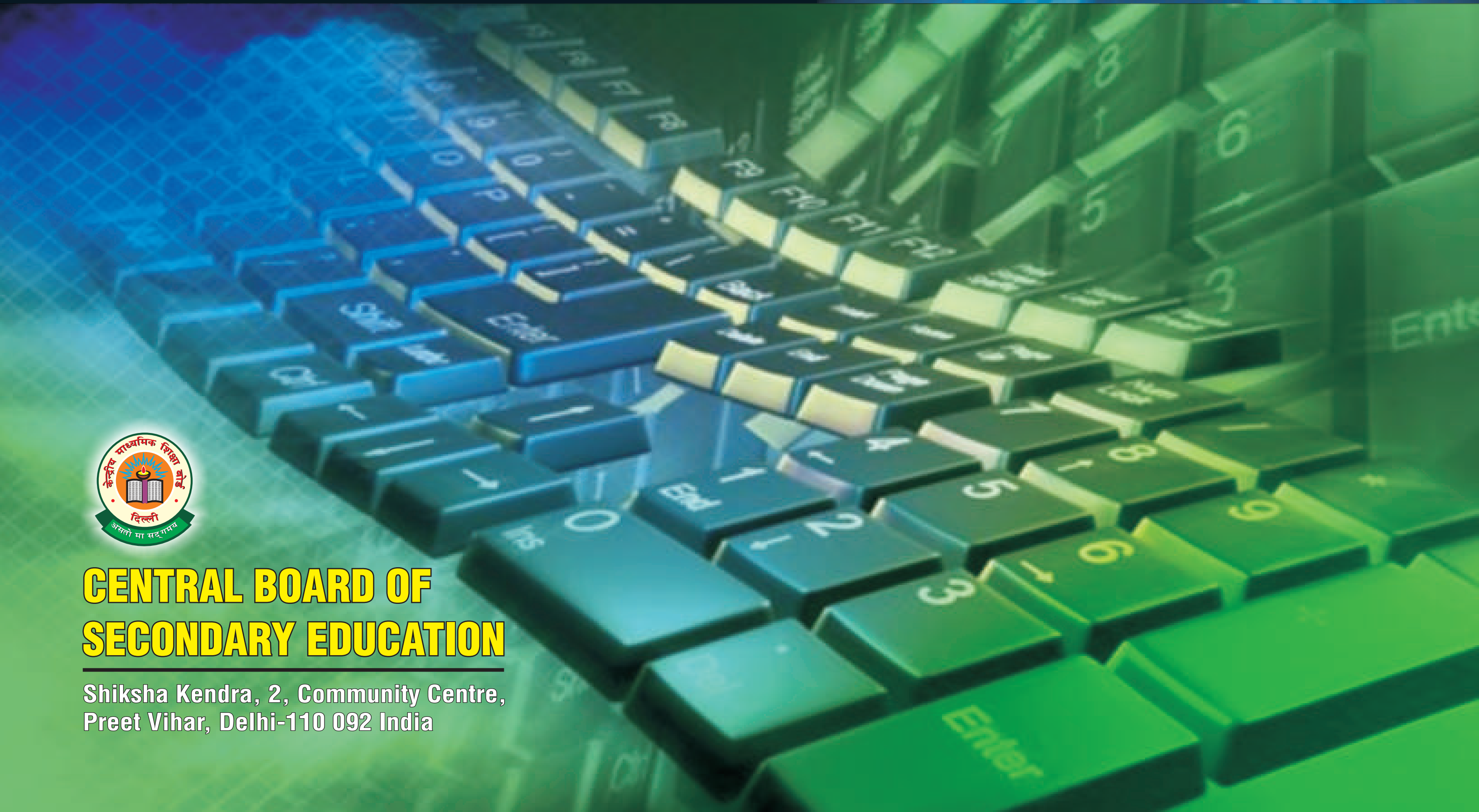
Computer Science

Class XI



CENTRAL BOARD OF SECONDARY EDUCATION

Shiksha Kendra, 2, Community Centre,
Preet Vihar, Delhi-110 092 India

नया आगाज़

आज समय की माँग पर
आगाज़ नया इक होगा
निरंतर योग्यता के निर्णय से
परिणाम आकलन होगा।

परिवर्तन नियम जीवन का
नियम अब नया बनेगा
अब परिणामों के भय से
नहीं बालक कोई डरेगा

निरंतर योग्यता के निर्णय से
परिणाम आकलन होगा।

बदले शिक्षा का स्वरूप
नई खिले आशा की धूप
अब किसी कोमल-से मन पर
कोई बोझ न होगा

निरंतर योग्यता के निर्णय से
परिणाम आकलन होगा।
नई राह पर चलकर मंज़िल को हमें पाना है
इस नए प्रयास को हमने सफल बनाना है
बेहतर शिक्षा से बदले देश, ऐसे इसे अपनाए
शिक्षक, शिक्षा और शिक्षित
बस आगे बढते जाएँ
बस आगे बढते जाएँ
बस आगे बढते जाएँ.........

CBSE
CONTINUOUS AND COMPREHENSIVE EVALUATION
KNOW - AS YOU GROW

COMPUTER SCIENCE

Class-XI

Computer Science - Class XI

केन्द्रीय माध्यमिक शिक्षा बोर्ड
दिल्ली
असतो मा सद्गमय

CENTRAL BOARD OF SECONDARY EDUCATION

Shiksha Kendra, 2, Community Centre, Preet Vihar, Delhi-110 092 India

Computer Science Class XI

PUBLISHED BY : The Secretary, Central Board of Secondary Education
Shiksha Kendra, 2, Community Centre,
Preet Vihar, Delhi-110092

DESIGN, LAYOUT : Multi Graphics, 5745/81, Reghar Pura, Karol Bagh,
New Delhi-110005, Phone : 25783846

PRINTED BY :

भारत का संविधान

उद्देशिका

हम, भारत के लोग, भारत को एक ¹[सम्पूर्ण प्रभुत्व-संपन्न समाजवादी पंथनिरपेक्ष लोकतंत्रात्मक गणराज्य] बनाने के लिए, तथा उसके समस्त नागरिकों को:

सामाजिक, आर्थिक और राजनैतिक न्याय,
विचार, अभिव्यक्ति, विश्वास, धर्म
और उपासना की स्वतंत्रता,
प्रतिष्ठा और अवसर की समता

प्राप्त कराने के लिए, तथा उन सब में, व्यक्ति की गरिमा और ²[राष्ट्र की एकता और अखण्डता] सुनिश्चित करने वाली बंधुता बढ़ाने के लिए दृढ़संकल्प होकर अपनी इस संविधान सभा में आज तारीख 26 नवम्बर, 1949 ई॰ को एतद्द्वारा इस संविधान को अंगीकृत, अधिनियमित और आत्मार्पित करते हैं।

1. संविधान (बयालीसवां संशोधन) अधिनियम, 1976 की धारा 2 द्वारा (3.1.1977) से "प्रभुत्व-संपन्न लोकतंत्रात्मक गणराज्य" के स्थान पर प्रतिस्थापित।
2. संविधान (बयालीसवां संशोधन) अधिनियम, 1976 की धारा 2 द्वारा (3.1.1977 से), "राष्ट्र की एकता" के स्थान पर प्रतिस्थापित।

भाग 4 क

मूल कर्त्तव्य

51 क. मूल कर्त्तव्य – भारत के प्रत्येक नागरिक का यह कर्त्तव्य होगा कि वह –

(क) संविधान का पालन करे और उसके आदर्शों, संस्थाओं, राष्ट्रध्वज और राष्ट्रगान का आदर करे;

(ख) स्वतंत्रता के लिए हमारे राष्ट्रीय आंदोलन को प्रेरित करने वाले उच्च आदर्शों को हृदय में संजोए रखे और उनका पालन करे;

(ग) भारत की प्रभुता, एकता और अखंडता की रक्षा करे और उसे अक्षुण्ण रखे;

(घ) देश की रक्षा करे और आह्वान किए जाने पर राष्ट्र की सेवा करे;

(ङ) भारत के सभी लोगों में समरसता और समान भ्रातृत्व की भावना का निर्माण करे जो धर्म, भाषा और प्रदेश या वर्ग पर आधारित सभी भेदभाव से परे हों, ऐसी प्रथाओं का त्याग करे जो स्त्रियों के सम्मान के विरुद्ध हैं;

(च) हमारी सामासिक संस्कृति की गौरवशाली परंपरा का महत्त्व समझे और उसका परीक्षण करे;

(छ) प्राकृतिक पर्यावरण की जिसके अंतर्गत वन, झील, नदी, और वन्य जीव हैं, रक्षा करे और उसका संवर्धन करे तथा प्राणिमात्र के प्रति दयाभाव रखे;

(ज) वैज्ञानिक दृष्टिकोण, मानववाद और ज्ञानार्जन तथा सुधार की भावना का विकास करे;

(झ) सार्वजनिक संपत्ति को सुरक्षित रखे और हिंसा से दूर रहे;

