

# Introduction to Programming

CSC 103



*University of Ibadan Distance Learning Centre  
Open and Distance Learning Course Series Development*

Copyright © 2016 by Distance Learning Centre, University of Ibadan, Ibadan.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior permission of the copyright owner.

ISBN: 978-021-592-1

*General Editor:* Prof. Bayo Okunade

**University of Ibadan Distance Learning Centre**  
University of Ibadan,  
Nigeria

Telex: 31128NG

Tel: +234 (80775935727)

E-mail: [ssu@dlc.ui.edu.ng](mailto:ssu@dlc.ui.edu.ng)

Website: [www.dlc.ui.edu.ng](http://www.dlc.ui.edu.ng)

## **Vice-Chancellor's Message**

The Distance Learning Centre is building on a solid tradition of over two decades of service in the provision of External Studies Programme and now Distance Learning Education in Nigeria and beyond. The Distance Learning mode to which we are committed is providing access to many deserving Nigerians in having access to higher education especially those who by the nature of their engagement do not have the luxury of full time education. Recently, it is contributing in no small measure to providing places for teeming Nigerian youths who for one reason or the other could not get admission into the conventional universities.

These course materials have been written by writers specially trained in ODL course delivery. The writers have made great efforts to provide up to date information, knowledge and skills in the different disciplines and ensure that the materials are user-friendly.

In addition to provision of course materials in print and e-format, a lot of Information Technology input has also gone into the deployment of course materials. Most of them can be downloaded from the DLC website and are available in audio format which you can also download into your mobile phones, IPod, MP3 among other devices to allow you listen to the audio study sessions. Some of the study session materials have been scripted and are being broadcast on the university's Diamond Radio FM 101.1, while others have been delivered and captured in audio-visual format in a classroom environment for use by our students. Detailed information on availability and access is available on the website. We will continue in our efforts to provide and review course materials for our courses.

However, for you to take advantage of these formats, you will need to improve on your I.T. skills and develop requisite distance learning Culture. It is well known that, for efficient and effective provision of Distance learning education, availability of appropriate and relevant course materials is a *sine qua non*. So also, is the availability of multiple plat form for the convenience of our students. It is in fulfilment of this, that series of course materials are being written to enable our students study at their own pace and convenience.

It is our hope that you will put these course materials to the best use.



**Prof. Abel Idowu Olayinka**

Vice-Chancellor

## **Foreword**

As part of its vision of providing education for “Liberty and Development” for Nigerians and the International Community, the University of Ibadan, Distance Learning Centre has recently embarked on a vigorous repositioning agenda which aimed at embracing a holistic and all encompassing approach to the delivery of its Open Distance Learning (ODL) programmes. Thus we are committed to global best practices in distance learning provision. Apart from providing an efficient administrative and academic support for our students, we are committed to providing educational resource materials for the use of our students. We are convinced that, without an up-to-date, learner-friendly and distance learning compliant course materials, there cannot be any basis to lay claim to being a provider of distance learning education. Indeed, availability of appropriate course materials in multiple formats is the hub of any distance learning provision worldwide.

In view of the above, we are vigorously pursuing as a matter of priority, the provision of credible, learner-friendly and interactive course materials for all our courses. We commissioned the authoring of, and review of course materials to teams of experts and their outputs were subjected to rigorous peer review to ensure standard. The approach not only emphasizes cognitive knowledge, but also skills and humane values which are at the core of education, even in an ICT age.

The development of the materials which is on-going also had input from experienced editors and illustrators who have ensured that they are accurate, current and learner-friendly. They are specially written with distance learners in mind. This is very important because, distance learning involves non-residential students who can often feel isolated from the community of learners.

It is important to note that, for a distance learner to excel there is the need to source and read relevant materials apart from this course material. Therefore, adequate supplementary reading materials as well as other information sources are suggested in the course materials.

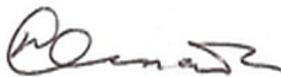
Apart from the responsibility for you to read this course material with others, you are also advised to seek assistance from your course facilitators especially academic advisors during your study even before the interactive session which is by design for revision. Your academic advisors will assist you using convenient technology including Google Hang Out, You Tube, Talk Fusion, etc. but you have to take advantage of these. It is also going to be of immense advantage if you complete assignments as at when due so as to have necessary feedbacks as a guide.

The implication of the above is that, a distance learner has a responsibility to develop requisite distance learning culture which includes diligent and disciplined self-study, seeking available administrative and academic support and acquisition of basic information technology skills. This is why you are encouraged to develop your computer skills by availing yourself the opportunity of training that the Centre’s provide and put these into use.

In conclusion, it is envisaged that the course materials would also be useful for the regular students of tertiary institutions in Nigeria who are faced with a dearth of high quality textbooks. We are therefore, delighted to present these titles to both our distance learning students and the university's regular students. We are confident that the materials will be an invaluable resource to all.

We would like to thank all our authors, reviewers and production staff for the high quality of work.

Best wishes.

A handwritten signature in black ink, appearing to read 'Bayo Okunade', written in a cursive style.

**Professor Bayo Okunade**

Director

# Course Development Team

Content Authoring

Yetunde Folajimi

Content Editor

Prof. Remi Raji-Oyelade

Production Editor

Ogundele Olumuyiwa Caleb

Learning Design/Assessment Authoring

Folajimi Olambo Fakoya

Managing Editor

Ogunmefun Oladele Abiodun

General Editor

Prof. Bayo Okunade

# Table of Contents

---

|   |           |
|---|-----------|
| <b>About this course manual</b>                               | <b>1</b>  |
| How this course manual is structured.....                     | 1         |
| <b>Course Overview</b>  | <b>3</b>  |
| Welcome to Introduction to Programming CIS104.....            | 3         |
| Course outcomes.....  | 3         |
| <b>Getting around this course manual</b>                      | <b>6</b>  |
| Margin icons.....   | 6         |
| <b>Study Session 1</b>  | <b>7</b>  |
| Overview of Programming.....                                  | 7         |
| Introduction.....   | 7         |
| Terminology.....  | 7         |
| 1.1 Definition of Programming.....                            | 7         |
| 1.1.1 Application Versus System Programming.....              | 8         |
| 1.2 The Computer Program.....                                 | 9         |
| 1.2.1 Writing a Computer Program.....                         | 10        |
| 1.3 Characteristics of Computer Program.....                  | 11        |
| 1.3.1 Portability.....  | 11        |
| 1.3.2 Readability.....  | 11        |
| 1.3.3 Efficiency.....   | 12        |
| 1.3.4 Structural.....   | 12        |
| 1.3.5 Flexibility.....  | 12        |
| 1.3.6 Generality.....   | 12        |
| 1.3.7 Documentation.....                                      | 12        |
| 1.4 The Computer Programmer.....                              | 14        |
| 1.4.1 The Work of a Computer Programmer.....                  | 14        |
| Study Session Summary.....                                    | 15        |
| Assessment.....   | 15        |
| Bibliography.....   | 16        |
| <b>Study Session 2</b>  | <b>17</b> |
| History of Programming Languages.....                         | 17        |
| Introduction.....   | 17        |
| Terminology.....  | 17        |
| 2.1 The Evolution of Programming Languages.....               | 17        |
| 2.1.1 The First Computer Language.....                        | 19        |
| 2.1.2 The Birth of Assembly Language.....                     | 19        |
| 2.1.3 The First Modern Computer Languages.....                | 20        |
| 2.1.4 More Recent Modern Languages.....                       | 21        |
| 2.2 Categorizing Computer Languages by Epochs.....            | 24        |
| 2.2.1 Summary of Important Programming Languages by Year..... | 26        |

|                            |    |
|----------------------------|----|
| Study Session Summary..... | 28 |
| Assessment.....            | 28 |
| Bibliography.....          | 28 |

## Study Session 3 29

|   |    |
|---|----|
| The Programming Process.....                    | 29 |
| Introduction.....                               | 29 |
| Terminology.....                                | 29 |
| 3.1 Problem Definition and Analysis.....        | 30 |
| 3.1.1 Planning and Outlining the Solution ..... | 30 |
| 3.2 Coding the Program.....                     | 31 |
| 3.3 Compilation.....                            | 32 |
| 3.4 Debugging.....                              | 33 |
| 3.4.1 Syntax .....                              | 33 |
| 3.4.2 Semantics.....                            | 34 |
| 3.4.3 Programming Errors.....                   | 35 |
| 3.5 Testing and Validation .....                | 37 |
| 3.5.1 Types of Programming Tests.....           | 38 |
| 3.6 Program Documentation.....                  | 42 |
| Study Session Summary.....                      | 43 |
| Assessment.....                                 | 43 |
| Bibliography.....                               | 44 |

## Study Session 4 44

|   |    |
|---|----|
| Types of Programming Languages.....                               | 45 |
| Introduction.....   | 45 |
| 4.1 Characteristics of Programming Languages.....                 | 45 |
| 4.2 Comparing Low-level and High-level Programming Languages..... | 46 |
| 4.3 Compiled code and interpreted code.....                       | 47 |
| 4.4 Classification of Programming Languages .....                 | 47 |
| 4.4.1 Modular Programming Languages.....                          | 48 |
| 4.4.2 Structured Programming Language .....                       | 48 |
| 4.4.3 Business Oriented Language .....                            | 49 |
| 4.4.4 Object oriented programming (OOP) language.....             | 50 |
| 4.4.5 Visual Programming Languages.....                           | 51 |
| Study Session Summary.....  | 52 |
| Assessment.....   | 52 |
| Credits.....  | 52 |

## Study Session 5 54

|  |                                     |
|--|-------------------------------------|
| Algorithms and Problem-Solving.....      | 54                                  |
| Introduction.....                        | 54                                  |
| Terminology.....                         | <b>Error! Bookmark not defined.</b> |
| 5.1 The Problem Solving Process.....     | 54                                  |
| 5.2 The Concept of Algorithm.....        | 55                                  |
| 5.2.1 Properties of Algorithm .....      | 56                                  |
| 5.2.2 Algorithmic Problem Solving .....  | 56                                  |
| 5.3 Pseudo-Codes.....                    | 58                                  |
| 5.3.1 Rules for Writing Pseudocode ..... | 58                                  |
| 5.3.2 Advantages of Pseudocode.....      | 60                                  |



|  |    |
|--|----|
| 5.3.3 Disadvantages of Pseudocode.....         | 60 |
| 5.4 Flowchart.....                             | 61 |
| 5.4.1 Advantages of Flowcharts.....            | 61 |
| 5.4.2 Disadvantages of Flowcharts.....         | 62 |
| 5.4.3 Flowchart Symbols .....                  | 62 |
| 5.4.4 General Guidelines in Flowcharting ..... | 63 |
| Study Session Summary .....                    | 66 |
| Assessment.....                                | 66 |
| Credits.....                                   | 66 |

## Study Session 6

67

|  |    |
|--|----|
| Basics of Computer Program .....                 | 67 |
| Introduction .....                               | 67 |
| Terminology.....                                 | 67 |
| 6.1 Programming Environment.....                 | 67 |
| 6.1.1 Text Editor.....                           | 68 |
| 6.1.2 Compiler .....                             | 69 |
| 6.1.3 Interpreter.....                           | 70 |
| 6.2 Basic Syntax of Programming .....            | 71 |
| 6.2.1 Program Entry Point .....                  | 71 |
| 6.2.2 Functions.....                             | 71 |
| 6.2.3 Comments.....                              | 72 |
| 6.2.4 Whitespaces.....                           | 72 |
| 6.2.5 Semicolons.....                            | 74 |
| 6.2.6 Syntax Error.....                          | 75 |
| 6.2.7 Hello World Program in Java .....          | 76 |
| 6.2.8 Hello World Program in Python.....         | 76 |
| 6.3 Variables.....                               | 76 |
| 6.3.1 Creating Variables.....                    | 77 |
| 6.3.2 Storing Values in Variables.....           | 78 |
| 6.3.3 Accessing Stored Values in Variables ..... | 79 |
| 6.3.4 Variables in Java.....                     | 80 |
| 6.3.5 Variables in Python.....                   | 80 |
| 6.4 Reserved Words.....                          | 81 |
| 6.4.1 C Programming Reserved Keywords .....      | 82 |
| 6.4.2 Java Programming Reserved Keywords .....   | 82 |
| 6.4.3 Python Programming Reserved Keywords ..... | 83 |
| 6.5 Operators .....                              | 84 |
| 6.5.1 Arithmetic Operators.....                  | 84 |
| 6.5.2 Relational Operators .....                 | 86 |
| 6.5.3 Logical Operators .....                    | 88 |
| 6.5.4 Operators in Java .....                    | 89 |
| 6.5.5 Operators in Python .....                  | 90 |
| 6.6 Functions .....                              | 91 |
| 6.6.1 Defining a Function.....                   | 93 |
| 6.6.2 Calling a Function.....                    | 93 |

|                            |    |
|----------------------------|----|
| Study Session Summary..... | 94 |
| Assessment.....            | 95 |
| Credits.....               | 95 |

## Study Session 7 96

|  |     |
|--|-----|
| Data Types .....                               | 96  |
| Introduction.....                              | 96  |
| Terminology.....                               | 96  |
| 7.1 Understanding Data Types.....              | 96  |
| 7.1.1 C and Java Data Types.....               | 98  |
| 7.1.2 Python Data Types.....                   | 98  |
| 7.2 Data Type and Numbers Manipulation.....    | 99  |
| 7.2.1 Math Operations on Numbers.....          | 100 |
| 7.2.2 Numbers in Java.....                     | 102 |
| 7.2.3 Numbers in Python.....                   | 103 |
| 7.3 Data Type and character manipulation ..... | 104 |
| 7.3.1 Escape Sequences.....                    | 105 |
| 7.3.2 Characters in Java.....                  | 107 |
| 7.3.3 Characters in Python.....                | 108 |
| 7.4 Data Types and String Manipulation.....    | 109 |
| 7.4.1 Basic String Concepts.....               | 111 |
| 7.4.2 Strings in Java.....                     | 112 |
| 7.4.3 Strings in Python.....                   | 112 |
| Study Session Summary.....                     | 113 |
| Assessment.....                                | 113 |
| Credits.....                                   | 114 |

## Study Session 8 114

|   |     |
|---|-----|
| Decision-Making and Loops .....           | 115 |
| Introduction.....                         | 115 |
| Terminology.....                          | 115 |
| 8.1 Conditional Statements.....           | 115 |
| 8.1.1 The if Statement.....               | 115 |
| 8.1.2 if...else statement .....           | 117 |
| 8.1.3 if...else if...else statement ..... | 118 |
| 8.1.4 The Switch Statement.....           | 119 |
| 8.1.5 Decisions in Java .....             | 121 |
| 8.1.6 Decisions in Python .....           | 122 |
| 8.2 Loops .....                           | 122 |
| 8.2.1 The while Loop .....                | 124 |
| 8.2.2 The do...while Loop.....            | 126 |
| 8.2.3 The break statement .....           | 127 |
| 8.2.4 The continue statement.....         | 129 |
| 8.2.5 Loops in Java.....                  | 130 |
| 8.2.6 Loops in Python.....                | 130 |

|                            |     |
|----------------------------|-----|
| Study Session Summary..... | 131 |
| Assessment.....            | 132 |
| Credit.....                | 132 |

## Study Session 9 133

|                                     |     |
|-------------------------------------|-----|
| Arrays.....                         | 133 |
| Introduction.....                   | 133 |
| Terminology.....                    | 133 |
| 9.1 Understanding Arrays.....       | 133 |
| 9.1.1 Creating Arrays.....          | 134 |
| 9.1.2 Initializing Arrays.....      | 135 |
| 9.1.3 Accessing Array Elements..... | 135 |
| 9.2 Arrays in Java.....             | 137 |
| 9.3 Arrays (Lists) in Python.....   | 138 |
| Study Session Summary.....          | 139 |
| Assessment.....                     | 139 |
| Credit.....                         | 139 |

## Study Session 10 140

|                                |     |
|--------------------------------|-----|
| Computer Files.....            | 140 |
| Introduction.....              | 140 |
| Terminology.....               | 140 |
| 10.1 File Input/output.....    | 140 |
| 10.2 File Operation Modes..... | 141 |
| 10.2.1 Opening Files.....      | 142 |
| 10.2.2 Closing a File.....     | 143 |
| 10.2.3 Writing a File.....     | 143 |
| 10.2.4 Reading a File.....     | 144 |
| 10.3 File I/O in Java.....     | 145 |
| 10.4 File I/O in Python.....   | 146 |
| Study Session Summary.....     | 147 |
| Assessment.....                | 147 |
| Credit.....                    | 147 |

## Notes on Self Assessment Questions 148

## About this course manual

---

Introduction to Programming CSC 1033 has been produced by University of Ibadan Distance Learning Centre. All course manuals produced by University of Ibadan Distance Learning Centre are structured in the same way, as outlined below.

---

### How this course manual is structured

#### The course overview

The course overview gives you a general introduction to the course. Information contained in the course overview will help you determine:

- If the course is suitable for you.
- What you will already need to know.
- What you can expect from the course.
- How much time you will need to invest to complete the course.

The overview also provides guidance on:

- Study skills.
- Where to get help.
- Course assignments and assessments.
- Margin icons.

---

We strongly recommend that you read the overview *carefully* before starting your study.

---

#### The course content

The course is broken down into Study Sessions. Each Study Session comprises:

- An introduction to the Study Session content.
- Study Session outcomes.
- Core content of the Study Session with a variety of learning activities.
- A Study Session summary.
- Assignments and/or assessments, as applicable.
- Bibliography

## Your comments

After completing Introduction to Programming we would appreciate it if you would take a few moments to give us your feedback on any aspect of this course. Your feedback might include comments on:

- Course content and structure.
- Course reading materials and resources.
- Course assignments.
- Course assessments.
- Course duration.
- Course support (assigned tutors, technical help, etc.)

Your constructive feedback will help us to improve and enhance this course.

# Course Overview

---

---

## Welcome to Introduction to Programming CSC103

Welcome to the Introduction to programming course! This course introduces the fundamental concepts, principles and techniques involved in modern computer programming. In addition, the course offers an introduction to the historical context of programming. A systematic approach is used to teach students various methods of algorithm development, program development, and program design in order to write programs that solve well-specified problems. Emphasis is placed on the mastery of basic programming skills, with a considerable attention to the fundamental building blocks of computer programs, and the associated concepts and principles. The course also provides the core knowledge to begin programming in any language. What we are exploring here are the core ideas and skills you will always need when programming on any platform with any language. Now there are dozens of programming languages you could choose from and we will see many of them in this course, languages like JavaScript, C, Ruby and Python. We will see what these different languages are good at and why you might pick one over another, but I won't be trying to make you an expert on any one of them. Instead in this course we will go through what's common across all of these languages. We will work with things like loops, conditionals, variables, and memory; see how to control the structure and the flow of a program; and what you need to know about what that program is doing under the hood.

Finally, the course compares how code is written in several different languages, the libraries and frameworks that have grown around them, and the reasons to choose each one. Topics include Algorithms and problem-solving, Fundamental programming constructs: Fundamental data structures: Primitive types; arrays; records; strings and string processing, Software development methodology: Fundamental design concepts and principles, data types, control structures, functions, arrays, files, and the mechanics of running, testing, and debugging.

---

## Course outcomes

Upon completion of Introduction to Programming CIS104, you will be able to:



- Understand the historical and modern perspectives of computer programming
- Develop comprehensive knowledge about the fundamental principles, characteristics, concepts and constructs of modern

**Outcomes**

- computer programming.
- Develop competencies for the design, coding and debugging of computer programs.
  - Describe the fundamental programming constructs and articulate how they are used to develop a program with a desired runtime execution flow.
  - Define the necessary program structure in terms of the basic building blocks
  - Articulate the advantages and limitations resulting from the use of different language constructs that embody similar programming concepts.





# Getting around this course manual

## Margin icons

While working through this course manual you will notice the frequent use of margin icons. These icons serve to “signpost” a particular piece of text, a new task or change in activity; they have been included to help you to find your way around this course manual.

A complete icon set is shown below. We suggest that you familiarize yourself with the icons and their meaning before starting your study.

|   |   |   |   |
|---|---|---|---|
|    |    |    |    |
| <b>Activity</b>   | <b>Assessment</b>   | <b>Assignment</b>   | <b>Case study</b>   |
|  |  |  |  |
| <b>Discussion</b>   | <b>Group Activity</b>   | <b>Help</b>   | <b>Outcomes</b>   |
|  |  |   |  |
| <b>Note</b>   | <b>Reflection</b>   | <b>Reading</b>  | <b>Study skills</b>   |
|  |  |  |  |
| <b>Summary</b>  | <b>Terminology</b>  | <b>Time</b>   | <b>Tip</b>  |

# Study Session 1

## Overview of Programming

### Introduction

In this study session, you will be discussing programming. You will begin by defining programming. You will also differentiate between application and system programming. Thereafter, you will examine computer programming and how to write a computer program. Lastly, you will describe who a computer programmer is.

### Learning Outcomes



#### Outcomes

When we have studied this session, we should be able to:

- 1.1 *define* programming
- 1.2 *describe* a computer program
- 1.3 *highlight* the characteristics of a computer program
- 1.4 *identify* a computer programmer.

### Terminology

|                   |   |
|-------------------|---|
| <b>Algorithm</b>  | A precise step by step plan for a computational procedure that possibly begins with an input value and yields an output value in a finite number of steps |
| <b>Programmer</b> | One who writes computer programs, a software developer  |
| <b>Database</b>   | A collection of organized information in a regular structure, usually but not necessarily in a machine-readable format                                    |
| <b>Debug</b>      | To search for and eliminate malfunctioning elements or errors in a computer program   |
| <b>Firmware</b>   | Computer hardware with non-volatile embedded computer applications  |

### 1.1 Definition of Programming

Programming is a creative process by which programmers to instruct a computer on how to do a task. This is usually done by converting the steps involved in solving a specific problem into a format that is

understandable by the computer through the use of a programming language, so that it can be executed by a computer. Although many programming languages and many different types of computers exist, the important first step is the need to have the solution in a format that is acceptable and understandable by the computer. Without an algorithm there can be no program. An algorithm is therefore, a process or set of rules to be followed in calculations or other problem-solving operations, especially by a computer.

### ITQ

#### Question

What is the first step and important step in programming?

#### Feedback

The important first step is the need to have the solution in a format that is acceptable and understandable by the computer.

Armed with the understanding of the creative process of turning an algorithm into a computer program and equipped with a specific language of communication, it is possible for a programmer to instruct a computer to do many interesting things, ranging from creating self-driving cars to performing medical diagnosis and treatment. As an example, a system has been running in the UK for several years that reads car number plates. The car is seen by a camera and the image captured then instantly processed so that the number plate details are extracted, run through a national car registration database of number plates and any stolen vehicle etc. alerts for that vehicle flagged up within few seconds.

The set of instructions normally used to perform these tasks is referred to as a computer program, and is normally written in a specific computer programming language. The essence of programming languages is to allow a programmer to manipulate numbers and texts (called variables) in different ways, share them over a network or store them on disks for future retrieval. In order to be able to write computer programs, it has to exist on an operating system such as Windows, Linux or Mac. While most older generation programming languages are built for specific operating system platforms, many modern programming languages such as Java now allow writing programs for multi platform operating systems

### ITQ

#### Question

What makes program work?

#### Feedback

Algorithm

## 1.1.1 Application Versus System Programming

Computer programming often are grouped into two broad categories: application programming and systems programming. Application

programming involves writing programs to handle a specific job, such as a program to track inventory within an organization. It also involves revising existing packaged software or customize generic applications which are frequently purchased from independent software vendors. Systems programming, in contrast, has to do with writing programs to maintain and control computer systems software, such as operating systems and database management systems. These programs are able to determine how the network, workstations, and CPU of the system handle the various jobs they have been given and how they communicate with peripheral equipment such as printers and disk drives.

### ITQ

#### Question

What are the two groups of computer programming?

#### Feedback

As described above, the two groups of computer programming could be Application programming or Systems programming.

## 1.2 The Computer Program

A computer program is a set of instructions, consisting of sequence of separate commands or instructions, one after the other. Each step tells the computer to perform a specific action. The idea of programming is to break the program apart into these individual steps that can be executed one after the other. In programming languages we write these instructions by writing what are called statements. Statements in programming languages are kind of like sentences in English. They use words, numbers, and punctuation to express one thought, one individual piece. Most programming statements are pretty short, just a few words. Now, exactly what words, numbers, and punctuation you use depends on the programming language. Some languages want each of your statements to end with a semicolon, like ending a sentence in English with a period, and others don't. You just go to the next line and start writing the next statement.

Every language has its own characteristics. Some languages are all uppercase, some languages are all lowercase, some languages just don't care. Now, understanding the rules of each language is understanding the syntax of a programming language. So programming is the ability to take this idea in your head, break it apart into its individual pieces, and know how to write those pieces in the programming language you are using at the time, writing your statements in the right order, using the right syntax. But what language? Well, sometimes you get to pick a language and sometimes it's kind of picked for you. This will be made clearer in this courseware.

The computer is only capable of processing binary, i.e. a series of 0s and 1s. Therefore we require a programming language to be able to write commands that the computer can execute, but in a legible manner, i.e.

commands that are understood by humans. These programs are then translated into machine code (in binary) by a compiler.

## ITQ

### Question

Explain the term “statements” with respect to programming?

### Feedback

Statements in programming languages have a similar role to what sentences do in English language. Statements uses words, numbers, and punctuation to express one thought, one individual piece in programming. Your understanding of the how statement relate to programming is key in grasping how a program functions.

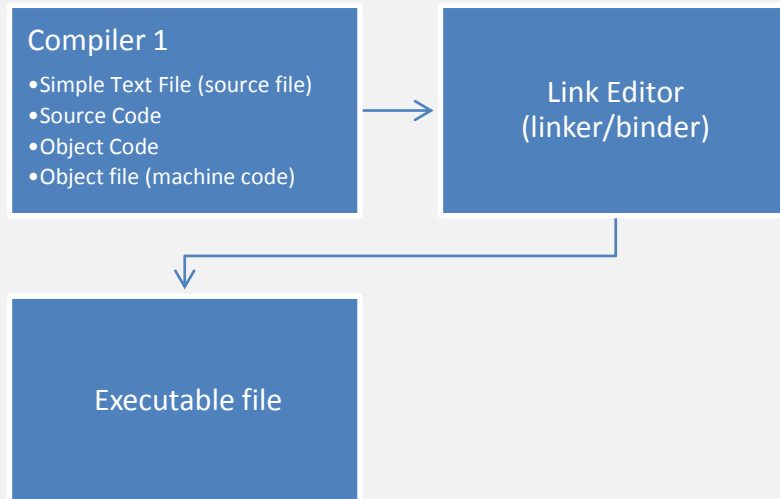
## 1.2.1 Writing a Computer Program

The method of writing a program is closely linked to the programming language chosen, there being many different types. Furthermore, the compiler should match the chosen language: each programming language has its own compiler (except interpreted languages). Generally speaking, the program is a simple text file (written using a word processor or a text editor), this is called the source file). The source file contains lines of program called source code. Once the source file has been completed it must be compiled. Compilation takes place in two stages:

1. The compiler transforms the source code into object code, and saves it in an **object file**, i.e. it translates the source file into machine code (some compilers also create a file in assembler), a language similar to machine code as it possesses basic functions but is legible by humans)
2. The compiler then makes a call to a **link editor** (or **linker** or **binder**) which enables it to embed all additional elements (functions or libraries)that are referenced in the program into the final file but which are not stored in the source file.
3. Then an **executable file** is created which contains all items required for the program to run on its own (in Microsoft Windows or MS-DOS this file will have the extension *.exe*).

**ITQ****Question**

Using a flow chart, outline the basic steps in writing a computer program

**Feedback**

## 1.3 Characteristics of Computer Program

Every computer requires appropriate instruction set (programs) to perform the required task. The quality of the processing depends upon the given instructions. If the instructions are improper or incorrect, then it is obvious that the result will be superfluous. Therefore, proper and correct instructions should be provided to the computer so that it can provide the desired output. Hence, a program should be developed in such a way that it ensures proper functionality of the computer. In addition, a program should be written in such a manner that it is easier to understand the underlying logic. The characteristics of a good computer program are listed by 79experts (<http://www.79xperts.com/blog/good-computer-program/>) as follows:

### 1.3.1 Portability

Portability refers to the ability of an application to run on different platforms (operating systems) with or without minimal changes. Due to rapid development in the hardware and the software, nowadays platform change is a common phenomenon. Hence, if a program is developed for a particular platform, then the life span of the program is severely affected.

### 1.3.2 Readability

The program should be written in such a way that it makes other programmers or users to follow the logic of the program without much effort. If a program is written structurally, it helps the programmers to understand their own program in a better way. Even if some computational efficiency needs to be sacrificed for better readability, it is

advisable to use a more user-friendly approach, unless the processing of an application is of utmost importance.

### **1.3.3 Efficiency**

Every program requires certain processing time and memory to process the instructions and data. As the processing power and memory are the most precious resources of a computer, a program should be laid out in such a manner that it utilizes the least amount of memory and processing time. Programs written in a good programming language are efficiently translated into machine code, are efficiently executed, and acquire as little space in the memory as possible. That is a good programming language is supported with a good language translator which gives due consideration to space and time efficiency.

