different arithmetic operators. The table below shows some of the more important operators in C language. For the purposes of this book, assume that variable “A” has a value of 10 and variable “B” has a value of 20:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Adds two operands</td>
<td>A + B will give 30</td>
</tr>
<tr>
<td>-</td>
<td>Subtracts second operand from the first</td>
<td>A - B will give -10</td>
</tr>
<tr>
<td>*</td>
<td>Multiplies both operands</td>
<td>A * B will give 200</td>
</tr>
<tr>
<td>/</td>
<td>Divides numerator by de-numerator</td>
<td>B / A will give 2</td>
</tr>
<tr>
<td>%</td>
<td>This gives remainder of an integer division</td>
<td>B % A will give 0</td>
</tr>
</tbody>
</table>

Next is an example of a program to help you understand these operators:

```c
#include <stdio.h>

main()
{
    int a, b, c;

    a = 10;
    b = 20;
```
c = a + b;
printf( “Value of c = %d\n”, c);

c = a - b;
printf( “Value of c = %d\n”, c);

c = a * b;
printf( “Value of c = %d\n”, c);

c = b / a;
printf( “Value of c = %d\n”, c);

c = b % a;
printf( “Value of c = %d\n”, c);

}

When we execute the above program, the following result is produced:

- Value of c = 30
- Value of c = -10
- Value of c = 200
- Value of c = 2
- Value of c = 0

Relational Operators

Think of a situation that would require you to create a pair of variables and then assign then values, like this:

- A = 20
- B = 10

In this example, it is clear that the value of “A” is greater than the value of “B”. But, if you wanted to write that in computer language, how would you do it? You would need to
use symbols to help you write this kind of expression, called relational expressions. In C language, they would be written like this:

- \((A > B)\)

We have used the symbol >, known as an operational operator. In its simplest form, it will produce a Boolean result, meaning that the result is not true nor is it false. The table below shows the more important relational operators in C language. Once again, assume that “A” has a value of 10 and “B” has a value of 20:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>Checks if the values of two operands are equal or not, if yes then condition becomes true.</td>
<td>((A == B)) is not true.</td>
</tr>
<tr>
<td>!=</td>
<td>Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.</td>
<td>((A != B)) is true.</td>
</tr>
<tr>
<td>&gt;</td>
<td>Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.</td>
<td>((A &gt; B)) is not true.</td>
</tr>
<tr>
<td>&lt;</td>
<td>Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.</td>
<td>((A &lt; B)) is true.</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.</td>
<td>((A &gt;= B)) is not true.</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.</td>
<td>((A &lt;= B)) is true.</td>
</tr>
</tbody>
</table>

Next, we are going to look at an example of programming in C language that sees the “if conditional statement”. This is used to check a condition – it that condition is true, the if statement body will be executed; if it is false, the statement will be skipped over.
```c
#include <stdio.h>

main()
{
  int a, b;

  a = 10;
  b = 20;

  /* Here we check whether a is equal to 10 or not */
  if( a == 10 )
  {
    /* if a is equal to 10 then this body will be executed */
    printf( "a is equal to 10\n" );
  }

  /* Here we check whether b is equal to 10 or not */
  if( b == 10 )
  {
    /* if b is equal to 10 then this body will be executed */
    printf( "b is equal to 10\n" );
  }

  /* Here we check if a is less b than or not */
  if( a < b )
  {
    /* if a is less than b then this body will be executed */
    printf( "a is less than b\n" );
  }

  /* Here we check whether a and b are not equal */
  if( a != b )
  {
    /* if a is not equal to b then this body will be executed */
    printf( "a is not equal to b\n" );
  }
}
```
When we execute this program, this is the result we get:

- a is equal to 10
- a is less than b
- a is not equal to b

**Logical Operators**

Logical operators hold a place of importance in all programming languages and they are used to help make decisions that are based on specific conditions. Let’s say that we want to combine the result of two separate conditions. We would use two logical operators to help us get to that result – AND and OR.

The table below shows all of the logical operators that are supported in C language. Again, A has a value of 10 while B has a value of 20

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;&amp;</td>
<td>Called Logical AND operator. If both the operands are non-zero, then condition becomes true.</td>
<td>(A &amp;&amp; B) is false.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>!</td>
<td>Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.</td>
<td>!(A &amp;&amp; B) is true.</td>
</tr>
</tbody>
</table>

Have a look at this example, to try to help you understand how the logical operators work

- #include <stdio.h>
- main()
```c
{  
    int a = 1;
    int b = 0;

    if ( a && b )  
    {  
        printf("This will never print because condition is false\n");  
    }  
    if ( a || b )  
    {  
        printf("This will be printed print because condition is true\n");  
    }  

    if ( !(a && b) )  
    {  
        printf("This will be printed print because condition is true\n");  
    }  
}
```

When you have compiled and the executed this program, the following result will be produced:

- This will be printed print because condition is true
- This will be printed print because condition is true
The following is a similar program that has been written in Java programming language. Both C and Java use a set of operators and conditional statements that are almost identical. The following program is going to declare two variables, “a” and “b” again and, in a similar way to C, we are going to assign the values of 10 and 20 to these variables. Lastly, we will make use relational and arithmetic operators.

See if you can compile and then execute this program to see what the output is. Bear in mind that it must be the same output as in the example in C programming above:

```java
public class DemoJava {
    public static void main(String[] args) {
        int a, b, c;
        a = 10;
        b = 20;
        c = a + b;
        System.out.println("Value of c = "+c);
        c = a - b;
        System.out.println("Value of c = "+c);
        c = a * b;
        System.out.println("Value of c = "+c);
        c = b / a;
        System.out.println("Value of c = "+c);
        c = b % a;
    }
}
```
System.out.println("Value of c = " + c);

if(a == 10)
{
    System.out.println("a is equal to 10");
}

}
Operators in SQL

Operators are reserved words or characters that are used in the WHERE clause of an SQL statement to perform one or more operations, like arithmetic and comparison operations. They are used as a way of specifying conditions in SQL statements and are also used as conjunctions for multiple conditions in statements. The following are the most common operators in SQL:

- Arithmetic operators
- Comparison operators
- Logical operators
- Operators used to negate conditions

### SQL Arithmetic Operators:

The table below shows the SQL arithmetic operators. As before, variable “a” has a value of 10 and “b” has a value of 20

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition - Adds values on either side of the operator</td>
<td>a + b will give 30</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction - Subtracts right hand operand from left hand operand</td>
<td>a - b will give -10</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication - Multiplies values on either side of the operator</td>
<td>a * b will give 200</td>
</tr>
<tr>
<td>/</td>
<td>Division - Divides left hand operand by right hand operand</td>
<td>b / a will give 2</td>
</tr>
<tr>
<td>%</td>
<td>Modulus - Divides left hand operand by right hand operand and returns remainder</td>
<td>b % a will give 0</td>
</tr>
</tbody>
</table>