(ञ) व्यक्तिगत और सामूहिक गतिविधियों के सभी क्षेत्रों में उत्कर्ष की ओर बढ़ने का सतत प्रयास करे जिससे राष्ट्र निरंतर बढ़ते हुए प्रयत्न और उपलब्धि की नई उंचाइयों को छू ले।

THE CONSTITUTION OF INDIA

PREAMBLE

WE, THE PEOPLE OF INDIA, having solemnly resolved to constitute India into a [1] **SOVEREIGN SOCIALIST SECULAR DEMOCRATIC REPUBLIC** and to secure to all its citizens :

JUSTICE, social, economic and political;

LIBERTY of thought, expression, belief, faith and worship;

EQUALITY of status and of opportunity; and to promote among them all

FRATERNITY assuring the dignity of the individual and the [2] [unity and integrity of the Nation];

IN OUR CONSTITUENT ASSEMBLY this twenty-sixth day of November, 1949, do **HEREBY TO OURSELVES THIS CONSTITUTION.**

1. Subs, by the Constitution (Forty-Second Amendment) Act. 1976, sec. 2, for "Sovereign Democratic Republic (w.e.f. 3.1.1977)
2. Subs, by the Constitution (Forty-Second Amendment) Act. 1976, sec. 2, for "unity of the Nation (w.e.f. 3.1.1977)

THE CONSTITUTION OF INDIA

Chapter IV A

Fundamental Duties

ARTICLE 51A

Fundamental Duties - It shall be the duty of every citizen of India-

(a) to abide by the Constitution and respect its ideals and institutions, the National Flag and the National Anthem;

(b) to cherish and follow the noble ideals which inspired our national struggle for freedom;

(c) to uphold and protect the sovereignty, unity and integrity of India;

(d) to defend the country and render national service when called upon to do so;

(e) to promote harmony and the spirit of common brotherhood amongst all the people of India transcending religious, linguistic and regional or sectional diversities; to renounce practices derogatory to the dignity of women;

(f) to value and preserve the rich heritage of our composite culture;

(g) to protect and improve the natural environment including forests, lakes, rivers, wild life and to have compassion for living creatures;

(h) to develop the scientific temper, humanism and the spirit of inquiry and reform;

(i) to safeguard public property and to abjure violence;

(j) to strive towards excellence in all spheres of individual and collective activity so that the nation constantly rises to higher levels of endeavour and achievement.

Foreword

This century is characterized with the emergence of knowledge based society wherein ICT plays a pivotal role. In its vision, the National Policy on ICT in School Education by MHRD, Govt. of India, states "The ICT Policy in School Education aims at preparing youth to participate creatively in the establishment, sustenance and growth of a knowledge society leading to all round socio economic development of the nation and global competitiveness". The policy envisages three stages of ICT implementations at school level - ICT literacy and Competency Enhancement, IT enabled teaching-learning, and introduction of ICT related elective subjects at Senior Secondary level.

With this backdrop a major paradigm shift is imperative in imparting ICT- enabled instructions, collaborative learning, multidisciplinary problem-solving and promoting critical thinking skills as envisaged in the National curriculum framework 2005. Foundation of these skills is laid at school level.

Ever since the invention of Charles Babbage's difference engine in 1822, computers have required a means of instructing them to perform a specific task. This is known as a programming language. Programs in computer programming language prepare people to write and design computer software. Computer languages were first composed of a series of steps to wire a particular program; these morphed into a series of steps keyed into the computer and then executed; later these languages acquired advanced features such as logical branching and object orientation.

Syllabus of Computer Sciences has been revisited accordingly with a focus on generic concepts with domain specific practical experiments and projects to ensure conceptual knowledge with practical skills. Learning to write programs stretches your mind, and helps you think better, creates a way of thinking about things that is helpful in all domains. Since Computers have permeated in every walk of life such as launching satellites, e-traiding, e-business and also enabling social networking it is imperative to study programming languages.

I am happy to release Part-1 of Computer Science Book for Class - XI. I would like to express my deep appreciation to the text book development team for their contribution. Appreciation is also due to Dr. Sadhana Parashar, Director (Academics, Research, Training and Innovation) and Kshipra Verma, Education Officer, CBSE in bringing out this publication.

It is hoped that all students and teachers will benefit by making best use of this publication. Their feedback will be highly appreciated for further improvement.

Vineet Joshi

Chairman, CBSE

Acknowledgements

CBSE ADVISORS

❖ **Shri Vineet Joshi, Chairman, CBSE**

❖ **Dr. Sadhana Parashar, Director**
(Academics, Research, Training & Innovation)

DEVELOPMENT TEAM

❖ **Ms. Anju Gupta, Rukmini Devi Public School, Pitam Pura, New Delhi.**

❖ **Ms. Mohini Arora, Air Force Golden Jubilee Institute, Subroto Park, New Delhi.**

❖ **Ms. S. Meena, Sachdeva Public School, Pitam Pura, New Delhi.**

❖ **Ms. Shally Arora, Delhi Public School, Gurgaon.**

MEMBER COORDINATOR

❖ Ms. **Kshipra Verma, Education Officer, CBSE, New Delhi.**

Content

UNIT 1

Computer Fundamentals

Chapter 1

Computer Fundamentals

After studying this session students will be able to:

- *Learn the evolution of computers*
- *Learn about various generations of computer*
- *Understand the basic operation of a computer*
- *Study the functional components and their interconnections*
- *Understand the concept of booting*
- *Learn about classification of computers*

Introduction

Computers are seen everywhere around us, in all spheres of life. May it be the field of education and research, travel and tourism, weather forecasting, social networking, e-commerce or any other, computers have now become an indispensable part of our lives. The manner, in which computers have revolutionised our lives because of their accuracy and speed of performing a job, is truly remarkable. Today no organization can function without a computer. In fact various organizations are trying to become paper free owing to benefits of computers. But the computers of today have evolved over the years from a simple calculating device to the portable high speed computers that we see today.

Evolution of Computers

The growth of computer industry started with the need for performing fast calculations. The manual method of computing was slow and prone to errors. So attempts were made to develop faster calculating devices. The journey that started from the first calculating device i.e. Abacus has led us today to extremely high speed calculating devices. Let us first have a look at some early calculating devices and then we will explore various generations of computer.

Abacus

Abacus was discovered by the Mesopotamians in around 3000 BC. An abacus consisted of beads on movable rods divided into two parts. (Fig-1) Addition and multiplication of numbers was done by using the place value of digits of the numbers and position of beads in an abacus.

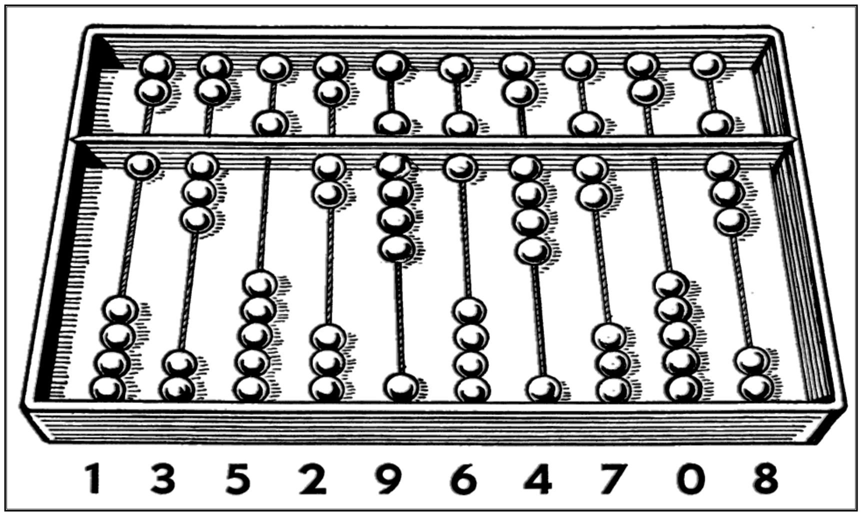


Fig: An Abacus

The Chinese further improved on the abacus so that calculations could be done more easily. Even today abacus is considered as an apt tool for young children to do calculations. In an abacus, each row is thought of as a ten's place. From right to left , row no-1 represents the one's column and the second column represents ten's place. The third column represents the hundred's place and so on. The starting position of the top beads (representing the value of five) is always towards the top wall of the abacus while the lower beads (representing the value of one) will always be pushed towards the lower wall as a starting position.

Napier's Logs and Bones

The idea of logarithm was developed by John Napier in 1617. He devised a set of numbering rods known as Napier's Bones through which both multiplication and division could be performed. These were numbered rods which could perform multiplication of any number by a number in the range of 2-9. There are 10 bones corresponding to the digits 0-9 and there is also a special eleventh bone that is used to represent the multiplier. By placing bones corresponding to the multiplier on the left side and the bones corresponding to the digits of the multiplicand on the right , the product of two numbers can be easily obtained.