### **1.3.4 Structural**

To develop a program, the task must be broken down into a number of subtasks. These subtasks are developed independently, and each subtask is able to perform the assigned job without the help of any other subtask. If a program is developed structurally, it becomes more readable, and the testing and documentation process also gets easier. A Structured program implies that the language should have necessary features to allow its users to write their programs based on the concepts of structured programming. This property of a moreover, it forces a programmer to look at a problem in a logical way, so that fewer errors are created while writing a program for the problem.

### **1.3.5 Flexibility**

A program should be flexible enough to handle most of the changes without having to rewrite the entire program. Most of the programs are developed for a certain period and they require modifications from time to time. For example, in case of payroll management, as the time progresses, some employees may leave the company while some others may join. Hence, the payroll application should be flexible enough to incorporate all the changes without having to reconstruct the entire application.

### **1.3.6 Generality**

Apart from flexibility, the program should also be general. Generality means that if a program is developed for a particular task, then it should also be used for all similar tasks of the same domain. For example, if a program is developed for a particular organization, then it should suit all the other similar organizations.

### **1.3.7 Documentation**

Documentation is one of the most important components of an application development. Even if a program is developed following the best programming practices, it will be rendered useless if the end user is not able to fully utilize the functionality of the application. A well-

documented application is also useful for other programmers because even in the absence of the author, they can understand it.

The followings are additional characteristics believed to be important for making a programming language good are:

1. **Simplicity:** A good programming language must be simple and easy to learn and use. It should provide a programmer with a clear, simple and unified set of **concepts**, which can be easily grasped. The overall simplicity of a programming language strongly affects the readability of the programs written in that language, and programs, which are easier to read and understand, are also easier to maintain. It is also easy to develop and implement a compiler or an interpreter for a programming language, which is simple. However, the power needed for the language should not be sacrificed for simplicity.
2. **Naturalness:** A good language should be natural for the application area, for which it has been designed. That is, it should provide appropriate operators, data structures, control structures, and a natural syntax to facilitate the users to code their problem easily and efficiently.
3. **Abstraction:** Abstraction means the ability to define and then use complicated structures or operations in ways that allow many of the details to be ignored. The degree of abstraction allowed by a programming language directly affect its writability. Object oriented language support high degree of abstraction. Hence, writing programs in object oriented language is much easier. Object oriented language also support reusability of program segments due to this features.
4. **Compactness:** In a good programming language, programmers should be able to express intended operations concisely. A verbose language is generally not liked by programmers, because they need to write too much.
5. **Locality:** A good programming language should be such that while writing a programmer concentrate almost solely on the part of the program around the statement currently being worked with.

## ITQ

### Question

What does the term 'flexibility' refer to?

### Feedback

Flexibility refers to the ability of a program to handle most of the changes without having to rewrite the entire program. This is necessary as most programs are developed for a certain period and they require modifications from time to time. Therefore, the payroll application should be flexible enough to incorporate all the changes without having to reconstruct the entire application.



## 1.4 The Computer Programmer

A computer programmer, also known as developer, coder, or software engineer is a person who writes computer software. The term computer programmer can refer to a specialist in one area of computer programming or to a generalist who writes code for many kinds of software and can be used to refer to a software developer, Web developer, mobile applications developer, embedded firmware developer, software engineer, computer scientist, or software analyst. A programmer's primary computer language is often prefixed to these titles, and those who work in a Web environment often prefix their titles with Web. As such, based on computer programming language expertise, we can name a computer programmers as follows –

1. C Programmer
2. C++ Programmer
3. Java Programmer
4. Python Programmer
5. PHP Programmer
6. Perl Programmer
7. Ruby Programmer

### ITQ

#### Question

What other names can we call a computer programmer?

#### Feedback

A computer programmer can also be called a developer, coder, or software engineer.

### 1.4.1 The Work of a Computer Programmer

The purpose of programming is to create a program that produces a certain desired behaviour (customization). The process of writing source code often requires expertise in many different subjects, including knowledge of the application domain, specialized algorithms and formal logic. A computer programmer figures out the process of designing, writing, testing, debugging/troubleshooting and maintaining the source code of computer programs. This source code is written in a programming language so the computer can 'understand' it. The code may be a modification of an existing source or something completely new.

The computer programmer also designs a graphical user interface (GUI) so that non-technical users can use the software through easy, point-and-click menu options. The GUI acts as a translator between the user and the software code. Some, especially those working on large projects that involve many programmers, use computer-assisted software engineering (CASE) tools to automate much of the coding process. These tools enable a programmer to concentrate on writing the unique parts of a program. A programmer working on smaller projects will often use “programmer

environments,” applications that increase productivity by combining compiling, code walk-through, code generation, test data generation, and debugging functions.

A programmer will also use libraries of basic code that can be modified or customized for a specific application. This approach yields more reliable and consistent programs and increases programmers' productivity by eliminating some routine steps. The programmer will also be responsible for maintaining the program's health. As software design has continued to advance, and some programming functions have become automated, programmers have begun to assume some of the responsibilities that were once performed only by software engineers. As a result, some computer programmers now assist software engineers in identifying user needs and designing certain parts of computer programs, as well as other functions. (Adapted from [www.sokanu.com](http://www.sokanu.com))

### ITQ

#### Question

What is the sole purpose of a computer programmer?

#### Feedback

The purpose of programming is to create a program that produces a certain desired behaviour (customization). The process of writing source code often requires expertise in many different subjects, including knowledge of the application domain, specialized algorithms and formal logic.

---

## Study Session Summary



### Summary

In this study session, you explored programming. You began the session by defining programming. You also distinguished between application and system programming. You continued by examining the computer program and how to write a computer program. Thereafter, you highlighted the characteristics of a computer program. You ended the session by describing who a programmer is and his functions.

---

## Assessment



### Assessment

#### SAQ 1.1 (tests Learning Outcome 1.1)

1. What is the essence of programming?
2. What is the essence of programming?

#### SAQ 1.2 (tests learning outcome 1.2)

What is a computer program?

#### SAQ 1.3 (tests learning outcome 1.3)

What are the characteristics of a computer program?

**SAQ 1.4 (tests learning outcome 1.4)**

1. Who is a computer programmer?
2. List four (4) core functions of a computer programmer?

---

## Bibliography



**Reading**

<http://www.cs.bham.ac.uk/~rx/java/intro/2programming.html>

<https://interactivepython.org/runestone/static/pythonds/Introduction/WhatIsProgramming.html>

<http://www.79xperts.com/blog/good-computer-program/>

[https://en.wikipedia.org/wiki/Semantics\\_\(computer\\_science\)](https://en.wikipedia.org/wiki/Semantics_(computer_science))

[https://en.wikipedia.org/wiki/Logic\\_error](https://en.wikipedia.org/wiki/Logic_error)  
[https://en.wikipedia.org/wiki/Logic\\_error](https://en.wikipedia.org/wiki/Logic_error)

[https://en.wikipedia.org/wiki/Integration\\_testing](https://en.wikipedia.org/wiki/Integration_testing)  
[https://en.wikipedia.org/wiki/Integration\\_testing](https://en.wikipedia.org/wiki/Integration_testing)

[https://en.wikipedia.org/wiki/unit\\_testing](https://en.wikipedia.org/wiki/unit_testing)  
[https://en.wikipedia.org/wiki/unit\\_testing](https://en.wikipedia.org/wiki/unit_testing)

[https://en.wikipedia.org/wiki/smoke\\_testing](https://en.wikipedia.org/wiki/smoke_testing)

Cem Kaner, James Bach, Bret Pettichord, Lessons learned in software testing: a context-driven approach. Wiley, 2001

## Study Session 2

# History of Programming Languages

## Introduction

In this study session, you will be tracing the history of programming language. You will describe the different stages of programming language. Under which you will look at the first computer language, the birth of assembly language and the first modern language. Lastly, you will highlight the different categories of computer languages by Epoch.

## Learning Outcomes



### Outcomes

When you have studied this session, you should be able to:

- 2.1 *trace* the evolution of programming languages
- 2.2 *categorize* computer languages by epochs

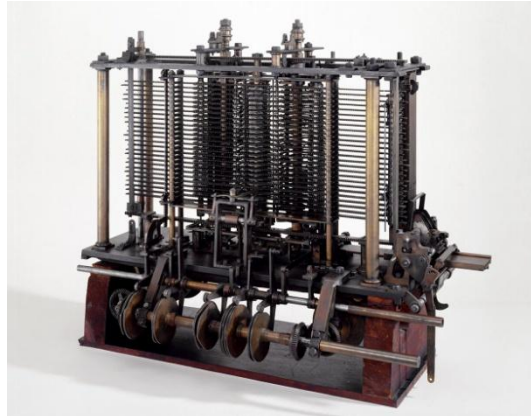
## Terminology

|                  |  |
|------------------|--|
| <b>Formulaic</b> | Closely following a formula or predictable pattern               |
| <b>Compiler</b>  | A computer program which transforms source code into object code |

## 2.1 The Evolution of Programming Languages

Thousands of different programming languages have been created, mainly in the computer field, and many more still are being created every year. The history of programming languages dates back to the 19th century, when Ada Lovelace (1815 – 1852) wrote a set of notes after translating the memoir of Italian mathematician Luigi Menabrea (1809 – 1896) about the Analytical Engine shown below during a period of nine months between 1842 and 1843. . The Analytical Engine was an invention of English mathematician and computer pioneer Charles Babbage (1791-1871). Ada Lovelace’s set of notes, has been recognized by some historians as the world’s first computer program as it contained a ground-breaking description of the possibilities of programming the machine to go beyond number-crunching to “computing”. Ada Lovelace was an English mathematician and writer, chiefly known for her work on Charles Babbage’s Analytical Engine as mentioned above. Her notes on

the engine include what is recognized as the first algorithm intended to be carried out by a machine. Because of this, she is often regarded as the first computer programmer. She also pointed out what otherwise might have been remembered as the first computer bug in Babbage's equations, which also makes her possibly the world's first debugger. The computer language **Ada**, created on behalf of the United States Department of Defense, was named after Ada Lovelace.



Analytical Engine



Ada Lovace- The world's first programmer

## ITQ

### Question

Why was Ada Lovelace's set of notes recognized as the world's first computer program?

### Feedback

Some historians have recognized Ada Lovelace's set of notes, as the world's first computer program because it contained a groundbreaking description of the possibilities of programming the machine to go beyond number crunching to "computing".

## 2.1.1 The First Computer Language

47 years after Ada Lovelace had annotated as what is considered the first computer program, Herman Hollerith (1860 – 1929) created what is considered the first computer language when he realized he could encode information on punch cards. Hollerith's punch card system was used to encode the 1890 census data on punch cards, saving millions of dollars and many years of manual work.

### ITQ

#### Question

Who created the first computer language?

#### Feedback

Herman Hollerith

## 2.1.2 The Birth of Assembly Language

In 1926, Alonzo Church (1903 – 1995) was able to express the lambda calculus (also written as  $\lambda$ -calculus) in a formulaic way. In the 1940s, machine-specific assembly language was probably the first (vaguely) human-readable programming language. An early high-level programming language to be designed for a computer was Plankalkül, developed for the German Z3 by Konrad Zuse between 1943 and 1945. However, it was not implemented until 1998 and 2000. John Mauchly's Short Code, proposed in 1949, was one of the first high-level languages ever developed for an electronic computer. Unlike machine code, Short Code statements represented mathematical expressions in understandable form. However, the program had to be translated into machine code every time it ran, making the process slower than running the equivalent machine code. The Manchester Mark 1 ran programs written in Autocode from 1952.

At the University of Manchester, Alick Glennie developed Autocode in the early 1950s. A programming language, it used as a compiler to automatically convert the language into machine code. The first code and compiler was developed in 1952 for the Mark 1 computer at the University of Manchester and is considered to be the first compiled high-level programming language. By the 1950s computer engineers realized that assembly language was too complex and a fallible process to build entire systems, and so the first modern programming language was born.

## ITQ

### Question

Where was Autocode developed?

### Feedback

Autocode was developed in the early 1950s at the University of Manchester.

## 2.1.3 The First Modern Computer Languages

John Warner Backus (1924 – 2007), an American computer scientist, assembled a team in 1954 to define and develop Fortran for the IBM 704 computer. In 1957, Backus and his team had created Fortran. Fortran was the first high-level programming language to be put to broad use. Lisp was invented by John McCarthy in 1958 while he was at the Massachusetts Institute of Technology (MIT). The name LISP derives from “List Processing”. Linked lists are one of Lisp’s major data structures, and Lisp source code is itself made up of lists. COBOL was later designed in 1959 by the Conference on Data Systems Languages (CODASYL) and was partly based on previous programming language design work by Grace Hopper, commonly referred to as “the (grand) mother of COBOL”. The name COBOL derives from “Common Business Oriented Language”. Cobol is a compiled English-like computer programming language designed for business use.

In 1970, Pascal was developed by Niklaus Wirth. He is a Swiss computer scientist, best known for designing several programming languages, including Pascal, and for pioneering several classic topics in software engineering. The name Pascal is in honor of the French mathematician and philosopher Blaise Pascal. Initially, Pascal was largely, but not exclusively, intended to teach students structured programming. A generation of students used Pascal as an introductory language in undergraduate courses.

The language C was developed in 1972. The name was based on an earlier language called B which is now almost extinct, the C was originally developed by Dennis Ritchie between 1969 and 1973 at AT&T Bell Labs, and used to re-implement the Unix operating system. It was renamed from C with Classes to C++. New features were added including virtual functions, function name and operator overloading, references, constants, type-safe free-store memory allocation (new/delete), improved type checking, and BCPL style single-line comments with two forward slashes “//”, etc.

**ITQ****Question**

Who was responsible for the development of FORTRAN?

**Feedback**

John Warner Backus, an American computer scientist.

## 2.1.4 More Recent Modern Languages

Many later languages have borrowed directly or indirectly from C, including C++, D, Go, Rust, Java, JavaScript, Limbo, LPC, C#, Objective-C, Perl, PHP, Python, Verilog (hardware description language), and Unix's C shell. C++ was created by Bjarne Stroustrup, a Danish computer scientist. The motivation for creating a new language originated from Stroustrup's experience in programming for his Ph.D. In 1983 Objective-C was created in 1983, primarily by Brad Cox and Tom Love in the early 1980s at their company Stepstone. Both had been introduced to Smalltalk while at ITT Corporation's Programming Technology Center in 1981. The earliest work on Objective-C traces back to around that time. Objective-C is a thin layer on top of C, and is a "strict superset" of C, meaning that it is possible to compile any C program with an Objective-C compiler, and to freely include C language code within an Objective-C class.

Perl was originally developed in 1987 by Larry Wall, a computer programmer and author, most widely known as the creator of the Perl programming language. It was developed as a general-purpose Unix scripting language to make report processing easier. Though Perl is not officially an acronym, there are various backronyms in use, the most well-known being "Practical Extraction and Reporting Language". It was named Perl because Pearl was already taken.

In 1991, Python was born. The name of the programming language was chosen by the author while being in a slightly irreverent mood (and because he was a big fan of Monty Python's Flying Circus). Python is a widely used general-purpose, high-level programming language. Its design philosophy emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code than it would be possible in languages such as C++ or Java.

Python was conceived in the late 1980s and its implementation was started in December 1989 by Guido van Rossum at CWI in the Netherlands as a successor to the ABC language capable of exception handling and interfacing with the Amoeba operating system. Van Rossum is Python's principal author, and his continuing central role in deciding the direction of Python is reflected in the title given to him by the Python community, benevolent dictator for life (BDFL). 1993 witnessed the evolution of Ruby. The name "Ruby" originated during an online chat session between Matsumoto and Keiju Ishitsuka. Initially two names were proposed: "Coral" and "Ruby". Matsumoto chose the latter because it was the birthstone of one of his colleagues. Ruby is a dynamic, reflective, object-oriented, general-purpose programming language.



According to its authors, Ruby was influenced by Perl, Smalltalk, Eiffel, Ada, and Lisp. Ruby was designed and developed by Yukihiro “Matz” Matsumoto in Japan. Matsumoto has said that Ruby is designed for programmer productivity and fun, following the principles of good user interface design.

1995 witnessed the evolution of three major modern languages; Java, PHP and JavaScript. Java is a general-purpose computer programming language that was named Java for the amount of coffee consumed while developing the language. It is concurrent, class-based, object-oriented, and specifically designed to have as few implementation dependencies as possible. It is intended to let application developers “write once, run anywhere” (WORA), meaning that compiled Java code can run on all platforms that support Java without the need for recompilation. Java was originally developed by James Gosling, a Canadian computer scientist at Sun Microsystems and released in 1995 as a core component of Sun Microsystems’ Java platform. The language derives much of its syntax from C and C++.

The name PHP derived originally from Personal Home Page, now it stands for Hypertext PreProcessor. PHP is a server-side scripting language designed for web development but also used as a general-purpose programming language. In 1994, Rasmus Lerdorf started the development of PHP when he wrote a series of Common Gateway Interface (CGI) binaries in C, which he used to maintain his personal homepage. He extended them to add the ability to work with web forms and to communicate with databases, and called this implementation “Personal Home Page/Forms Interpreter” or PHP/FI.

JavaScript It was developed under the name Mocha, and then the language was officially called LiveScript when it first shipped in beta releases of Netscape Navigator 2.0 in September 1995, but it was renamed JavaScript when it was deployed in the Netscape browser version 2.0B3. JavaScript is a high level, dynamic, untyped, and interpreted programming language. It has been standardized in the ECMAScript language specification. Alongside HTML and CSS, it is one of the three essential technologies of World Wide Web content production; the majority of websites employ it and it is supported by all modern web browsers without plug-ins.

Despite some naming, syntactic, and standard library similarities, JavaScript and Java are otherwise unrelated and have very different semantics. The syntax of JavaScript is actually derived from C, while the semantics and design are influenced by the Self and Scheme programming languages. JavaScript was originally developed by Brendan Eich, while he was working for Netscape Communications Corporation. Indeed, while competing with Microsoft for user adoption of web technologies and platforms, Netscape considered their client-server offering a distributed OS with a portable version of Sun Microsystems’ Java providing an environment in which applets could be run.

In 2000, The name “C sharp” was inspired by musical notation where a sharp indicates that the written note should be made a semitone higher in pitch. C# is intended to be a simple, modern, general-purpose, object-oriented programming language. Its development team is led by Anders

Hejlsberg. The most recent version is C# 6.0, which was released on July 20, 2015.

In January 1999, Anders Hejlsberg formed a team to build a new language at the time called Cool, which stood for “C-like Object Oriented Language”. Microsoft had considered keeping the name “Cool” as the final name of the language, but chose not to do so for trademark reasons. By the time the .NET project was publicly announced at the July 2000 Professional Developers Conference, the language had been renamed C#.

## ITQ

### Question

Who created C++?

### Feedback

Bjarne Stroustrup created C++.

Born in 2003, Scala is a programming language for general software applications. The name Scala is a portmanteau of “scalable” and “language”, signifying that it is designed to grow with the demands of its users. Scala has full support for functional programming and a very strong static type system. This allows programs written in Scala to be very concise and thus smaller in size than other general-purpose programming languages. The design of Scala started in 2001 at the École Polytechnique Fédérale de Lausanne (EPFL) by Martin Odersky, following on from work on Funnel, a programming language combining ideas from functional programming and Petri nets. Odersky had previously worked on Generic Java and javac, Sun’s Java compiler.

In 2009, the Go language was born. It is also commonly referred to as `golang`, is a programming language developed at Google. Go is recognizably in the tradition of C, but makes many changes to improve conciseness, simplicity, and safety. Go was developed at Google in 2007 by Robert Griesemer, Rob Pike, and Ken Thompson. It is a statically typed language with syntax loosely derived from that of C, adding garbage collection, type safety, some dynamic-typing capabilities, additional built-in types such as variable-length arrays & key-value maps, and a large standard library. Swift was introduced at Apple’s 2014 Worldwide Developers Conference (WWDC), It underwent an upgrade to version 1.2 during 2014, and a more major upgrade to Swift 2 at WWDC 2015. Swift is a multi-paradigm, compiled programming language created by Apple Inc. for iOS, OS X, and watch OS development. Swift is designed to work with Apple’s Cocoa and Cocoa Touch frameworks and the large body of existing Objective-C code written for Apple products. Swift is intended to be more resilient to erroneous code (“safer”) than Objective-C, and more concise. Development on Swift began in 2010 by Chris Lattner, with the eventual collaboration of many other programmers at Apple. Swift took language ideas “from Objective-C, Rust, Haskell, Ruby, Python, C#, CLU, and far too many others to list”. On June 2, 2014, the Worldwide Developers Conference (WWDC) application became the first publicly released app written in Swift.

**ITQ**

**Question**

When was Python created?

**Feedback**

1991

## 2.2 Categorizing Computer Languages by Epochs

Simon Raik-Allen divided the history of computer languages into three primary epochs as follows:

1. **The High-Level Dawning:** In the 1940s computer programs were cryptic and lengthy streams of low-level machine instructions. That changed in 1955 and through the 60s with the advent of Fortran, good for scientific calculations, and then Cobol for commercial data processing. These languages gave early programmers their first glimpse of the expressive power of abstracting away underlying details and using higher-level constructs to describe their needs.
2. **The C Era:** Around 1972 the C language was born and has probably had more impact on programming and language design than any other language to date. Many subsequent languages derived their syntax (appearance) from C and there are also a number of C dialects (such as C++ in 1980) which collectively still dominate the list of most popular programming languages today. With C one can create high-performing and low-memory-using applications so it's often used for writing other languages, operating systems, and small device controllers.
3. **The Age of Java:** In 1995 the Java language hit the scene, rapidly rose to popularity and is widely considered the most popular programming language in existence today. I put its success down to three reasons. Firstly, it was based on the C/C++ syntax so it was familiar looking. Secondly, it omitted a number of its predecessor's more complex language features greatly simplifying the language (although at the price of performance) but thus opening it up to a wider audience of programmers. And lastly due to the web. Netscape Navigator, the first web browser, entered the market around the same time and allowed embedding of Java programs into web pages to extend their functionality and the two rode the tech boom together. A few years later Java made its way into the enterprise market (some say basically replacing COBOL) and the rest is history.

## ITQ

### Question

Which primary epoch had more impact on programming and language design than any other language?

### Feedback

The C Era.

According to Raik-Allen, the historical perspective above is one way to look at it, but there are a few others. Which he sliced across some other dimensions:

1. The Web Dimension: For rapidly building web sites Perl (1987) and PHP (1995) have been two popular choices for a number of years. Among other things they make it easy to merge business data and page layout for delivery to web browsers. There are also many Web Application Frameworks, which are pre-built components, which provide many of the features required to build and manage web sites. Different versions of these have been built in most of the popular programming languages.
2. The Corporate Dimension : The big technology companies have each largely aligned themselves with different languages stacks. Oracle and IBM are aligned with Java (Oracle actually owns Java). Google are known for their use of Python (1997), a very versatile, dynamic and extensible language, although in reality they are also heavy users of C++ and Java. They have also created their own language called Go (2009). Developers who use Microsoft technology tend to use their systems, tools, and languages exclusively, and for over a decade now Microsoft have focused their efforts on their own language C# (pronounced 'see sharp'). Although its namesake is C it began life (debatably) closer to Java and was reportedly developed as a response to Java's growing popularity in 2000. That said, C# has since evolved rapidly into a rich and eloquent language and if there was any similarity to Java, it's gone. C# has been enjoying solid growth in popularity since day one. Another language worth mentioning is BASIC. Originally from 1964 BASIC took off when it found its way onto almost every Personal Computer from the 70s and 80s and two variants are still extremely popular today. Both are made by Microsoft.

## ITQ

### Question

Which language is regarded as the most popular programming language in existence today?

### Feedback

The Java Language

3. The Mobile Dimension: This is the language of Apple's mobile phones and tablets. Although officially published in 1986 it

remained almost dormant until the rise of the App Store in 2009. Since then it has been one of the fastest growing languages each year. On the other hand, Google's Android mobile platform uses Java to write applications which is surely contributing to that language's current popularity in a significant way.

**ITQ**

**Question**

What programming language has Oracle and IBM aligned themselves?

**Feedback**

Oracle and IBM are aligned with Java (Oracle actually owns Java).

4. The New Age Dimension: There is of late an emerging backlash to the highly verbose and structured C-derived languages. Many developers are wanting to do more with less and hence we are seeing a rise in a number of more succinct, streamlined, and potentially more productive languages. Three worth mentioning are Clojure (2007), Scala (2003), and Ruby (1993). Clojure is a modern interpretation of an older language called LISP (from the Fortran days). Scala was designed to be a 'better Java' and has added many new features to the language over the last few years, making it also more C#-like. Ruby is definitely the most popular of the bunch and has been sitting just inside the top 10 for a few years.

**ITQ**

**Question**

What is Google known with and what other languages do they make use of?

**Feedback**

Google is known for their use of Python (1997), a very versatile, dynamic and extensible language.

They are also heavy users of C++ and Java.

### 2.2.1 Summary of Important Programming Languages by Year

Below is a Summary of important programming languages by year:

|                                   |               |  |
|-----------------------------------|---------------|--|
| 1951 – Regional Assembly Language | 1970 – Pascal | 1993 – Ruby                            |
| 1952 – Autocode                   | 1972 – C      | 1994 – CLOS (part of ANSI Common Lisp) |
| 1954 – IPL (forerunner to LISP)   | 1972 – Prolog | 1995 – Ada 95                          |

|                                     |   |                               |
|-------------------------------------|---|-------------------------------|
| 1955 – FLOW-MATIC (led to COBOL)    | 1972 – Smalltalk                                | 1995 – Delphi (Object Pascal) |
| 1957 – COMTRAN (precursor to COBOL) | 1973 – ML                                       | 1995 – Java                   |
| 1957 – FORTRAN (First compiler)     | 1975 – Scheme                                   | 1995 – JavaScript             |
| 1958 – ALGOL 58                     | 1978 – SQL (a query language, later extended)   | 1995 – PHP                    |
| 1958 – LISP                         | 1980 – C++ (as C with classes, renamed in 1983) | 1996 – WebDNA                 |
| 1959 – COBOL                        | 1983 – Ada                                      | 1997 – Rebol                  |
| 1959 – FACT (forerunner to COBOL)   | 1984 – Common Lisp                              | 1999 – D                      |
| 1959 – RPG                          | 1984 – MATLAB                                   | 2000 – ActionScript           |
| 1962 – APL                          | 1985 – Eiffel                                   | 2000 – C#                     |
| 1962 – Simula                       | 1986 – Erlang                                   | 2001 – Visual Basic .NET      |
| 1962 – SNOBOL                       | 1986 – Objective-C                              | 2003 – Groovy                 |
| 1963 – CPL (forerunner to C)        | 1987 – Perl                                     | 2003 – Scala                  |
| 1964 – BASIC                        | 1988 – Mathematica                              | 2005 – F#                     |
| 1964 – PL/I                         | 1988 – Tcl                                      | 2007 – Clojure                |
| 1966 – JOSS                         | 1989 – FL (Backus)                              | 2009 – Go                     |
| 1967 – BCPL (forerunner to C)       | 1990 – Haskell                                  | 2011 – Dart                   |
| 1968 – Logo                         | 1991 – Python                                   | 2012 – Rust                   |
| 1969 – B (forerunner to C)          | 1991 – Visual Basic                             | 2014 – Swift                  |
| 1970 – Forth                        | 1993 – Lua                                      |                               |

---

## Study Session Summary



### Summary

In this study session, you traced the history of programming languages. You discussed the different developmental stages of program development. You examined the first computer language, the birth of assembly language and the first modern computer language. You also highlighted the different categories of computer languages by Epoch.

---

## Assessment



### Assessment

#### SAQ 2.1 (tests Learning Outcome 2.1)

What are the key points in the evolution of programming languages?

#### SAQ 2.2 (tests Learning Outcome 2.2)

1. How do you categorize computer languages by epochs?
2. Give a summary of the important languages by year.

---

## Bibliography



### Reading

[https://en.wikipedia.org/wiki/Programming\\_language](https://en.wikipedia.org/wiki/Programming_language)[https://en.wikipedia.org/wiki/Programming\\_language](https://en.wikipedia.org/wiki/Programming_language)

<http://www.ricardosanchez.com/2015/08/31/programming-a-short-history-of-computer-languages/>

## Study Session 3

# The Programming Process

## Introduction

In this study session, you will examine the programming process. You will begin by discussing problem definition and analysis. Thereafter, you will point out planning and outlining the solution. In addition, you will examine how to code the program, compilation and debugging. You will conclude the session by exploring testing, validation, and program documentation.

## Learning Outcomes



### Outcomes

When you have studied this session, you should be able to:

- 3.1 *explain* problem definition and analysis
- 3.2 *discuss* the process of coding the program
- 3.3 *define* compilation
- 3.4 *define* debugging
- 3.5 *explain* testing and validation
- 3.6 *describe* programming documentation

## Terminology

|                       |   |
|-----------------------|---|
| <b>Flowchart</b>      | A schematic representation of how the different stages in a process are interconnected  |
| <b>Pseudo code</b>    | A description of a computer programming algorithm that uses the structural conventions of programming languages   |
| <b>Word Processor</b> | A software that provides word processing functions on a computer, typically including typeface selection, line justification and other formatting, pagination and numerous other features |
| <b>Syntax</b>         | A set of rules that govern how words are combined to form phrases and sentences   |
| <b>Error</b>          | A failure to complete a task, usually involving a premature termination   |



## 3.1 Problem Definition and Analysis

The first thing in the programming life cycle is to identify and analyze the specific problem to be solved. If you have no problem, you cannot concentrate your capability on programming work. If you have a problem, you engage yourself in programming. The first thing is an analytical brain that can help a programmer to quickly understand problems and analyze problem for the next step.