### SQL Comparison Operators:
This table shows the comparison operators in SQL – “a” has a value of 10 and “b” has a value of 20

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>Checks if the values of two operands are equal or not, if yes then condition becomes true.</td>
<td>(a = b) is not true.</td>
</tr>
<tr>
<td>!=</td>
<td>Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.</td>
<td>(a != b) is true.</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.</td>
<td>(a &lt;&gt; b) is true.</td>
</tr>
<tr>
<td>&gt;</td>
<td>Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.</td>
<td>(a &gt; b) is not true.</td>
</tr>
<tr>
<td>&lt;</td>
<td>Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.</td>
<td>(a &lt; b) is true.</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.</td>
<td>(a &gt;= b) is not true.</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.</td>
<td>(a &lt;= b) is true.</td>
</tr>
<tr>
<td>!&lt;</td>
<td>Checks if the value of left operand is not less than the value of right operand, if yes then condition becomes true.</td>
<td>(a !&lt; b) is false.</td>
</tr>
<tr>
<td>!&gt;</td>
<td>Checks if the value of left operand is not greater than the value of right operand, if yes then condition becomes true.</td>
<td>(a !&gt; b) is true.</td>
</tr>
</tbody>
</table>

**SQL Logical Operators:**
This table shows all the logical operators used in SQL:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>The ALL operator is used to compare a value to all values in another value set.</td>
</tr>
<tr>
<td>AND</td>
<td>The AND operator allows the existence of multiple conditions in an SQL statement’s WHERE clause.</td>
</tr>
<tr>
<td>ANY</td>
<td>The ANY operator is used to compare a value to any applicable value in the list according to the condition.</td>
</tr>
<tr>
<td>BETWEEN</td>
<td>The BETWEEN operator is used to search for values that are within a set of values, given the minimum value and the maximum value.</td>
</tr>
<tr>
<td>EXISTS</td>
<td>The EXISTS operator is used to search for the presence of a row in a specified table that meets certain criteria.</td>
</tr>
<tr>
<td>IN</td>
<td>The IN operator is used to compare a value to a list of literal values that have been specified.</td>
</tr>
<tr>
<td>LIKE</td>
<td>The LIKE operator is used to compare a value to similar values using wildcard operators.</td>
</tr>
<tr>
<td>NOT</td>
<td>The NOT operator reverses the meaning of the logical operator with which it is used. Eg: NOT EXISTS, NOT BETWEEN, NOT IN, etc. This is a negate operator.</td>
</tr>
<tr>
<td>OR</td>
<td>The OR operator is used to combine multiple conditions in an SQL statement’s WHERE clause.</td>
</tr>
<tr>
<td>IS NULL</td>
<td>The NULL operator is used to compare a value with a NULL value.</td>
</tr>
<tr>
<td>UNIQUE</td>
<td>The UNIQUE operator searches every row of a specified table for uniqueness (no duplicates).</td>
</tr>
</tbody>
</table>

Braces
{} – also termed curly braces. These are used as a way of controlling the program flow. Everything that is inside a pair of curly braces has to be executed as a group, not individuals.

**Functions**

These are also similar to those in Algebra and are used, sometimes on a variable, to do the work necessary to obtain a result.

**Include Files**

Most programming languages will use this and you will normally find it at the start of a set of code. Include Files has information that is needed by the program so that it can run the code that is in the program. Instead of having all the code copied to the program (as this takes up a lot of space and can get quite complex). The compiler simply looks for the instructions it needs to execute the item that is in the Include Files plus the code you wrote.
Basic Syntax
Java and C++

Java and C++ programs are defined as a set of objects that talk to each other through methods. Let’s take a brief look at what objects, class, methods and instance variables mean:

- **Object** – Each object has a state and a behavior. An example – a dog has colors, breeds and names, which are states, and they eat, bark or wag their tails, which are behaviors. An object is an instance of a class
- **Class** – A class is a template that carries a description of the states and behaviors that are supported by its type
- **Methods** – A behavior. Classes can contain unlimited methods and they are where data manipulation, logics and execution of an action take place
- **Instance variables** – Every object has its own set of these and it is the values that are assigned to an instance variable that creates the state of the object.

When it comes to writing code in Java, there are some basis syntax rules you must follow:

- **Case** - Java is a case sensitive language. This means that the identifier “hello” would have a different meaning to the identifier “Hello”
- **Class Names** – All class names should begin with an upper case letter. If you use more than one word to form a class name, each word should begin with an upper case letter, for example MyFirstJavaClass
- **Method Names** – Method names should being with a lower case letter and, similar to class names, if more than one word is used, each word must have a lower case letter at the beginning.
- **Program File Names** – The name of the program file name should be exactly the same as the class name so remember that when you save it. Don’t forget to use the right case and to add.java to the end of the name. If the class and the file name are not the same, your program is not going to compile.

**Java Identifiers**

Every Java component has to have a name and the names that we use for variables, classes and methods are known as identifiers. You need to remember these points:

- All identifiers must start with a letter – A – Z or a – z, or they should start with an underscore (_), or a currency character
• The rest of the identifier name can be made up of any combination of different characters
• You cannot use any of the reserved keywords as identifiers
• Identifiers are case sensitive
• A few examples of identifiers that are legal are age, $wage, _value, ___1__value
• An illegal identifier would look something like 123abc, -wage

Java Modifiers

In a similar way to other computer programming languages, you can modify methods, classes, etc. by using something called a modifier. In Java, we use two categories of modifiers:

• Access Modifiers - private, public, protected, default
• Non-Access Modifiers - abstract, final, strictfp

Java Variables

In Java language, you will see these variable types:

• Local
• Class
• Instance

Java Arrays

An array is an object that is used to store multiple values that are of the same type.

Java Enums

 Enums are a relatively new thing in Java and they restrict variables so that they can only have one of a small number of pre-defined values. These values are called enums. By using these, it is possible to cut down on the number of bugs that are in your code. Let’s take an example here of a shop that sells fresh juice. You can restrict the size of the juice glass to small, medium or large and this would ensure that this was the only size that people could buy. An example of the code:
• class FreshJuice {
  
  • enum FreshJuiceSize{ SMALL, MEDIUM, LARGE }
  • FreshJuiceSize size;
  • }

• public class FreshJuiceTest {
  
  • public static void main(String args[]){
  • FreshJuice juice = new FreshJuice();
  • juice.size = FreshJuice.FreshJuiceSize.MEDIUM ;
  • System.out.println(“Size: ” + juice.size);
  • }
  • }

The result from this particular example would be:

• Size: MEDIUM

It is possible to declare an enum inside a class or on its own and methods, variables and constructors can also be defined inside an enum.

**Java Comments**

The Java language supports both multi and single line comments and the Java compiler will ignore any character that is inside a comment:

• public class MyFirstJavaProgram{
  
  • /* This is my first java program.
  • This will print ‘Hello World’ as the output
  • This is an example of multi-line comments.
  • */
Using Blank Lines in Java

A line that contains nothing but white space and perhaps a comment is called a blank line and is ignored completely by Java.