Fig: Napier's Logs

Pascaline

Blaise Pascal, a French mathematician invented an adding machine in 1642 that was made up of gears and was used for adding numbers quickly. This machine was also called Pascaline and was capable of addition and subtraction along with carry-transfer capability. It worked on clock

Fig: Pascaline

work mechanism principle. It consisted of various numbered toothed wheels having unique position values. The addition and subtraction operations was performed by controlled rotation of these wheels.

Leibnitz's Calculator

In 1673 Gottfried Leibnitz, a German mathematician extended the capabilities of the adding machine invented by Pascal to perform multiplication and division as well. The multiplication was done through repeated addition of numbers using a stepped cylinder each with nine teeth of varying lengths.

Fig: Leibnitz's Calculator

Jacquard's Loom

In order to make the cotton weaving process automatic, Joseph Jaquard devised punch cards and used them to control looms in 1801. The entire operation was under a program's control. Through this historic invention, the concept of storing and retrieving information started.

Difference engine and Analytical Engine

Charles Babbage, an English mathematician developed a machine called Difference Engine in 1822 which could calculate various mathematical functions, do polynomial evaluation by finite difference and theoretically could also solve differential equations.

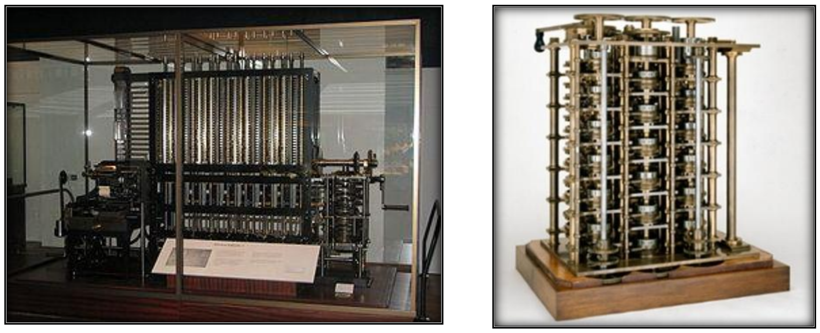


Fig: Difference Engine and Analytical engine

Thereafter in 1833, he designed the Analytical Engine which later on proved to be the basis of modern computer. This machine could perform all the four arithmetic operations as well as comparison. It included the concept of central processor, memory storage and input-output devices. Even the stored information could be modified. Although the analytical engine was never built that time but Babbage established the basic principles on which today's modern computers work.

Both these great inventions earned him the title of **'Father of Modern Computers'.**

Mark 1

In 1944 Prof Howard Aiken in collaboration with IBM constructed an electromechanical computer named Mark 1 which could multiply two 10 digit numbers in 5 seconds. This machine was based on the concept of Babbage's Analytical engine and was the first operational general purpose computer which could execute preprogrammed instructions automatically without any human intervention.

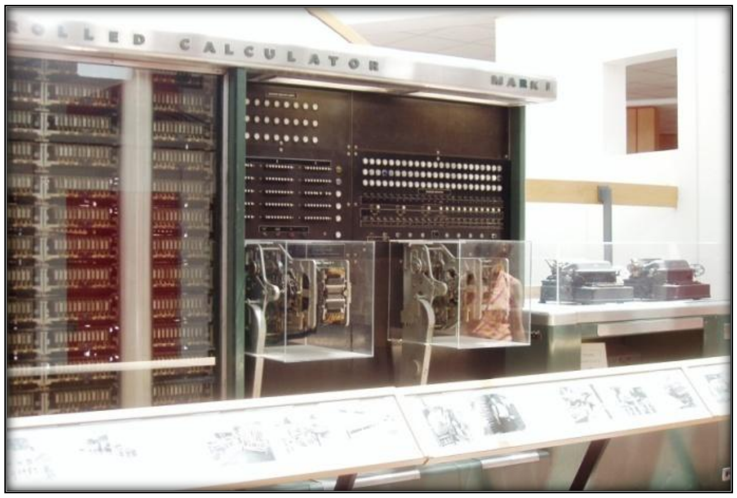


Fig: Mark 1

In 1945, Dr. John Von Neumann proposed the concept of a stored program computer. As per this concept the program and data could be stored in the same memory unit. The basic architecture of the Von Neumann computer is shown in the figure below

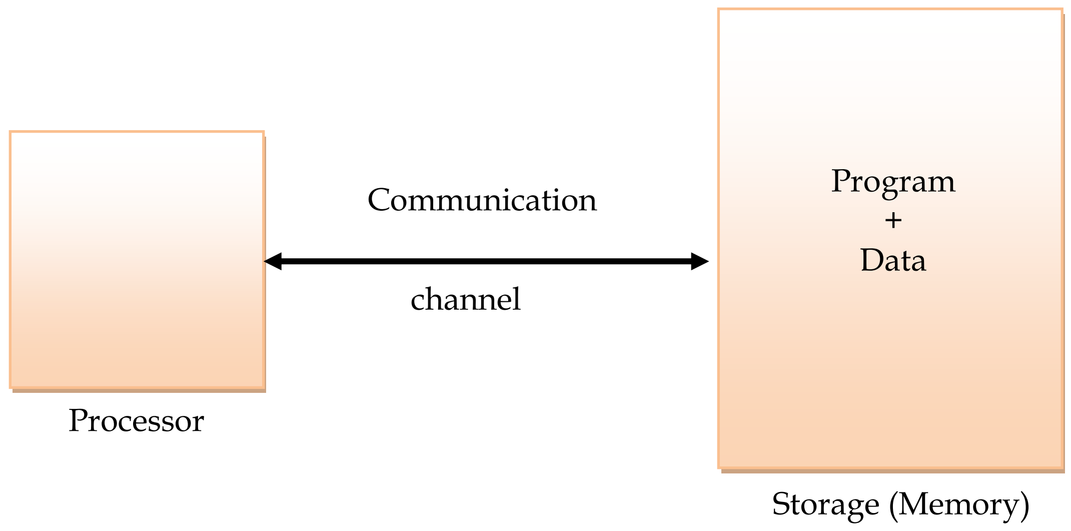


Fig: Von Neumann Computer

According to Von Neumann architecture, the processor executes instructions stored in the memory of the computer. Since there is only one communication channel, the processor at a time can either fetch data or an instruction. That means at one point of time either the data or an instruction can be picked (fetched) from the storage unit for execution by the processor. Hence execution takes place in sequential manner. This limitation of Von Neumann Computer is known as Von Neumann bottleneck. EDVAC (**E**lectronic **D**iscrete **V**ariable **A**utomatic **C**omputer) was the first stored program computer developed in 1952. After the invention of first electronic computer ENIAC (**E**lectronic **N**umerical **I**ntegrator and **C**alculator) in 1946, the computer technology improved tremendously and at very fast pace.

Generation of Computers

Growth in the computer industry is determined by the development in technology. Each phase/generation of computer development is characterized by one or more hardware/software developments that distinctly improved the performance of the computers of that generation. Based on various stages of development, computers can be divided into different generations.

The First Generation (1942-1955)

The first generation computers used the concept of 'stored program' and were characterized by vacuum tubes. A vacuum tube is a delicate glass device that can control and amplify electronic signals. The first generation computers were made using thousands of vacuum tubes and were the fastest calculating devices of their time. These computers were very large in size, consumed lot of electricity and generated lot of heat. UNIVAC 1 was the first electronic computer of this generation and was used for business applications.

Salient features of First generation computers:

- Used vacuum tubes to control and amplify electronic signals
- Huge computers that occupied lot of space
- High electricity consumption and high heat generation
- Were unreliable since they were prone to frequent hardware failures
- Commercial production was difficult
- They were very costly and required constant maintenance
- Continuous air conditioning was required
- Programming was done in machine language although assembly language also started at the end of this generation Example : ENIAC , EDVAC , UNIVAC 1

Note: ENIAC weighed about 27 tons, was of the size 8 feet * 100 feet * 3 feet and consumed around 150 watts of power.

The Second Generation (1955–1964)

The second generation computers were characterized by transistors. A transistor is a solid state semiconductor device that revolutionized the electronic industry. Transistors were smaller, highly reliable, consumed less electricity and generated less heat. Also magnetic core memories were developed during this generation. These are tiny ferrite rings that can be magnetized in either clockwise or anticlockwise direction so as to represent binary 1 or binary 0. Magnetic cores were used as primary memories. Later magnetic disks also came into existence and were used as secondary storage devices. All these new developments – transistors, magnetic core memory and magnetic disk storage devices made the computers more powerful and reliable. This further led to the existence of operating systems. Programming languages like FORTRAN, COBOL, Algol etc. also developed. Commercial applications of the computer increased and now the computers were used in business and industries for applications like payroll, employee management, inventory control etc. IBM 1401 and IBM 1620 were popular computers of this generation.