Suppose that, as a programmer, you have been contacted because your services are needed. You meet with users from the client organization to analyze the problem, or you meet with a systems analyst who outlines the project. Specifically, the task of defining the problem consists of identifying what it is you know (input-given data), and what it is you want to obtain (output-the result). Eventually, you produce a written agreement that, among other things, specifies the kind of input, processing, and output required. This is not a simple process.

### ITQ

#### Question

How do you identify a problem in programming?

#### Feedback

To identify a problem, there must be a need. The need from the end-user/clients determines what the problem is and how to solve it

### 3.1.1 Planning and Outlining the Solution

If you have understood your problem, outline of solution is next important steps. The problem is divided into several modules or tasks and assigned to each programmer. The software industries are based on efficient system analyst, and competent programmers. Two common ways of planning the solution to a problem are to draw a flowchart and to write pseudocode, or possibly both. Essentially, a flowchart is a pictorial representation of a step-by-step solution to a problem. It consists of arrows representing the direction the program takes and boxes and other symbols representing actions. It is a map of what your program is going to do and how it is going to do it. Pseudocode is an English-like nonstandard language that lets you state your solution with more precision than you can in plain English but with less precision than is required when using a formal programming language. Pseudocode permits you to focus on the program logic without having to be concerned just yet about the precise syntax of a particular programming language. However, pseudocode is not executable on the computer until it is developed into an executable program.

**ITQ****Question**

What is the uniqueness of the Pseudocode?

**Feedback**

Pseudocode allows you state your solution with precision and permits you to focus on the program logic

## 3.2 Coding the Program

As the programmer, your next step is to code the program—that is, to express your solution in a programming language. You will translate the logic from the flowchart or pseudocode—or some other tool—to a programming language. Coding is the act of translating the design into an actual program, written in some form of programming language. This is the step where you actually have to sit down at the computer and type! Coding is a little bit like writing an essay. In most cases you write your program using something a bit like a word processor. And, like essays, there are certain things that you always need to include in your program (a bit like titles, contents pages, introductions, references etc.). When you've finished translating your design into a program (usually filling in lots of details in the process) you need to submit it to the computer to see what it makes of it.

As we have already noted, a programming language is a set of rules that provides a way of instructing the computer what operations to perform. There are many programming languages: BASIC, COBOL, Pascal, FORTRAN, and C are some examples. Although programming languages operate grammatically, somewhat like the English language, they are much more precise. To get your program to work, you have to follow exactly the rules—the syntax—of the language you are using. Of course, using the language correctly is no guarantee that your program will work, any more than speaking grammatically correct English means you know what you are talking about. The point is that correct use of the language is the required first step. Then your coded program must be keyed, probably using a terminal or personal computer, in a form the computer can understand. Programmers usually use a text editor, which is somewhat like a word processing program, to create a file that contains the program. However, as a beginner, you will probably want to write your program code on paper first. Generally, programmers choose general purpose programming languages. The specific type's languages are used for specific types of software. Before doing the coding, the programmer has to choose a computer language based on the following considerations:

1. The nature of the problem.
2. The programming language available on the computer.
3. The facilities and limitations of the computer installation.

**ITQ****Question**

What is the idea behind coding a program?

**Feedback**

It allows you not just to state your solution in plain English but to express your solution in a programming language.

## 3.3 Compilation

Every programming language has to take a set of programming specifications and translate them, i.e. create a means to execute those specifications, by using a compiler. Technically, the term, compiler generally means a program that produces a separate executable from the compiler (that may require a run time library or subsystem to operate), a compiler that merely executes the original specifications is usually referred to as an "interpreter", although because of differing methods of analyzing what represents compilation and what represents interpretation, there is some overlap between the two terms. Compilation turns the program into the instructions made up of 0's and 1's that the computer can actually follow. This is necessary because the chip that makes your computer work only understands binary machine code - something that most humans would have a great deal of trouble using since it looks something like:

```
01010101 11111101 10111100 10000001 10010001
```

Early programmers actually used to write their programs in that sort of a style - but luckily they soon learnt how to create programs that could take something written in a more understandable language and translate it into this gobbled gook. These programs are called compilers, and you can think of them simply as translators that can read a programming language, translate it and write out the corresponding machine code.

Compilers are notoriously pedantic though - if you don't write very correct programs, they will complain. Think of them as the strictest sort of English teacher, who picks you up on every single missing comma, misplaced apostrophe and grammatical error. This is where debugging makes its first appearance, since once the compiler has looked at your program it is likely to come back to you with a list of mistakes, the next thing is to correct these mistakes through a process known as debugging,, otherwise the program will not execute.

**ITQ****Question**

What can a compiler be compared with?

**Feedback**

I don't know what you have compared a compiler with, but a compiler can be compared with a translator. This is because they can read a program language, translate it, and write out the corresponding machine code.

## 3.4 Debugging

Debugging is the process of detecting, locating and fixing or bypassing errors, typically known as bugs in the process of compiling a computer program. When we compile program, we debug the program by looking at the error messages that are displayed due to syntax error, and removing the bug or error so that no erroneous messages are displayed. It is a common daily life event in software industries. When a program is coded carefully, bugs can still creep stealthily into the program. Some compilers are equipped with debugger to remove programming errors. These compilers diagnose errors and debug them carefully.

It should be noted that it is not actually necessary to write the entire program before you start to compile and debug. In most cases it is better to write a small section of the code first, get that to work, and then move on to the next stage. This reduces the amount of code that needs to be debugged each time and generally creates a good feeling of "getting there" as each section is completed. Finally, the compiler presents you with a program that the computer can run, hopefully, your solution!

To clearly understand the concept of debugging in computer programs, it is important to first understand the concepts of Syntax, Semantics, and program errors.

**ITQ****Question**

How do you improve outcome and save time while programming?

**Feedback**

You could save time and improve the speed of programming by debugging in bits. Do not wait until you finish writing your program before you start debugging. That could be overwhelming! To avoid that, once you have written up to a particular level, you should commence the process of debugging.

### 3.4.1 Syntax

The syntax of a computer language is the set of rules that defines the combinations of symbols that are considered to be a correctly structured document or fragment in that language. Simply put, syntax is the basic

spelling and grammar of a particular programming language in order to indicate the program's validity. Generally, syntax differs from language to language. A program follows valid syntax rules in order to be able to function, otherwise the program cannot execute successfully.

## ITQ

### Question

What is syntax?

### Feedback

Syntax is the basic spelling and grammar of a particular programming language in order to indicate the program's validity.

## 3.4.2 Semantics

Semantics of a programming language refer to the rules that give meaning to programs. We can say syntax has specific output oriented meaning, which is called semantics. It refers to the meaning of the language elements in terms of what they formally mean as regards computation (operational semantics). This means that it expresses what a term of your language effectively does assuming an underlying kind of model that depends on which semantic we are talking about.

Wikipedia describes 3 kinds of semantics as follows:

1. operational semantics express the meaning of the language by specifying how an abstract virtual machine behaves whenever it executes a term. (eg: +: pops two elements from the stack and push the sum. This is NOT formal and it is NOT how you should really consider it, it's just to give you an idea). This is the most used one to describe semantics of "normal" programming languages. For example for Java you could have, for every possible term, a sequence of JVM instruction meant to be executed to model that term. Probably when you asked for the meaning of semantics this is the one you were looking for.
2. Denotational semantics is a different approach: you give for every term of the language a meaning that is represented by a mathematical function. So for previous example you would have a function  $f$  associated with + that contains what is the semantic (effective meaning) of the term
3. axiomatic semantics is a way to annotate the terms of your language expressing how they alter the validity of some logical formulas you want to verify over your program. You should consider reading this just because the inference rules and axioms used are similar in how you develop this kind of semantics but it's explained in a practical way

From this description above, you understand that a semantic is something well defined inside a context, and you need a specified context otherwise you couldn't give you language a formal definition of what its terms do..

**Example 3.1**

The semantics of the following equation tells that sum of x and y is assigned to variable sum:

$$\text{Sum} = x + y$$

**Example 3.2**

Consider another equation below:

$$p = q * r + 1 / 2 * s * r * r$$

It is a syntax which has two set of variables. In right set has 'q', 'r' and 's' variables and left set has only one variable 'p'. The semantics tells that right side has operators +, / and \* working over variables. In first step, variables are multiplied (one set 'q' and 'r' and other set s, r and r), then 1 is divided by  $2 * s * r * r$  and added to  $q * r$ , and sum is assigned to p. Syntax is the rule used by programming grammar. So programming statement may be correct on the basis of syntax, but it may be semantically wrong.

**ITQ****Question**

What is the most important rule in programming with respect to Semantics?

**Feedback**

Syntax is the rule used by programming grammar. So programming statement may be correct because of syntax, but it may be semantically wrong. Therefore, it is essential that you understand the rules of semantics, how it applies to programming, and syntax.

**3.4.3 Programming Errors**

There are three types of errors in a program:

1. Syntax Error
2. Run time Error or Execution Error, and
3. Logical Errors

Syntax errors are the result of avoiding rules of program coding. These errors occur due to lack of concentration on language syntax. Some QBASIC experts learn C or C++ , avoid semicolon at the end of statements or use crater (^) sign for exponential. These types of syntax errors occurred due to transference of skill of one language into others and syntax errors are occurred.

The followings are some General causes of Syntax errors:

1. Missing parenthesis or extra parenthesis causes syntax errors.
2. In some language (Pascal, C, C++, C#, JAVA), statements' terminators are used. Missing terminators or extra terminators cause syntax errors.

3. Misspelled keywords.
4. Extra blanks or missing blanks also cause syntax errors.
5. All type of operators is very sensitive to produce syntax errors, when not used properly.
6. The use of colon (:) instead of semi-colon (;) causes syntax errors.

The syntax errors are displayed at the time of program compilation because compilers and interpreters are equipped with error-diagnostic features. These errors are easily debugged from programs and some compilers check erroneous input and recover errors also. The 50 to 75% overall programming times are to be spent on debugging. 8023588240 8167518347 7034993743

Run-time Errors are the errors appeared at the time of program execution. Compilers do not detect these types of errors because these types of errors are not caused by syntax errors. Rather, they are noticed at the time of execution.

A logic error is a bug in a program that causes it to operate incorrectly and the compiler or interpreter is unable to detect these type of errors. A logic error produces an incorrect or unintended or undesired output or other behaviour, although it may not immediately be recognised as such. As an example, telling a computer to repeat an operation but not telling it how to stop repeating may result in a situation where the program runs in an endless cycle. Logic errors occur in both compiled and interpreted languages. The only clue to the existence of logic errors is the production of wrong solutions. One of the ways to find these type of errors is to output the program's variables to a file or on the screen in order to define the error's location in code. Although this will not work in all cases, for example when calling the wrong subroutine, it is the easiest way to find the problem if the program uses the incorrect results of a bad mathematical calculation.

### Example 3.3

This example function in C to calculate the average of two numbers contains a logic error. It is missing brackets in the calculation, so it compiles and runs but does not give the right answer due to operator precedence (division is evaluated before addition).

```
int average(int a, int b)
{
    return a + b / 2;    /* should be (a + b) / 2 */
}
```

### Example 3.4

The following statements are intended to be used to find the roots of a quadratic equation. They are syntactically acceptable and would not cause any error message but produce wrong outputs.

```
X1 = -b+sqrt(b^2-4*a*c/2*a);
X2 = -b-sqrt(b^2-4*a*c)/2*a;
```

**ITQ****Question**

Describe briefly the types of error in a program?

**Feedback**

The errors in a program are described as follows:

- Syntax errors, which occur because rules guiding program coding have not been followed.
- Run-time Errors often appear at the time of program execution. Compilers do not detect these types of errors because these types of errors are not caused by syntax errors.
- A logic error is a bug in a program that causes it to operate incorrectly and the compiler or interpreter is unable to detect these type of errors.

## 3.5 Testing and Validation

It is important to test your program to check that it does what you want it to do. This step is necessary because though the compiler has checked that your program is correctly written, it can't check whether what you've written actually solves your original problem. This is because it is quite possible to write a sentence in any language that is perfectly formed with regards to the language that it's written in (syntactically correct) but at the same time be utter nonsense (semantically incorrect). For example, 'Fish trousers go sideways.' is a great sentence - it's got a capital letter and a full stop - but it doesn't mean a lot. Similarly, 'Put the ice cube tray in the oven.' has verbs and nouns and so on - but it's pretty useless if you wanted to make ice cubes. So your program needs to be tested, and this is often initially done informally (or perhaps, haphazardly) by running it and playing with it for a bit to see if it seems to be working correctly. After this has been done, it should also be checked more thoroughly by subjecting it to carefully worked out set of tests that put it through its paces, and check that it meets the requirements and specification. Where mistakes are identified, there is need to figure out where in the code the mistake is. Once identified, the problem should be fixed by changing the code and recompiling. Care should be taken at this point that this fix doesn't break something else, so careful retesting is important. This process is also known as debugging.

Once all the testing and debugging has been completed, you should be pretty certain that your program works according to your requirements and your specification and so you should finally have a solution to your problem! Software testing involves the execution of a software component or system component to evaluate one or more properties of interest. In general, these properties indicate the extent to which the component or system under test:

1. meets the requirements that guided its design and development,
2. responds correctly to all kinds of inputs,
3. performs its functions within an acceptable time,



4. is sufficiently usable,
5. can be installed and run in its intended environments, and
6. achieves the general result its stakeholders desire.

## ITQ

### Question

Mention two properties needed before a software can be tested.

### Feedback

1. The software must meet the requirements that guided its design and development, and
2. The software should respond correctly to all kinds of inputs.

## 3.5.1 Types of Programming Tests

Many methods exist which are used for program testing:

1. **Unit Testing:** Unit testing is a software testing method by which individual units of source code, sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures, are tested to determine whether they are fit for use. Intuitively, one can view a unit as the smallest testable part of an application. The goal of unit testing is to isolate each part of the program and show that the individual parts are correct. A unit test provides a strict, written contract that the piece of code must satisfy.
2. **Integration testing:** The purpose of integration testing is to verify functional, performance, and reliability requirements placed on major design items. In integration testing, software modules are combined and tested as a group. It occurs after unit testing and before validation testing. Integration testing takes as its input modules that have been unit tested, groups them in larger aggregates, applies tests defined in an integration test plan to those aggregates, and delivers as its output the integrated system ready for system testing.
3. **Regression Testing:** Regression testing is a type of software testing that verifies that software previously developed and tested still performs correctly after it was changed or interfaced with other software. Changes may include software enhancements, patches, configuration changes, etc. During regression testing, new software bugs or regressions may be uncovered. Sometimes a Software Change Impact Analysis is performed to determine what areas could be affected by the proposed changes. These areas may include functional and non-functional areas of the system. The purpose of regression testing is to ensure that changes such as those mentioned above have not introduced new faults. One of the main reasons for regression testing is to determine whether a change in one part of the software affects other parts of the software.
4. **Smoke Testing:** smoke testing (also confidence testing, sanity testing) is preliminary testing to reveal simple failures severe

enough to (for example) reject a prospective software release or determine whether it is reasonable to proceed with further testing. Smoke testing consists of minimal attempts to operate the software, designed to determine whether there are any basic problems that will prevent it from working at all. Such tests can be used as build verification test. A smoke tester will select and run a subset of test cases that cover the most important functionality of a component or system, to ascertain if crucial functions of the software work correctly. When used to determine if a computer program should be subjected to further, more fine-grained testing, a smoke test may be called an intake test. For example, a smoke test may address basic questions like "Does the program run?", "Does it open a window?", or "Does clicking the main button do anything?" The process of smoke testing aims to determine whether the application is so badly broken as to make further immediate testing unnecessary. As the book *Lessons Learned in Software Testing* puts it, "smoke tests broadly cover product features in a limited time ... if key features don't work or if key bugs haven't yet been fixed, your team won't waste further time installing or testing".

5. **Recovery Testing:** recovery testing is the activity of testing how well an application is able to recover from crashes, hardware failures and other similar problems. Recovery testing is the forced failure of the software in a variety of ways to verify that recovery is properly performed. Recovery testing should not be confused with reliability testing, which tries to discover the specific point at which failure occurs. Recovery testing is basically done in order to check how fast and better the application can recover against any type of crash or hardware failure etc. Type or extent of recovery is specified in the requirement specifications. It is basically testing how well a system recovers from crashes, hardware failures, or other catastrophic problems: Examples of recovery testing:
  - i. While an application is running, suddenly restart the computer, and afterwards check the validness of the application's data integrity.
  - ii. While an application is receiving data from a network, unplug the connecting cable. After some time, plug the cable back in and analyze the application's ability to continue receiving data from the point at which the network connection disappeared.
  - iii. Restart the system while a browser has a definite number of sessions. Afterwards, check that the browser is able to recover all of them.
6. **Security Testing:** Security testing is a process intended to reveal flaws in the security mechanisms of an information system that protect data and maintain functionality as intended. Due to the logical limitations of security testing, passing security testing is not an indication that no flaws exist or that the system adequately satisfies the security requirements.
7. **Stress testing:** Stress testing, in general, should put computer hardware under exaggerated levels of stress in order to ensure

stability when used in a normal environment. These can include extremes of workload, type of task, memory use, thermal load (heat), clock speed, or voltages. Memory and CPU are two components that are commonly stress tested in this way. There is considerable overlap between stress testing software and benchmarking software, since both seek to assess and measure maximum performance. Of the two, stress testing software aims to test stability by trying to force a system to fail; benchmarking aims to measure and assess the maximum performance possible at a given task or function.

8. Installation testing: An installation test assures that the system is installed correctly and working at actual customer's hardware.
9. Compatibility testing: This ensures that a software is compatible with other application software, operating systems (or operating system versions, old or new), or target environments that differ greatly from the original (such as a terminal or GUI application intended to be run on the desktop now being required to become a web application, which must render in a web browser).
10. Regression testing: Regression testing focuses on finding defects after a major code change has occurred. Specifically, it seeks to uncover software regressions, as degraded or lost features, including old bugs that have come back.
11. Acceptance testing: Acceptance testing can be divided into 2 categories; Smoke testing and acceptance testing. A smoke test is used as an acceptance test prior to introducing a new build to the main testing process, i.e., before integration or regression. Acceptance testing performed by the customer, often in their lab environment on their own hardware, is known as user acceptance testing (UAT).

## ITQ

### Question

What is the central theme of program testing?

### Feedback

Program testing is an important tool used by either program developers or end users to check whether a program would function properly amidst several constraints that should make the program not function properly. The weak and faulty areas are identified, corrected, and improved for an enjoyable experience by the end users.

12. Alpha testing: Alpha testing is simulated or actual operational testing by potential users/customers or an independent test team at the developer's site.
13. Beta testing: Beta testing comes after alpha testing and can be considered a form of external user acceptance testing. Versions of the software, known as beta versions, are released to a limited audience outside of the programming team known as beta testers. The software is released to groups of people so that further testing can ensure the product has few faults or bugs.

14. **Functional vs non-functional testing:** Functional testing refers to activities that verify a specific action or function of the code. These are usually found in the code requirements documentation, although some development methodologies work from user cases or user stories. Non-functional testing refers to aspects of the software that may not be related to a specific function or user action, such as scalability or other performance, behavior under certain constraints, or security. Testing will determine the breaking point, the point at which extremes of scalability or performance leads to unstable execution.
15. **Continuous testing:** Continuous testing is the process of executing automated tests as part of the software delivery pipeline to obtain immediate feedback on the business risks associated with a software release candidate
16. **Destructive testing:** Destructive testing attempts to cause the software or a sub-system to fail. It verifies that the software functions properly even when it receives invalid or unexpected inputs, thereby establishing the robustness of input validation and error-management routines.
17. **Software performance testing:** Performance testing is generally executed to determine how a system or sub-system performs in terms of responsiveness and stability under a particular workload. It can also serve to investigate, measure, validate or verify other quality attributes of the system, such as scalability, reliability and resource usage.
18. **Usability testing:** Usability testing is to check if the user interface is easy to use and understand. It is concerned mainly with the use of the application.
19. **Security testing:** Security testing is essential for software that processes confidential data to prevent system intrusion by hackers.
20. **Development testing:** Development Testing involves synchronized application of a broad spectrum of defect prevention and detection strategies in order to reduce software development risks, time, and costs.
21. **A/B testing:** A/B testing is basically a comparison of two outputs, generally when only one variable has changed: run a test, change one thing, run the test again, compare the results. This is more useful with more small-scale situations, but very useful in fine-tuning any program.
22. **Concurrent testing:** In concurrent testing, the focus is on the performance while continuously running with normal input and under normal operational conditions, as opposed to stress testing, or fuzz testing.
23. **Conformance testing:** In software testing, conformance testing verifies that a product performs according to its specified standards. Compilers, for instance, are extensively tested to determine whether they meet the recognized standard for that language.

**ITQ****Question**

What is Regression testing?

**Feedback**

Regression testing is a type of software testing that verifies that software previously developed and tested still performs correctly after it was changed or interfaced with other software. Changes may include software enhancements, patches, configuration changes, etc. During regression testing, new software bugs or regressions may be uncovered. Sometimes a Software Change Impact Analysis is performed to determine what areas could be affected by the proposed changes.

## 3.6 Program Documentation

Program documentation is a written text or illustration that accompanies computer software. It either explains how it operates or how to use it, and may mean different things to people in different roles. Documenting is an ongoing, necessary process, although, as many programmers are, you may be eager to pursue more exciting computer-centered activities. Documentation is a written detailed description of the programming cycle and specific facts about the program. Typical program documentation materials include the origin and nature of the problem, a brief narrative description of the program, logic tools such as flowcharts and pseudocode, data-record descriptions, program listings, and testing results. Comments in the program itself are also considered an essential part of documentation. Many programmers document as they code. In a broader sense, program documentation can be part of the documentation for an entire system.

Types of documentation include:

1. Requirements documentation - Statements that identify attributes, capabilities, characteristics, or qualities of a system. This is the foundation for what will be or has been implemented.
2. Architecture/Design documentation - Overview of software. Includes relations to an environment and construction principles to be used in design of software components.
3. Technical documentation - Documentation of code, algorithms, interfaces, and APIs.
4. End user documentation - Manuals for the end-user, system administrators and support staff.
5. Marketing documentation - How to market the product and analysis of the market demand.

The wise programmer continues to document the program throughout its design, development, and testing. Documentation is needed to supplement human memory and to help organize program planning. Also, documentation is critical to communicate with others who have an interest in the program, especially other programmers who may be part of a programming team. And, since turnover is high in the computer

industry, written documentation is needed so that those who come after you can make any necessary modifications in the program or track down any errors that you missed.

### ITQ

#### Question

What is program documentation?

#### Feedback

Program documentation is a written text or illustration that accompanies computer software.

### ITQ

#### Question

What is the smart thing to do while undergoing the process of program documentation?

#### Feedback

The wise programmer continues to document the program throughout its design, development, and testing. This is necessary, as documentation is needed to supplement human memory and to help organize program planning.

## Study Session Summary



### Summary

In this study session, you discussed programming process. You started by discussing how to define and analyze a problem. You also discussed how to code a program. You furthered the discussion by examining how to compile and debug a program. You concluded the session by discussing how to test and validate a programming problem.

## Assessment



### Assessment

#### SAQ 3.1 (tests learning outcome 3.1)

1. What does Problem definition and Analysis entails?
2. How would you plan and outline solutions to the problems you have identified?

#### SAQ 3.2 (tests learning outcome 3.2)

What are the processes of coding a program?

#### SAQ 3.3 (tests learning outcome 3.3)

What is compilation?

**SAQ 3.4 (tests learning outcome 3.4)**

What is Debugging?

**SAQ 3.5 (tests learning outcome 3.5)**

What is the aim of testing?

**SAQ 3.6 (tests learning outcome 3.6)**

1. What do you understand by the term program documentation?
2. Mention the types of documentation that you know

---

## Bibliography



**Reading**

[https://en.wikipedia.org/wiki/Software\\_documentation](https://en.wikipedia.org/wiki/Software_documentation)

## Study Session 4

---

# Types of Programming Languages

## Introduction

In this study session, you will be highlighting the different types of programming languages. You will discuss the different characteristics of programming language. You will also compare low-level and high-level programming language. In addition, you will examine the compiled code interpreted codes. You will conclude the session by describing the different classifications of programming language.

## Learning Outcomes



### Outcomes

When you have studied this session, you should be able to:

- 4.1 *highlight* the characteristics of programming languages
- 4.2 *compare* low-level and high-level programming languages
- 4.3 *describe* the compiled code and interpreted code
- 4.4 *identify* the classifications of programming languages

## 4.1 Characteristics of Programming Languages

You can think of programming languages just like spoken languages, as they both share many of the same characteristics, such as:

1. **Functionality across languages:** Programming languages can all create the same functionality similar to how spoken languages can all express the same objects, phrases, and emotions.
2. **Syntax and structure:** Commands in programming languages can overlap just like words in spoken languages overlap. To output text to screen in Python or Ruby you use the print command, just like *imprimer* and *imprimir* are the verbs for “print” in French and Spanish.
3. **Natural lifespan:** Programming languages are born when a programmer thinks of a new or easier way to express a computational concept. If other programmers agree, they adopt the language for their own programs and the programming language spreads. However, just like Latin or Aramaic, if the programming language is not adopted by other programmers or a better language comes along, then the programming language slowly dies from lack of use.



Despite these similarities, programming languages also differ from spoken languages in a few key ways:

1. **One creator:** Unlike spoken languages, programming languages can be created by one person in a short period of time, sometimes in just a few days. Popular languages with a single creator include JavaScript (Brendan Eich), Python (Guido van Rossum), and Ruby (Yukihiro Matsumoto).
2. **Written in English:** Unlike spoken languages (except, of course, English), almost all programming languages are written in English. Whether they're programming in HTML, JavaScript, Python, or Ruby, Brazilian, French, or Chinese programmers all use the same English keywords and syntax in their code. Some non-English programming languages exist, such as languages in Hindi or Arabic, but none of these languages are widespread or mainstream.

### ITQ

#### Question

List two characteristics of programming language

#### Feedback

Syntax and structure; Natural lifespan

## 4.2 Comparing Low-level and High-level Programming Languages

One way to classify programming languages is either as low-level languages or high-level languages. Low-level languages interact directly with the computer processor or CPU, are capable of performing very basic commands, and are generally hard to read. Machine code, one example of a low-level language, uses code that consists of just two numbers — 0 and 1. The figure below shows an example of machine code. Assembly language, another low-level language, uses keywords to perform basic commands like read data, move data, and store data.

```
011010100110101000101101100100101011001010101001010101
011110001010111100011011101110001010100101001101010100
01010100010010010110101000101001011100011001010100100110
00110101010111101011011110100100100010110101010100000101
00110101001101010001011011001001010110010101010100101010
10111100010101111000110111011100010101001010100110101010
00101010001001001011010100010100101110001100101010010011
00011010101011110101101111010010010001011010101010000010
00110101001101010001011011001001010110010101010100101010
101111000101011110001101110111000101010010100110101010
00101010001001001011010100010100101110001100101010010011
00011010101011110101101111010010010001011010101010000010
00110101001101010001011011001001010110010101010100101010
10111100010101111000110111011100010101001010100110101010
00101010001001001011010100010100101110001100101010010011
00011010101011110101101111010010010001011010101010000010
00110101001101010001011011001001010110010101010100101010
10111100010101111000110111011100010101001010100110101010
00101010001001001011010100010100101110001100101010010011
00011010101011110101101111010010010001011010101010000010
00110101001101010001011011001001010110010101010100101010
10111100010101111000110111011100010101001010100110101010
10111100010101111000110111011100010101001010100110101010
```

By contrast, high-level languages use natural language so it is easier for people to read and write. Once code is written in a high-level language, like C++, Python, or Ruby, an interpreter or compiler translates this high-level language into low-level code a computer can understand.

## 4.3 Compiled code and interpreted code

High-level programming languages must be converted to low-level programming languages using an interpreter or compiler, depending on the language. Interpreted languages are considered more portable than compiled languages, while compiled languages execute faster than interpreted languages. However, the speed advantage compiled languages have is starting to fade in importance as improving processor speeds make performance differences between interpreted and compiled languages negligible. High-level programming languages like JavaScript, Python, and Ruby are interpreted. For these languages the interpreter executes the program directly, translating each statement one line at a time into machine code. High-level programming languages like C++, COBOL, and Visual Basic are compiled. For these languages, after the code is written a compiler translates all the code into machine code, and an executable file is created. This executable file is then distributed via the Internet, CD-ROMs, or other media and run. Software you install on your computer, like Microsoft Windows or Mac OS X, are coded using compiled languages, usually C or C++.

### ITQ

#### Question

What is the peculiarity of program languages?

#### Feedback

Some program languages (JavaScript, Python) are interpreted while others (C++, COBOL) are compiled.