Java Inheritance

In the Java language, it is possible to derive a class from a class. Let’s say that you needed to create a new class but there is already one that contains a bit of the code that you need. It is possible to derive the new class from the existing one. This allows you to use fields and methods from existing class without the need to write a new code. The original class is called the superclass and the new class that you have derived from it is called the subclass.

Java Interfaces

In Java, an interface is defined as a contract that sets out how objects communicate with one another. Interfaces are vital in the concept of inheritance because the interface will define the methods that the subclass uses. However, implementation of those methods is entirely up to the subclass.

C++ Semicolons and Blocks

In the C++ language, the semicolon is used to terminate a statement and, to that end, all statements must be finished off with a semicolon, to indicate that the logical entity has ended. For example, have a look at these three statements:

- $x = y$;
- $y = y+1$;
- $\text{add}(x, y)$;
A block is a collection of statements that are logically connected and are enclosed in braces:

```c++
{
    cout << "Hello World"; // prints Hello World
    return 0;
}
```

C++ will not see the end of each statement as a terminator because it is not a semicolon. So, for this reason, you can put a statement anywhere on a line. For example:

```c++
x = y;
y = y+1;
add(x, y);
```

is exactly the same as:

```c++
x = y; y = y+1; add(x, y);
```

**C++ Identifiers**

These are used to identify a class, function, variable, module or any other item that is user-defined. A few points to remember about C++ identifiers:

- They must begin with a letter – A to Z or a to z, an underscore (_) that is followed by a zero or, more underscores, letters or digits (0 to 9)
- Punctuation characters such a $, @, or % are not allowed in C++ identifiers
- C++ is case sensitive, which means that, for example, Manpower is a different identifier to manpower.

**Examples of identifiers that are acceptable are:**

- Mohr
- Zara
C++ Trigraphs

Some characters have another representation in C++ and this is known as a trigraph sequence. A trigraph is a sequence of three characters, representing one single character. The sequence will always begin with a pair of question marks (??). The most common of the trigraph sequences are:

?? = replaces #
?? / replaces \
??’ replaces ^
?? ( replaces [
??) replaces ]
??! replaces |
?? < replaces {
?? > replaces }
?? - replaces ~

Not all C++ compilers will support trigraphs and, in all fairness, it is probably better that you do not use the, simply because they can be confusing. I have mentioned them because you are likely to come across them and will need to know what they are.

C++ White Space

As in Java, a line that contains whitespace only, maybe with a comment is a blank line and is ignored by C++. The language uses whitespace as the term to describe tabs, banks, comments and newline characters. It is used to separate a statement into parts and allows the compiler to see where one part of a statement ends and another part begins. For
example, in this statement:

- int age;

there has to be at least one whitespace character, normally a space, in between int and age so that the compiler knows to distinguish between them and can read the statement correctly. However, in this statement:

- fruit = apples + oranges; // Get the total fruit

it isn’t necessary to put in a whitespace character between fruit and = or in between = and apples. You can if you wish but it won’t make any difference to the way C++ reads it.
SQL

SQL is a little different to Java and C++ and it follow a unique set of guidelines and rules. All statements in SQL have to begin with one of the keywords, such as INSERT, SELECT, CREATE, SHOW, ALTER, USE, DELETE, or UPDATE for example, and every statement must end with a semicolon.

It is important to remember that SQL is NOT case sensitive, which means that SELECT and select both have the same meaning. Take a look at the following examples of SQL statements and clauses to see how they are set out:

- SELECT Statement:
  ```sql
  SELECT column1, column2….columnN
  FROM  table_name;
  ```

- DISTINCT Clause:
  ```sql
  SELECT DISTINCT column1, column2….columnN
  FROM  table_name;
  ```

- WHERE Clause:
  ```sql
  SELECT column1, column2….columnN
  FROM  table_name
  WHERE  CONDITION;
  ```

- AND/OR Clause:
  ```sql
  SELECT column1, column2….columnN
  FROM  table_name
  WHERE  CONDITION-1 {AND|OR} CONDITION-2;
  ```

- IN Clause:
  ```sql
  SELECT column1, column2….columnN
  FROM  table_name
  WHERE  column_name IN (val-1, val-2,…val-N);
  ```

- BETWEEN Clause:
  ```sql
  SELECT column1, column2….columnN
  FROM  table_name
  ```
WHERE column_name BETWEEN val-1 AND val-2;

LIKE Clause:
SELECT column1, column2....columnN 
FROM table_name 
WHERE column_name LIKE { PATTERN };

ORDER BY Clause:
SELECT column1, column2....columnN 
FROM table_name 
WHERE CONDITION 
ORDER BY column_name {ASC|DESC};

GROUP BY Clause:
SELECT SUM(column_name) 
FROM table_name 
WHERE CONDITION 
GROUP BY column_name;

COUNT Clause:
SELECT COUNT(column_name) 
FROM table_name 
WHERE CONDITION;

HAVING Clause:
SELECT SUM(column_name) 
FROM table_name 
WHERE CONDITION 
GROUP BY column_name 
HAVING (arithmetic function condition);

CREATE TABLE Statement:
CREATE TABLE table_name( 
column1 datatype, 
column2 datatype, 
column3 datatype, 
..... 
columnN datatype, 
PRIMARY KEY( one or more columns )
- );

- DROP TABLE Statement:
  DROP TABLE table_name;

- CREATE INDEX Statement:
  CREATE UNIQUE INDEX index_name
  ON table_name ( column1, column2,…columnN);

- DROP INDEX Statement:
  ALTER TABLE table_name
  DROP INDEX index_name;

- DESC Statement:
  DESC table_name;

- TRUNCATE TABLE Statement:
  TRUNCATE TABLE table_name;

- ALTER TABLE Statement:
  ALTER TABLE table_name {ADD|DROP|MODIFY} column_name {data_type};

- ALTER TABLE Statement (Rename):
  ALTER TABLE table_name RENAME TO new_table_name;

- INSERT INTO Statement:
  INSERT INTO table_name( column1, column2….columnN)
  VALUES ( value1, value2….valueN);

- UPDATE Statement:
  UPDATE table_name
  SET column1 = value1, column2 = value2….columnN=valueN
  [ WHERE CONDITION ];

- DELETE Statement:
  DELETE FROM table_name
WHERE {CONDITION};

CREATE DATABASE Statement:
CREATE DATABASE database_name;

DROP DATABASE Statement:
DROP DATABASE database_name;

USE Statement:
USE database_name;

COMMIT Statement:
COMMIT;

ROLLBACK Statement:
ROLLBACK;

**SQL Loops**

There may come a time when you will have to execute a code block repeatedly. This is called a loop. In SQL, we use the following loop types:

- **SQL Basic LOOP** - The statement sequence is in between LOOP and END LOOP. The sequence is executed at each iteration and control begins again at the top of the loop.
- **SQL WHILE LOOP** - Provided a given condition is true, single or group of statements will repeat. The condition is tested before the loop body is executed.
- **SQL FOR LOOP** - Will perform multiple executions of a statement sequence and abbreviates the loop variable management code.
- **Nested loops in SQL** - Allows you to use a single or multiple loops inside any other basic, WHILE, or FOR loop.