Salient Features of Second generation computers:

- ✡ Use transistor based technology
- ✡ Were smaller and less expensive as compared to first generation
- ✡ Consumed less electricity and emitted less heat
- ✡ Magnetic core memories and magnetic disks were used as primary and secondary storage respectively
- ✡ First operating system developed
- ✡ Programming in assembly language and in the later part high level languages were used
- ✡ Wider commercial use but commercial production was still difficult
- ✡ They also required constant air-conditioning.

Examples: IBM 1401, IBM 1620, UNIVAC 1108

The Third Generation (1964-1975)

In 1964, the Integrated Circuits or ICs or chips revolutionized the electronic industry and started the third generation of computers. An IC is a small silicon chip or wafer made up of extremely purified silicon crystals. It has numerous transistors, capacitors, resistors and other elements of an electronic circuit. A small scale integration (SSI) chip used to have about 10 transistors on a single chip and a medium scale integration (MSI) chip had about 100 transistors per chip. The size of memories also increased. Various mainframe computers and minicomputers were developed during this generation. Even operating systems with multitasking and multiprogramming features (you will learn about these terms in the next chapter) were developed. Since ICs made the computers highly reliable, relatively inexpensive and faster, computers these days were found in areas of education, small businesses and offices along with industrial and business applications. IBM 360 was a very popular third generation computer.

Salient Features of Third Generation computers:

- ✡ Used integrated circuits
- ✡ Computers were smaller , faster and more reliable
- ✡ Low power consumption and less emission of heat as compared to previous generations

Examples: IBM 360 series, Honeywell 6000 series

The Fourth Generation (1975 onwards)

In this generation Large Scale Integration (LSI) and Very Large scale integration (VLSI) technology was used by which up to 300,000 transistors were used on a single chip. Thus integration of complete CPU on a single chip was achieved in 1971 and was named microprocessor which marked the fourth generation of computers. The computers based on microprocessor technology had faster accessing and processing speeds. In addition to this the increased memory capacity further made the computers more powerful and also more efficient operating systems were developed for these computers. New concepts of microprogramming, application software, databases, virtual memory etc were developed and used.

The computers that we use today belong to this generation. These portable computers can be carried from one place to another owing to their compact size. They are much more accurate. Even memory sizes have become phenomenal. Commercial production of these computers is easier and they are the least expensive, compared to the earlier generation computers.

Also computer networks starting coming up during this generation. It is today one of the most popular means to interact and communicate with people.

Salient features of Fourth generation Computers

- ✡ ICs with LSI and VLSI technology
- ✡ Microprocessors developed
- ✡ Portable computers developed
- ✡ Networking and data communication became popular

- Different types of secondary memory with high storage capacity and fast access developed
- Very reliable ,powerful and small in size
- Negligible power consumption and heat generation
- Very less production cost

Fifth Generation Computers

Fifth Generation computers are still under development. This generation is based on the concept of artificial intelligence. In simple terms the computers of this generation are supposed to behave like humans. The principles of parallel processing (many processors are grouped together) and superconductivity are being used to develop devices that respond to human languages and will have the ability to apply previously gained knowledge to execute a task. They will let them make decisions of their own to execute a task. Some applications like voice recognition, visual recognition are a step in this very direction.

Salient features of fifth generation computers:

- Parallel Processing
- Superconductivity
- Artificial Intelligence

Computer - Data and Information

We all know what a computer is? It is an electronic device that processes the input according to the set of instructions provided to it and gives the desired output at a very fast rate. Computers are very versatile as they do lot of different tasks such as storing data, weather forecasting, booking airline, railway or movie tickets and even playing games.

Data: It is the term used for raw facts and figures. For example, 134, + 9, 'Raju', 'C' are data. Definition of information should start from next line as given in the word file. In composed file it is starting from the same line immediately after the definition of data. **Information:** Data represented in useful and meaningful form is information. In simple words we can say that data is the raw material that is processed to give meaningful, ordered or structured information. For example Raju is 9 years old. This is information about Raju and conveys some meaning. This conversion of data to information is called data processing.

Functional Components of a Computer

The computer is the combination of hardware and software. Hardware are the physical components of a computer like motherboard, memory devices, monitor, keyboard etc. while software is the set of programs or instructions. Both hardware and software together make the computer system function. Let us first have a look at the functional components of a computer.

Every task given to a computer follows an Input- Process- Output Cycle (IPO cycle). It needs certain input, processes that input and produces the desired output. The input unit takes the input, the central processing unit does the processing of data and the output unit produces the output. The memory unit holds the data and instructions during the processing.

Fig below shows the basic structure of the computer.

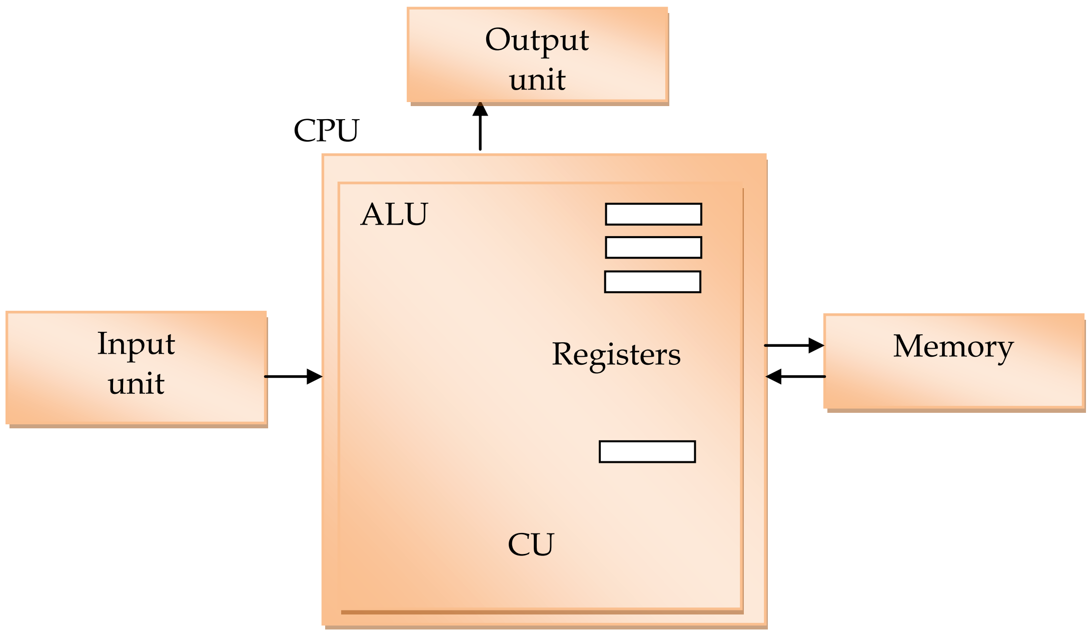


Fig: Functional Components of a computer

Input Unit

The input unit consists of input devices that are attached to the computer. These devices take input and convert it into binary language that the computer understands. Some of the common input devices are keyboard, mouse, joystick, scanner etc.

Central Processing Unit (CPU)

Once the information is entered into the computer by the input device, the processor processes it. The CPU is called the brain of the computer because it is the control centre of the computer. As the CPU is located on a small chip, it is also called the microprocessor. It first fetches instructions from memory and then interprets them so as to know what is to be done. If required, data is fetched from memory or input device. Thereafter CPU executes or performs the required computation and then either stores the output or displays on the output device. The CPU has three main components which are responsible for different functions – Arithmetic Logic Unit (ALU) , Control Unit (CU) and Memory registers.

Arithmetic and Logic Unit (ALU)

The ALU, as its name suggests performs mathematical calculations and takes logical decisions. Arithmetic calculations include addition, subtraction, multiplication and division. Logical decisions involve comparison of two data items to see which one is larger or smaller or equal.

Control Unit

The Control unit coordinates and controls the data flow in and out of CPU and also controls all the operations of ALU, memory registers and also input/output units. It is also responsible for carrying out all the instructions stored in the program. It decodes the fetched instruction, interprets (understands) it and sends control signals to input/output devices until the required operation is done properly by ALU and memory.

Memory Registers

A register is a temporary unit of memory in the CPU. These receive data/information and then this data/information is held in them as per the requirement. Registers can be of different sizes(16 bit , 32 bit , 64 bit and so on) and each register inside the CPU has a specific function like storing data, storing an instruction, storing address of a location in memory etc. The user registers can be used by an assembly language programmer for storing operands, intermediate results etc. Accumulator (ACC) is the main register in the ALU and contains one of the operands of an operation to be performed in the ALU.

Memory

Memory attached to the CPU is used for storage of data and instructions and is called internal memory. During processing, it is the internal memory that holds the data. The internal memory is divided into many storage locations, each of which can store data or instructions. Each memory location is of the same size and has an address. With the help of the address, the computer can find any data easily without having to search the entire memory. The internal memory is also called the ***Primary memory*** or Main memory. When the task is performed, the CU makes the space available for storing data and instructions, thereafter the memory is cleared and the memory space is then available for the next task. The time of access of data is independent of its location in memory, therefore this memory is also called Random Access memory (RAM). Primary memory is volatile in nature. That means when the power is switched off, the data stored in this memory is permanently erased. That is why secondary memory is needed to store data and information permanently for later use. Some of the examples of secondary storage devices are hard disk, compact disks, pen drives etc.