## 4.4 Classification of Programming Languages

Designing, developing and using a programming language is always dynamic in nature and no particular programming languages have ever been under constant attraction to the programmers/users. Depending on the size of the program and its complexity, users prefer to choose their programming languages. It is well known that more than 2000 programming languages have been designed, developed and used till date for various purposes. But each language has drawn the attention of specific group of users and has been used for solving problems in scientific, business and engineering fields. In general, high level programming languages fall into any of the categories explained in the following subsections:

### 4.4.1 Modular Programming Languages

Modular programming languages have been very influential in introducing certain fundamental concepts in contemporary programming languages. It is the first programming language that introduced subprogram concepts and variable declaration, etc. In real life situations, problems in scientific, engineering and business areas are usually large and complex. To solve the entire problem in one shot may not be a feasible option. Modular programming methodology advocates the breakdown structure approach of a large project into several smaller and manageable modules. Then each small module is programmed comfortably and then put together in order to generate program and hence the answer to the entire problem. Modular programming has several unique advantages as given below: It is easier to understand· It is easier to code· It is easier to document· It is easier to modify· In order to implement each module in a programming language, subprograms are used. In programming, a subprogram is a series of instructions that performs a particular task. An example for the modular programming language is ALGOL. ALGOL stands for ALGOrithmic Oriented Language. ALGOL had its unique name and place among programmers in late 1960s. A notable feature of ALGOL was that it laid the foundation for many advanced programming languages of today and some of its salient features were embedded into new languages like Pascal and C.

#### ITQ

##### Question

What are the unique advantages of methodology programming language?

##### Feedback

It is easier to understand, easier to code, document, and modify.

### 4.4.2 Structured Programming Language

Since the early 1970s a new methodology has become very common in programming environments. This new methodology advocates GOTO programming. The very purpose is to eliminate the use of GOTO statements in programs. Proponents of this methodology believe that the GOTO statements make the program complex and difficult to debug or modify. In structured programming, four structures are normally used for performing any tasks. These include the following:

1. Sequence
2. Selection
3. Iteration
4. CASE structure·

Sequence means going from step1 step2 and so on. There is no looping or branching involved. Selection is used whenever choices are to be executed between two options. Iteration indicates a loop for performing a specific task. The CASE structure is used when there are more than two options from which one has to be selected based on the value of an

expression. An example of the structured programming language is Pascal. The following are the salient features of Pascal that identifies it as a structured language. These features allow a programmer to write Pascal programs in a top-down modular design:

1. It is a block structured language in which blocks of statements can be used as a unit by control structures.
2. It relies on a very heavy use of subroutines which it calls as procedures and functions.
3. It also supports both local variables known and valid only within a procedure and global variables known and valid throughout a program.

### ITQ

#### Question

How is 'sequence' different from 'selection'?

#### Feedback

Sequence means going from step one to step two and so on. It does not involve looping or branching, while Selection is used whenever choices are to be executed between two options.

### 4.4.3 Business Oriented Language

An example for the business oriented language is COBOL. COBOL stands for COMmon Business Oriented Language. Advantages of COBOL can be listed in the following points:

1. COBOL programs can be easily understood even by non-programmers due to its self-documenting nature and verbose of the grammar.
2. It is an ideal language for processing voluminous data files. Therefore it is an excellent language for commercial data processing.
3. Editing facility is very powerful and hence even the manipulation of non-numeric data is very easy.
4. COBOL is a universally standardized language.
5. It is best suited for time sharing computer systems. It is easy to debug and maintain programs in COBOL.
6. It is one of the most structured languages.
7. COBOL provides scope for effective documentation.
8. Functional/logic programming language

One of the well known and commercially available logic programming languages is PROLOG which stands for PROgramming in LOGic. Its unique property is its level of abstraction. Other main feature of PROLOG include:

1. It is exclusively meant for Artificial Intelligence (AI) and knowledge representation techniques.
2. It supports rule based expert systems and knowledge bases (data bases with inferential components).

3. It provides a foundation for understanding the semantics of other kinds of rules and is itself a good language for writing rule processors with various search strategies.
4. It is well suited for symbolic manipulation and information representation tasks.
5. Construction and extraction of data structures are the principal mechanism used in the computation processes of PROLOG.
6. Another example for the functional and logic programming language used in AI applications is the LISP. LISP stands for LISt Processing. LISP takes functions as arguments or return functions as results. It provides for defining functions that are based directly on the lambda-calculus. LISP is characterized by the following ideas:
  - i. Computing with symbolic expressions rather than with numbers.
  - ii. Representing symbolic expressions and other information by list structure in the memory of a computer.
  - iii. The use of lambda expression for naming functions.
  - iv. Representation of LISP programs as LISP data that can be manipulated by other programs.
  - v. Conditional expression interpretation of Boolean connection and others.

### ITQ

#### Question

Mention one example of Logic Programming Language?

#### Feedback

A known and commercially available logic programming language is PROLOG, which stands for PROgramming in LOGic. Its unique property is its level of abstraction.

## 4.4.4 Object oriented programming (OOP) language

The aim of using an object oriented programming language is to handle complex software design projects in a very easy, simple and efficient manner. Redesigning and maintaining the source code costs much more than reusability of the source code. The turn over time and software cost are drastically brought down due to the use of OOP language. Some of the famous objects oriented programming languages are: Object Pascal, C++, Smalltalk, Simula, Eiffel, Java, Ada. A major advantage of C++ is its ability to support object oriented programming, while retaining the high level of compactness and speed offered by the C programming language.

**ITQ****Question**

What is the aim of using Object-Oriented Programming Language?

**Feedback**

The aim of using an object oriented programming language is to handle complex software design projects in a very easy, simple and efficient manner.

### 4.4.5 Visual Programming Languages

Visual programming is one of the key points for developing any software product with multimedia and graphical user interface. Visual effects are one of the most salient features and characteristics of the modern software especially on microcomputer systems. Icons, graphics, animated pictures and texts form the foundation for designing a visual system. Modern computer is not just an electronic information processing machine but it is a powerful tool for information gathering, collection, storing and retrieving the items in any form such as text, picture, audio and video. Visual programming systems are computer systems which support both visual programming and visualization. Visual programming means the use of visual expressions (such as icons, drawings or gestures) in the process of programming and visualization and hence the use of visual representation (such as graphics, images or animation sequences) to illustrate programs, data, the structure of a complex system or the dynamic behaviour of a complex system. In visual programming languages, objects have logical meaning but not visual image. Objects are then assigned a visual representation so that it can be visualized. Some examples of visual programming languages are Visual Basic, Visual C++, Visual Foxpro, Visual J++/Java. Their application domains include the following:

1. Image processing and image communication
2. Computer vision
3. Robotics
4. Image data base management
5. Office automation
6. Computer graphics
7. Database interface
8. Form management
9. Computer aided design

**ITQ****Question**

Mention examples of visual programming languages and their application domains?

**Feedback**

Examples of visual programming languages include Visual Basic, Visual

C++, Visual Foxpro, Visual J++/Java. Their application domains could be image processing and communication, robotics, database interface just to mention a few.

---

## Study Session Summary



### Summary

In this study session, you describe the different types of programming language. You compared the low-level and high-level programming language. You also described the compiled and interpreted code. You brought the session to an end by highlighting the different classifications of programming language.

---

## Assessment



### Assessment

#### SAQ 4.1 (tests Learning Outcome 4.1)

What are the characteristics of programming languages?

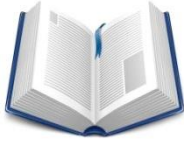
#### SAQ 4.2 (tests learning outcome 4.2)

How will you compare low-level and high level programming languages?

#### SAQ 4.3 (tests learning outcome 4.3)

Identify the classifications of programming languages?

## Credits



### Reading

This lesson has been adapted from:

[http://www.uew.edu.gh/sites/default/files/teaching%20materials/A\\_BRIEF\\_HISTORY\\_OF\\_PROGRAMMING\\_LANGUAGES\\_Sem2\\_2012.pdf](http://www.uew.edu.gh/sites/default/files/teaching%20materials/A_BRIEF_HISTORY_OF_PROGRAMMING_LANGUAGES_Sem2_2012.pdf)  
[http://www.uew.edu.gh/sites/default/files/teachingmaterials/A\\_BRIEF\\_HISTORY\\_OF\\_PROGRAMMING\\_LANGUAGE\\_S\\_Sem2\\_2012.pdf](http://www.uew.edu.gh/sites/default/files/teachingmaterials/A_BRIEF_HISTORY_OF_PROGRAMMING_LANGUAGE_S_Sem2_2012.pdf)

[http://www.uew.edu.gh/sites/default/files/teachingmaterials/A\\_BRIEF\\_HISTORY\\_OF\\_PROGRAMMING\\_LANGUAGE\\_S\\_Sem2\\_2012.pdf](http://www.uew.edu.gh/sites/default/files/teachingmaterials/A_BRIEF_HISTORY_OF_PROGRAMMING_LANGUAGE_S_Sem2_2012.pdf)

<http://www.dummies.com/how-to/content/the-types-of-programming-languages.html>  
<http://www.dummies.com/how-to/content/the-types-of-programming-languages.html>

[http://www.uew.edu.gh/sites/default/files/teaching%20materials/A\\_BRIEF\\_HISTORY\\_OF\\_PROGRAMMING\\_LANGUAGES\\_Sem2\\_2012.pdf](http://www.uew.edu.gh/sites/default/files/teaching%20materials/A_BRIEF_HISTORY_OF_PROGRAMMING_LANGUAGES_Sem2_2012.pdf)  
[http://www.uew.edu.gh/sites/default/files/teachingmaterials/A\\_BRIEF\\_HISTORY\\_OF\\_PROGRAMMING\\_LANGUAGE\\_S\\_Sem2\\_2012.pdf](http://www.uew.edu.gh/sites/default/files/teachingmaterials/A_BRIEF_HISTORY_OF_PROGRAMMING_LANGUAGE_S_Sem2_2012.pdf)

<http://www.dummies.com/how-to/content/the-types-of-programming-languages.html>



## Study Session 5

# Algorithms and Problem-Solving

## Introduction

In this study session, you will examine algorithms and problem solving. You will explain algorithm and highlight its properties. Thereafter, you will describe the pseudo-codes. In doing this, you will discuss the rules for writing pseudocodes, its advantages and disadvantages. You will conclude the session by describing what a flow chart is.

## Learning Outcomes



### Outcomes

When you have studied this session, you should be able to:

- 5.1 *explain* the problem solving process
- 5.2 *describe* algorithm
- 5.3 *discuss* pseudo-codes
- 5.4 *illustrate* flowchart

## 5.1 The Problem Solving Process

The problems in the preceding section offer a glimpse into the craft of problem solving. The Hungarian mathematician George Pólya (1887-1985) conceptualized the process and captured it in the famous book “How To Solve It” by enumerating four phases a problem solver has to go through. The four phases are outlined below.

1. Phase 1: Understanding the problem.
  - i. What is the unknown? What are the data?
  - ii. What is the condition? Is it possible to satisfy the condition? Is the condition sufficient to determine the unknown? Or is it insufficient? Or redundant? Or contradictory?
  - iii. Draw a figure. Introduce suitable notation.
  - iv. Separate the various parts of the condition. Can you write them down?
2. Phase 2: Devising a plan.
  - i. Have you seen it before? Or have you seen the same problem in slightly different form?
  - ii. Do you know a related problem?
  - iii. Look at the unknown! Try to think of a familiar problem having the same or similar unknown.
  - iv. Split the problem into smaller, simpler sub-problems.

- v. If you cannot solve the proposed problem try to solve first some related problem. Or solve a more general problem. Or a special case of the problem. Or solve a part of the problem.
3. Phase 3: Carrying out the plan.
  - i. Carrying out your plan of the solution, check each step.
  - ii. Can you see clearly that the step is correct?
  - iii. Can you prove that it is correct?
4. Phase 4: Looking back.
  - i. Can you check the result?
  - ii. Can you derive the result differently?
  - iii. Can you use the result, or the method, for some other problem?

**ITQ****Question**

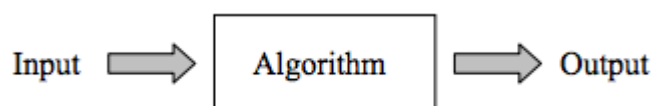
How many phases are involved in the problem-solving process?

**Feedback**

Four (4)

## 5.2 The Concept of Algorithm

The term ‘algorithm’ was derived from the name of Mohammed al-Khowarizmi, a Persian mathematician in the ninth century. An algorithm is a well-defined computational procedure consisting of a set of instructions, that takes some value or set of values, as input, and produces some value or set of values, as output. In other word, an algorithm is a procedure that accepts data, manipulate them following the prescribed steps, so as to eventually fill the required unknown with the desired value(s).



Tersely put, an algorithm, a jargon of computer specialists, is simply a procedure. People of different professions have their own form of procedure in their line of work, and they call it different names. A cook, for instance, follows a procedure commonly known as a recipe that converts the ingredients (input) into some culinary dish (output), after a certain number of steps. An algorithm is a form that embeds the complete logic of the solution. From programming point of view, an algorithm is a step-by-step procedure to resolve any problem. An algorithm is an effective method expressed as a finite set of well-defined instructions.

## ITQ

### Question

How can do programmers view Algorithms?

### Feedback

Programmers often see algorithm as a systematic procedure to resolve any problem. They also agree that it is an effective method expressed as a finite set of well-defined instructions.

## 5.2.1 Properties of Algorithm

Donald Ervin Knuth provided a list of five properties for algorithm. They are:

1. **Finiteness:** An algorithm must always terminate after a finite number of steps. It means after every step one reach closer to solution of the problem and after a finite number of steps algorithm reaches to an end point.
2. **Definiteness:** Each step of an algorithm must be precisely defined. It is done by well thought actions to be performed at each step of the algorithm. In addition, the actions are defined unambiguously for each activity in the algorithm.
3. **Input:** Any operation you perform need some beginning value/quantities associated with different activities in the operation. So, the value/quantities are given to the algorithm before it begins.
4. **Output:** One always expects output/result (expected value/quantities) in terms of output from an algorithm. The result may be obtained at different stages of the algorithm. If some result is from the intermediate stage of the operation then it is known as intermediate result and result obtained at the end of algorithm is known as end result. The output is expected to have value/quantities that always have a specified relation to the inputs
5. **Effectiveness:** Algorithms are to be developed/written using basic operations. Actually, operations should be basic, so that they can in principle, be done exactly, and in a finite amount of time by a person by using paper and pencil only.

## ITQ

### Question

List the essential properties of Algorithms?

### Feedback

Finiteness, definiteness, input, output, and effectiveness.

## 5.2.2 Algorithmic Problem Solving

Webopedia defines an algorithm as “A formula or set of steps for solving a particular problem. To be an algorithm, a set of rules must be unambiguous and have a clear stopping point”. There may be more than one way to solve a problem, so there may be more than one algorithm for a problem. Now, if we take definition of algorithm as: “A sequence of activities to be processed for getting desired output from a given input.” Then we can say that:

1. Getting specified output is essential after algorithm is executed.
2. One will get output only if algorithm stops after finite time.
3. Activities in an algorithm to be clearly defined in other words for it to be unambiguous. Before writing an algorithm for a problem, one should find out what is/are the inputs to the algorithm and what is/are expected output after running the algorithm.

Algorithmic problem solving actually comes in two phases: derivation of an algorithm that solves the problem, and conversion of the algorithm into code. The latter, usually known as coding, is comparatively easier, since the logic is already present – it is just a matter of ensuring that the syntax rules of the programming language are adhered to. The first phase is what stumbles most people, for two main reasons. Firstly, it challenges the mental faculties to search for the right solution, and secondly, it requires the ability to articulate the solution concisely into systematic instructions, a skill that is acquired only through lots of practice. A computer programmer lists down all the steps required to resolve a problem before writing the actual code.

### Example 5.1

The following is a simple example of an algorithm to find out the largest number from a given list of numbers:

1. Get a list of numbers  $L_1, L_2, L_3, \dots, L_N$
2. Assume  $L_1$  is the largest,  $\text{Largest} = L_1$
3. Take next number  $L_i$  from the list and do the following
4. If  $\text{Largest}$  is less than  $L_i$
5.  $\text{Largest} = L_i$
6. If  $L_i$  is last number from the list then
7. Print value stored in  $\text{Largest}$  and come out
8. Else repeat same process starting from step 3

The above algorithm has been written in a crude way to help beginners understand the concept. You will come across more standardized ways of writing computer algorithms as you move on to advanced levels of computer programming.

### Example 5.2:

The following algorithm finds the volume of a cube where the input to the algorithm is length, width and height, and expected output is volume of the cube.

Step 1: input the length, width and height of the cube

Step 2:  $\text{Volume} \leftarrow \text{length} * \text{width} * \text{height}$

## Step 3: Print Volume

**ITQ****Question**

What is the most important rule for a computer programmer?

**Feedback**

A computer programmer lists down all the steps required to resolve a problem before writing the actual code.

## 5.3 Pseudo-Codes

Pseudocode is an informal high-level description of the operating principle of a computer program or other algorithm. It uses the structural conventions of a normal programming language, but is intended for human reading rather than machine reading. Pseudocode typically omits details that are essential for machine understanding of the algorithm, such as variable declarations, system-specific code and some subroutines. The programming language is augmented with natural language description details, where convenient, or with compact mathematical notation. The purpose of using pseudocode is that it is easier for people to understand than conventional programming language code, and that it is an efficient and environment-independent description of the key principles of an algorithm. It is commonly used in textbooks and scientific publications that are documenting various algorithms, and also in planning of computer program development, for sketching out the structure of the program before the actual coding takes place.

No standard for pseudocode syntax exists, as a program in pseudocode is not an executable program. Pseudocode resembles, but should not be confused with skeleton programs which can be compiled without errors. Flowcharts, drakon-charts and Unified Modeling Language (UML) charts can be thought of as a graphical alternative to pseudocode, but are more spacious on paper.

**ITQ****Question**

What augments Pseudo-Code?

**Feedback**

The programming language (Pseudo-Code) is augmented with natural language description details, where convenient, or with compact mathematical notation.

### 5.3.1 Rules for Writing Pseudocode

1. Write only one statement per line: As illustrated in example 5.3, each statement in pseudocode should express just one action for the computer.

**Example 5.3:**

The gross pay of a worker is calculated in the pseudocode below where name, hours worked and pay rate are the input and name, hours worked and gross pay are the output.

```
READ name, hoursWorked, payRate
```

```
gross = hoursWorked * payRate
```

```
WRITE name, hoursWorked, gross
```

2. Capitalize the keywords. In example 5.3, note that the words: READ and WRITE are keywords. These are just a few of the keywords to use, others can be:

IF, ELSE, ENDIF, WHILE, ENDWHILE

3. Indent to show hierarchy: The indentation pattern in each of the 3 major design structures; sequence, selection and iteration is commonly used to show hierarchy. In sequence pattern, statements that are “stacked” in sequence all start in the same column. In selection pattern, statements that fall inside the selection structure, but not the keywords that form the selection are indented. Lastly, in the looping pattern, statements that fall inside the loop, but not the keywords that form the loop are indented. This is illustrated in example 5.4

**Example 5.4**

The pseudocode below illustrates indentation using example 5.3 above where, employees whose grossPay is less than 100 do not have any deduction.

```
READ name, grossPay, taxes
```

```
IF taxes > 0
```

```
net = grossPay – taxes
```

```
ELSE
```

```
net = grossPay
```

```
ENDIF
```

```
WRITE name, net
```

4. End multiline structures: As shown in example 5.4 above, the ENDIF (or END whatever) always is in line with the IF (or whatever starts the structure).
5. Keep stats language independent: Resist the urge to write in whatever language you are most comfortable with. In the long run, you will save time! There may be special features available in the language you plan to eventually write the program in; if

you are SURE it will be written in that language, then you can use the features. If not, then avoid using the special features.

## ITQ

### Question

Highlight the rules for writing Pseudo-codes?

### Feedback

1. Write only one statement per line
2. Capitalize the keywords
3. Indent to show hierarchy
4. End multiline structures
5. Keep stats language independent

## 5.3.2 Advantages of Pseudocode

1. It can be easily in any word processor.
2. It can be easily modified as compared to flowchart.
3. It's implementation is very useful in structured design elements. It implements structured concepts well
4. It can be written easily.
5. It can be read and understood easily
6. Converting a pseudocode to programming language is very easy as compared with converting a flowchart to programming language.

## 5.3.3 Disadvantages of Pseudocode

1. It is not visual.
2. We do not get a picture of the design.
3. There is no standardized style or format, so one pseudocode may be different from another.
4. For a beginner, it is more difficult to follow the logic or write pseudocode as compared to flowchart.

### Example 5.5

The pseudocode below prints “failed” or “passed” depending on whether or not the student’s grade is less than 60.

```
1. If student's grade is greater than or equal to 60
   Print "passed"
Else
   Print "failed"
```

### Example 5.6

The pseudocode below finds the class average by using the grades of all students in the class as input:

```
Set total to zero
Set grade counter to one
While grade counter is less than or equal to ten
   Input the next grade
   Add the grade into the total
Set the class average to the total divided by ten
```

Print the class average.

### ITQ

#### Question

What are the disadvantages of using Pseudo-Codes?

#### Feedback

1. Pseudo-Code is not visual.
2. Pseudo-Code does not reveal a picture of the design.

## 5.4 Flowchart

Algorithms are nothing but sequence of steps for solving problems. So a flow chart can be used for representing an algorithm. A flowchart is a graphical representation of an algorithm, which will describe the operations (and in what sequence) are required to solve a given problem. These flowcharts play a vital role in the programming of a problem and are quite helpful in understanding the logic of complicated and lengthy problems. You can see a flow chart as a blueprint of a design you have made for solving a problem. For example suppose you are going for a picnic with your friends then you plan for the activities you will do there. If you have a plan of activities then you know clearly when you will do what activity. Similarly when you have a problem to solve using computer or in other word you need to write a computer program for a problem then it will be good to draw a flowchart prior to writing a computer program. Once the flowchart is drawn, it becomes easy to write the program in any high level language. Often we see how flowcharts are helpful in explaining the program to others. Hence, it is correct to say that a flowchart is a must for the better documentation of a complex program.

### ITQ

#### Question

Explain the importance of flow charts in the documentation of a program?

#### Feedback

A flowchart, once prepared, makes it easy to write programs in any high level language. This important feature of flow charts makes it an indispensable tool and helps in explaining the program to others.

### 5.4.1 Advantages of Flowcharts

The pictorial representation of a solution/system is having many advantages. These advantages are as follows:

1. Communication: A Flowchart can be used as a better way of communication of the logic of a system and steps involve in the solution, to all concerned particularly to the client of system.
2. Effective analysis: A flowchart of a problem can be used for effective analysis of the problem.



3. Documentation of Program/System: Program flowcharts are a vital part of a good program documentation. Program document is used for various purposes like knowing the components in the program, complexity of the program etc.
4. Efficient Program Maintenance: Once a program is developed and becomes operational it needs time to time maintenance. With help of flowchart maintenance become easier.
5. Coding of the Program: Any design of solution of a problem is finally converted into computer program. Writing code referring the flowchart of the solution become easy.

### 5.4.2 Disadvantages of Flowcharts

1. Complex logic: Sometimes, the program logic is quite complicated. In that case, flowchart becomes complex and clumsy. This will become a pain for the user, resulting in a waste of time and money trying to correct the problem
2. Alterations and Modifications: If alterations are required the flowchart may require re-drawing completely. This will usually waste valuable time.
3. Reproduction: As the flowchart symbols cannot be typed, reproduction of flowchart becomes a problem.

#### ITQ

##### Question







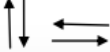
What are the advantages of Flow Charts?

##### Feedback

Documentation of program/system; and effective analysis.

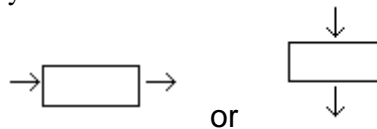
### 5.4.3 Flowchart Symbols

Flowcharts are usually drawn using some standard symbols;

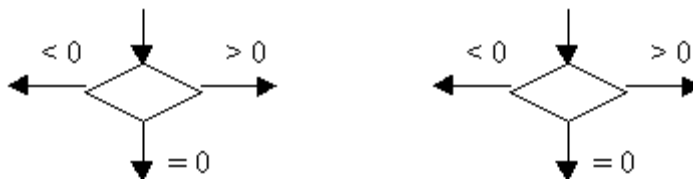
| Symbol   | Name               | Function   |
|--|--------------------|--|
|   | <b>Process</b>     | Indicates any type of internal operation inside the Processor or Memory                                      |
|   | input/output       | Used for any Input / Output (I/O) operation. Indicates that the computer is to obtain data or output results |
|   | Decision           | Used to ask a question that can be answered in a binary format (Yes/No, True/False)                          |
|   | Connector          | Allows the flowchart to be drawn without intersecting lines or without a reverse flow.                       |
|   | Predefined Process | Used to invoke a subroutine or an Interrupt program.   |
|   | Terminal           | Indicates the starting or ending of the program, process, or interrupt program                               |
|  | Flow Lines         | Shows direction of flow.   |

### 5.4.4 General Guidelines in Flowcharting

1. In drawing a proper flowchart, all necessary requirements should be listed out in logical order.
2. The flowchart should be clear, neat and easy to follow. There should not be any room for ambiguity in understanding the flowchart.
3. The usual direction of the flow of a procedure or system is from left to right or top to bottom.
4. Only one flow line should come out from a process symbol



5. Only one flow line should enter a decision symbol, but two or three flow lines, one for each possible answer, should leave the decision symbol.



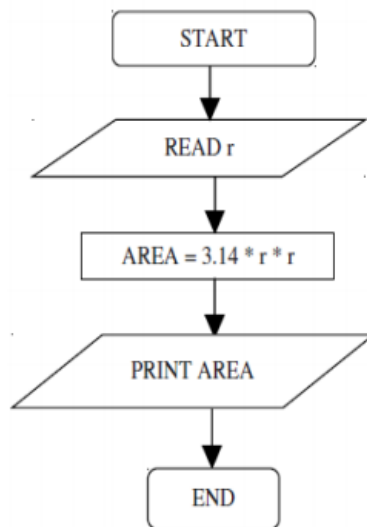
- Only one flow line is used in conjunction with terminal symbol.



- All boxes of the flowchart are connected with Arrows. (Not lines). If the flowchart becomes complex, it is better to use connector symbols to reduce the number of flow lines. Avoid the intersection of flow lines if you want to make it more effective and better way of communication.
- Ensure that the flowchart has a logical start and finish.
- It is useful to test the validity of the flowchart by passing through it with a simple test data.

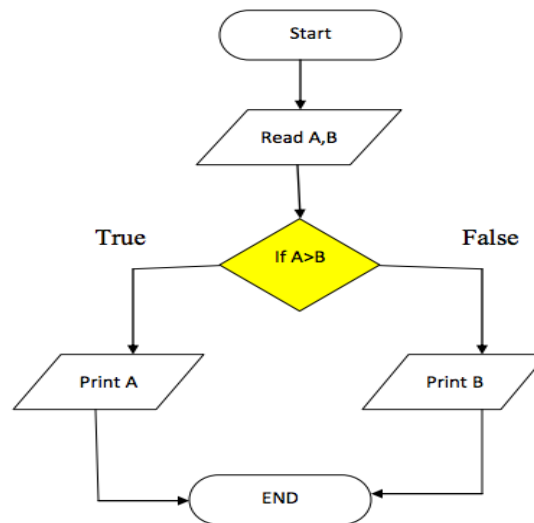
Example 5.7

The following flowchart finds the area of a circle of radius  $r$ .



Example 5.8

The flowchart below finds the greater of two numbers



### Example 5.9

Write an algorithm and draw the flowchart for finding the average of two numbers

Algorithm:

Input: two numbers  $n_1$  and  $n_2$

Output: the average of  $n_1$  and  $n_2$

Steps:

1. input  $n_1$
2. input  $n_2$
3.  $sum = n_1 + n_2$
4.  $average = sum / 2$
5. output average

## ITQ

### Question

Mention two (2) guidelines in flowcharting

### Feedback

- The flowchart should be clear, neat and easy to follow. There should not be any room for ambiguity in understanding the flowchart, and
- The usual direction of the flow of a procedure or system is from left to right or top to bottom.

---

## Study Session Summary



### Summary

In this study session, you examined algorithm and problem solving. You started by defining algorithm and highlighted its different properties. You also discussed the pseudocode. Under this, you looked at rules for writing pseudocodes, its advantages and disadvantages. You concluded the session by describing what a flow chart is, its advantages and disadvantages.

---

## Assessment



### Assessment

#### SAQ 5.1 (tests Learning Outcome 5.1)

Describe the problem solving process?

#### SAQ 5.2 (tests learning outcome 5.2)

Explain the concept of Algorithm?

#### SAQ 5.3 (tests learning outcome 5.3)

Describe Pseudo-codes?

#### SAQ 5.4 (test learning outcome 5.4)

What is a Flowchart?