**Labeling a Loop**

SQL loops can be given a label that must be enclosed in double angle brackets (<< and >>) and should be at the beginning of the statement or at the end. Labels can be used in
the EXIT statement to get out of the loop. The following shows how that works:

- DECLARE
  - i number(1);
  - j number(1);
- BEGIN
  - << outer_loop >>
  - FOR i IN 1..3 LOOP
    - << inner_loop >>
    - FOR j IN 1..3 LOOP
      - dbms_output.put_line('i is: ' || i || ' and j is: ' || j);
    - END loop inner_loop;
  - END loop outer_loop;
- END;
- /

When this piece of code is executed, you will see the following result:

- i is: 1 and j is: 1
- i is: 1 and j is: 2
- i is: 1 and j is: 3
- i is: 2 and j is: 1
- i is: 2 and j is: 2
- i is: 2 and j is: 3
- i is: 3 and j is: 1
- i is: 3 and j is: 2
- i is: 3 and j is: 3

PL/SQL procedure successfully completed.

Loop Control Statements

A loop control statement changes the execution of the code from it would normally be. As soon as execution has left a scope, all the automatic objects in the scope will be destroyed. The following control statements are supported by SQL:

- EXIT statement - This completes the loop and passes control to the statement right after END LOOP
- CONTINUE statement - This makes the loop skip over the rest of its body and makes it test its condition before it restarts
- GOTO statement - Gives control to a labeled statement but be aware that it isn’t recommended for you to use a GOTO statement in the program you write
Java

On occasion, you may find it necessary to repeatedly execute a particular block of code in Java. As a rule, all statements are executed in a sequence with the first statement in the function being executed first, followed by the others in turn. Java supports the following loops:

- **while loop** - Provided a given condition is true, this will repeat a single or multiple statements and will test the condition before the loop body is executed.
- **for loop** - Allows a statement sequence to be execute repeatedly and abbreviates the loop variable management code
- **do…while loop** - Similar to a while statement but will test out the condition at the end of the loop

**Loop Control Statements:**

As in SQL. These also change the normal sequence of the execution. When the scope is left, all the automatic objects in it are destroyed. These are the control statements supported by Java:

- **break statement** – Stops the switch or loop statement and then immediately transfers execution to the statement that directly follows it
- **continue statement** - Makes the loop skip the rest of the body ad retest condition before restarting

**Java Enhanced For Loop**

Again, this is relatively new in Java and is used to traverse through a collection of elements. The following example shows you how it is written:

```
for(declaration : expression)
{
    //Statements
}
```

**Declaration** – this is the new declared block variable, which is a type that is compatible with all of the elements in the array that you are trying to access. The variable is in the for
block and has a value that is the same as the current element in the array.

**Expression** – this is that evaluates to the array that you are going to loop through. It can be a method call that returns an array or it can be an array variable.

An example:

- public class Test {
  
  - public static void main(String args[]) {
  -   int [] numbers = {10, 20, 30, 40, 50};
  
  -   for(int x : numbers ) {
  -     System.out.print( x );
  -     System.out.print(“,”);
  -   }
  -   System.out.print(“n”);
  -   String [] names = {“James”, “Larry”, “Tom”, “Lacy”};
  -   for( String name : names ) {
  -     System.out.print( name );
  -     System.out.print(“,”);
  -   }
  -   }
  - }

Executing this would give the following result:

- 10,20,30,40,50,
- James,Larry,Tom,Lacy,
As with the other languages, you may want to repeat a code execution a number of times and the statements are executed in sequence. The following loops are supported by C++:

- **while loop** – this will repeat a single or group of statements so long as a given condition is true and will test the condition before it begins the execution
- **for loop** – will repeatedly execute a sequence of statements and abbreviates the loop variable management code
- **do…while loop** – similar to the while statement but will test the condition that is at the end of the body
- **nested loops** – using this, you can use more than one loop inside any do…while, for or while loop

**Loop Control Statements:**

These change the normal sequence of the execution in exactly the same way as Java and SQL. These are control statements that are supported by C++:

- break statement - This stops the switch or loop statement and pushes the execution to the statement that immediately follows
- continue statement - This one makes the loop skip the rest of the body and will retest condition before it restarts
- goto statement - Puts control to the labelled statement but it isn’t advisable to use this statement

**The Infinite Loop:**

If a condition can never become false then the loop will become an infinite loop. Traditionally in C++, we use the for loop. It is easy to make an endless loop with this because you do not need any of the three expressions that make up the for loop, meaning you can leave the conditional expression empty:

- #include <iostream>
- using namespace std;

- int main ()
When there is no conditional expression, it is automatically assumed that it is true. You can have an increment and initialization expression but most C++ programmers will use the construct for(;;) to signal that it is an infinite or endless loop. You can terminate infinite loops easily by pressing on CTRL+C on your keyboard.
Java Arrays

An array is a data structure that stores a sequence of fixed size elements that are all the same size. In layman’s terms, it is a storage place for a collection of data but is better thought of as a collection of variables that are all the same type.

In Java, instead of declaring the individual variables, like number0, number1, etc., we declare a single array variable, perhaps numbers, and then use number0, number1, etc. as a representation for the individual variables.

Declaring Array Variables

If you want to use an array in your program, you have to declare a variable that will reference the array and you must also specify the array type that the variable is able to reference. This is the syntax you would use:

- `dataType[] arrayRefVar; //`

Or

- `dataType arrayRefVar[]; //`

The top version is the preferred way. The following are examples of code showing the syntax:

- `double[] myList;       //`

Or

- `double myList[];       //`

Creating Arrays:
To create an array, you can use the new operator with this syntax:

- `arrayRefVar = new dataType[arraySize];`

This statement does two things:

- First, it creates the array using `new dataType[arraySize];`
- Second, the reference for the newly create array is given to the variable `arrayRefVar`

You can do all of this – declare an array variable, create an array and assign the reference to the variable in one single statement, as such:

- `dataType[] arrayRefVar = new dataType[arraySize];`

Or you can do this:

- `dataType[] arrayRefVar = {value0, value1, …, valuek};`

The array elements can be accessed via the index. All array indices are 0-based which means that they begin from 0 to `arrayRefVar.length-1`. An example of this:

The following statement declares the array variable, creates the array containing 10 elements, each of double type, and then assigns the reference:

- `double[] myList = new double[10];`

**Processing Arrays:**

When we process the elements of an array we tend to use the for loop or the foreach loop, simply because all of those elements are the same types and we know what the array size is.
The following is an example that shows you how to create an array, followed by initialization and processing:

- public class TestArray {
  
  - public static void main(String[] args) {
  - double[] myList = {1.9, 2.9, 3.4, 3.5};
  
  - // Print all the array elements
  - for (int i = 0; i < myList.length; i++) {
  - System.out.println(myList[i] + " ");
  - }
  
  - // Summing all elements
  - double total = 0;
  - for (int i = 0; i < myList.length; i++) {
  - total += myList[i];
  - }
  - System.out.println("Total is " + total);
  
  - // Finding the largest element
  - double max = myList[0];
  - for (int i = 1; i < myList.length; i++) {
  - if (myList[i] > max) max = myList[i];
  - }
  - System.out.println("Max is " + max);
  - }
  
  }

Executing this code would give this result:

- 1.9
- 2.9
- 3.4
- 3.5
- Total is 11.7
- ax is 3.5

The foreach Loops:
Foreach lops were introduced in the Java Development Kit v 1.5 and they allow you to go through an entire array in sequence without the need to use an index variable.