Output Unit

The output unit consists of output devices that are attached with the computer. It converts the binary data coming from CPU to human understandable from. The common output devices are monitor, printer, plotter etc.

Interconnection between Functional Components

The interconnection between the functional components of a computer can be done in many ways. In microcomputers we generally see a Common Bus Architecture as shown in the figure below. As we have seen before that a computer consists of input unit that takes input, a CPU that processes the input and an output unit that produces output. All these devices communicate with each other through a common bus. A bus is a transmission path (set of conducting wires) over which data or information in the form of electric signals, is passed from one component to another in a computer. The bus can be of three types – Address bus, Data bus and Control Bus.

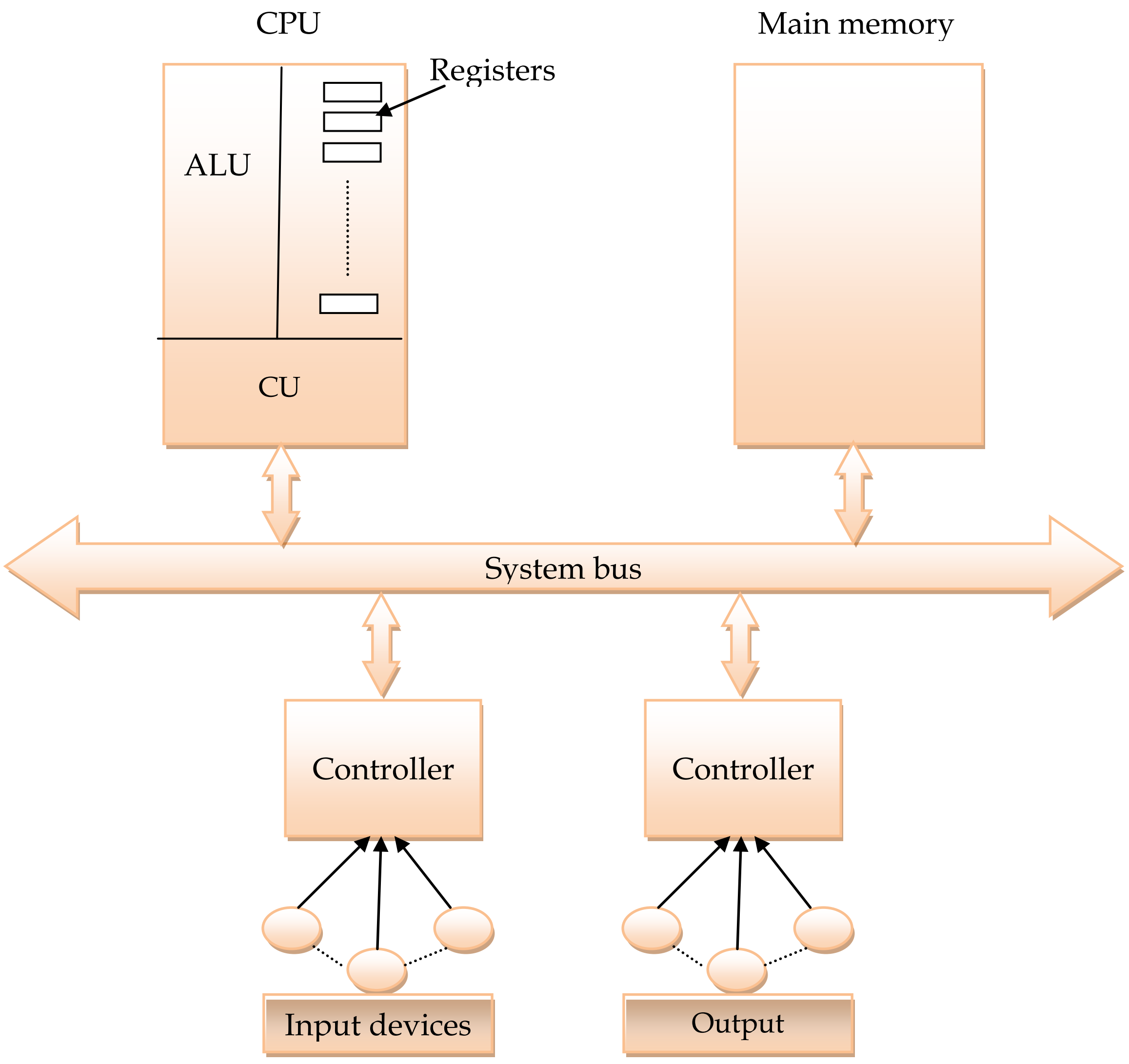


Fig: Common Bus Architecture

The address bus carries the address location of the data or instruction. The data bus carries data from one component to another and the control bus carries the control signals. As shown in the figure above, the system bus is the common communication path that carries signals to/from CPU, main memory and input/output devices. The input/output devices communicate with the system bus through the controller circuit. This controller circuit helps to manage various input/output devices attached to the computer.

Concept of Booting

When the computer is switched on, a copy of boot program is brought from ROM into the main memory. This process is called booting. The CPU first runs a jump instruction that transfers to BIOS (Basic Input output System) and it starts executing. The BIOS conducts a series of self diagnostic tests called POST (Power On Self Test). These tests include memory tests, configuring and starting video circuitry, configuring the system's hardware and checking other devices that help to function the computer properly. Thereafter the BIOS locates a bootable drive to load the boot sector. The execution is then transferred to the Boot Strap Loader program on the boot sector which loads and executes the operating system. If the boot sector is on the hard drive then it will have a Master Boot record (MBR) which checks the partition table for active partition. If found, the MBR loads that partition's boot sector and executes it.

Booting Process is of two types – Warm and Cold

Cold Booting: When the system starts from initial state i.e. it is switched on, we call it cold booting or Hard Booting. When the user presses the Power button, the instructions are read from the ROM to initiate the booting process.

Warm Booting: When the system restarts or when Reset button is pressed, we call it Warm Booting or Soft Booting. The system does not start from initial state and so all diagnostic tests need not be carried out in this case. There are chances of data loss and system damage as the data might not have been stored properly.

Classification of Computers

The computers can be classified based on the technology being used as: Digital, Analog and Hybrid

Digital Computers

These computers are capable of processing information in discrete form. In digital technology data which can be in the form of letters, symbols or numbers is represented in binary form i.e. 0s and 1s. Binary digits are easily expressed in a digital computer by the presence (1) or absence (0) of current or voltage. It computes by counting and adding operations. The digital computers are used in industrial, business and scientific applications. They are quite suitable for large volume data processing.

Analog Computers

An analog computer works on continuously changeable aspects of physical phenomenon such as fluid pressure, mechanical motion and electrical quantities. These computers measure changes in continuous physical quantities say current and voltage. These computers are used to process data generated by ongoing physical processes. A thermometer is an example of an analog computer since it measures the change in mercury level continuously. Although the accuracy of an analog computer is less as compared to digital computers, yet it is used to process data generated by changing physical quantities especially when the response to change is fast. Most present day analog computers are well suited to simulating systems. A simulator helps to conduct experiments repeatedly in real time environment. Some of the common examples are simulations in aircrafts, nuclear power plants, hydraulic and electronic networks.

Hybrid Computers

These use both analog and digital technology. It has the speed of analog computer and the accuracy of a digital computer. It may accept digital or analog signals but an extensive conversion of data from digital to analog and analog to digital has to be done. Generally the analog components provide efficient processing of differential equations while the digital part deals with logical operations of the system. Hence benefits of both analog and digital computing are readily available. Hybrid Computers are used as a cost effective means for complex simulations.

Classification of Digital Computers

The digital computers are classified according to their computing capabilities. The various types of digital computers are discussed below:

Micro Computers

These are also known as Personal Computers. These type of digital computer uses a microprocessor (a CPU on a single chip) and include both desktops and laptops. These computers can work on small volume of data, are very versatile and can handle variety of applications. These computers are being used as work stations, CAD, multimedia and advertising applications. Small portable computers such as PDAs (Personal Digital Assistants) and tablets with wireless computing technology are increasingly becoming popular.

Mini Computers

These computers can support multiple users working simultaneously on the same machine. These are mainly used in an organization where computers installed in various departments are interconnected. These computers are useful for small business organizations.

Main Frames

These computers are large and very powerful computers with very high memory capacity. These can process huge databases such as census at extremely fast rate. They are suitable for big organizations, banks, industries etc. and can support hundreds of users simultaneously on the network.

Super Computers

These are fastest and very expensive computers. They can execute billions of instructions per second. These are multiprocessor, parallel systems suitable for specialized complex scientific applications involving huge amounts of mathematical applications such as weather forecasting. The main difference between a supercomputer and a mainframe is that a supercomputer executes fewer programs as fast as possible whereas a mainframe executes many programs concurrently.