---

## Credits



### Reading

This lesson has been adapted from:

[http://www.comp.nus.edu.sg/~cs1101x/4\\_misc/jumpstart/chap2.pdf](http://www.comp.nus.edu.sg/~cs1101x/4_misc/jumpstart/chap2.pdf)

<http://faradars.org/wp-content/uploads/2015/07/Algorithm-and-Flow-Chart.pdf>

[http://www.tutorialspoint.com/computer\\_programming/computer\\_programming\\_overview.htm](http://www.tutorialspoint.com/computer_programming/computer_programming_overview.htm)

[http://www.comp.nus.edu.sg/~cs1101x/4\\_misc/jumpstart/chap3.pdf](http://www.comp.nus.edu.sg/~cs1101x/4_misc/jumpstart/chap3.pdf)<http://faradars.org/wp-content/uploads/2015/07/Algorithm-and-Flow-Chart.pdf>

<http://faculty.ccri.edu/mkelly/COMI1150/PseudocodeBasics.pdf><http://faculty.ccri.edu/mkelly/COMI1150/PseudocodeBasics.pdf>

[http://www.comp.nus.edu.sg/~cs1101x/4\\_misc/jumpstart/chap2.pdf](http://www.comp.nus.edu.sg/~cs1101x/4_misc/jumpstart/chap2.pdf)

[http://www.comp.nus.edu.sg/~cs1101x/4\\_misc/jumpstart/chap2.pdf](http://www.comp.nus.edu.sg/~cs1101x/4_misc/jumpstart/chap2.pdf)

## Study Session 6

# Basics of Computer Program

## Introduction

In this study session, you will be examining the basics of computer programming. You will begin by exploring the programming environment. Under which you will examine text editor, compiler and interpreter. Thereafter, you will discuss the basic syntax of programming. You will also explain variable. In doing so, you will describe how to create, store and access variables. Moving on, you will examine reserve words. In addition, you will describe operator. This will lead you to focusing on arithmetic, relational, logical operations, and functions.

## Learning Outcomes



### Outcomes

When you have studied this session, you should be able to:

- 6.1 *describe* a programming environment
- 6.2 *explain* basic syntax of programming
- 6.3 *define* variables
- 6.4 *discuss* reserved words
- 6.5 *define* operators
- 6.6 *explain* functions

## Terminology

|                    |  |
|--------------------|--|
| <b>Interpreter</b> | A program that executes another program written in a programming language other than machine code. |
| <b>Editors</b>     | A program for creating and making changes to files, especially text files                          |

## 6.1 Programming Environment

Though Environment Setup is not an element of any Programming Language, it is the first step to take before setting on to write a program.

When we say Environment Setup, it simply implies a base on top of which we can do our programming. Thus, we need to have the required software setup, i.e., installation on our PC which will be used to write

computer programs, compile, and execute them. For example, if you need to browse Internet, then you need the following setup on your machine –

1. A working Internet connection to connect to the Internet
2. A Web browser such as Internet Explorer, Chrome, Safari, etc.

Similarly, you will need the following setup to start with programming using any programming language.

1. A text editor to create computer programs.
2. A compiler to compile the programs into binary format.
3. An interpreter to execute the programs directly.

In case you do not have sufficient exposure to computers, you will not be able to set up either of this software. Therefore, we suggest you take the help from any technical person around you to set up the programming environment on your machine from where you can start. However, for you, it is important to understand what these items are.

## ITQ

### Question

What is the first step to take before writing a program?

### Feedback

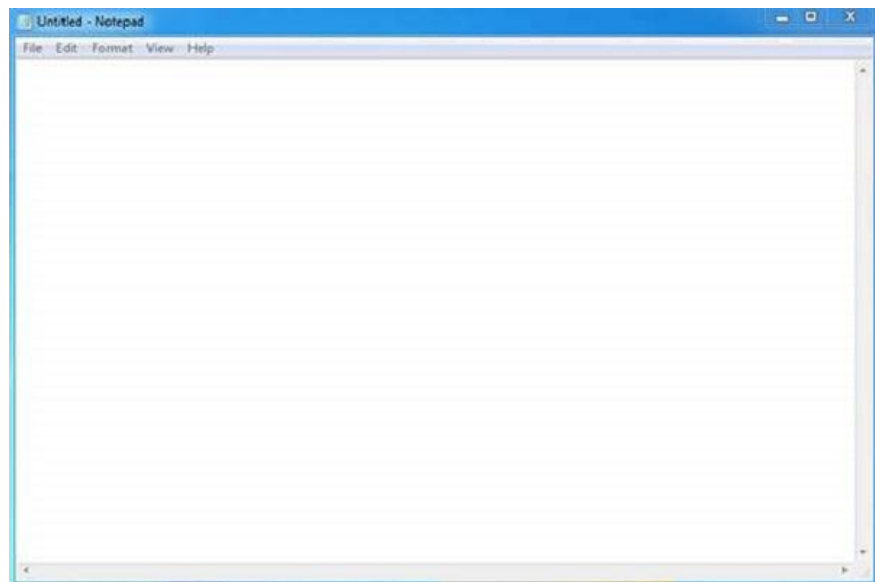
Environment Set-up

## 6.1.1 Text Editor

A text editor is a software that is used to write computer programs. Common example is Notepad, which can be used to type programs. You can launch it by following these steps –

Start Icon → All Programs → Accessories → Notepad → Mouse Click on Notepad

It will launch Notepad with the following window –

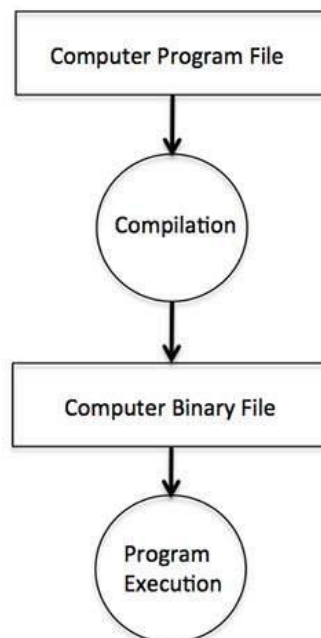


You can use this software to type your computer program and save it in a file at any location. You can download and install other good editors like Notepad++, which is freely available. If you are a Mac user, then you will have TextEdit or you can install some other commercial editor like BBEdit to start with.

### 6.1.2 Compiler

You write your computer program using your favorite programming language and save it in a text file called the program file. Now let us try to get a little more detail on how the computer understands a program written by you using a programming language. Actually, the computer cannot understand your program directly given in the text format, so we need to convert this program in a binary format, which can be understood by the computer. The conversion from text program to binary file is done by another software called Compiler and this process of conversion from text formatted program to binary format file is called program compilation. Finally, you can execute binary file to perform the programmed task.

The following flow diagram gives an illustration of the compilation process –



So, if you are going to write your program in any such language, which needs compilation like C, C++, Java and Pascal, etc., then you will need to install their compilers before you start programming.



**ITQ****Question**

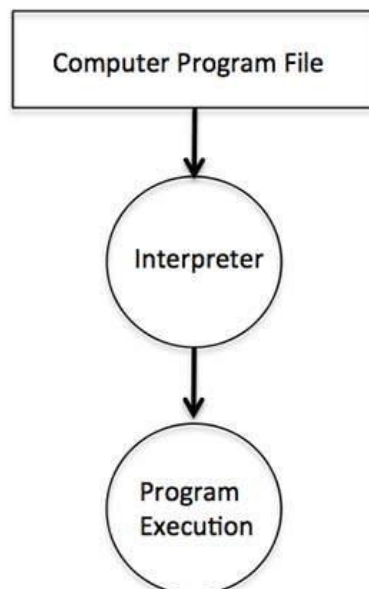
What does Program Compilation entail?

**Feedback**

Ordinarily, the computer cannot understand your program directly given in the text format, so there is a need to convert the program in a binary format, which the computer understands. This conversion from text program to binary file is done using a software called Compiler and this process of conversion from text formatted program to binary format file is called program compilation.

### 6.1.3 Interpreter

We just discussed about compilers and the compilation process. Compilers are required in case you are going to write your program in a programming language that needs to be compiled into binary format before its execution. There are other programming languages such as Python, PHP, and Perl, which do not need any compilation into binary format, rather an interpreter can be used to read such programs line by line and execute them directly without any further conversion.



So, if you are going to write your programs in PHP, Python, Perl, Ruby, etc., then you will need to install their interpreters before you start programming.

**ITQ****Question**

Mention two programming languages that does not require compilation into a binary format.

**Feedback**

Python and PHP? Yes, you are correct.

## 6.2 Basic Syntax of Programming

Let's start with a little coding, which will really make you a computer programmer. We are going to write a single-line computer program to write Hello, World! on your screen. Let's see how it can be written using different programming languages.

```
#include <stdio.h>

main() {
    /* printf() function to write Hello, World! */
    printf( "Hello, World!" );
}
```

This little Hello World program will help us understand various basic concepts related to C Programming.

### 6.2.1 Program Entry Point

For now, just forget about the `#include <stdio.h>` statement, but keep a note that you have to put this statement at the top of a C program.

Every C program starts with `main()`, which is called the main function, and then it is followed by a left curly brace. The rest of the program instruction is written in between and finally a right curly brace ends the program.

The coding part inside these two curly braces is called the program body. The left curly brace can be in the same line as `main(){` or in the next line like it has been mentioned in the above program.

**ITQ****Question**

What follows the main function { main() }?

**Feedback**

A left curly brace

### 6.2.2 Functions

Functions are small units of programs and they are used to carry out a specific task. For example, the above program makes use of two

functions: main() and printf(). Here, the function main() provides the entry point for the program execution and the other function printf() is being used to print an information on the computer screen. You can write your own functions which we will see in a separate chapter, but C programming itself provides various built-in functions like main(), printf(), etc., which we can use in our programs based on our requirement. Some of the programming languages use the word sub-routine instead of function, but their functionality is more or less the same.

### ITQ

#### Question

What do we utilize functions?

#### Feedback

Functions are used to carry out a specific task.

## 6.2.3 Comments

A C program can have statements enclosed inside `/*.....*/`. Such statements are called comments and these comments are used to make the programs user friendly and easy to understand. The good thing about comments is that they are completely ignored by compilers and interpreters. So you can use whatever language you want to write your comments.

### ITQ

#### Question

How do we utilize comments?

#### Feedback

Comments is used to make the program more user friendly, and easy to understand.

## 6.2.4 Whitespaces

When we write a program using any programming language, we use various printable characters to prepare programming statements. These printable characters are a, b, c,.....z, A, B, C,.....Z, 1, 2, 3,..... 0, !, @, #, \$, %, ^, &, \*, (, ), -, \_, +, =, \, |, {, }, [, ], :, ;, <, >, ?, /, \, ~, ` . " , ' . Hope I'm not missing any printable characters from your keyboard.

Apart from these characters, there are some characters which we use very frequently but they are invisible in your program and these characters are spaces, tabs (`\t`), new lines (`\n`). These characters are called whitespaces. These three important whitespace characters are common in all the programming languages and they remain invisible in your text document

| Whitespace | Explanation          | Representation |
|------------|----------------------|----------------|
| New Line   | To create a new line | \n             |
| Tab        | To create a tab.     | \t             |
| Space      | To create a space.   | empty space    |

A line containing only whitespace, possibly with a comment, is known as a blank line, and a C compiler totally ignores it. Whitespace is the term used in C to describe blanks, tabs, newline characters, and comments. So you can write `printf("Hello, World!");` as shown below. Here all the created spaces around "Hello, World!" are useless and the compiler will ignore them at the time of compilation.

```
#include <stdio.h>
main() {
    /* printf() function to write Hello, World! */
    printf( "Hello, World!" );
}
```

If we make all these whitespace characters visible, then the above program will look like this and you will not be able to compile it –

```
#include <stdio.h>\n
\n
main()\n
{\n
\n
\t/* printf() function to write Hello, World! */\n
\n
\tprintf(\t"Hello, World!\t);\n
\n
}\n
```

**ITQ****Question**

What do we refer to as “Whitespaces”?

**Feedback**

These often refer to characters, which we use very frequently but are not visible in your program. These characters could be spaces, tabs (\t), new lines(\n).

## 6.2.5 Semicolons

Every individual statement in a C Program must be ended with a semicolon (;), for example, if you want to write "Hello, World!" twice, then it will be written as follows –

```
#include <stdio.h>
main() {
    /* printf() function to write Hello, World! */
    printf( "Hello, World!\n" );
    printf( "Hello, World!" );
}
```

This program will produce the following result –

Hello, World!

Hello, World!

Here, we are using a new line character \n in the first printf() function to create a new line. Let us see what happens if we do not use this new line character –

```
#include <stdio.h>
main() {
    /* printf() function to write Hello, World! */
    printf( "Hello, World!" );
    printf( "Hello, World!" );
}
```

This program will produce the following result –

Hello, World! Hello, World!

We will learn identifiers and keywords in next few chapters.

**Program Explanation**

Let us understand how the above C program works. First of all, the above program is converted into a binary format using C compiler. So let's put this code in test.c file and compile it as follows –

```
$gcc test.c -o demo
```

If there is any grammatical error (Syntax errors in computer terminologies), then we fix it before converting it into binary format. If everything goes fine, then it produces a binary file called demo. Finally, we execute the produced binary demo as follows –

```
./demo
```

which produces the following result –

```
Hello, World!
```

Here, when we execute the binary a.out file, the computer enters inside the program starting from main() and encounters a printf() statement. Keep a note that the line inside /\*...\*/ is a comment and it is filtered at the time of compilation. So printf() function instructs the computer to print the given line at the computer screen. Finally, it encounters a right curly brace which indicates the end of main() function and exits the program.

## 6.2.6 Syntax Error

If you do not follow the rules defined by the programming language, then at the time of compilation, you will get syntax errors and the program will not be compiled. From syntax point of view, even a single dot or comma or a single semicolon matters and you should take care of such small syntax as well. In the following example, we have skipped a semicolon, let's try to compile the program –

```
#include <stdio.h>

main() {

    printf("Hello, World!")
}
```

This program will produce the following result –

```
main.c: In function 'main':
main.c:7:1: error: expected ';' before '}' token
}
^
```

So the bottom-line is that if you are not following proper syntax defined by the programming language in your program, then you will get syntax errors. Before attempting another compilation, you will need to fix them and then proceed.

### ITQ

#### Question

How does Syntax Error occur?

#### Feedback

Syntax Error often occur when the rules defined by the programming

language are not followed to letter. The result is that at the time of compilation, you will get syntax errors and the program would not be compiled leading to a failed execution

### 6.2.7 Hello World Program in Java

Following is the equivalent program written in Java. This program will also produce the same result Hello, World!.

```
public class HelloWorld {
    public static void main(String []args) {
        /* println() function to write Hello, World! */
        System.out.println("Hello, World!");
    }
}
```

### 6.2.8 Hello World Program in Python

Following is the equivalent program written in Python. This program will also produce the same result Hello, World!.

```
# print function to write Hello, World! */
print "Hello, World!"
```

Note that for C and Java examples, first we are compiling the programs and then executing the produced binaries, but in Python program, we are directly executing it. As we explained in the previous chapter, Python is an interpreted language and it does not need an intermediate step called compilation. Python does not require a semicolon (;) to terminate a statement, rather a new line always means termination of the statement.

#### ITQ

##### Question

Mention one important point to note when using a Python program

##### Feedback

Python program uses an interpreted language; therefore, it does not require the compilation process/stage unlike the C and Java programs that need the intermediate stage of compiling before execution.

## 6.3 Variables

Variables are the names you give to computer memory locations which are used to store values in a computer program. For example, assume you want to store two values 10 and 20 in your program and at a later stage, you want to use these two values. Let us see how you will do it. Here are the following three simple steps:

1. Create variables with appropriate names.
2. Store your values in those two variables.
3. Retrieve and use the stored values from the variables.

### 6.3.1 Creating Variables

Creating variables is also called declaring variables in C programming. Different programming languages have different ways of creating variables inside a program. For example, C programming has the following simple way of creating variables –

```
#include <stdio.h>

main() {
    int a;
    int b;
}
```

The above program creates two variables to reserve two memory locations with names a and b. We created these variables using int keyword to specify variable data type which means we want to store integer values in these two variables. Similarly, you can create variables to store long, float, char or any other data type. For example –

```
/* variable to store long value */
long a;
/* variable to store float value */
float b;
```

You can create variables of similar type by putting them in a single line but separated by comma as follows –

```
#include <stdio.h>

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

Listed below are the key points about variables that you need to keep in mind:

1. A variable name can hold a single type of value. For example, if variable a has been defined int type, then it can store only integer.
2. C programming language requires a variable creation, i.e., declaration before its usage in your program. You cannot use a variable name in your program without creating it, though programming language like Python allows you to use a variable name without creating it.
3. You can use a variable name only once inside your program. For example, if a variable a has been defined to store an integer value, then you cannot define a again to store any other type of value.
4. There are programming languages like Python, PHP, Perl, etc., which do not want you to specify data type at the time of creating variables. So you can store integer, float, or long without specifying their data type.



5. You can give any name to a variable like age, sex, salary, year1990 or anything else you like to give, but most of the programming languages allow to use only limited characters in their variables names. For now, we will suggest to use only a....z, A....Z, 0....9 in your variable names and start their names using alphabets only instead of digits.
6. Almost none of the programming languages allow to start their variable names with a digit, so 1990year will not be a valid variable name whereas year1990 or ye1990ar are valid variable names.

Every programming language provides more rules related to variables and you will learn them when you will go in further detail of that programming language.

## ITQ

### Question

In C programming language, what other term can be used to replace 'Creating variables'?

### Feedback

Creating variables is also called declaring variables in C programming.

## 6.3.2 Storing Values in Variables

You have seen how we created variables in the previous section. Now, let's store some values in those variables –

```
#include <stdio.h>
main() {
    int a;
    int b;
    a = 10;
    b = 20;
}
```

The above program has two additional statements where we are storing 10 in variable a and 20 is being stored in variable b. Almost all the programming languages have similar way of storing values in variable where we keep variable name in the left hand side of an equal sign = and whatever value we want to store in the variable, we keep that value in the right hand side.

Now, we have completed two steps, first we created two variables and then we stored required values in those variables. Now variable a has value 10 and variable b has value 20. In other words we can say, when above program is executed, the memory location named a will hold 10 and memory location b will hold 20.

### 6.3.3 Accessing Stored Values in Variables

If we do not use the stored values in the variables, then there is no point in creating variables and storing values in them. We know that the above program has two variables a and b and they store the values 10 and 20, respectively. So let's try to print the values stored in these two variables. Following is a C program, which prints the values stored in its variables –

```
#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 the above program is executed, it produces the following result –

Value of a = 10

Value of b = 20

You must have seen printf() function in the previous chapter where we had used it to print "Hello, World!". This time, we are using it to print the values of variables. We are making use of %d, which will be replaced with the values of the given variable in printf() statements. We can print both the values using a single printf() statement as follows –

```
#include <stdio.h>

main() {
    int a;
    int b;
    a = 10;
    b = 20;
    printf( "Value of a = %d and value of b = %d\n", a, b );
}
```

When the above program is executed, it produces the following result –

Value of a = 10 and value of b = 20

If you want to use float variable in C programming, then you will have to use %f instead of %d, and if you want to print a character value, then you will have to use %c. Similarly, different data types can be printed using different % and characters.

**ITQ****Question**

What is necessary after creating variable?

**Feedback**

It is important to store variables after creating them else, the purpose of creation would be defeated.

### 6.3.4 Variables in Java

Following is the equivalent program written in Java programming language. This program will create two variables a and b and very similar to C programming, it will assign 10 and 20 in these variables and finally print the values of the two variables in two ways –

```
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);
    }
}
```

### 6.3.5 Variables in Python

Following is the equivalent program written in Python. This program will create two variables a and b and at the same time, assign 10 and 20 in those variables.

Python does not want you to specify the data type at the time of variable creation and there is no need to create variables in advance.

```
a = 10
b = 20
print "Value of a = ", a
print "Value of b = ", b
print "Value of a = ", a, " and value of b = ", b
```

You can use the following syntax in C and Java programming to declare variables and assign values at the same time –

```
#include <stdio.h>
main() {
```

```

int a = 10;
int b = 20;
printf( "Value of a = %d and value of b = %d\n", a, b );
}

```

## 6.4 Reserved Words

Reserved word (also known as a keyword) is a word in programming that cannot be used as an identifier, such as the name of a variable, function, or label – it is "reserved from use". Different programming languages provide different set of reserved keywords, but there is one important & common rule in all the programming languages that we cannot use a reserved keyword to name our variables, which means we cannot name our variable like `int` or `float` rather these keywords can only be used to specify a variable data type.

For example, if you will try to use any reserved keyword for the purpose of variable name, then you will get a syntax error.

```

#include <stdio.h>
main() {
    int float;
    float = 10;
    printf( "Value of float = %d\n", float);
}

```

When you compile the above program, it produces the following error –

main.c: In function 'main':

main.c:5:8: error: two or more data types in declaration specifiers

```

    int float;

```

.....

Let's now give a proper name to our integer variable, then the above program should compile and execute successfully –

```

#include <stdio.h>
main() {
    int count;
    count = 10;
    printf( "Value of count = %d\n", count);
}

```

The following sections give the list of reserved words in C, Java and Python programming languages. We know you cannot memorize all these keywords, but we have listed them down for your reference purpose and to explain the concept of reserved keywords. So just be careful while

giving a name to your variable, you should not use any reserved keyword for that programming language.

### ITQ

#### Question

What is the common rule in all programming languages?

#### Feedback

The important and common rule in all the programming languages is that a reserved keyword cannot be used to name variables.

## 6.4.1 C Programming Reserved Keywords

Here is a table having almost all the keywords supported by C Programming language:

|          |        |          |          |
|----------|--------|----------|----------|
| Auto     | Else   | long     | switch   |
| Break    | Enum   | register | typedef  |
| Case     | Extern | return   | union    |
| Char     | Float  | short    | unsigned |
| Const    | For    | signed   | void     |
| continue | Goto   | sizeof   | volatile |
| Default  | If     | static   | while    |
| Do       | Int    | struct   | _Packed  |
| Double   |        |          |          |

## 6.4.2 Java Programming Reserved Keywords

Here is a table having almost all the keywords supported by Java Programming language –

|          |        |         |       |
|----------|--------|---------|-------|
| 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    | Static       | strictfp | super      |
| Switch   | Synchronized | this     | throw      |
| Throws   | Transient    | try      | void       |
| Volatile | While        |          |            |

### 6.4.3 Python Programming Reserved Keywords

Here is a table having almost all the keywords supported by Python Programming language –

|        |         |       |
|--------|---------|-------|
| And    | Exec    | not   |
| Assert | Finally | or    |
| Break  | For     | pass  |
| Class  | From    | print |

|          |        |        |
|----------|--------|--------|
| Continue | Global | raise  |
| Def      | If     | return |
| Del      | Import | try    |
| Elif     | In     | while  |
| Else     | Is     | with   |
| Except   | Lambda | yield  |

## 6.5 Operators

An operator in a programming language is a symbol that tells the compiler or interpreter to perform specific mathematical, relational or logical operation and produce final result. This lesson will explain the concept of operators and it will take you through the important arithmetic and relational operators available in C, Java, and Python.

### 6.5.1 Arithmetic Operators

Computer programs are widely used for mathematical calculations. We can write a computer program which can do simple calculation like adding two numbers ( $2 + 3$ ) and we can also write a program, which can solve a complex equation like  $P(x) = x^4 + 7x^3 - 5x + 9$ . Even if you have been a poor student, you must be aware that in first expression 2 and 3 are operands and + is an operator. Similar concepts exist in Computer Programming.

Take a look at the following two examples –

$$2 + 3$$

$$P(x) = x^4 + 7x^3 - 5x + 9.$$

These two statements are called arithmetic expressions in a programming language and plus, minus used in these expressions are called arithmetic operators and the values used in these expressions like 2, 3 and x, etc., are called operands. In their simplest form, such expressions produce numerical results. Similarly, a programming language provides various arithmetic operators. The following table lists down a few of the important arithmetic operators available in C programming language. Assume variable A holds 10 and variable B holds 20, then –

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

Following is a simple example of C Programming to understand the above mathematical operators –

```
#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 the above program is executed, it produces the following result –

Value of c = 30

Value of c = -10

Value of c = 200

Value of c = 2



Value of  $c = 0$

## 6.5.2 Relational Operators

Consider a situation where we create two variables and assign them some values as follows –

$A = 20$

$B = 10$

Here, it is obvious that variable  $A$  is greater than  $B$  in values. So, we need the help of some symbols to write such expressions which are called relational expressions. If we use C programming language, then it will be written as follows –

$(A > B)$

Here, we used a symbol  $>$  and it is called a relational operator and in their simplest form, they produce Boolean results which means the result will be either true or false. Similarly, a programming language provides various relational operators. The following table lists down a few of the important relational operators available in C programming language. Assume variable  $A$  holds 10 and variable  $B$  holds 20, then –

| Operator | Description   | Example                 |
|----------|---|-------------------------|
| $==$     | Checks if the values of two operands are equal or not, if yes then condition becomes true.                                      | $(A == B)$ is not true. |
| $!=$     | Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.                     | $(A != B)$ is true.     |
| $>$      | Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.             | $(A > B)$ is not true.  |
| $<$      | Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.                | $(A < B)$ is true.      |
| $>=$     | Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true. | $(A >= B)$ is not true. |

|    |  |                   |
|----|--|-------------------|
| <= | Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true. | (A <= B) is true. |
|----|--|-------------------|

Here, we will show you one example of C Programming which makes use of if conditional statement. Though this statement will be discussed later in a separate chapter, but in short, we use if statement to check a condition and if the condition is true, then the body of if statement is executed, otherwise the body of if statement is skipped.

```
#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 the above program is executed, it produces the following result –

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

## ITQ

### Question

What does a programming language provide?

### Feedback

A programming language provides various relational operators.

## 6.5.3 Logical Operators

Logical operators are very important in any programming language and they help us take decisions based on certain conditions. Suppose we want to combine the result of two conditions, then logical AND and OR logical operators help us in producing the final result.

The following table shows all the logical operators supported by the C language. Assume variable A holds 1 and variable B holds 0, then –

| Operator | Description  | Example            |
|----------|--|--------------------|
| &&       | Called Logical AND operator. If both the operands are non-zero, then condition becomes true.   | (A && B) is false. |
|          | Called Logical OR Operator. If any of the two operands is non-zero, then condition becomes true.   | (A    B) is true.  |
| !        | 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. | !(A && B) is true. |

Try the following example to understand all the logical operators available in C programming language –

```
#include <stdio.h>
main() {
    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 compile and execute the above program, it produces the following result –

This will be printed print because condition is true

This will be printed print because condition is true

### ITQ

#### Question

How do logical operators help programmers?

#### Feedback

Logical Operators serves as a reliable guide when taking decisions based on certain conditions.

## 6.5.4 Operators in Java

Following is the equivalent program written in Java. C programming and Java provide almost identical set of operators and conditional statements. This program will create two variables a and b, very similar to C programming, then we assign 10 and 20 in these variables and finally, we will use different arithmetic and relational operators –

You can try to execute the following program to see the output, which must be identical to the result generated by the above example.

```

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" );
}
}
}

```

### ITQ

#### Question

What does Java provide?

#### Feedback

Java provide almost identical set of operators and conditional statements.

## 6.5.5 Operators in Python

Following is the equivalent program written in Python. This program will create two variables a and b and at the same time, assign 10 and 20 in those variables. Fortunately, C programming and Python programming languages provide almost identical set of operators. This program will create two variables a and b, very similar to C programming, then we assign 10 and 20 in these variables and finally, we will use different arithmetic and relational operators.

You can try to execute the following program to see the output, which must be identical to the result generated by the above example.