The following example shows an array called myList and all the elements included in it:

```java
public class TestArray {
    public static void main(String[] args) {
        double[] myList = {1.9, 2.9, 3.4, 3.5};

        // Print all the array elements
        for (double element : myList) {
            System.out.println(element);
        }
    }
}
```

Execution of this code would give this result:

```
1.9
2.9
3.4
3.5
```

**Passing Arrays to Methods:**

In the same way that you pass a primitive value to a method, you can also pass an array to a method. The following example shows the elements that are in an int array:

```java
public static void printArray(int[] array) {
    for (int i = 0; i < array.length; i++) {
        System.out.print(array[i] + " ");
    }
}
```

We invoke this by passing the array. This example shows a statement that invokes a
printArray method with the result of displaying 3, 1, 2, 6, 4, 2:

- printArray(new int[]{3, 1, 2, 6, 4, 2});

Returning an Array from a Method:

It is also possible for methods to return arrays. This example shows a method that returns an array that is actually the reverse of a different array:

- public static int[] reverse(int[] list) {
  int[] result = new int[list.length];

  for (int i = 0, j = result.length - 1; i < list.length; i++, j--) {
    result[j] = list[i];
  }
  return result;
}

The Arrays Class:

Included in the java.util.arrays class, there are a number of static methods that allow you to search, sort, compare and fill any array element:

- public static int binarySearch(Object[] a, Object key) - Allows you to search a specified object array for a specified value by using the binary search algorithm. Before this call is made, the array must be sorted. The result will be the index of the search key provided it is in the list already.
- public static boolean equals(long[] a, long[] a2) - This will return true if the specified arrays of two longs are the same as each other. Arrays are considered to be equal if each contains the same amount of elements and all the corresponding element pairs in the arrays are equal.
- public static void fill(int[] a, int val) - This assigns the int value that is specified to each separate element of the array of ints specified.
- public static void sort(Object[] a) - This allows you to sort the objects of a specified array into ascending order.
In SQL, we get a data structure that is called VARRAY and this can store a collection of fixed size sequential elements that are all the same type. Every VARRAY has contiguous memory locations, which means that the lowest address will correspond to the first element and the highest to the last. Each separate element in a VARRAY has an associated index and a maximum size that can be dynamically changed.

Creating a VARRAY Type

VARRAY types are created using the CREATE TYPE statement. You need to specify the element types that will be stored in the array and the maximum size. To create a VARRAY type at schema level, the basic syntax should be:

```
CREATE OR REPLACE TYPE varray_type_name IS VARRAY(n) of <element_type>
```

Where,

- varray_type_name is a valid attribute name,
- n is the number of elements (maximum) in the varray,
- element_type is the data type of the elements of the array.

To change the maximum size, you can use the ALTER TYPE statement, as such:

```
CREATE Or REPLACE TYPE namearray AS VARRAY(3) OF VARCHAR2(10);
/
```

- Type created.

To create a VARRAY type in an SQL block, you would use this basic syntax:

```
TYPE varray_type_name IS VARRAY(n) of <element_type>
```
An example of that is:

- TYPE namearray IS VARRAY(5) OF VARCHAR2(10);
- Type grades IS VARRAY(5) OF INTEGER;

This example shows how to use VARRAYS:

- DECLARE
  - type namesarray IS VARRAY(5) OF VARCHAR2(10);
  - type grades IS VARRAY(5) OF INTEGER;
  - names namesarray;
  - marks grades;
  - total integer;
  - BEGIN
  - names := namesarray('Kavita', 'Pritam', 'Ayan', 'Rishav', 'Aziz');
  - marks:= grades(98, 97, 78, 87, 92);
  - total := names.count;
  - dbms_output.put_line('Total ' || total || ' Students');
  - FOR i in 1 .. total LOOP
  - dbms_output.put_line('Student: ' || names(i) || ' Marks: ' || marks(i));
  - END LOOP;
  - END;
  - /

Execution of this code is done at the SQL prompt and it will give this result:

- Student: Kavita  Marks: 98
- Student: Pritam  Marks: 97
- Student: Ayan   Marks: 78
- Student: Rishav  Marks: 87
- Student: Aziz   Marks: 92

- PL/SQL procedure successfully completed.

Points to note:
- In the Oracle environment, the beginning index for a VARRAY is always 1
- You can use the constructor method to initialize the elements in a VARRAY type so long as it has the same name as the VARRAY does
- VARRAYS are always one-dimensional
- VARRAYS are NULL when they are declared – before the elements can be referenced it must be initialized

In this example, we see the concept of a VARRAY being a %ROWTYPE of a database table or %TYPE of a database table field:

```sql
Select * from customers;
```

```
+----+----------+-------+----------+-------+
<table>
<thead>
<tr>
<th>ID</th>
<th>NAME</th>
<th>AGE</th>
<th>ADDRESS</th>
<th>SALARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Ramesh</td>
<td>32</td>
<td>Ahmedabad</td>
<td>2000.00</td>
</tr>
<tr>
<td>2</td>
<td>Khilan</td>
<td>25</td>
<td>Delhi</td>
<td>1500.00</td>
</tr>
<tr>
<td>3</td>
<td>kaushik</td>
<td>23</td>
<td>Kota</td>
<td>2000.00</td>
</tr>
<tr>
<td>4</td>
<td>Chaitali</td>
<td>25</td>
<td>Mumbai</td>
<td>6500.00</td>
</tr>
<tr>
<td>5</td>
<td>Hardik</td>
<td>27</td>
<td>Bhopal</td>
<td>8500.00</td>
</tr>
<tr>
<td>6</td>
<td>Komal</td>
<td>22</td>
<td>MP</td>
<td>4500.00</td>
</tr>
</tbody>
</table>
+----+----------+-------+----------+--------+
```

This example uses cursor:

```sql
DECLARE CURSOR c_customers is
SELECT name FROM customers;
type c_list is varray (6) of customers.name%type;
```
name_list := c_list();
counter := 0;
BEGIN
FOR n IN c_customers LOOP
  counter := counter + 1;
  name_list.extend;
  name_list(counter) := n.name;
  dbms_output.put_line('Customer('||counter||'):'||name_list(counter));
END LOOP;
END;
/