EXERCISE

Answer the following questions

a) Name at least four early calculating devices.

b) Name the first operational general purpose computer.

c) Who first proposed the concept of 'Stored Program Computer'?

d) Define the IPO cycle.

e) Differentiate between data and information.

f) Explain the Von Neumann Computer.

g) Compare the salient features of first and second generation computers.

h) Why is Charles Babbage known as the Father of Modern Computers?

i) Explain the functional components of a computer with the help of a block diagram.

j) What are the functions of the control unit?

k) Where are the instructions needed to start a computer stored?

l) Explain booting process and its types.

m) Differentiate between :

   a) Digital computers and analog computers.

   b) Microcomputers and Mini Computers

Chapter 2

Software Concepts

After studying this chapter the student will be able to:

- *Learn different types of Software*
- *System Software (Operating system, Language Processors)*
- *Utility Software (Antivirus, Compression tools, Backup, Disk Defragmentor)*
- *Application Software (General Purpose and Customized)*
- *Study the need, functions and types of operating system*
- *Study some commonly used operating systems- UNIX, LINUX, Windows, Solaris, BOSS*
- *Study mobile operating systems – Android and Symbian*
- *Understand Open Source Concepts - Open Source Software, Freeware, Shareware, Proprietary Software.*

Hardware and Software

A computer consists of both hardware and software and both are equally important for the working of the computer system. The electronic components of a computer system that we can see and touch are called hardware. Software is a general term used for computer programs that control the operations of the computer. A program is a sequence of instructions that perform a particular task. A set of programs form a software. It is the software which gives hardware its capability. Hardware is of no use without software and software cannot be used without hardware.

Types of Software

Software can be broadly are categorized as: (Fig 1)

- System Software
- Application Software
- Utility Software

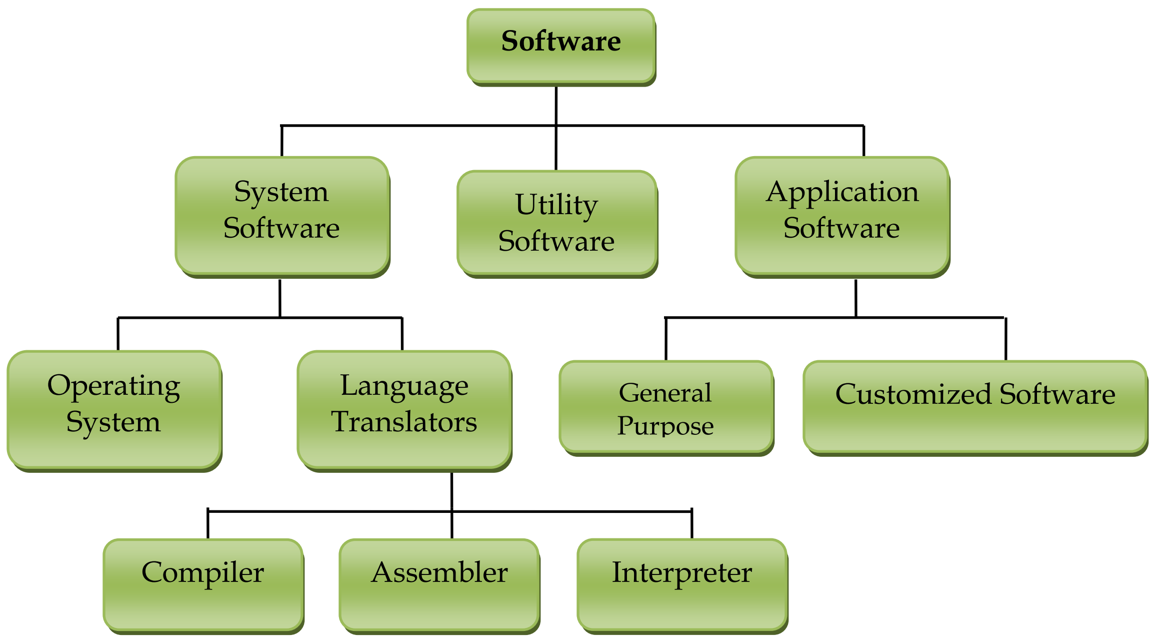


Fig: Types of Software

System Software

System Software is the software that is directly related to coordinating computer operations and performs tasks associated with controlling and utilizing computer hardware. These programs assist in running application programs and are designed to control the operation of a computer system. System software directs the computer what to do, when to do and how to do. System software can be further categorized into

- Operating System
- Language Translators

Operating System

An Operating system is the most important system software. It is a set of programs that control and supervise the hardware of a computer and also provide services to application software, programmers and users. It manages all hardware and software, input, output and processing activities within the computer system, the flow of information to and from the processor, sets priorities for handling different tasks, and so on. Without operating system a computer cannot do anything useful. When a computer is switched on, the operating system is the first program that is loaded onto its memory. A user cannot communicate directly with the computer hardware, so the operating system acts as an interface between the user and the computer hardware.

Some of the popular operating systems used in personal computers are DOS, Windows, Unix, Linux, Solaris, etc.

An operating system can be a Single User or a Multiuser operating system. A single user operating system allows only one user to work at any time but a multiuser operating system allows two or more users to use a powerful computer at the same time. For example Windows 7 is a single user operating system while Linux is a multiuser operating system.

Need for an Operating System

Operating system provides a platform, on top of which, other programs, called application programs can run. As discussed before, it acts as an interface between the computer and the user. It is designed in such a manner that it operates, controls and executes various applications on the computer. It also allows the computer to manage its own resources such as memory, monitor, keyboard, printer etc.

Our choice of operating system, therefore, depends to a great extent on the CPU and the other attached devices and the applications we want to run. The operating system controls the various system hardware and software resources and allocates them to the users or programs as per their requirement.

Functions of an Operating System

An operating system has variety of functions to perform. Some of the prominent functions of an operating system can be broadly outlined as:

- **Processor Management:** This deals with management of the Central Processing Unit (CPU). The operating system takes care of the allotment of CPU time to different processes. This is called **scheduling**. Two types of scheduling techniques are employed by an operating system :
    - **Priority Scheduling:** Each task is given CPU time according to the priority assigned to that task. The program with higher priority will be given CPU time before a program with lower priority. The CPU executes the task till it is completed or there is some interrupt request i.e. till the time operating

system has to stop (interrupt) the current task due to an unavoidable job request. The major drawback of Priority scheduling is that even a small job has to wait for a long time when a long duration job with higher priority is being executed.

- **Round Robin Scheduling:** This type of scheduling technique is also known as Time Sharing Scheduling. In this, each program or task is given a fixed amount of time to execute. The CPU continues with the execution till either the allotted time is over or there is some interrupt request or the task is completed before the allotted time. If the task is not completed at the end of the allotted time, it is put at the end of the queue. So each task gets its allotted share of CPU time. This scheduling technique improves the response time and provides an interactive environment. Hence time sharing operating system is very useful in network environment as each user is allowed to share the network resources.

✡ **Device Management:** The Operating System communicates with hardware and the attached devices and maintains a balance between them and the CPU. This is all the more important because the CPU processing speed is much higher than that of I/O devices. In order to optimize the CPU time, the operating system employs two techniques - Buffering and Spooling.

- **Buffering:** In this technique the temporary storage of input and output data is done in Input Buffer and Output Buffer. Once the signal for input or output is sent to or from the CPU respectively, the operating system through the device controller moves the data from the input device to the input buffer and for the output device to the output buffer. When the signal is sent to/from the operating system to the respective device controllers, the program doesn't wait rather it returns to its processing. In case of input, if the buffer is full, the operating system sends a signal to the program which processes the data stored in the buffer. When the buffer becomes empty, the program informs the operating system which reloads the buffer and the input operation continues. Similarly for output when the program being executed has to display some output, it fills the buffer and then informs the operating system. Thereafter the operating system empties the buffer by sending data to the output device and in the meantime the program fills another buffer. This technique is called **overlapped processing**. This is because while the operating system reloads one buffer, the executing program doesn't stop as it is able to retrieve/fill data from/in another buffer.

- **Spooling (Simultaneous Peripheral Operation on Line):** This is a device management technique used for processing of different tasks on the same input/output device. Say for example there are various users on a network sharing the same printer. At one point of time more than one user might give print command. The speed of the printer is very slow as compared to the CPU processing. So the operating system temporarily stores the data of every user on the hard disk of the computer to which the printer is attached. The individual users need not wait for the printing process to be complete. Instead the operating system sends the data from to hard disk to the printer one by one.

✡ **Memory management:** In a computer, both the CPU and the I/O devices interact with the memory. When a program needs to be executed it is loaded onto the main memory till the execution is complete. Thereafter that memory space is freed and is available for other programs. The common memory management techniques used by the operating system are Partitioning and Virtual Memory.

- **Partitioning:** The total memory is divided into various partitions of same size or different sizes. This helps to accommodate number of programs in the memory. The partition can be fixed i.e. remains same for all the programs in the memory or variable i.e. memory is allocated when a program is loaded on to the memory. The later approach causes less wastage of memory but in due course of time, it may become fragmented.