```

a = 10
b = 20
c = a + b
print "Value of c = ", c
c = a - b
print "Value of c = ", c
c = a * b
print "Value of c = ", c

c = a / b
print "Value of c = ", c

```

```

c = a % b
print "Value of c = ", c
if( a == 10 ):
    print "a is equal to 10"

```

## 6.6 Functions

A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing. You have already seen various functions like `printf()` and `main()`. These are called built-in functions provided by the language itself, but we can write our own functions as well and this tutorial will teach you how to write and use those functions in C programming language.

Good thing about functions is that they are famous with several names. Different programming languages name them differently, for example, functions, methods, sub-routines, procedures, etc. If you come across any such terminology, then just imagine about the same concept, which we are going to discuss in this tutorial.

Let us start with a program where we will define two arrays of numbers and then from each array, we will find the biggest number. Given below are the steps to find out the maximum number from a given set of numbers –

- 1) Get a list of numbers L1, L2, L3...LN
- 2) Assume L1 is the largest, Set max = L1
- 3) Take next number Li from the list and do the following
- 4) If max is less than Li
- 5) Set max = Li
- 6) If Li is last number from the list then
- 7) Print value stored in max and come out
- 8) Else prepeat same process starting from step 3

Let us translate the above program in C programming language –

```

#include <stdio.h>

main() {
    int set1[5] = {10, 20, 30, 40, 50};
    int set2[5] = {101, 201, 301, 401, 501};
    int i, max;
    /* Process first set of numbers available in set1[] */
    max = set1[0];
    i = 1;
    while( i < 5 ) {
        if( max < set1[i] ) {
            max = set1[i];

```

```

    }
    i = i + 1;
}
printf("Max in first set = %d\n", max );
/* Now process second set of numbers available in set2[] */
max = set2[0];
i = 1;
while( i < 5 ) {
    if( max < set2[i] ) {
        max = set2[i];
    }
    i = i + 1;
}
printf("Max in second set = %d\n", max );
}

```

When the above code is compiled and executed, it produces the following result –

Max in first set = 50 Max in second set = 501

If you are clear about the above example, then it will become easy to understand why we need a function. In the above example, there are only two sets of numbers, set1 and set2, but consider a situation where we have 10 or more similar sets of numbers to find out the maximum numbers from each set. In such a situation, we will have to repeat, processing 10 or more times and ultimately, the program will become too large with repeated code. To handle such situation, we write our functions where we try to keep the source code which will be used again and again in our programming.

Now, let's see how to define a function in C programming language and then in the subsequent sections, we will explain how to use them.

### ITQ

#### Question

Define Functions?

#### Feedback

Functions are small units of programs that are used to carry out a specific task. Kindly note that C programming provides various built-in functions like main(), printf(), etc., which are used in programs based on the requirement.

## 6.6.1 Defining a Function

The general form of a function definition in C programming language is as follows –

```
return_type function_name( parameter list ) {    body of the function
return [expression]; }
```

A function definition in C programming consists of a function header and a function body. Here are all the parts of a function:

1. Return Type – A function may return a value. The return\_type is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the return\_type is the keyword void.
2. Function Name – This is the actual name of the function. The function name and the parameter list together constitute the function signature.
3. Parameter List – A parameter is like a placeholder. When a function is invoked, you pass a value as a parameter. This value is referred to as the actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.
4. Function Body – The function body contains a collection of statements that defines what the function does.

### ITQ

#### Question

List two parts of a function

#### Feedback

The two parts of a function are Return Type, and Function Name.

## 6.6.2 Calling a Function

While creating a C function, you give a definition of what the function has to do. To use a function, you will have to call that function to perform a defined task.

Now, let us write the above example with the help of a function –

```
#include <stdio.h>

int getMax( int set[] ) {
    int i, max;
    max = set[0];
    i = 1;
    while( i < 5 ) {
        if( max < set[i] ) {
            max = set[i];
        }
    }
}
```



```

    }
    i = i + 1;
}
return max;
}
main() {
    int set1[5] = {10, 20, 30, 40, 50};
    int set2[5] = {101, 201, 301, 401, 501};
    int max;
    /* Process first set of numbers available in set1[] */
    max = getMax(set1);
    printf("Max in first set = %d\n", max );
    /* Now process second set of numbers available in set2[] */
    max = getMax(set2);
    printf("Max in second set = %d\n", max );
}

```

When the above code is compiled and executed, it produces the following result –

Max in first set = 50 Max in second set = 501

### ITQ

#### Question

What must you bear in mind while creating a C function?

#### Feedback

You must always remember that while creating a C function, you should give a definition of what the function has to do.

## Study Session Summary



### Summary

In this Study Session, you discussed the basics of computer programming. You started by describing the programming environment. You also highlighted the basic syntax of programming. You examined variables and reserved words. You concluded the session by discuss operators and functions.

---

## Assessment



### Assessment

#### SAQ 6.1 (tests Learning Outcome 6.1)

Identify a program environment?

#### SAQ 6.2 (tests learning outcome 6.2)

Explain the basic syntax of programming?

#### SAQ 6.3 (tests learning outcome 6.3)

Explain Reserved Words?

#### SAQ 6.4 (tests learning outcome 6.4)

Define Operators?

#### SAQ 6.5 (tests learning outcome 6.5)

Define Functions?

#### SAQ 6.6 (tests learning outcome 6.6)

Mention the parts of a function?

---

## Credits



### Reading

This lesson has been adapted from:

Program Structure- <http://ccm.net/contents/317-program-structure>

[https://en.wikipedia.org/wiki/Statement\\_\(computer\\_science\)](https://en.wikipedia.org/wiki/Statement_(computer_science))

<http://www.webopedia.com/TERM/S/statement.html><http://www.webopedia.com/TERM/S/statement.html>

[https://en.wikipedia.org/wiki/Control\\_flow](https://en.wikipedia.org/wiki/Control_flow)[https://en.wikipedia.org/wiki/Control\\_flow](https://en.wikipedia.org/wiki/Control_flow)

Introduction to Problem Solving and Control Statements in Visual Basic 2010 By Paul Deitel and Harvey Deitel Apr 27, 2011

[http://www.teach-ict.com/gcse\\_computing/ocr/216\\_programming/control\\_flow/miniweb/p7.htm](http://www.teach-ict.com/gcse_computing/ocr/216_programming/control_flow/miniweb/p7.htm)

[http://www.tutorialspoint.com/computer\\_programming/](http://www.tutorialspoint.com/computer_programming/)

# Study Session 7

## Data Types

### Introduction

In this study session, you will discuss the different data types. You will describe the C and java data types and python data types. You will then proceed to explaining data types and numbers manipulations. Under which you will examine math operations on numbers, numbers in java and numbers in python. Subsequently, you will discuss data types and character manipulation. You will end the session by describing data types and string manipulations.

### Learning Outcomes



#### Outcomes

When we have studied this session, we should be able to:

- 7.1 *identify* data types
- 7.2 *describe* data types and numbers manipulation
- 7.3 *examine* data types and character manipulation
- 7.4 *explain* data types and string manipulation

### Terminology

|             |  |
|-------------|--|
| <b>Data</b> | A representation of facts or ideas in a formalized manner capable of being communicated or manipulated by some processes |
|-------------|--|

## 7.1 Understanding Data Types

Consider an easy example of adding two whole numbers 10 & 20, which can be done simply as follows:

$$10 + 20$$

Let's take another problem where we want to add two decimal numbers 10.50 & 20.50, which will be written as follows –

$$10.50 + 20.50$$

The two examples are straightforward. Now let's take another example where we want to record student information in a notebook. Here we would like to record the following information –

Name:

Class:

Section:

Age:

Sex:

Now, let's put one student record as per the given requirement –

Name: Zara Ali

Class: 6th

Section: J

Age: 13

Sex: F

The first example dealt with whole numbers, the second example added two decimal numbers, whereas the third example is dealing with a mix of different data. Let's put it as follows:

1. Student name "Zara Ali" is a sequence of characters which is also called a string.
2. Student class "6th" has been represented by a mix of whole number and a string of two characters. Such a mix is called alphanumeric.
3. Student section has been represented by a single character which is 'J'.
4. Student age has been represented by a whole number which is 13.
5. Student sex has been represented by a single character which is 'F'.

This way, we realized that in our day-to-day life, we deal with different types of data such as strings, characters, whole numbers (integers), and decimal numbers (floating point numbers). Similarly, when we write a computer program to process different types of data, we need to specify its type clearly; otherwise the computer does not understand how different operations can be performed on that given data. Different programming languages use different keywords to specify different data types. For example, C and Java programming languages use int to specify integer data, whereas char specifies a character data type.

Subsequent sections will show you how to use different data types in different situations. For now, let's check the important data types available in C, Java, and Python and the keywords we will use to specify those data types.

## ITQ

### Question

What must we specify when writing a computer program?

### Feedback

Whenever you are to write a computer program to process different types of data, you need to specify its type clearly; otherwise the computer does not understand how different operations should be

performed on that given data.

### 7.1.1 C and Java Data Types

C and Java support almost the same set of data types, though Java supports additional data types. For now, we are taking a few common data types supported by both the programming languages –

| Type           | Keyword | Value range which can be represented by this data type |
|----------------|---------|--|
| Character      | Char    | -128 to 127 or 0 to 255                                |
| Number         | int     | -32,768 to 32,767 or -2,147,483,648 to 2,147,483,647   |
| Small Number   | short   | -32,768 to 32,767                                      |
| Long Number    | long    | -2,147,483,648 to 2,147,483,647                        |
| Decimal Number | float   | 1.2E-38 to 3.4E+38 till 6 decimal places               |

These data types are called primitive data types and you can use these data types to build more complex data types, which are called user-defined data type, for example a string will be a sequence of characters.

#### ITQ

##### Question

What are called primitive data types?

##### Feedback

The following are referred to as primitive data types: Character, Number, Small Number, Long Number, and Decimal Number.

### 7.1.2 Python Data Types

Python has five standard data types but this programming language does not make use of any keyword to specify a particular data type, rather Python is intelligent enough to understand a given data type automatically.

1. Numbers
2. String
3. List

4. Tuple
5. Dictionary

Here, Number specifies all types of numbers including decimal numbers and string represents a sequence of characters with a length of 1 or more characters. For now, let's proceed with these two data types and skip List, Tuple, and Dictionary, which are advanced data types in Python.

## 7.2 Data Type and Numbers Manipulation

Every programming language provides support for manipulating different types of numbers such as simple whole integers and floating-point numbers. C, Java, and Python categorize these numbers in several categories based on their nature.

Let us go back and check the data types chapter, where we listed down the core data types related to numbers –

| Type           | Keyword | Value range which can be represented by this data type |
|----------------|---------|--|
| Number         | int     | -32,768 to 32,767 or -2,147,483,648 to 2,147,483,647   |
| Small Number   | short   | -32,768 to 32,767                                      |
| Long Number    | long    | -2,147,483,648 to 2,147,483,647                        |
| Decimal Number | float   | 1.2E-38 to 3.4E+38 till 6 decimal places               |

These data types are called primitive data types and you can use these data types to build more data types, which are called user-defined data types.

We have seen various mathematical and logical operations on numbers during a discussion on operators. So we know how to add numbers, subtract numbers, divide numbers, etc.

**ITQ****Question**

What must every programming language provide?

**Feedback**

Every programming language should provide support for manipulating different types of numbers such as simple whole integers, and floating point numbers.

## 7.2.1 Math Operations on Numbers

The following table lists down various useful built-in mathematical functions available in C programming language which can be used for various important mathematical calculations. For example, if you want to calculate the square root of a number, for example, 2304, then you have a built-in function available to calculate the square root.

| S.N. | Function & Purpose   |
|------|--|
| 1    | double cos(double);<br>This function takes an angle (as a double) and returns the cosine.  |
| 2    | double sin(double);<br>This function takes an angle (as a double) and returns the sine.  |
| 3    | double tan(double);<br>This function takes an angle (as a double) and returns the tangent.   |
| 4    | double log(double);<br>This function takes a number and returns the natural log of that number.  |
| 5    | double pow(double, double);<br>The first is a number you wish to raise and the second is the power you wish to raise it to.                          |
| 6    | double hypot(double, double);<br>If you pass this function the length of two sides of a right triangle, it will return the length of the hypotenuse. |

|    |   |
|----|---|
| 7  | <pre>double sqrt(double);</pre> <p>You pass this function a number and it returns its square root.</p>              |
| 8  | <pre>int abs(int);</pre> <p>This function returns the absolute value of an integer that is passed to it.</p>        |
| 9  | <pre>double fabs(double);</pre> <p>This function returns the absolute value of any decimal number passed to it.</p> |
| 10 | <pre>double floor(double);</pre> <p>Finds the integer which is less than or equal to the argument passed to it.</p> |

Following is a simple example to show a few mathematical operations. To utilize these functions, you need to include the math header file `<math.h>` in your program in the same way you included `stdio.h` –

```
#include <stdio.h>
#include <math.h>
main() {
    short s;
    int i;
    long l;
    float f;
    double d;
    s = 10;
    i = 1000;
    l = 1000000;
    f = 230.47;
    d = 2.374;
    printf( "sin(s): %f\n", sin(s));
    printf( "abs(i): %f\n", abs(i));
    printf( "floor(f): %f\n", floor(f));
    printf( "sqrt(f): %f\n", sqrt(f));
    printf( "pow(d, 2): %f\n", pow(d, 2));
}
```



When the above program is executed, it produces the following result –

```
sin(s): -0.544021
abs(i): -0.544021
floor(f): 230.000000
sqrt(f): 15.181238
pow(d, 2): 5.635876
```

Besides the above usage, you will use numbers in loop counting, flag representation, true or false values in C programming.

## 7.2.2 Numbers in Java

Following is the equivalent program written in Java. Java provides almost all the numeric data types available in C programming.

You can try to execute the following program to see the output, which is identical to the result generated by the above C example.

```
public class DemoJava {
    public static void main(String []args) {
        short s;
        int i;
        long l;
        float f;
        double d;
        s = 10;
        i = 1000;
        l = 1000000L;
        f = 230.47f;
        d = 30949.374;
        System.out.format( "s: %d\n", s);
        System.out.format( "i: %d\n", i);
        System.out.format( "l: %d\n", l);
        System.out.format( "f: %f\n", f);
        System.out.format( "d: %f\n", d);
    }
}
```

When the above program is executed, it produces the following result –

```
s: 10
i: 1000
l: 1000000
```

f: 230.470001

d: 30949.374000

Java also provides a full range of built-in functions for mathematical calculation and you can use them in the same way as you did in C programming.

## ITQ

### Question

What additional feature does Java provide?

### Feedback

Java also provides a full range of built-in functions for mathematical calculation.

## 7.2.3 Numbers in Python

Python is a little different from C and Java; it categorizes numbers in int, long, float and complex. Here are some examples of numbers in Python:

| Int    | Long                  | Float      | complex    |
|--------|-----------------------|------------|------------|
| 10     | 51924361L             | 0.0        | 3.14j      |
| 100    | -0x19323L             | 15.20      | 45.j       |
| -786   | 0122L                 | -21.9      | 9.322e-36j |
| 080    | 0xDEFABCECBDAECBFBAE1 | 32.3+e18   | .876j      |
| -0490  | 535633629843L         | -90.       | -.6545+0J  |
| -0x260 | -052318172735L        | -32.54e100 | 3e+26J     |
| 0x69   | -4721885298529L       | 70.2-E12   | 4.53e-7j   |

Following is the equivalent program written in Python –

s = 10

i = 1000

l = 1000000

f = 230.47

```

d = 30949.374
print "s: ", s
print "i: ", i
print "l: ", l
print "f: ", f
print "d: ", d

```

When the above program is executed, it produces the following result –

```

s: 10
i: 1000
l: 1000000
f: 230.47
d: 30949.374

```

Python also provides a full range of built-in functions for mathematical calculations and you can use them in the same way you have used them in C programming.

## 7.3 Data Type and character manipulation

If it was easy to work with numbers in computer programming, it would be even easier to work with characters. Characters are simple alphabets like a, b, c, d,...., A, B, C, D,....., but with an exception. In computer programming, any single digit number like 0, 1, 2,....and special characters like \$, %, +, -.... etc., are also treated as characters and to assign them in a character type variable, you simply need to put them inside single quotes. For example, the following statement defines a character type variable ch and we assign a value 'a' to it –

```
char ch = 'a';
```

Here, ch is a variable of character type which can hold a character of the implementation's character set and 'a' is called a character literal or a character constant. Not only a, b, c,.... but when any number like 1, 2, 3.... or any special character like !, @, #, #, \$,.... is kept inside single quotes, then they will be treated as a character literal and can be assigned to a variable of character type, so the following is a valid statement –

```
char ch = '1';
```

A character data type consumes 8 bits of memory which means you can store anything in a character whose ASCII value lies in between -127 to 127, so it can hold any of the 256 different values. A character data type can store any of the characters available on your keyboard including special characters like !, @, #, #, \$, %, ^, &, \*, (, ), \_, +, {, }, etc.

Note that you can keep only a single alphabet or a single digit number inside single quotes and more than one alphabets or digits are not allowed inside single quotes. So the following statements are invalid in C programming –

```
char ch1 = 'ab';
char ch2 = '10';
```

Given below is a simple example, which shows how to define, assign, and print characters in C Programming language –

```
#include <stdio.h>
main() {
    char ch1;
    char ch2;
    char ch3;
    char ch4;
    ch1 = 'a';
    ch2 = '1';
    ch3 = '$';
    ch4 = '+';
    printf( "ch1: %c\n", ch1);
    printf( "ch2: %c\n", ch2);
    printf( "ch3: %c\n", ch3);
    printf( "ch4: %c\n", ch4);
}
```

Here, we used %c to print a character data type. When the above program is executed, it produces the following result –

```
ch1: a
ch2: 1
ch3: $
ch4: +
```

## ITQ

### Question

How many bits of memory does a character Data type consumes?

### Feedback

A character data type consumes 8 bits of memory, which means you can store anything in a character whose ASCII value lies in between -127 to 127, so it can hold any of the 256 different values.

## 7.3.1 Escape Sequences

Many programming languages support a concept called Escape Sequence. When a character is preceded by a backslash (\), it is called an escape sequence and it has a special meaning to the compiler. For example, \n in

the following statement is a valid character and it is called a new line character –

```
char ch = '\n';
```

Here, character n has been preceded by a backslash (\), it has special meaning which is a new line but keep in mind that backslash (\) has special meaning with a few characters only. The following statement will not convey any meaning in C programming and it will be assumed as an invalid statement –

```
char ch = '\1';
```

The following table lists the escape sequences available in C programming language –

| Escape Sequence | Description   |
|-----------------|---|
| \t              | Inserts a tab in the text at this point.                    |
| \b              | Inserts a backspace in the text at this point.              |
| \n              | Inserts a newline in the text at this point.                |
| \r              | Inserts a carriage return in the text at this point.        |
| \f              | Inserts a form feed in the text at this point.              |
| \'              | Inserts a single quote character in the text at this point. |
| \"              | Inserts a double quote character in the text at this point. |
| \\              | Inserts a backslash character in the text at this point.    |

The following example shows how the compiler interprets an escape sequence in a print statement –

```
#include <stdio.h>
main() {
    char ch1;
    char ch2;
    char ch3;
```

```

char ch4;
ch1 = '\t';
ch2 = '\n';
printf( "Test for tabspace %c and a newline %c will start here", ch1,
ch2);
}

```

When the above program is executed, it produces the following result –  
 Test for tabspace    and a newline  
 will start here

## ITQ

### Question

What concept does many programming support?

### Feedback

Many programming languages support a concept called Escape Sequence

## 7.3.2 Characters in Java

Following is the equivalent program written in Java. Java handles character data types much in the same way as we have seen in C programming. However, Java provides additional support for character manipulation.

You can try to execute the following program to see the output, which must be identical to the result generated by the above C example.

```

public class DemoJava {
    public static void main(String []args) {
        char ch1;
        char ch2;
        char ch3;
        char ch4;
        ch1 = 'a';
        ch2 = '1';
        ch3 = '$';
        ch4 = '+';
        System.out.format( "ch1: %c\n", ch1);
        System.out.format( "ch2: %c\n", ch2);
        System.out.format( "ch3: %c\n", ch3);
        System.out.format( "ch4: %c\n", ch4);
    }
}

```

```

    }
}

```

When the above program is executed, it produces the following result –

```

ch1: a
ch2: 1
ch3: $
ch4: +

```

Java also supports escape sequence in the same way you have used them in C programming.

### ITQ

#### Question

How does Java handle character data types?

#### Feedback

Java handles character data types much in the same way as we have seen in C programming. However, Java provides additional support for character manipulation.

### 7.3.3 Characters in Python

Python does not support any character data type but all the characters are treated as string, which is a sequence of characters. We will study strings in a separate chapter. You do not need to have any special arrangement while using a single character in Python.

Following is the equivalent program written in Python –

```

ch1 = 'a';
ch2 = '1';
ch3 = '$';
ch4 = '+';

print "ch1: ", ch1
print "ch2: ", ch2
print "ch3: ", ch3
print "ch4: ", ch4

```

When the above program is executed, it produces the following result –

```

ch1: a
ch2: 1
ch3: $
ch4: +

```

Python supports escape sequences in the same way as you have used them in C programming.

**ITQ****Question**

How does Python handle character data types?

**Feedback**

Python does not support any character data type but all the characters are treated as string, which is a sequence of characters.

## 7.4 Data Types and String Manipulation

During our discussion about characters, we learnt that character data type deals with a single character and you can assign any character from your keyboard to a character type variable.

Now, let's move a little bit ahead and consider a situation where we need to store more than one character in a variable. We have seen that C programming does not allow to store more than one character in a character type variable. So the following statements are invalid in C programming and produce syntax errors –

```
char ch1 = 'ab';
```

```
char ch2 = '10';
```

We have also seen how to use the concept of arrays to store more than one value of similar data type in a variable. Here is the syntax to store and print five numbers in an array of int type –

```
#include <stdio.h>
main() {
    int number[5] = {10, 20, 30, 40, 50};
    int i = 0;
    while( i < 5 ) {
        printf("number[%d] = %d\n", i, number[i] );
        i = i + 1;
    }
}
```

When the above code is compiled and executed, it produces the following result –

```
number[0] = 10
```

```
number[1] = 20
```

```
number[2] = 30
```

```
number[3] = 40
```

```
number[4] = 50
```



Now, let's define an array of five characters in the same way as we did for numbers and try to print them –

```
#include <stdio.h>
main() {
    char ch[5] = {'H', 'e', 'l', 'l', 'o'};
    int i = 0;
    while( i < 5 ) {
        printf("ch[%d] = %c\n", i, ch[i] );
        i = i + 1;
    }
}
```

Here, we used %c to print character value. When the above code is compiled and executed, it produces the following result –

```
ch[0] = H
ch[1] = e
ch[2] = l
ch[3] = l
ch[4] = o
```

If you are done with the above example, then I think you understood how strings work in C programming, because strings in C are represented as arrays of characters. C programming simplified the assignment and printing of strings. Let's check the same example once again with a simplified syntax –

```
#include <stdio.h>
main() {
    char ch[5] = "Hello";
    int i = 0
    /* Print as a complete string */
    printf("String = %s\n", ch);
    /* Print character by character */
    while( i < 5 ) {
        printf("ch[%d] = %c\n", i, ch[i] );
        i = i + 1;
    }
}
```

Here, we used %s to print the full string value using array name ch, which is actually the beginning of the memory address holding ch variable as shown below –

|          |         |         |         |         |         |         |
|----------|---------|---------|---------|---------|---------|---------|
| Index    | 0       | 1       | 2       | 3       | 4       | 5       |
| Variable | H       | e       | l       | l       | o       | \0      |
| Address  | 0x23451 | 0x23452 | 0x23453 | 0x23454 | 0x23455 | 0x23456 |

Although it's not visible from the above examples, a C program internally assigns null character '\0' as the last character of every string. It indicates the end of the string and it means if you want to store a 5 character string in an array, then you must define an array size of 6 as a good practice, though C does not complain about it.

If the above code is compiled and executed, it produces the following result –

String = Hello

ch[0] = H

ch[1] = e

ch[2] = l

ch[3] = l

ch[4] = o

## ITQ

### Question

What must you remember with reference to data types and strings manipulation?

### Feedback

You need to remember that character data type deals with a single character and you can assign any character from your keyboard to a character type variable.

## 7.4.1 Basic String Concepts

Based on the above discussion, we can conclude the following important points about strings in C programming language:

1. Strings in C are represented as arrays of characters.
2. We can constitute a string in C programming by assigning character by character into an array of characters.
3. We can constitute a string in C programming by assigning a complete string enclosed in double quote.
4. We can print a string character by character using an array subscript or a complete string by using an array name without subscript.
5. The last character of every string is a null character, i.e., '\0'.

6. Most of the programming languages provide built-in functions to manipulate strings, i.e., you can concatenate strings, you can search from a string, you can extract sub-strings from a string, etc. For more, you can check our detailed tutorial on C programming or any other programming language.

### 7.4.2 Strings in Java

Though, you can use character arrays to store strings, but Java is an advanced programming language and its designers tried to provide additional functionality. Java provides strings as a built-in data type like any other data type. It means you can define strings directly instead of defining them as array of characters.

Following is the equivalent program written in Java. Java makes use of the new operator to create string variables as shown in the following program.

You can try to execute the following program to see the output –

```
public class DemoJava {
    public static void main(String []args) {
        String str = new String("Hello");
        System.out.println( "String = " + str );
    }
}
```

When the above program is executed, it produces the following result –

```
String = Hello
```

### 7.4.3 Strings in Python

Creating strings in Python is as simple as assigning a string into a Python variable using single or double quotes.

Given below is a simple program that creates two strings and prints them using print() function –

```
var1 = 'Hello World!'
var2 = "Python Programming"
print "var1 = ", var1
print "var2 = ", var2
```

When the above program is executed, it produces the following result –

```
var1 = Hello World!
var2 = Python Programming
```

Python does not support character type; these are treated as strings of length one, thus also considered a substring.

To access substrings, use the square brackets for slicing along with the index or indices to obtain your substring. Take a look at the following code segment –

```
var1 = 'Hello World!'
var2 = "Python Programming"
print "var1[0]: ", var1[0]
print "var2[1:5]: ", var2[1:5]
```

When the above code is executed, it produces the following result –

```
var1[0]: H
var2[1:5]: ytho
```

### ITQ

#### Question

Mention two conclusions that we can deduce from the Basic String Concept

#### Feedback

From the aforementioned, we can deduce that

1. Strings in C are represented as arrays of characters.
2. We can constitute a string in C programming by assigning character by character into an array of characters.

## Study Session Summary



### Summary

In this Study Session, you examined how to deal with data types, and how to manipulate number data types (integer and float/real). In addition, you explored character manipulation and string data types manipulation. You concluded the session by describing data types in python and gave examples of programs on data type manipulation.

## Assessment



### Assessment

#### SAQ 7.1 (tests Learning Outcome 7.1)

1. Discuss Data types?
2. Discuss C and Java Data types?
3. Discuss about Python Data types?

#### SAQ 7.2 (tests learning outcomes 7.2)

Discuss data types, numbers, character, and string manipulation?

#### SAQ 7.3 (tests learning outcome 7.3)

List the Basic Strings Concept that you know?

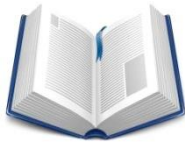
#### SAQ 7.4 (tests learning outcome 7.4)

1. What is important about Strings in Java?

2. What is important about strings in Python?

---

## Credits



### Reading

This content of this lesson is adapted from <http://www.tutorialspoint.com/>

## Study Session 8

# Decision-Making and Loops

## Introduction

In this study session, you will discuss the concept of decision-making and loops. You will start the session by examining conditional statement. Under which you will examine the *if* statement. You will end the session by exploring loops. Here, you will look at the *while loop*, *do-while-loop*, *loops in java* and *loops in python*.

## Learning Outcomes



### Outcomes

When you have studied this session, you should be able to:

- 8.1 *identify* Conditional statements and their usefulness in programming
- 8.2 *explain* and implement loops in programming

## Terminology

|               |   |
|---------------|---|
| <b>Loop</b>   | A programmed sequence of instructions that is repeated until or while a particular condition is satisfied |
| <b>Switch</b> | A programming construct that takes different actions depending on the value of an expression              |

## 8.1 Conditional Statements

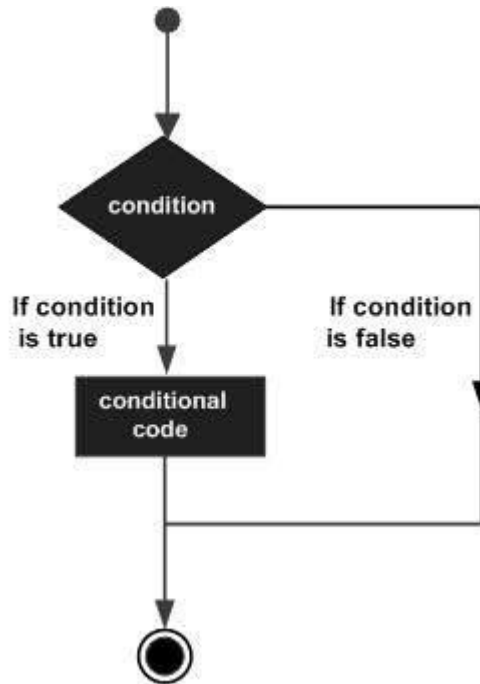
A conditional statement is a mechanism that allows for conditional execution of instructions based upon the outcome of a conditional statement, which can either be true or false.

### 8.1.1 The if Statement

Assuming we want to print a remark about a student based on his secured marks. Following is the situation:

1. Assume given marks are  $x$  for a student:
2. If given marks are more than 95, then
3. Student is brilliant
4. If given marks are less than 30, then
5. Student is poor
6. If given marks are less than 95 and more than 30, then
7. Student is average

Now, the question is how to write a programming code to handle such situations. Almost all the programming languages provide conditional statements that work based on the following flow diagram –



Let's write a C program with the help of if conditional statements to convert the above given situation into a programming code –

```

#include <stdio.h>
main() {
    int x = 45;
    if( x > 95) {
        printf( "Student is brilliant\n");
    }
    if( x < 30) {
        printf( "Student is poor\n");
    }
    if( x < 95 && x > 30 ) {
        printf( "Student is average\n");
    }
}
  
```

When the above program is executed, it produces the following result –

Student is average

The above program uses if conditional statements. Here, the first if statement checks whether the given condition i.e., variable x is greater

than 95 or not and if it finds the condition is true, then the conditional body is entered to execute the given statements. Here we have only one `printf()` statement to print a remark about the student.

Similarly, the second if statement works. Finally, the third if statement is executed, here we have the following two conditions –

1. First condition is  $x > 95$
2. Second condition is  $x < 30$

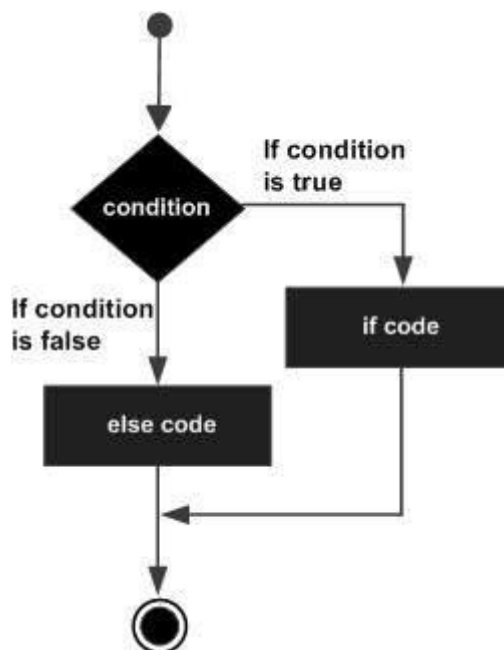
The computer evaluates both the given conditions and then, the overall result is combined with the help of the binary operator `&&`. If the final result is true, then the conditional statement will be executed, otherwise no statement will be executed.

### 8.1.2 if...else statement

An if statement can be followed by an optional else statement, which executes when the Boolean expression is false. The syntax of an if...else statement in C programming language is –