When we execute this code, we get this result:

- Customer(1): Ramesh
- Customer(2): Khilan
- Customer(3): kaushik
- Customer(4): Chaitali
- Customer(5): Hardik
- Customer(6): Komal

- PL/SQL procedure successfully completed.
C++

C++ is similar to Java in the data structure of the array, storing sequential fixed size collection of elements. And, it is the same as SQL in that all arrays have contiguous memory locations

Declaring Arrays:

In C++, you must specify the element types and the number of elements that are required in the array:

- type arrayName [ arraySize ];

This a one-dimension array. The arraySize has to be an integer constant that is more than zero ad the type is any C++ data type, so long as it is valid. For example, if you wanted to declare an array with 10 elements, that is called a balance of type double, you would use this statement:

- double balance[10];

Initializing Arrays:

In C++, we initialize array elements in two ways – one at a time or by using one statement:

- double balance[5] = {1000.0, 2.0, 3.4, 17.0, 50.0};

The values between { } cannot be any bigger in number than the number of the elements that are declared between [ ]. This example shows you how to assign one ingle element of the array. Note that if you leave out the array size, an array that is just large enough to hold the initialization is created. So, if you were to write:

- double balance[] = {1000.0, 2.0, 3.4, 17.0, 50.0};
You are creating the same array that you did in the initial example:

- \( \text{balance}[4] = 50.0; \)

This statement assigns the element number that is fifth in the array with a value of 50.0. So, because arrays begin with 0 as the first element index, an array that has a 4th index will be numbered 5th.

**Accessing Array Elements:**

Elements of an array can be accessed through indexing the name of the array. We do this by putting the element index inside square brackets following the array name. An example of that:

- \( \text{double salary} = \text{balance}[9]; \)

This statement takes element 10 out of the array and gives the value to salary variable. The following example shows all of the above concepts – declaring, assigning and accessing the arrays:

```cpp
#include <iostream>
using namespace std;

#include <iomanip>
using std::setw;

int main ()
{
    int n[10]; // n is an array of 10 integers

    // initialize elements of array n to 0
    for ( int i = 0; i < 10; i++ )
    {
        n[ i ] = i + 100; // set element at location i to i + 100
    }
    cout << "Element" << setw(13) << "Value" << endl;
```
// output each array element’s value
for ( int j = 0; j < 10; j++ )
{
    cout << setw( 7 ) << j << setw( 13 ) << n[ j ] << endl;
}

return 0;
}

When we compile and execute this code, we get this result:

<table>
<thead>
<tr>
<th>Element</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>1</td>
<td>101</td>
</tr>
<tr>
<td>2</td>
<td>102</td>
</tr>
<tr>
<td>3</td>
<td>103</td>
</tr>
<tr>
<td>4</td>
<td>104</td>
</tr>
<tr>
<td>5</td>
<td>105</td>
</tr>
<tr>
<td>6</td>
<td>106</td>
</tr>
<tr>
<td>7</td>
<td>107</td>
</tr>
<tr>
<td>8</td>
<td>108</td>
</tr>
<tr>
<td>9</td>
<td>109</td>
</tr>
</tbody>
</table>
Strings
Java

Strings are used a lot in Java and they are objects that contain a sequence of characters. Java allows users to both create and manipulate a string.

**Creating a String**

The easiest way to create a string is this way:

- String greeting = “Hello world!”;

Whenever the Java compiler comes across a string literal written in your code, it will create a string object; in the case of the syntax above it would be “Hello, World!”.

In the same way you can with any object, you can use a constructor and the new keyword to create string objects. The string class in Java has 11 constructors in it, which enable you to use different sources to give the initial string value:

- public class StringDemo{

  - public static void main(String args[]){
  -   String helloString = new String(helloArray);
  -   System.out.println( helloString );
  - }
  - }

Compiling and executing this code would give this result:

- hello.

Be aware that the string class is immutable. This means that, once you have created the string object, you can’t change it. If you find that you have to make changes to character strings then you should use the string builder and the string buffer classes.
**String Length:**

A method that is used to get information about a particular object is called an accessor. You can use the `length()` accessor method with strings, allowing you to return the value of the number of characters that are in the string object.

Below is an example of this method:

```java
public class StringDemo {
    public static void main(String args[]) {
        String palindrome = "Dot saw I was Tod";
        int len = palindrome.length();
        System.out.println("String Length is : "+ len);
    }
}
```

Which would give this result:

String Length is : 17

**Concatenating Strings:**

There is also a method in the string class for concatenating two strings:

```java
  string1.concat(string2);
```

This would return a new string, with string1 and the beginning and string 2 added to the end. You can also use this `concat()` method with string literals, as in this example:

```java
  "My name is ".concat("Zara");
```

More commonly, we see strings concatenated with the `+` operator:

```java
  "Hello," + " world" + "!
```
And that gives this result:

- “Hello, world!”

Have a look at this example:

- public class StringDemo {

  - public static void main(String args[]) {
    - String string1 = “saw I was “;
    - System.out.println(“Dot ” + string1 + “Tod”);
  - }  
  - }

Which would give this result:

- Dot saw I was Tod

Creating Format Strings:

You have two methods to print an output with formatted numbers – printf() and format(). In Java, the string class contains a class method that will return a string object instead of a printStream object – format(). This is a static method and using this lets you create a reusable formatted string instead of a one-time only print statement. As an example, instead of writing this:

```
System.out.printf(“The value of the float variable is ” +
  “%f, while the value of the integer ” +
  “variable is %d, and the string ” +
  “is %s”, floatVar, intVar, stringVar);
```

You can do this:

```
String fs;
fs = String.format(“The value of the float variable is ” +
```
• “%f, while the value of the integer ” +
• “variable is %d, and the string ” +
• “is %s”, floatVar, intVar, stringVar);
• System.out.println(fs);

String Methods:

These are the methods that are supported by the string class in Java:

• char charAt(int index) - Will return the character at a specified index
• int compareTo(Object o) - Compares the specified string to another object
• int compareTo(String anotherString) - compares a pair of strings lexicographically
• int compareToIgnoreCase(String str) - compares a pair of strings lexicographically but ignoring any differences in case
• String concat(String str) - Concatenates the string that is specified to the end of the given string
• boolean contentEquals(StringBuffer sb) - Will return true if, and only if, the string has the same character sequence as the StringBuffer that is specified
• static String copyValueOf(char[] data) - Will return a string that shows the sequence of characters in the specified array
• static String copyValueOf(char[] data, int offset, int count) - Will return a string that shows the sequence of characters in the specified array
• boolean endsWith(String suffix) - Will test that the string finishes with the suffix that is specified
• boolean equals(Object anObject) - Will compare the string to the object that is specified
• boolean equalsIgnoreCase(String anotherString) - Will compare the string to another one but will ignore case
• byte getBytes() - Will encode the string to a byte sequence and uses the charset that is named. The result is stored in a new byte array
• byte[] getBytes(String charsetName) - Does the same as the above method
• void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin) - Will copy specified characters from the string into the specified character array
• int hashCode() - Will return a hashcode for the particular string
• int indexOf(int ch) - Will return the index that is in this string of the first instance of the character specified
• int indexOf(int ch, int fromIndex) - Same as the above method but beginning the search at the index specified
• int indexOf(String str) - As above but from the first instance of the substring specified
• int indexOf(String str, int fromIndex) - As above but starting from the index that
is specified