- **Virtual Memory:** This is a technique used by the operating system by virtue of which the user can load the programs which are larger than the main memory of the computer. In this technique the program is executed even if the complete program is not loaded on to the main memory. The operating system divides the main memory into equal sizes called pages. A part of the program resides in the main memory and is called the **active set.** The rest is in the secondary storage device in the form of tracks/sectors or blocks. With the help of **Page Map Tables (PMT)**, the operating system keeps track which page of main memory is storing which block of secondary memory. A virtual address (which is not the real physical address) is mapped either to the main memory or the secondary memory. Hence virtual memory allows more programs and even larger programs to be executed in the main memory leading to efficient memory utilization.

- **File Management:** The operating System manages the files, folders and directory systems on a computer**.** Any data on a computer is stored in the form of files and the operating system keeps information about all of them using **File Allocation Table (FAT).** The FAT stores general information about files like filename, type (text or binary), size, starting address and access mode (sequential/indexed sequential/direct/relative). The file manager of the operating system helps to create, edit, copy, allocate memory to the files and also updates the FAT. The operating system also takes care that files are opened with proper access rights to read or edit them.

Types of Operating System

OS are classified into the following types depending on their capability of processing

- **Single User** and **Single Task OS:** It is used on a standalone single computer for performing a single task. Operating systems for Personal Computers (PC) are single-user OS. Single user OS are simple operating system designed to manage one task at a time. MS-DOS is an example of single user OS.

- **Multiuser OS** is used in mini computers or mainframes that allow same data and applications to be accessed by multiple users at the same time. The users can also communicate with each other. Linux and UNIX are examples of multiuser OS.

- **Multiprocessing OS** have two or more processors for a single running process. Processing takes place in parallel and is also called *parallel processing*. Each processor works on different parts of the same task, or, on two or more different tasks. Since execution takes place in parallel, they are used for high speed execution, and to increase the power of computer. Linux, UNIX and Windows 7 are examples of multiprocessing OS.

- **Time sharing Operating System:** It allows execution of more than one tasks or processes concurrently. For this, the processor time is divided amongst different tasks. This division of time is also called **time sharing**. The processor switches rapidly between various processes. After the stipulated time is over, the CPU shifts to next task in waiting, So this type of operating system employs **round robin scheduling technique.** The system switches rapidly from one user to another but still each user feels that it is getting a dedicated CPU time. Virtual Memory techniques are used in this type of operating system. For example, the user can listen to music on the computer while writing an article using a word processing software. The user can switch between the applications and also transfer data between them. Time sharing operating system can be both single user and multiuser. Windows 95 and all later versions of Windows are examples of multitasking OS.

- **Real Time Operating System:** It is a multitasking operating system designed for real time applications like robotics. In this type of operating system, the tasks have to be done within a fixed deadline. System performance is good if task is finished within this deadline. If it is not done, the situation is called Deadline Overrun. Lesser the deadline over run, better is the system efficiency. Hence Real Time operating systems depend not only on the logical result of the computation but also on the time in which the results are produced.

- **Distributed Operating System:** On a network data is stored and processed on multiple locations. The Distributed Operating System is used on networks as it allows shared data/files to be accessed from any machine on the network in a transparent manner. We can insert and remove the data and can even access all the input and output devices. The users feel as if all data is available on their workstation itself.

- **Interactive Operating System:** This is the operating system that provides a Graphic User Interface (GUI) through which the user can easily navigate and interact. The computer responds almost immediately after an instruction has been entered, and the user can enter new instructions after seeing the results of the previous instructions.

Commonly Used Operating Systems

Some of the commonly used operating systems are discussed below:

1. **Windows:** Microsoft launched Windows 1.0 operating system in 1985 and since then Windows has ruled the world's software market. It is a GUI (Graphic User Interface) and various versions of Windows have been launched like Windows 95, Windows 98, Win NT, Windows XP, Windows 7 and the latest being Windows 8.

2. **Linux:** Linux is a free and open software which means it is freely available for use and since its source code is also available so anybody can use it, modify it and redistribute it. It can be downloaded from www.linux.org. It is a very popular operating system used and supported by many companies. The defining component of this operating system is the Linux kernel.

3. **BOSS (Bharat Operating System Solutions):** This is an Indian distribution of GNU/Linux. It consists of Linux operating system kernel, office application suite, Bharateeya OO, Internet browser (Firefox), multimedia applications and file sharing.

4. **UNIX:** It is a multitasking, multiuser operating system originally developed in 1969 at Bell Labs. It was one of the first operating systems developed in a high level language, namely C. Due to its portability, flexibility and power, UNIX is widely being used in a networked environment. Today, "UNIX" and "Single UNIX Specification" interface are owned and trademarked by The Open Group. There are many different varieties of UNIX, although they share common similarities, the most popular being GNU/Linux and Mac OS X.

5. **Solaris:** It is a free Unix based operating system introduced by Sun Microsystems in 1992. It is now also known as Oracle Solaris. Solaris is registered as compliant with Single UNIX Specification. It is quite scalable and is used on virtual machines.

Mobile Operating Systems (Mobile OS)

It is the operating system that operates on digital mobile devices like smart phones and tablets. It extends the features of a normal operating system for personal computers so as to include touch screen, Bluetooth, WiFi, GPS mobile navigation, camera, music player and many more. The most commonly used mobile operating systems are – Android and Symbian

Android: It is a Linux derived Mobile OS released on 5th November 2007 and by 2011 it had more than 50% of the global Smartphone market share. It is Google's open and free software that includes an operating system, middleware and some key applications for use on mobile devices. Android applications are quiet user friendly and even one can easily customize the Smartphone with Android OS. Various versions of Android OS have been released like 1.0, 1.5, 1.6, 2. x, 3.0 etc. Most Android phones use the 2.x release while Android 3.0 is available only for tablets. The latest Android version released is 4.2.2. The Android releases have dessert inspired codenames like Cupcake, Honeycomb, Ice Cream sandwich and Jelly Bean.

Symbian: This Mobile OS by Nokia (currently being maintained by Accenture) designed for smartphones. It offers high level of functional integration between communication and personal information management. It has an integrated mail box and it completely facilitates the usage of all Google applications in your smartphone easily. Symbian applications are easy to shut down as compared to Android applications. Various versions like S60 series, S80 series, S90 series, Symbian Anna etc have been released. The latest Symbian releases (Symbian Belle) can support 48 languages.

Language Processors

We know that computer understands instructions in machine code, i.e. in the form of 0s and 1s. It is difficult for us to write computer program directly in machine code. The programs are written mostly in high-level languages, i.e. BASIC, C++, Python etc. A program written in any high-level programming language (or written in assembly language) is called the Source Program or Source Code.

The source code cannot be executed directly by the computer. The source code must be converted into machine language to be executed. The program translated into machine code is known as Object Program or Object code.

The special translator system software that is used to translate the program written in high-level language (or Assembly language) into machine code is called language processor or translator program.

The language processors can be any of the following three types- Assembler, Compiler and Interpreter.

Assembler

The Assembler is used to translate the program written in Assembly language into machine code. The input of Assembler is a source program that contains assembly language instructions. The output generated by assembler is the object code or machine code understandable by the computer.

Compiler

The language processor that translates the complete source program as a whole in one go into machine code is called compiler. Some of the examples are C and C++ compilers.

The program translated into machine code is called the object program. The source code is translated to object code successfully if it is free of errors. If there are any errors in the source code, the compiler specifies the errors at the end of compilation with line numbers. The errors must be removed before the compiler can successfully recompile the source code again.

Interpreter

The language processor that translates a single statement of source program into machine code and executes it immediately before moving on to the next line is called an Interpreter. If there is an error in the statement the interpreter terminates its translating process at that statement and displays an error message.

Only after removal of the error, the interpreter moves on to the next line for execution.

Utilities

A utility software is one which provides certain tasks that help in proper maintenance of the computer. The job of utility programs is to keep the computer system running smoothly. Nowadays many utility softwares are part of the operating system itself. Even if there is no utility software on your computer, the computer works but with the right kind of utility software loaded, the computer becomes more reliable and even its processing speed increases. Some of the commonly use utility softwares are antivirus, Disk defragmenter, backup, compression etc.

✡ Antivirus

An antivirus is utility software which detects and removes computer viruses. If the software is not able to remove the virus, it is neutralized. The antivirus keeps a watch on the functioning of the computer system. If a virus is found it may alert the user, flag the infected program or kill the virus. Some of the common types of viruses are:

- **Boot Sector Virus:** A boot sector virus displaces the boot record and copies itself to the boot sector i.e. where the program to boot the machine is stored. So first the virus is loaded on to the main memory and then the operating system. Whenever a new disk is inserted the virus copies itself to the new disk. The antivirus overwrites the correct boot record on the infected boot sector and also cleans the bad sectors.