```
if(boolean_expression) {
    /* Statement(s) will execute if the boolean expression is true */
}
else {
    /* Statement(s) will execute if the boolean expression is false */
}
```

The above syntax can be represented in the form of a flow diagram as shown below –





An if...else statement is useful when we have to take a decision out of two options. For example, if a student secures more marks than 95, then the student is brilliant, otherwise no such situation can be coded, as follows –

```
#include <stdio.h>
main() {
    int x = 45;
    if( x > 95) {
        printf( "Student is brilliant\n");
    }else {
        printf( "Student is not brilliant\n");
    }
}
```

When the above program is executed, it produces the following result –

Student is not brilliant

## ITQ

### Question

What follows an 'if..else statement'?

### Feedback

An Optional Else Statement

## 8.1.3 if...else if...else statement

An if statement can be followed by an optional else if...else statement, which is very useful to test various conditions.

While using if, else if, else statements, there are a few points to keep in mind:

1. An if can have zero or one else's and it must come after an else if.
2. An if can have zero to many else...if's and they must come before the else.
3. Once an else...if succeeds, none of the remaining else...if's or else's will be tested.

The syntax of an if...else if...else statement in C programming language is –

```
if(boolean_expression 1) {
    /* Executes when the boolean expression 1 is true */
}
else if( boolean_expression 2) {
    /* Executes when the boolean expression 2 is true */
}
```

```

else if( boolean_expression 3) {
    /* Executes when the boolean expression 3 is true */
}
else {
    /* Executes when the none of the above condition is true */
}

```

Now with the help of if...elseif...else statement, the very first program can be coded as follows –

```

#include <stdio.h>
main() {
    int x = 45;
    if( x > 95) {
        printf( "Student is brilliant\n");
    }else if( x < 30) {
        printf( "Student is poor\n");
    }else if( x < 95 && x > 30 ) {
        printf( "Student is average\n");
    }
}

```

When the above program is executed, it produces the following result –

Student is average

## ITQ

### Question

What follows an ‘If’ statement?

### Feedback

An if statement can be followed by an optional else if...else statement.

## 8.1.4 The Switch Statement

A switch statement is an alternative of if statements which allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each switch case. It has the following syntax –

```

switch(expression){
    case ONE :
        statement(s);
        break;

```

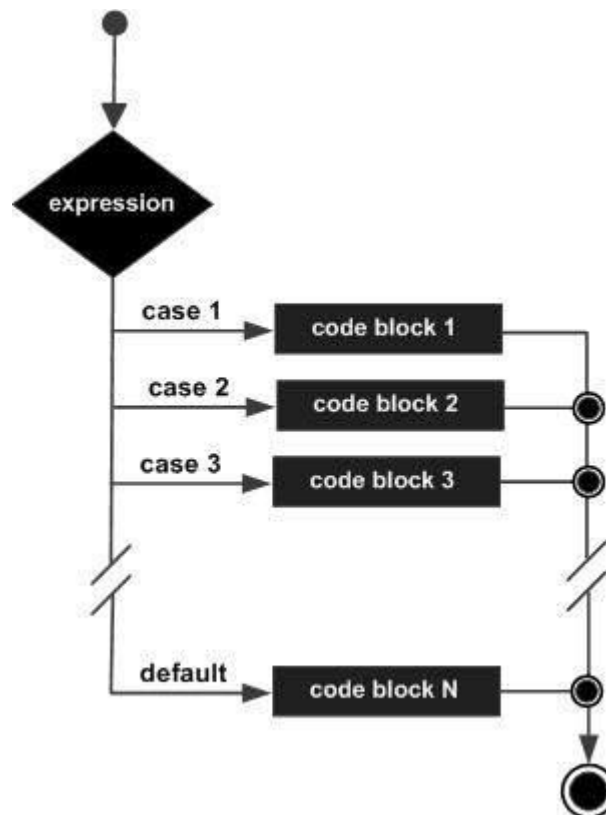
```

case TWO:
    statement(s);
    break;
.....
default :
    statement(s);
}

```

The expression used in a switch statement must give an integer value, which will be compared for equality with different cases given. Wherever an expression value matches with a case value, the body of that case will be executed and finally, the switch will be terminated using a break statement. If no break statements are provided, then the computer continues executing other statements available below to the matched case. If none of the cases matches, then the default case body is executed.

The above syntax can be represented in the form of a flow diagram as shown below –



Now, let's consider another example where we want to write the equivalent English word for a given number. Then, it can be coded as follows –

```

#include <stdio.h>
main() {
    int x = 2;

```

```

switch( x ){
    case 1 :
        printf( "One\n");
        break;
    case 2 :
        printf( "Two\n");
        break;
    case 3 :
        printf( "Three\n");
        break;
    case 4 :
        printf( "Four\n");
        break;
    default :
        printf( "None of the above...\n");
}
}

```

When the above program is executed, it produces the following result –

Two

### ITQ

#### Question

What are Switch Statement?

#### Feedback

A switch statement is an alternative of if statements which allows a variable to be tested for equality against a list of values.

## 8.1.5 Decisions in Java

The following is the equivalent program written in Java which too supports if,if...else, if...elseif...else, and switch statements.

You can try to execute the following program to see the output, which must be identical to the result generated by the above C example.

```

public class DemoJava {
    public static void main(String []args) {
        int x = 45;
        if( x > 95) {
            System.out.println( "Student is brilliant");

```

```

}else if( x < 30) {
    System.out.println( "Student is poor");
}else if( x < 95 && x > 30 ) {
    System.out.println( "Student is average");
}
}
}

```

### 8.1.6 Decisions in Python

Following is the equivalent program written in Python. Python provides if,if...else, if...elif...else, and switch statements. Here, you must note that Python does not make use of curly braces for conditional body, instead it simply identifies the body of the block using indentation of the statements.

You can try to execute the following program to see the output –

```

x = 45
if x > 95:
    print "Student is brilliant"
elif x < 30:
    print "Student is poor"
elif x < 95 and x > 30:
    print "Student is average"
print "The end"

```

When the above program is executed, it produces the following result –

Student is average

The end

#### ITQ

##### Question

What does Python provide?

##### Feedback

Python provides if,if...else, if...elif...else, and switch statements

## 8.2 Loops

Let us consider a situation when you want to print Hello, World! five times. Here is a simple C program to do the same –

```

#include <stdio.h>

main() {

```

```
printf( "Hello, World!\n");  
printf( "Hello, World!\n");  
printf( "Hello, World!\n");  
printf( "Hello, World!\n");  
printf( "Hello, World!\n");  
}
```

When the above program is executed, it produces the following result –

Hello, World!

Hello, World!

Hello, World!

Hello, World!

Hello, World!

It was simple, but again, let's consider another situation when you want to write Hello, World! a thousand times. We can certainly not write printf() statements a thousand times. Almost all the programming languages provide a concept called loop, which helps in executing one or more statements up to a desired number of times. All high-level programming languages provide various forms of loops, which can be used to execute one or more statements repeatedly.

Let's write the above C program with the help of a while loop and later, we will discuss how this loop works

```
#include <stdio.h>  
  
main() {  
    int i = 0;  
    while ( i < 5 ) {  
        printf( "Hello, World!\n");  
        i = i + 1;  
    }  
}
```

When the above program is executed, it produces the following result –

Hello, World!

Hello, World!

Hello, World!

Hello, World!

Hello, World!

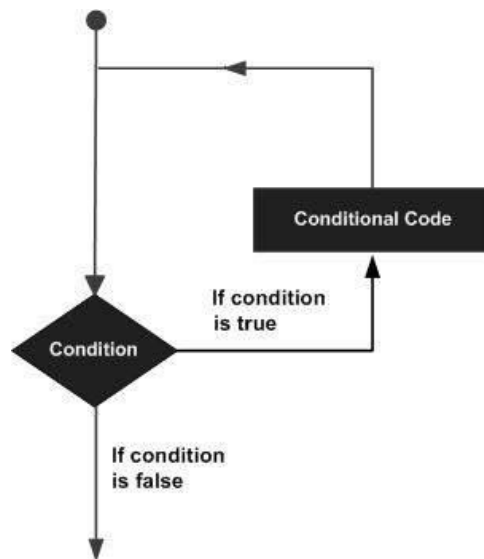
The above program makes use of a while loop, which is being used to execute a set of programming statements enclosed within {...}. Here, the computer first checks whether the given condition, i.e., variable "a" is

less than 5 or not and if it finds the condition is true, then the loop body is entered to execute the given statements. Here, we have the following two statements in the loop body –

1. First statement is printf() function, which prints Hello World!
2. Second statement is  $i = i + 1$ , which is used to increase the value of variable  $i$

After executing all the statements given in the loop body, the computer goes back to while(  $i < 5$  ) and the given condition, (  $i < 5$  ), is checked again, and the loop is executed again if the condition holds true. This process repeats till the given condition remains true which means variable "a" has a value less than 5.

To conclude, a loop statement allows us to execute a statement or group of statements multiple times. Given below is the general form of a loop statement in most of the programming languages –



This lesson has been designed to present programming's basic concepts to non-programmers, so let's discuss the two most important loops available in C programming language. Once you are clear about these two loops, then you can pick-up C programming tutorial or a reference book and check other loops available in C and the way they work.

## ITQ

### Question

How does a loop help a programmer save time?

### Feedback

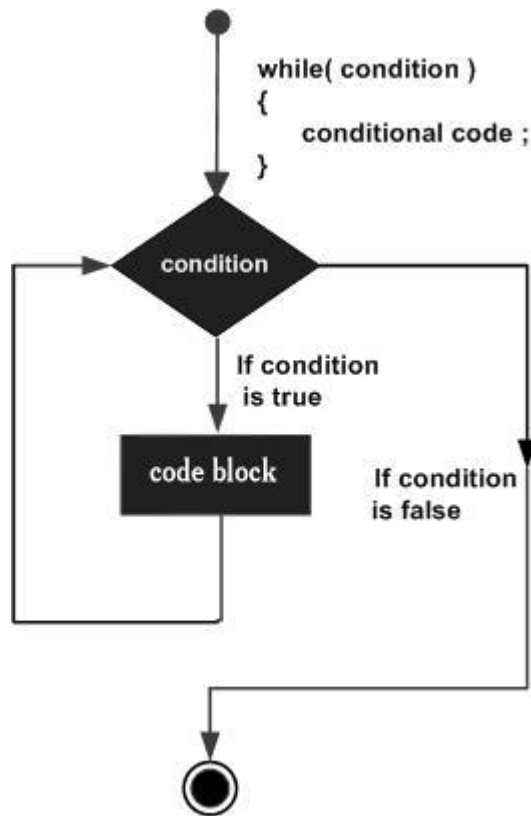
A loop statement allows the programmer to execute a statement or group of statements multiple times.

## 8.2.1 The while Loop

A while loop available in C Programming language has the following syntax –

```
while ( condition ) {
    /*...while loop body ...*/
}
```

The above code can be represented in the form of a flow diagram as shown below –



The following important points are to be noted about a while loop:

1. A while loop starts with a keyword while followed by a condition enclosed in ( ).
2. Further to the while() statement, you will have the body of the loop enclosed in curly braces {...}.
3. A while loop body can have one or more lines of source code to be executed repeatedly.
4. If the body of a while loop has just one line, then its optional to use curly braces {...}.
5. A while loop keeps executing its body till a given condition holds true. As soon as the condition becomes false, the while loop comes out and continues executing from the immediate next statement after the while loop body.
6. A condition is usually a relational statement, which is evaluated to either true or false. A value equal to zero is treated as false and any non-zero value works like true.



**ITQ****Question**

Mention one important to note with respect to While Loops

**Feedback**

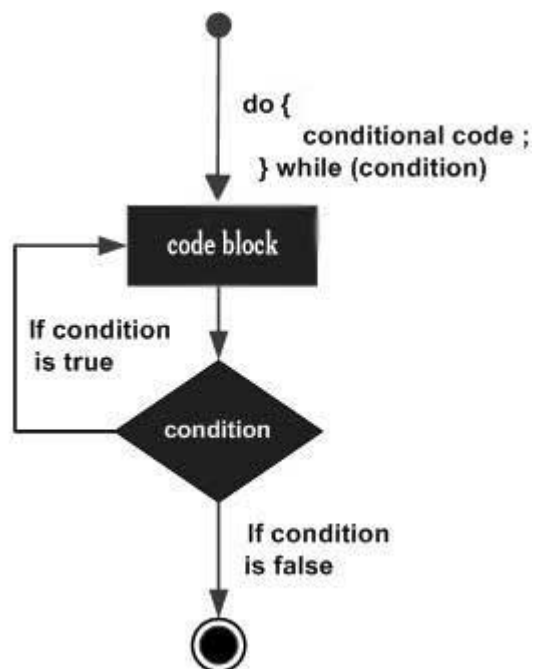
A while loop starts with a keyword while followed by a condition enclosed in ( ).

**8.2.2 The do...while Loop**

A while loop checks a given condition before it executes any statements given in the body part. C programming provides another form of loop, called the do...while that allows to execute a loop body before checking a given condition. It has the following syntax –

```
do {
    /*....do...while loop body ...*/
} while ( condition );
```

The above code can be represented in the form of a flow diagram as shown below –



If you will write the above example using do...while loop, then Hello, World will produce the same result –

```
#include <stdio.h>
main() {
    int i = 0;
    do {
        printf("Hello, World!\n");
```

```
    i = i + 1;  
}while ( i < 5 );  
}
```

When the above program is executed, it produces the following result –

Hello, World!

Hello, World!

Hello, World!

Hello, World!

Hello, World!

### ITQ

#### Question

How does a ‘do...while loop’ function?

#### Feedback

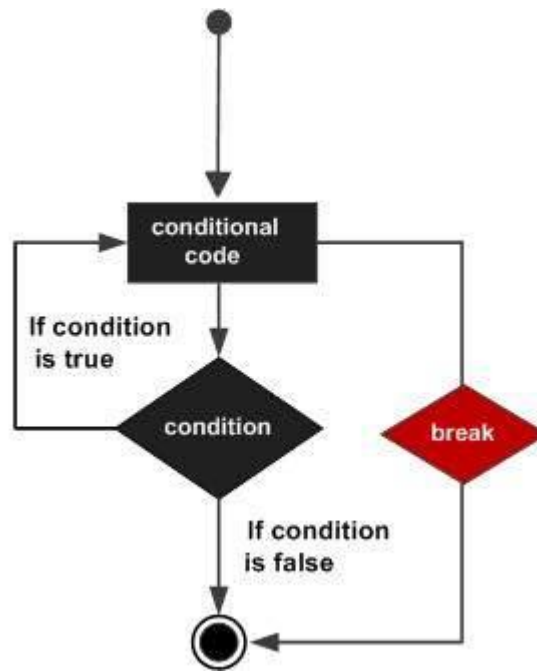
A do...while loop allows to execute a loop body before checking a given condition.

## 8.2.3 The break statement

When the break statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop. The syntax for a break statement in C is as follows:

```
break;
```

A break statement can be represented in the form of a flow diagram as shown below –



Following is a variant of the above program, but it will come out after printing Hello World! only three times –

```

#include <stdio.h>
main() {
    int i = 0;
    do {
        printf( "Hello, World!\n");
        i = i + 1;
        if( i == 3 ) {
            break;
        }
    }while ( i < 5 );
}
  
```

When the above program is executed, it produces the following result –

Hello, World!

Hello, World!

Hello, World!

## ITQ

### Question

What do we encounter in a loop?

### Feedback

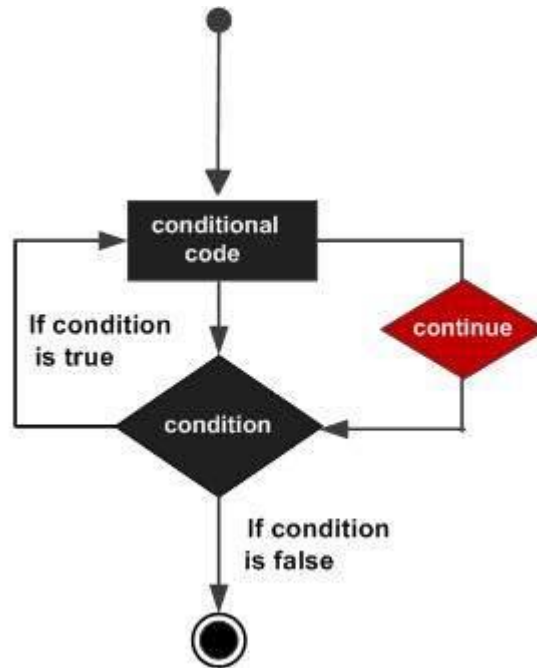
The Break Statement

## 8.2.4 The continue statement

The continue statement in C programming language works somewhat like the break statement. Instead of forcing termination, continue forces the next iteration of the loop to take place, skipping any code in between. The syntax for a continue statement in C is as follows –

```
continue;
```

A continue statement can be represented in the form of a flow diagram as shown below



Following is a variant of the above program, but it will skip printing when the variable has a value equal to 3 –

```
#include <stdio.h>
```

```
main() {

    int i = 0;

    do {

        if( i == 3 ) {

            i = i + 1;
            continue;
        }
    }
}
```

```

printf( "Hello, World!\n");
i = i + 1;
}while ( i < 5 );
}

```

When the above program is executed, it produces the following result –

Hello, World!

Hello, World!

Hello, World!

Hello, World!

## 8.2.5 Loops in Java

Following is the equivalent program written in Java that too supports while and do...while loops. The following program prints Hello, World! five times as we did in the case of C Programming –

You can try to execute the following program to see the output, which must be identical to the result generated by the above example.

```

public class DemoJava {
    public static void main(String []args) {
        int i = 0;
        while ( i < 5 ) {
            System.out.println("Hello, World!");
            i = i + 1;
        }
    }
}

```

The break and continue statements in Java programming work quite the same way as they work in C programming.

### ITQ

#### Question

What is peculiar about the break and continue statements in Java programming?

#### Feedback

The break and continue statements in Java programming work quite the same way as they work in C programming.

## 8.2.6 Loops in Python

Following is the equivalent program written in Python. Python too supports while and do...while loops. The following program prints Hello, World! five times as we did in case of C Programming. Here you must note that Python does not make use of curly braces for the loop body, instead it simply identifies the body of the loop using indentation of the statements.

You can try to execute the following program to see the output. To show the difference, we have used one more print statement, which will be executed when the loop will be over.

```
i = 0
while (i < 5):
    print "Hello, World!"
    i = i + 1
print "Loop ends"
```

When the above program is executed, it produces the following result –

```
Hello, World!
Hello, World!
Hello, World!
Hello, World!
Hello, World!
Loop ends
```

The break and continue statements in Python work quite the same way as they do in C programming.

### ITQ

#### Question

What important point should you bear in mind about loops in python?

#### Feedback

Python does not make use of curly braces for the loop body; instead, it simply identifies the body of the loop using indentation of the statements.

## Study Session Summary



### Summary

In this Study Session, you discussed decision-making and loops, with example of programs in C, Java and Python. You also examined the necessary framework for you to use loops in other languages.

---

## Assessment



### Assessment

#### SAQ 8.1 (tests Learning Outcome 8.1)

1. What are Conditional Statements?

#### SAQ 8.2 (tests learning outcome 8.2)

1. Explain and implement loops in programming?
2. What is peculiar about the 'break' and 'continue' statement in Java Programming?

---

## Credit



### Reading

This content of this lesson is adapted from <http://www.tutorialspoint.com/>

# Study Session 9

## Arrays

### Introduction

In this study session, your focus will be on Arrays. You will begin by explaining what an array is. You will then create, initialize and access arrays. Thereafter, you will examine arrays in java and python.

### Learning Outcomes



#### Outcomes

When you have studied this session, you should be able to:

- 9.1 *explain* the term arrays
- 9.2 *discuss* arrays in java
- 9.3 *describe* arrays (lists) in python.

### Terminology

|       |   |
|-------|---|
| Array | A data structure that contains a group of elements. |
|-------|---|

## 9.1 Understanding Arrays

Consider a situation where we need to store five integer numbers. If we use programming's simple variable and data type concepts, then we need five variables of int data type and the program will be as follows –

```
#include <stdio.h>
main() {
    int number1;
    int number2;
    int number3;
    int number4;
    int number5;
    number1 = 10;
    number2 = 20;
    number3 = 30;
    number4 = 40;
    number5 = 50;
```



```

printf( "number1: %d\n", number1);
printf( "number2: %d\n", number2);
printf( "number3: %d\n", number3);
printf( "number4: %d\n", number4);
printf( "number5: %d\n", number5);
}

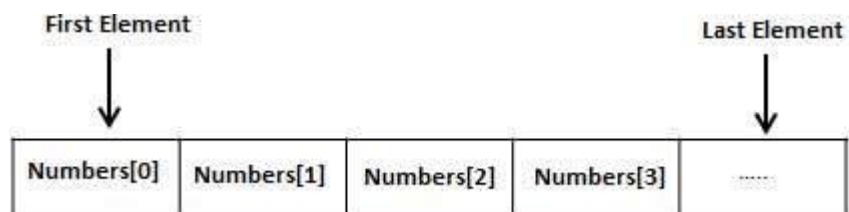
```

It was simple, because we had to store just five integer numbers. Now let's assume we have to store 5000 integer numbers. Are we going to use 5000 variables?

To handle such situations, almost all the programming languages provide a concept called array. An array is a data structure, which can store a fixed-size collection of elements of the same data type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Instead of declaring individual variables, such as `number1`, `number2`, ..., `number99`, you just declare one array variable `number` of integer type and use `number[0]`, `number[1]`, and ..., `number[99]` to represent individual variables. Here, 0, 1, 2, .....99 are index associated with var variable and they are being used to represent individual elements available in the array.

All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.



### 9.1.1 Creating Arrays

To create an array variable in C, a programmer specifies the type of the elements and the number of elements to be stored in that array. Given below is a simple syntax to create an array in C programming:

```
type arrayName [ arraySize ];
```

This is called a single-dimensional array. The array Size must be an integer constant greater than zero and type can be any valid C data type. For example, now to declare a 10-element array called `number` of type `int`, use this statement:

```
int number[10];
```

Here, `number` is a variable array, which is sufficient to hold up to 10 integer numbers.

**ITQ****Question**

1. What do most programming languages provide?
2. What do all arrays consist of?

**Feedback**

1. Arrays
2. All arrays consist of contiguous memory locations.

## 9.1.2 Initializing Arrays

You can initialize an array in C either one by one or using a single statement as follows:

```
int number[5] = {10, 20, 30, 40, 50};
```

The number of values between braces { } cannot be larger than the number of elements that we declare for the array between square brackets [ ].

If you omit the size of the array, an array just big enough to hold the initialization is created. Therefore, if you write –

```
int number[] = {10, 20, 30, 40, 50};
```

You will create exactly the same array as you did in the previous example. Following is an example to assign a single element of the array –

```
number[4] = 50;
```

The above statement assigns element number 5th in the array with a value of 50. All arrays have 0 as the index of their first element which is also called the base index and the last index of an array will be the total size of the array minus 1. The following image shows the pictorial representation of the array we discussed above –

| Index  | 0  | 1  | 2  | 3  | 4  |
|--------|----|----|----|----|----|
| number | 10 | 20 | 30 | 40 | 50 |

## 9.1.3 Accessing Array Elements

An element is accessed by indexing the array name. This is done by placing the index of the element within square brackets after the name of the array. For example –

```
int var = number[9];
```

The above statement will take the 10th element from the array and assign the value to var variable. The following example uses all the above-mentioned three concepts viz. creation, assignment, and accessing arrays –

```
#include <stdio.h>
```

```
int main () {
    int number[10]; /* number is an array of 10 integers */
    int i = 0;
    /* Initialize elements of array n to 0 */
    while( i < 10 ) {
        /* Set element at location i to i + 100 */
        number[ i ] = i + 100;
        i = i + 1;
    }
    /* Output each array element's value */
    i = 0;
    while( i < 10 ) {
        printf("number[%d] = %d\n", i, number[i] );
        i = i + 1;
    }
    return 0;
}
```

When the above code is compiled and executed, it produces the following result –

```
number[0] = 100
number[1] = 101
number[2] = 102
number[3] = 103
number[4] = 104
number[5] = 105
number[6] = 106
number[7] = 107
number[8] = 108
number[9] = 109
```

### ITQ

#### Question

How do we access an element?

#### Feedback

Elements are generally accessed by indexing the names of the array.

## 9.2 Arrays in Java

Following is the equivalent program written in Java. Java supports arrays, but there is a little difference in the way they are created in Java using the new operator.

You can try to execute the following program to see the output, which must be identical to the result generated by the above C example.

```
public class DemoJava {
    public static void main(String []args) {
        int[] number = new int[10];
        int i = 0;
        while( i < 10 ) {
            number[ i ] = i + 100;
            i = i + 1;
        }
        i = 0;
        while( i < 10 ) {
            System.out.format( "number[%d] = %d\n", i, number[i] );
            i = i + 1;
        }
    }
}
```

When the above program is executed, it produces the following result –

```
number[0] = 100
number[1] = 101
number[2] = 102
number[3] = 103
number[4] = 104
number[5] = 105
number[6] = 106
number[7] = 107
number[8] = 108
number[9] = 109
```

**ITQ****Question**

Does Java support arrays?

**Feedback**

Java support arrays though there is a little difference in the way they are created.

## 9.3 Arrays (Lists) in Python

Python does not have a concept of Array, instead Python provides another data structure called list, which provides similar functionality as arrays in any other language.

Following is the equivalent program written in Python –

# Following defines an empty list.