- **String intern()** - Will return a canonical representation for the specified string object
- **int lastIndexOf(int ch)** - Will return the index that is in the string, from the last instance of the character specified
- **int lastIndexOf(int ch, int fromIndex)** - As above but searches backwards, from the index specified
- **int lastIndexOf(String str)** - as above but returns the index that is the rightmost instant of the substring specified
- **int lastIndexOf(String str, int fromIndex)** - Will return the index in the string of the last instant of the substring specified, but searches backwards from the index specified
- **int length()** - will return the string length
- **boolean matches(String regex)** - will tell you if this string matches the regular expression that is given
- **boolean regionMatches(boolean ignoreCase, int toffset, String other, int ooffset, int len)** - Will test if two strings are equal to one another
- **boolean regionMatches(int toffset, String other, int ooffset, int len)** - Will test if two string regions ae equal to one another
- **String replace(char oldChar, char newChar)** - Will return a new string that is the result of the replacement of all the instants of oldChar with newChar
- **String replaceAll(String regex, String replacement** - Will replace all the substrings of the string that match up to the regular expression with the specified replacement
- **String replaceFirst(String regex, String replacement)** - As above but only replaces the first substring
- **String[] split(String regex)** - Will split the string around any matches of the specified regular expression
- **String[] split(String regex, int limit)** - As above
- **boolean startsWith(String prefix)** - Tests to see if the string begins with the prefix that is specified
- **boolean startsWith(String prefix, int toffset)** - As above but on a specified index
- **CharSequence subSequence(int beginIndex, int endIndex)** - Will return a new sequence of characters that is as a subsequence of the given sequence
- **String substring(int beginIndex)** - Will return a new string that will be a substring of the given string
- **String substring(int beginIndex, int endIndex)** - As above
- **char[] toCharArray()** - Will convert the string to a new character array
- **String toLowerCase()** - Will convert all characters in the string to lower case with the use of the rules for the given locale
- **String toLowerCase(Locale locale)** - As above
- **String toString()** - The object, a string, returns itself
- **String toUpperCase()** - Converts the characters int eh string to upper case, using the default locale rules
- **String toUpperCase(Locale locale)** - As above but using the given locale
- **String trim()** - Will return a copy of the string but omitting the leading and the trailing whitespaces
- **static String valueOf(primitive data type x)** - Will return the string representation of the specified data type argument
C++

C++ has support for two string representations:

- C-style character string.
- String class type that was introduced along with standard C++

**The C-Style Character String:**

This came with the original C language and has been given full support in C++. It is a one-dimensional character array, terminated by a null character. The following example is of a declaration and initialization that creates a string with the word “Hello” in it. In order for the null character to be held at the end, the character array size must be one more than the amount of characters in the word “Hello”:

- char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};

If you were to follow the array initialization rule, you could write that statement as this:

- char greeting[] = “Hello”;

You do not actually put the null character on the end of the string constant because, when it initializes the array the compiler will place it at the end of the string for you.

The following example shows you how to print that string:

- #include <iostream>

- using namespace std;

- int main ()
  {
   char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
   }
cout << “Greeting message: “;
cout << greeting << endl;

return 0;
}
}

When we compile and execute this piece of code, we get this result:

- Greeting message: Hello

C++ provides support for a number of functions that are able to manipulate strings that are null-terminated:

- strcpy(s1, s2); - will copy string s2 to string s1
- strcat(s1, s2); - Will concatenate string s2 to the end of string s1
- strlen(s1); - Will return the length of the specified string
- strcmp(s1, s2); - Will return 0 provided s1 and s2 are identical and will return less than 0 if s1 is less than s2, greater than 0 if s1 is greater than s2.
- strchr(s1, ch); - Will return a pointer to the first instance of the specified character, ch, in string s1
- strstr(s1, s2); - will return a pointer to the first instance of string s2 in string s1

The following example shows some of these functions in use:

- #include <iostream>

- #include <cstring>

- using namespace std;

- int main ()
  {
    char str1[10] = “Hello”;
    char str2[10] = “World”;
    char str3[10];
    int len ;
• // copy str1 into str3
• strcpy( str3, str1);
• cout << "strcpy( str3, str1) : " << str3 << endl;

• // concatenates str1 and str2
• strcat( str1, str2);
• cout << "strcat( str1, str2): " << str1 << endl;

• // total length of str1 after concatenation
• len = strlen(str1);
• cout << "strlen(str1) : " << len << endl;

• return 0;
• }
• When we compile and execute this code, we get a result like this one:
• strcpy( str3, str1) : Hello
• strcat( str1, str2): HelloWorld
• strlen(str1) : 10

**The String Class in C++:**

This string class supports all of those functions that we mentioned above as well as many more besides. The following example uses objects and classes:

• #include <iostream>
• #include <string>

• using namespace std;

• int main ()
• {
• string str1 = “Hello”;  
• string str2 = “World”; 
• string str3; 
• int len;
When compiled and executed, the above code produces this result:

- `str3 : Hello`
- `str1 + str2 : HelloWorld`
- `str3.size() : 10`
SQL strings are character sequences with an optional specification of size. These characters can be alpha, numeric, special characters, blank spaces, or a combination of all of these. SQL supports three string types:

- **Fixed Length** - You specify how long the string is while declaring it. Strings are right-padded with spaces that correspond to the specified length
- **Variable Length** - Can be a maximum length of 32,767 characters and there is no padding
- **CLOBs – Character Large Objects** - Variable in length up to 128 terabytes in size

SQL strings can be literal or they can be variable. String literals must be enclosed inside quotation marks, as such:

- ‘This is a string literal.’ Or ‘hello world’

If you want to include a single quote in a string literal, you must use a pair of single quotes beside each other:

- ‘this isn’t what it looks like’

**Declaring String Variables**

There are quite a number of string datatypes in the Oracle database, including NVARCHAR, VARCHAR2, CHAR, CHAR, CLOB and NCLOB. Those that have an N as a prefix are “national character types” and will store character data in Unicode.

If you have to declare a string that is variable length, you have to provide the maximum string length. This example demonstrates the declaration and use of some of the string variables:

- DECLARE
- name varchar2(20);
- company varchar2(30);
introduction clob;
choice char(1);
BEGIN
name := 'John Smith';
company := 'Infotech';
introduction := 'Hello! I”m John Smith from Infotech.';
choice := 'y';
IF choice = 'y' THEN
  dbms_output.put_line(name);
  dbms_output.put_line(company);
  dbms_output.put_line(introduction);
END IF;
END;
/

When we compile and execute this code, we get this result:

- John Smith
- Infotech Corporation
- Hello! I’m John Smith from Infotech.

- PL/SQL procedure successfully completed

You would use the CHAR datatype to declare a fixed length string and you do not need to specify the maximum length. If you omit the length constraint, Oracle will automatically use the maximum length that is required. These two examples are actually the same:

- red_flag CHAR(1) := 'Y';
- red_flag CHAR := 'Y';

**SQL String Functions and Operators**

SQL uses the concatenation operator to join two strings together - (||). Below are the SQL string functions.

- **ASCII(x);** - Will return the ASCII value of the specified character
- **CHR(x)**; - Will return the character with the specified ASCII value
- **CONCAT(x, y)**; - will concatenate the two specified strings and return the new string – one string with the other appended
- **INITCAP(x)**; - Will convert the first letter of each word in the specified string to uppercase and then returns the new string
- **INSTR(x, find_string [, start [, occurrence]]);** - Will search for the specified string and return the position it finds it at
- **INSTRB(x)**; - will return the location of the specified string inside another string but the result is the value in bytes
- **LENGTH(x)**; - will return the number of characters in the specified string
- **LENGTHB(x)**; - Will return the character string length in bytes for character sets that are single byte
- **LOWER(x)**; - Will convert the letters in the specified string to lower case and then return the new string
- **LPAD(x, width [, pad_string])** ; - Will pad the specified string with spaces on the left to make the total number of characters match the width characters
- **LTRIM(x [, trim_string])**; - Will trim off the characters to the left of the specified string
- **NANVL(x, value)**; - Will return the value provided x is the same as the NaN special value. If not an x is returned
- **NLS_INITCAP(x)**; - This is the same as INITCAP function but uses a different method of sorting
- **NLS_LOWER(x)**; - This is the same as the LOWER function with a different sort method
- **NLS_UPPER(x)**; - This is the same as the UPPER function with a different sort method
- **NLSSORT(x)**; - This changes the method used to sort the characters and must be specified before NLS functions. If not, the default sort method is used
- **NVL(x, value)**; - Will return a value if x is a null. If not then x will be returned
- **NVL2(x, value1, value2)**; - will return value1 if x is not a null but, if it is, it will return value2
- **REPLACE(x, search_string, replace_string)**; - Will search the specified string and replace it with the specified text
- **RPAD(x, width [, pad_string])**; - Will pad x on the right
- **RTRIM(x [, trim_string])**; - Will trim x from the right
- **SOUNDEX(x)** ; - Will return a string that contains the phonetic representation
- **SUBSTR(x, start [, length])**; - Will return a substring that starts at the specified position. You can supply an length for the substring if you wish but this is optional
- **SUBSTRB(x)**; - This is the same as SUBSTR except for the parameter expression n bytes instead of in characters where the character system is single byte
- **TRIM([trim_char FROM) x]**; - Will trim the characters from both left and right
- **UPPER(x)**; - Will convert the letters to uppercase and then return the new string
Below are some examples that show these functions in use.

Example 1

- DECLARE
greetings varchar2(11) := ‘hello world’;
- BEGIN
dbms_output.put_line(UPPER(greetings));

dbms_output.put_line(LOWER(greetings));

dbms_output.put_line(INITCAP(greetings));

/* retrieve the first character in the string */
dbms_output.put_line ( SUBSTR (greetings, 1, 1));

/* retrieve the last character in the string */
dbms_output.put_line ( SUBSTR (greetings, -1, 1));

/* retrieve five characters, starting from the seventh position. */
dbms_output.put_line ( SUBSTR (greetings, 7, 5));

/* retrieve the remainder of the string, starting from the second position. */
dbms_output.put_line ( SUBSTR (greetings, 2));

/* find the location of the first “e” */
dbms_output.put_line ( INSTR (greetings, ‘e’));
END;
/

Execution of the above code results in this:

- HELLO WORLD
- hello world
- Hello World
- h
d
- World
ello World
- 2

- PL/SQL procedure successfully completed.

**Example 2**

- DECLARE
greetings varchar2(30) := ‘……Hello World…..’;
- BEGIN
dbms_output.put_line(RTRIM(greetings,’.’));
dbms_output.put_line(LTRIM(greetings, ‘.’));
dbms_output.put_line(TRIM( ‘.’ from greetings));
- END;
- /
- And the result of this code is this:
  - ……Hello World
  - Hello World…..
  - Hello World

- PL/SQL procedure successfully completed.
Glossary of Common Programming Terms

There are a large number of terms that you will come across in your programming but the following are the most common and the ones that you need to learn in order to get started:
Compiler

The program that converts your written code into computer language so that it can be executed
Database

A file that holds information in a structured manner, similar in many ways to a spreadsheet. These are used to store data in and to enable you to retrieve that data for the program to use.
Algorithm

A set of steps or instructions that are used in solving a problem. If you are asked how to do something, you told that person to “do this, the do this, and then do that, and if you see this, you should do that” you have basically given them an algorithm. This is the kind of thing that computers use to gain results based on the data that is contained in the program.
Object Oriented

OOP or Object Oriented Programming is fairly recent in computer programming languages, basically a new design on the way that programmers think about how they are going to solve a particular problem. Instead of using the tried and tested method of functions and algorithms, you think instead about each “object” in the program and what it needs to do. As a beginner, you do not need to worry too much about the concepts of OOP – the time for leaning them will become clear as you work your way up the ladder as a programmer.
A platform is the specific type of operating system and hardware that you are creating a program for. The most common ones are Windows, Mac, iOS and Android.
Conclusion

I hope that you found my introduction of computer programming helpful. It is a very basic start but it should have given you some idea on how to begin. It should also have shown you that computer programming really isn’t all that difficult and can be quite exciting, especially as you start to see your results appear on screen and see your computer, in short, doing what it is told to do!

If you found that this has given you a good idea of what to expect then you may want to move on to more advanced programming in your chosen language. A word of warning here – do not try to learn more than one language at a time, otherwise you will find yourself in a muddle. The only other piece of advice I will give you at this stage is to practice…and keep on practicing. The more you do the more you will learn, and the more you will want to learn.

Thank you for downloading my book; if you found it helpful please consider leaving me a review at Amazon.com.
Check Out My Other Books

Below you’ll find some of my other popular books that are popular on Amazon and Kindle as well. Simply click on the links below to check them out. Alternatively, you can visit my author page on Amazon to see other work done by me.

*Raspberry Pi 2: The Definitive Beginner’s Guide to Get Started with Raspberry Projects*

*Hacking: Hacking for Beginners - Computer Virus, Cracking, Malware, IT Security*

*Python: The Definitive Guide to Learning Python Programming for Beginners*

If the above links do not work, you can simply search for these titles on the Amazon website to find them.