```
number = []
```

```
i = 0
```

```
while i < 10:
```

```
    # Appending elements in the list
```

```
    number.append(i + 100)
```

```
    i = i + 1
```

```
i = 0
```

```
while i < 10:
```

```
    # Accessing elements from the list
```

```
    print "number[" , i, "] = ", number[ i ]
```

```
    i = i + 1
```

When the above program is executed, it produces the following result –

```
number[ 0 ] = 100
```

```
number[ 1 ] = 101
```

```
number[ 2 ] = 102
```

```
number[ 3 ] = 103
```

```
number[ 4 ] = 104
```

```
number[ 5 ] = 105
```

```
number[ 6 ] = 106
```

```
number[ 7 ] = 107
```

```
number[ 8 ] = 108
```

```
number[ 9 ] = 109
```

## ITQ

### Question

Can arrays be created using the Python programming?

### Feedback

Python does not have a concept of Array; instead, Python provides another data structure called LIST, which provides similar functionality as arrays in any other language.

## Study Session Summary



### Summary

You began this session by describing what array means. You also attempted to create, initialize and access arrays. You concluded the session by pointing-out arrays in java and python.

## Assessment



### Assessment

#### SAQ 9.1 (tests learning outcome 9.1)

1. Explain the term array?
2. How do you initialize and access array elements?

#### SAQ 9.2 (tests learning outcome 9.2 and 9.3)

Discuss arrays in Java and Python?

## Credit



### Reading

This content of this lesson is adapted from <http://www.tutorialspoint.com/>

# Study Session 10

## Computer Files

### Introduction

In this study session, you will discuss computer files in relation to programming language. You will also examine the opening, closing, writing and reading of computer files. You will end the session by exploring file I/O in java and file I/O in python.

### Learning Outcomes



#### Outcomes

When you have studied this session, you should be able to

- 10.1 *discuss* file input/output
- 10.2 *explain* operation modes
- 10.3 *examine* file I/O in Java
- 10.4 *describe* file I/O python

### Terminology

|             |  |
|-------------|--|
| <b>File</b> | An aggregation of data on a storage device, identified by a name |
|-------------|--|

## 10.1 File Input/output

Usually, you create files using text editors such as notepad, MS Word, MS Excel or MS PowerPoint, etc. However, many times, we need to create files using computer programs as well. We can modify an existing file using a computer program. File input means data that is written into a file and file output means data that is read from a file. Actually, input and output terms are more related to screen input and output. When we display a result on the screen, it is called output. Similarly, if we provide some input to our program from the command prompt, then it is called input. For now, it is enough to remember that writing into a file is file input and reading something from a file is file output.

**ITQ****Question**

How else can we create files if we decide not to use computer programs?

**Feedback**

Files cannot only be created using computer programs. We could also create files using MS Word, MS PowerPoint, and MS Excel.

## 10.2 File Operation Modes

Before we start working with any file using a computer program, either we need to create a new file if it does not exist or open an already existing file. In either case, we can open a file in the following modes –

1. Read-Only Mode – If you are going to just read an existing file and you do not want to write any further content in the file, then you will open the file in read-only mode. Almost all the programming languages provide syntax to open files in read-only mode.
2. Write-Only Mode – If you are going to write into either an existing file or a newly created file but you do not want to read any written content from that file, then you will open the file in write-only mode. All the programming languages provide syntax to open files in write-only mode.
3. Read & Write Mode – If you are going to read as well as write into the same file, then you will open file in read & write mode.
4. Append Mode – When you open a file for writing, it allows you to start writing from the beginning of the file; however it will overwrite existing content, if any. Suppose we don't want to overwrite any existing content, then we open the file in append mode. Append mode is ultimately a write mode, which allows content to be appended at the end of the file. Almost all the programming languages provide syntax to open files in append mode.

In the following sections, we will learn how to open a fresh new file, how to write into it, and later, how to read and append more content into the same file.

**ITQ****Question**

Which file mode would you select if you were ONLY interested in writing into either an existing file, or a newly created file?

**Feedback**

In such a case as described above, you would open the file in a Write-Only Mode.



## 10.2.1 Opening Files

You can use the `fopen()` function to create a new file or to open an existing file. This call will initialize an object of the type `FILE`, which contains all the information necessary to control the stream. Here is the prototype, i.e., signature of this function call –

```
FILE *fopen( const char * filename, const char * mode );
```

Here, `filename` is string literal, which you will use to name your file and access mode can have one of the following values –

| Mode | Description  |
|------|--|
| R    | Opens an existing text file for reading purpose.   |
| W    | Opens a text file for writing. If it does not exist, then a new file is created. Here, your program will start writing content from the beginning of the file.                   |
| A    | Opens a text file for writing in appending mode. If it does not exist, then a new file is created. Here, your program will start appending content in the existing file content. |
| r+   | Opens a text file for reading and writing both.  |
| w+   | Opens a text file for both reading and writing. It first truncates the file to zero length, if it exists; otherwise creates the file if it does not exist.                       |
| a+   | Opens a text file for both reading and writing. It creates a file, if it does not exist. The reading will start from the beginning, but writing can only be appended.            |

### ITQ

#### Question

How would you open a file?

#### Feedback

Files are often opened using the `fopen()` function.

## 10.2.2 Closing a File

To close a file, use the `fclose( )` function. The prototype of this function is –

```
int fclose( FILE *fp );
```

The `fclose( )` function returns zero on success, or EOF, special character, if there is an error in closing the file. This function actually flushes any data still pending in the buffer to the file, closes the file, and releases any memory used for the file. The EOF is a constant defined in the header file `stdio.h`.

There are various functions provided by C standard library to read and write a file character by character or in the form of a fixed length string. Let us see a few of them in the next section.

### ITQ

#### Question

What is the function for closing a file?

#### Feedback

```
int fclose( FILE *fp );
```

## 10.2.3 Writing a File

Given below is the simplest function to write individual characters to a stream –

```
int fputc( int c, FILE *fp );
```

The function `fputc()` writes the character value of the argument `c` to the output stream referenced by `fp`. It returns the written character on success, otherwise EOF if there is an error. You can use the following functions to write a null-terminated string to a stream –

```
int fputs( const char *s, FILE *fp );
```

The function `fputs()` writes the string `s` into the file referenced by `fp`. It returns a non-negative value on success, otherwise EOF is returned in case of any error. You can also use the function `int fprintf(FILE *fp, const char *format, ...)` to write a string into a file. Try the following example –

```
#include <stdio.h>

main() {
    FILE *fp;
    fp = fopen("/tmp/test.txt", "w+");
    fprintf(fp, "This is testing for fprintf...\n");
    fputs("This is testing for fputs...\n", fp);
    fclose(fp);
}
```

```
}

```

When the above code is compiled and executed, it creates a new file test.txt in /tmp directory and writes two lines using two different functions. Let us read this file in the next section.

## ITQ

### Question

What is the key function in writing a file?

### Feedback

```
int fputc( int c, FILE *fp );
```

## 10.2.4 Reading a File

Given below is the simplest function to read a text file character by character –

```
int fgetc( FILE * fp );
```

The fgetc() function reads a character from the input file referenced by fp. The return value is the character read; or in case of any error, it returns EOF. The following function allows you to read a string from a stream –

```
char *fgets( char *buf, int n, FILE *fp );
```

The function fgets() reads up to n - 1 characters from the input stream referenced by fp. It copies the read string into the buffer buf, appending a null character to terminate the string.

If this function encounters a newline character '\n' or EOF before they have read the maximum number of characters, then it returns only the characters read up to that point including the new line character. You can also use int fscanf(FILE \*fp, const char \*format, ...) to read strings from a file, but it stops reading after encountering the first space character.

```
#include <stdio.h>
```

```
main() {
    FILE *fp;
    char buff[255];
    fp = fopen("/tmp/test.txt", "r");
    fscanf(fp, "%s", buff);
    printf("1 : %s\n", buff );
    fgets(buff, 255, (FILE*)fp);
    printf("2: %s\n", buff );
    fgets(buff, 255, (FILE*)fp);
    printf("3: %s\n", buff );
    fclose(fp);
}
```

}When the above code is compiled and executed, it reads the file created in the previous section and produces the following result –

```
1 : This 2: is testing for fprintf... 3: This is testing for fputs...
```

Let's analyze what happened here. First, the `fscanf()` method reads This because after that, it encountered a space. The second call is for `fgets()`, which reads the remaining line till it encountered end of line. Finally, the last call `fgets()` reads the second line completely.

## ITQ

### Question

What is the key function in reading a file?

### Feedback

```
int fgetc( FILE * fp );
```

## 10.3 File I/O in Java

Java provides even richer set of functions to handle File I/O. For more on this topic, we suggest you to check our Java Tutorials.

Here, we will see a simple Java program, which is equivalent to the C program explained above. This program will open a text file, write a few text lines into it, and close the file. Finally, the same file is opened and then read from an already created file. You can try to execute the following program to see the output –

```
import java.io.*

public class DemoJava {
    public static void main(String []args) throws IOException {
        File file = new File("/tmp/java.txt");
        // Create a File
        file.createNewFile();
        // Creates a FileWriter Object using file object
        FileWriter writer = new FileWriter(file);
        // Writes the content to the file
        writer.write("This is testing for Java write...\n");
        writer.write("This is second line...\n");
        // Flush the memory and close the file
        writer.flush();
        writer.close();
        // Creates a FileReader Object
```

```

        FileReader reader = new FileReader(file);
        char [] a = new char[100];
        // Read file content in the array
        reader.read(a);
        System.out.println( a );
        // Close the file
        reader.close();
    }
}

```

When the above program is executed, it produces the following result –  
This is testing for Java write... This is second line...

## 10.4 File I/O in Python

The following program shows the same functionality to open a new file, write some content into it, and finally, read the same file –

```

# Create a new file
fo = open("/tmp/python.txt", "w")
# Writes the content to the file
fo.write( "This is testing for Python write...\n");
fo.write( "This is second line...\n");
# Close the file
fo.close()
# Open existing file
fo = open("/tmp/python.txt", "r")
# Read file content in a variable
str = fo.read(100);
print str
# Close opened file
fo.close()

```

When the above code is executed, it produces the following result –  
This is testing for Python write... This is second line...

## ITQ

### Question

What more can we say about file input/output in Java?

### Feedback

Java provides even richer set of functions to handle File Input/Output.

---

## Study Session Summary



### Summary

In this study session, you discussed computer files in relation to programming language. You also examine the opening, closing, writing and reading of computer files. You finally described file I/O in java and python.

---

## Assessment



### Assessment

#### SAQ 10.1 (tests learning outcome 10.1)

Discuss file Input/Output?

#### SAQ 10.2 (tests learning outcome 10.2)

1. Explain Operation Modes?
2. How would you open/close a file?
3. How would you write/read a file?

---

## Credit



### Reading

This content of this lesson is adapted from <http://www.tutorialspoint.com/>

# Notes on Self Assessment Questions

---

## SAQ 1.1

1. The essence of programming languages is to allow a programmer to manipulate numbers and texts (called variables) in different ways, share them over a network or store them on disks for future retrieval.
2. The essence of programming languages is to allow a programmer to manipulate numbers and texts (called variables) in different ways, share them over a network or store them on disks for future retrieval.

## SAQ 1.2

A computer program is a set of instructions, consisting of sequence of separate commands or instructions, one after the other. Each step tells the computer to perform a specific action.

## SAQ 1.3

1. The core characteristics of a computer program includes portability, readability, efficiency, structural, flexibility, generality, and documentation. Additional features include simplicity, naturalness, abstraction, compactness, and locality.
2. The core characteristics helps the computer obtain appropriate instruction set (programs) needed to perform the required task. The quality of the processing depends upon the given instructions.

## SAQ 1.4

1. A computer programmer, also known as developer, coder, or software engineer is a person who writes computer software. The term computer programmer can refer to a specialist in one area of computer programming or to a generalist who writes code for many kinds of software. A programmer's primary computer language is often prefixed to these titles, and those who work in a Web environment often prefix their titles with Web. As such, based on computer programming language expertise, we can name a computer programmers as follows –
  - i. C Programmer
  - ii. C++ Programmer
  - iii. Java Programmer
  - iv. Python Programmer
  - v. PHP Programmer
  - vi. Perl Programmer
  - vii. Ruby Programmer
2. The function of a computer programmer are:
  - i. A computer programmer figures out the process of designing, writing, testing, debugging/troubleshooting and maintaining the source code of computer programs.

- ii. The computer programmer also designs a graphical user interface (GUI) so that non-technical users can use the software through easy, point-and-click menu options. The GUI acts as a translator between the user and the software code.
- iii. A programmer will also use libraries of basic code that can be modified or customized for a specific application. This approach yields more reliable and consistent programs and increases programmers' productivity by eliminating some routine steps.
- iv. The programmer will also be responsible for maintaining the program's health. As software design advances, some programming functions has become automated, and programmers have begun to assume some of the responsibilities once performed only by software engineers.

### SAQ 2.1

1. The history of programming languages dates back to the 19th century, when Ada Lovelace (1815 – 1852) wrote a set of notes after translating the memoir of Italian mathematician Luigi Menabrea (1809 – 1896) about the Analytical Engine shown below during a period of nine months between 1842 and 1843. . The Analytical Engine was an invention of English mathematician and computer pioneer Charles Babbage (1791-1871). Some historians have recognized Ada Lovelace's set of notes, as the world's first computer program as it contained a ground-breaking description of the possibilities of programming the machine to go beyond number-crunching to "computing".
2. Her notes on the engine include what is recognized as the first algorithm intended to be carried out by a machine. Because of this, she is often regarded as the first computer programmer.
3. 47 years after Ada Lovelace, Herman Hollerith (1860 – 1929) created what is considered the first computer language when he realized he could encode information on punch cards.
4. In the 1940s, machine-specific assembly language was probably the first (vaguely) human-readable programming language.
5. John Mauchly's Short Code, proposed in 1949, was one of the first high-level languages ever developed for an electronic computer.
6. By the 1950s computer engineers realized that assembly language was too complex and a fallible process to build entire systems, and so the first modern programming language was born.
7. In 1957, Backus and his team had created FORTRAN. Fortran was the first high-level programming language to be put to broad use.
8. John McCarthy invented lisp in 1958.
9. COBOL was later designed in 1959 by the Conference on Data Systems Languages (CODASYL).
10. In 1970, Niklaus Wirth developed Pascal.



11. The language C was developed in 1972. The name was based on an earlier language called B, which is now almost extinct.
12. Bjarne Stroustrup, a Danish computer scientist, created C++. The motivation for creating a new language originated from Stroustrup experience in programming for his Ph.D.
13. Primarily Brad Cox and Tom Love in the early 1980s at their company Stepstone created objective-C in 1983.
14. Larry Wall originally developed Perl in 1987; it was developed as a general-purpose Unix scripting language to make report processing easier.
15. Python, a widely used general-purpose, high-level programming language whose design philosophy emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code was born in 1991.
16. Ruby, a dynamic, reflective, object-oriented, general-purpose programming language came about in 1993. Ruby was designed for programmer productivity and fun, following the principles of good user interface design.
17. The year 1995 witnessed the evolution of three major modern languages; Java, PHP and JavaScript.
18. Java is intended to let application developers “write once, run anywhere” (WORA), meaning that compiled Java code can run on all platforms that support Java without the need for recompilation. The language derives much of its syntax from C and C++.
19. Hypertext PreProcessor, PHP, is a server-side scripting language designed for web development but also used as a general-purpose programming language.
20. JavaScript is a high level, dynamic, untyped, and interpreted programming language. It has been standardized in the ECMAScript language specification. Alongside HTML and CSS, it is one of the three essential technologies of World Wide Web content production; the majority of websites employ it and all modern web browsers without plug-ins support it.
21. C-sharp, C#, is intended to be a simple, modern, general-purpose, object-oriented programming language.
22. Born in 2003, Scala is a programming language for general software applications. Scala has full support for functional programming and a very strong static type system. This allows programs written in Scala to be very concise and thus smaller than other general-purpose programming languages.
23. The Go language, commonly referred to as golang, was born in 2009. It is a programming language developed at Google, and is recognizably in the tradition of C, but makes many changes to improve conciseness, simplicity, and safety.
24. Swift was introduced at Apple’s 2014 Worldwide Developers Conference (WWDC). It is a multi-paradigm, compiled programming language created by Apple Inc. for iOS, OS X, and watch OS development. Swift was designed to work with Apple’s Cocoa and Cocoa Touch frameworks and the large body of existing Objective-C code written for Apple products. It is also

intended to be more resilient to erroneous code (“safer”) than Objective-C, and more concise.

### SAQ 2.2

1. According to Simon Raik-Allen, computer languages are categorized thus:
  - i. The High-Level Dawning
  - ii. The C Era
  - iii. The Age of Java

or

According to Raik-Allen, we have:

- i. The Web Dimension
  - ii. The Corporate Dimension
  - iii. The Mobile Dimension
  - iv. The New Age Dimension
2. In approaching this question, your attention is being pointed to the fact that a thorough mastery of the important languages by year is highly essential. As a way of helping you, the table has been brought below to help with a quick check of the important dates.

Below is a Summary of important programming languages by year:

1951 – Regional Assembly Language  
1970 – Pascal  
1993 – Ruby  
1952 – Autocode  
1972 – C  
1994 – CLOS (part of ANSI Common Lisp)  
1954 – IPL (forerunner to LISP)  
1972 – Prolog  
1995 – Ada 95  
1955 – FLOW-MATIC (led to COBOL)  
1972 – Smalltalk  
1995 – Delphi (Object Pascal)  
1957 – COMTRAN (precursor to COBOL)  
1973 – ML  
1995 – Java  
1957 – FORTRAN (First compiler)  
1975 – Scheme  
1995 – JavaScript

1958 – ALGOL 58  
1978 – SQL (a query language, later extended)  
1995 – PHP  
1958 – LISP  
1980 – C++ (as C with classes, renamed in 1983)  
1996 – WebDNA  
1959 – COBOL  
1983 – Ada  
1997 – Rebol  
1959 – FACT (forerunner to COBOL)  
1984 – Common Lisp  
1999 – D  
1959 – RPG  
1984 – MATLAB  
2000 – ActionScript  
1962 – APL  
1985 – Eiffel  
2000 – C#  
1962 – Simula  
1986 – Erlang  
2001 – Visual Basic .NET  
1962 – SNOBOL  
1986 – Objective-C  
2003 – Groovy  
1963 – CPL (forerunner to C)  
1987 – Perl  
2003 – Scala  
1964 – BASIC  
1988 – Mathematica  
2005 – F#  
1964 – PL/I  
1988 – Tcl  
2007 – Clojure  
1966 – JOSS  
1989 – FL (Backus)  
2009 – Go

1967 – BCPL (forerunner to C)  
1990 – Haskell  
2011 – Dart  
1968 – Logo  
1991 – Python  
2012 – Rust  
1969 – B (forerunner to C)  
1991 – Visual Basic  
2014 – Swift  
1970 – Forth  
1993 – Lua

### SAQ 3.1

1. Specifically, the task of defining the problem consists of identifying what it is that you know (input-given data), and what it is you want to obtain (output-the result). This vital information guides you in problem identity and providing a solution to the identified problem.
2. In carrying out a thorough task, it is expected of you to be able to plan and outline solutions to the problems you have identified. First, the problem is divided into several modules or tasks and assigned to each programmer. Two common ways of planning the solution to a problem is either to draw a flowchart or write a pseudocode, or possibly both. Flow chart is a map of what your program is going to do and how it is going to do it while Pseudocode permits you to focus on the program logic without having to be concerned just yet about the precise syntax of a particular programming language.

### SAQ 3.2

The step by step process of coding a program involves:

- i. Translation of the logic from the flowchart or pseudocode-or some other tool-to a programming language.
- ii. This is the step where you actually have to sit down at the computer and type! Coding is a little bit like writing an essay.
- iii. To achieve this, you would require a programming language. A programming language is a set of rules that provides a way of instructing the computer what operations to perform. Examples of programming languages include BASIC, COBOL, Pascal, etc.
- iv. The correct use of the language is the required first step. Then, your coded program must be keyed, probably using a terminal or personal computer, in a form the computer can understand. This is done using a text editor or a paper for beginners.

- v. Finally, before doing the coding, the programmer has to choose a computer language based on the nature of the problem, the programming language available on the computer, and the facilities and limitations of the computer installation.

**SAQ 3.3**

Compilation turns the program into the instructions made up of 0's and 1's that the computer can actually follow. This is necessary because the chip that makes your computer work only understands binary machine code.

**SAQ 3.4**

Debugging is the process of correcting the mistakes made in the course of programming. A compiler must have identified these mistakes. This is important as failure to do so will affect the execution of the program.

**SAQ 3.5**

The aim of testing is to ensure that what you have written solves the original problem. This is important as it is possible for you to write a program correctly without errors but the program would fail in its bid to solve the problem it has been designed for. That is why we test to ensure this purpose has not been defeated.

This can be achieved by subjecting the program to carefully work out set of tests that put it through its paces, and check that it meets the requirements and specification. Where mistakes are identified, there is a need to figure out where in the code the mistake is. Once identified, the problem should be fixed by changing the code and recompiling.

**SAQ 3.6**

Documentation is a written detailed description of the programming cycle and facts about the program. Typical program documentation materials include the origin and nature of the problem, a brief narrative description of the program, logic tools such as flowcharts and pseudocode, data-record descriptions, program listings, and testing results. Essentially, program documentation instructs you on how to interact with a software or a program.

2. The following are some types of documentation that exist:
  - Requirement documentation
  - Architecture/Design documentation
  - Technical documentation
  - End-user documentation
  - Marketing documentation

**SAQ 4.1**

The characteristics of programming languages are:

- i. Functionality across languages

- ii. Syntax and structure
- iii. Natural lifespan
- iv. One creator
- v. Written in English and all use the same English keywords and syntax in their code

#### SAQ 4.2

One way to classify programming languages is either as low-level languages or high-level languages. Low-level languages interact directly with the computer processor or CPU, are capable of performing very basic commands, and are generally hard to read. By contrast, high-level languages use natural language so it is easier for people to read and write.

High-level programming languages must be converted to low-level programming languages using an interpreter or compiler, depending on the language. Interpreted languages are considered more portable than compiled languages, while compiled languages execute faster than interpreted languages.

#### SAQ 4.3

The classification of programming languages are:

- i. Modular Programming language: It is the first programming language that introduced subprogram concepts and variable declaration. Modular programming methodology advocates the breakdown structure approach of a large project into several smaller and manageable modules. Then each small module is programmed comfortably and then put together in order to generate program and hence the answer to the entire problem.
- ii. Structured Programming Language: Structured programming utilizes structures normally used for performing any tasks. These include the Sequence, Selection, Iteration, and CASE structure. An example of the structured programming language is Pascal.
- iii. Business Oriented Language: An example for the business-oriented language is Common Business Oriented Language (COBOL). It has a peculiar feature such that COBOL programs can be easily understood even by non-programmers due to its self-documenting nature and verbose of the grammar. It is also an ideal language for processing voluminous data files making it an excellent language for commercial data processing. Another example is PROLOG, which stands for PROgramming in LOGic. It has a core feature making it exclusively for Artificial Intelligence (AI) and knowledge representation techniques.
- iv. Object Oriented Programming (OOP) Language: The aim of using an object oriented programming language is to handle complex software design projects in a very easy, simple and efficient manner. Some of the famous objects oriented programming languages are: Object Pascal, C++, Smalltalk, Simula, Eiffel, Java, Ada. A major advantage of C++ is its ability to support object oriented programming, while retaining the high

- level of compactness and speed offered by the C programming language.
- v. Visual Programming Languages: Visual programming is one of the key points for developing any software product with multimedia and graphical user interface. Visual effects are one of the most salient features and characteristics of the modern software especially on microcomputer systems. Icons, graphics, animated pictures and texts form the foundation for designing a visual system. Visual programming systems are computer systems, which support both visual programming and visualization. Visual programming means the use of visual expressions (such as icons, drawings or gestures) in the process of programming and visualization. In visual programming languages, objects have logical meaning but not visual image. Objects are then assigned a visual representation so that it can be visualized

### **SAQ 5.1**

The problem solving process is a series of steps to be taken to arrive a logical resolution of the identified problem. These steps have been divided into phases and they include:

- i. Phase 1: Understanding the problem.
- ii. Phase 2: Devising a plan.
- iii. Phase 3: Carrying out the plan.
- iv. Phase 4: Looking back.

### **SAQ 5.2**

An algorithm is a well-defined computational procedure consisting of a set of instructions, which takes some value or set of values, as input, and produces some value or set of values, as output. In other words, an algorithm is a procedure that accepts data; manipulate them following the prescribed steps, to eventually fill the required unknown with the desired value(s).

### **SAQ 5.3**

Pseudo-codes uses the structural conventions of a normal programming language, but is intended for human reading rather than machine reading. Pseudocode typically omits details that are essential for machine understanding of the algorithm, such as variable declarations, system-specific code and some subroutines.

The purpose of using pseudocode is that it is easier for people to understand than conventional programming language code, and that it is an efficient and environment-independent description of the key principles of an algorithm.

### **SAQ 5.4**

Flowcharts are often used as illustrations for algorithms. A flowchart is a graphical representation of an algorithm, which will describe the operations (and in what sequence) are required to solve a given problem. These flowcharts play a vital role in the programming of a problem and

are quite helpful in understanding the logic of complicated and lengthy problems

### **SAQ 6.1**

A program environment refers to a set-up such as a base on top of which programs are created. Thus, it is essential to have to have the required software setup in order to facilitate the proper execution of this function. The following are necessary set-ups to start with in programming using any language.

- A text editor to create computer programs.
- A compiler to compile the programs into binary format.
- An interpreter to execute the programs directly.

### **SAQ 6.2**

The basic syntax of programming refers to codes that written on a program to generate a specified output. These codes comes in forms of commands that are then interpreted to carry out the specified outcome. It is also important to note that if there are any grammatical error (Syntax errors in computer terminologies), then they must be fixed before conversion into a binary format.

### **SAQ 6.3**

Reserved word (also known as a keyword) is a word in programming that cannot be used as an identifier, such as the name of a variable, function, or label – it is "reserved from use"

### **SAQ 6.4**

An operator in a programming language is a symbol that tells the compiler or interpreter to perform specific mathematical, relational or logical operation and produce result.

Logical operators are very important in any programming language as they help us in taking decisions based on certain conditions. C programming and Java provide almost identical set of operators and conditional statements.

### **SAQ 6.5**

Functions are small units of programs used to carry out a specific task. Kindly note that C programming provides various built-in functions like main(), printf(), etc., which are used in programs based on the requirement. A function is a block of organized, reusable code used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.

### **SAQ 6.6**

The parts of a function have been described as follows:

- 1) Return Type
- 2) Function Name
- 3) Parameter List
- 4) Function Body



### SAQ 7.1

1. The types of data we deal with on a daily basis include strings, characters, whole numbers (integers), and decimal numbers (floating point numbers). Different programming languages use different keywords to specify different data types. For example, C and Java programming languages use `int` to specify integer data, whereas `char` specifies a character data type.
2. C and Java support almost the same set of data types. The common types include Character, Number, Small Number, Long Number, and Decimal Number.
3. Python has five standard data types. These include Numbers, String, List, Tuple, and Dictionary. Please note that Number specifies all types of numbers including decimal numbers and string represents a sequence of characters with a length of 1 or more characters.

### SAQ 7.2

- i. Every programming language provides support for manipulating different types of numbers such as simple whole integers and floating point numbers. The major types of data include Character, Number, Small Number, Long Number, and Decimal Number. These data types are called primitive data types and you can use these data types to build more data types, which are called user-defined data types.
- ii. Note that Java provides almost all the numeric data types available in C programming. Java also provides a full range of built-in functions for mathematical calculation and you can use them in the same way as you did in C programming.
- iii. Python is a little different from C and Java; it categorizes numbers in `int`, `long`, `float` and `complex`. Python also provides a full range of built-in functions for mathematical calculations and you can use them in the same way you have used them in C programming.
- iv. Characters are simple alphabets like a, b, c, d..., A, B, C, D... but with an exception. In computer programming, any single digit number like 0, 1, 2 ...and special characters like \$, %, +, -... etc., are also treated as characters and to assign them in a character type variable, you simply need to put them inside single quotes. A character data type consumes 8 bits of memory, which means you can store anything in a character whose ASCII value lies in between -127 to 127, so it can hold any of the 256 different values. Note that you can keep only a single alphabet or a single digit number inside single quotes and more than one alphabets or digits are not allowed inside single quotes.
- v. Escape Sequence refers to when a character is preceded by a backslash (`\`) and many programming languages support this concept. Java handles character data types much in the same way as we have seen in C programming. However, Java provides additional support for character manipulation. Java also supports escape sequence in the same way you have used them in C programming. Python does not support any character data type

but all the characters are treated as string, which is a sequence of characters. Python supports escape sequences in the same way as you have used them in C programming

### SAQ 7.3

The basic string concepts include:

- Strings in C are represented as arrays of characters.
- We can constitute a string in C programming by assigning character by character into an array of characters.
- We can constitute a string in C programming by assigning a complete string enclosed in double quote.
- We can print a string character by character using an array subscript or a complete string by using an array name without subscript.
- The last character of every string is a null character, i.e., '\0'.
- Most of the programming languages provide built-in functions to manipulate strings

### SAQ 7.4

1. Java provides strings as a built-in data type like any other data type. It means you can define strings directly instead of defining them as array of characters.
2. Creating strings in Python is as simple as assigning a string into a Python variable using single or double quotes. Python does not support character type; these are treated as strings of length one, thus also considered a substring.

### SAQ 8.1

1. A conditional statement is a mechanism that allows for conditional execution of instructions based upon the outcome of a conditional statement, which can be either true or false. Examples include:
  - The 'If Statement'
  - The 'If...Else Statement'
  - The 'If...elseif....Else Statement'
  - The Switch Statement

### SAQ 8.2

1. A Loop is an important tool in programming. Almost all the programming languages provide this concept (i.e. loop), which helps in executing one or more statements up to a desired number of times e.g. rather than repeating the statement 'hello world' a thousand times by writing 'printf()' a thousand times, you can bypass this challenge by simply using a loop. All high-level programming languages provide various forms of loops, which can be used to execute one or more statements repeatedly. The

most important types of loop are ‘The While Loop’, and the ‘The Do...While Loop’

2. The break and continue statements in Java programming work quite the same way as they work in C programming.

### SAQ 9.1

1. An array is a data structure, which can store a fixed-size collection of elements of the same data type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type. All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.
2. To create an array variable in C, a programmer specifies the type of the elements and the number of elements to be stored in that array. Array in C can be initialized either one by one or using a single statement. You should know that the number of values between braces { } cannot be larger than the number of elements that we declare for the array between square brackets [ ]. If you omit the size of the array, an array just big enough to hold the initialization is created. An element is accessed by indexing the array name. This is done by placing the index of the element within square brackets after the name of the array.

### SAQ 9.2

Java support arrays though there is a little difference in the way they are created

Python does not have a concept of Array, instead Python provides another data structure called list, which provides similar functionality as arrays in any other language.

### SAQ 10.1

File input means data that is written into a file and file output means data that is read from a file. When we provide some input to our program from the command prompt, then it is called input.

### SAQ 10.2

1. Operation modes refer to a format through which a file can be opened. This include the Read-Only Mode, Write-Only Mode, Read & Write Mode, and the Append Mode.
2. Usually, we use the fopen() function to create a new file or to open an existing file. This call will initialize an object of the type FILE, which contains all the information necessary to control the stream. In closing files, To close a file, we adopt the fclose( ) function. The prototype of this function is – int fclose( FILE \*fp );. The fclose( ) function returns zero on success, or EOF, special character, if there is an error in closing the file. This function actually flushes any data still pending in the buffer to the file, closes the file, and releases any memory used for the file.

3. In writing a file, we make use of the function `int fputc( int c, FILE *fp );`. The function `fputc()` writes the character value of the argument `c` to the output stream referenced by `fp`. It returns the written character written on success, otherwise EOF if there is an error. In reading a file, we make use of the function `int fgetc( FILE * fp );`. The `fgetc()` function reads a character from the input file referenced by `fp`. The return value is the character read; or in case of any error, it returns EOF.