- **File Virus:** A file virus generally attacks executable files. They can attach to various locations of the original file, replace code, fill in open spaces in the code, or create companion files to work with an executable file.   Most of the file viruses are memory resident and wait in the memory until the user runs another program. While another program is running, the virus replicates.

- **Macro Virus:** This virus infects an important file called normal.dot of MS Word. As soon as the application is opened the virus gets activated. It damages the formatting of documents and even may not allow editing or saving of documents.

- **Trojan Horse:** It is a code generally hidden in games or spreadsheets. Since they are hidden, the program seems to function as the user wants but actually it is destroying the program. A Trojan horse does not require a host program to embed itself. It is a complete program. Its main objective is to cause harm to the data. They can create bad sectors on the disk, destroy file allocation tables and cause the system to hang.

- **Worm:** Worm is a program capable of replicating itself on a computer network. A worm also does not require a host as it is a self contained program. They generally travel from one computer to another across communication links on a network. They generally disrupt routine services.

- **Disk Defragmenter**

  The memory is used in small chunks randomly. Sometimes when a memory chunk of appropriate size is not available, the operating system breaks or fragments the files resulting in slower access to files. A disk defragmenter scans the hard disk for fragmented files and brings all the fragments together.

- **Backup Utility**

  This utility is used to create the copy of the complete or partial data stored in a disk or CD on any other disk. In case the hard disk crashes or some other system failure occurs, the files can be restored using backup software.

- **Compression Utility**

  This utility is used to compress large files. Compression is useful because it helps reduce resources usage and the file transmission on the network becomes easier.

- **Disk Cleaner**

  This utility scans for file that have not been accessed/used since long. Such files might be occupying huge amount of memory space. In that case the Disk Cleaner utility prompts the user to delete such files so as to create more space on the disk. If the files are important, the user might take a backup before deleting them.

- **File Management Tools**

  This utility helps the user in storing, indexing, searching and sorting files and folders on the system. The most commonly used tool is the Windows Explorer and Google Desktop.

Application Software

An application software is bought by the user to perform specific applications or tasks, say for example making a document or making a presentation or handling inventory or managing the employee database. An application software can be of two types – General Purpose Application Software and Customized Application software.

General Purpose Application Software

Some of the application software is made for the common users for day to day applications and uses. These are also referred as Office Tools. The users may use them in the manner they want. Some of the popular types of general purpose application software are discussed below:

- **Word Processor:** Word processor is a general purpose application software used to create documents. It allows us to create , edit and format documents. We can use different types of fonts of various sizes; underline or make bold a certain part of the text. We can add clipart and other graphics into the document. Popular examples of Word processing software are Writer (Open Office) and Microsoft Word.

  We use word processing software for various uses like writing a simple document to designing special art effects. Since we can attach images and different shapes, can use different colors, even a poster can be designed using word processing software. Features like Mail Merge, Macro has further enhanced the word processing software and made it very useful.

- **Presentation Tools:** Presentation tools is a general purpose application software that lets us create presentations on any topic. We can not only create a presentation and add slides into that but also can use different types of background, fonts, animations, audio, video, etc. We can add clipart and other graphics into our document. Even audio video files can be added on to the presentations. Popular examples of Presentation tools software are Impress (open office) and Microsoft Power Point.

- **Spreadsheet Packages:** Spreadsheet is a general purpose application software that lets us create and store data in tabular form. Both text and numerical values can be entered in that tables known as a spreadsheet. We can not only create a document and add data into that but also can create different types of charts and graphs based upon the numerical data stored in that page. All common mathematical and statistical formulae can be used on the numeric data. Popular examples of Spreadsheet software are Calc (Open Office) and Microsoft Excel.

- **Database Management System:** Database Management System is general purpose application software that lets us create computer programs that control the creation, maintenance, and the use of database for an organization and its end users. We can not only store data but can also manage data in a database. We can also import and export the data to many formats including Excel, Outlook, ASCII, dBase, FoxPro, Oracle, SQL Server, ODBC, etc. Popular examples of Database Management System are Base (Open Office) and Microsoft Access.

Customized Software

Customized Software is one which is tailor made as per the user's requirement. Such type of software is customer specific. It is made keeping in mind the individual needs of the user and so are also referred as Domain Specific Tools. Such software cannot be installed and used by any other user/customer since the requirements may differ. Some examples of customized software are discussed below:

- **Inventory Management System & Purchasing System:** Inventory Management System is generally used in departmental stores or other organizations to keep the record of the stock of all the physical resources. For Example, in a Computer store, it keeps record of the number of computers, printers, printing sheet, printer cartridge available. It also helps to place purchase orders, bills, invoices etc. Various reports as to position of stock, sales made in a particular period, profit earned etc. can be generated.

- **School Management System:** School Management System (sometimes called a School Information System or SIS) is a system that manages all of a school's data in a single, integrated application. Having all of the information in a single system allows schools to more easily connect data together. For example, when viewing a student's record, the user can follow a link to the student's class, and from there a link to the student's teacher, and from there a link to the teacher's other classes, and so on

- **Payroll System:** Payroll Management System software is used by all modern organizations to keep track of employees of the organization who receives wages or salary. All different payment amounts are calculated by the payroll software and the record is maintained. The software keeps track of personal records of employees viz. name, address, date of birth, qualification, date of joining etc. It also keeps track of professional record viz. allowances, perks, income tax, insurance etc. Different reports, pay slips etc can be generated through this software.

- **Financial Accounting:** Financial accounting System is used to prepare accounting information, maintain different accounts ledger, and account books. It also helps an organization to make budget.

- **Hotel Management:** Hotel management software refers to management techniques used in the hotel sector. These can include hotel administration, accounts, billing, marketing, housekeeping, front office or front desk, food and beverage management, catering and maintenance. Even advance bookings can be made through this software. Customers can have a look at the hotel and the rooms before making bookings. At any point of time the room availability, tariff for each type of room and even booking status can be checked.

- **Reservation System:** Reservation System is software used to book (reserve) air flights, railway seats, movie tickets, tables in a restaurant, etc. In the case of a booking system, the inputs are booking requests. The processing involves checking if bookings are possible, and if so making the bookings. The outputs are booking confirmations/rejections.

- **Weather Forecasting system:** This software makes it possible to forecast the weather for days and even months in advance. The detailed weather reports can also be generated.

Open Source Concepts

Software are mainly categorised into the following categories based on their licenses:

1. Proprietary
2. Shareware
3. Freeware
4. Open source
5. Free Software

Proprietary

We pay a supplier for a copy of the software which these days may be supplied on physical media (disks) or downloaded from the Internet. We get the permission to use the software on one or sometimes more than one machines. Examples of this type of software include Microsoft Office and Microsoft Windows.

✡ Shareware

Shareware is basically a software for trial purpose that the user is allowed to try for free, for a specified period of time. It is usually downloaded from the Internet. When the trial period ends, the software must be purchased or uninstalled.

✡ Freeware

Freeware software is free of cost and is usually bundled up with some operating system or any other software. Examples of freeware include Microsoft Internet Explorer which comes bundled up with any Microsoft operating system. The author of the freeware software is the owner of the software, though people may use it for free. The source code is not available, so no modifications can be done. Freeware should not be mistaken with Open Source Software or Free Software.

✡ Open source

Open Source Software (OSS) is the software which gives the users freedom to run/use the software for any purpose and in any manner. They can be used, modified and even redistributed. In simple terms it can be freely used but it may not be free of charge. The source code is freely available to the customer. Python, Tux Paint etc are examples of Open Source Software.

✡ Free Software

This type of software is freely accessible and can be freely used, modified, copied or distributed by anyone. And no licence fee or any other form of payment need to be made for a free software. The source code is also accessible in case of free softwares.

EXERCISE

Answer the following questions

a) 'Hardware is of no use without software and software cannot be used without hardware.' Explain.

b) How can the software be classified? Name at least one software in each of the categories.

c) What is an operating system? Write names of any two popular operating systems.

d) What is the role of a Page Map Table in Virtual Memory Management?

e) Explain the major functions of an operating system.

f) What is the purpose of a language processor?

g) Differentiate between:

   (i) An interpreter and a compiler.

   (ii) Priority Scheduling and Round Robin Scheduling

   (iii) Buffering and SPOOLING

   (iv) Time Sharing and Real Time Operating System

h) Explain any two utilities.

i) What is word processing? Discuss the purpose of word processing software.

j) What is the difference between an Open source Software and a Freeware. Write 2 examples of each.

k) How are Freeware and Free Software different?

Chapter 3

Data Representation in Computers

After studying this chapter the student will be able to:

**Learn about binary, octal, decimal and hexadecimal number systems*

**Learn conversions between two different number systems*

**Understand internal storage encoding of characters: ASCII, ISCII and UNICODE*

Binary Representation of Data

In order to work with data, the data must be represented inside the computer. Digital computers represent data by means of an easily identified symbol called a digit.

Numbering Systems

Each number system has a base also called a Radix. A decimal number system is a system of base 10; binary is a system of base 2; octal is a system of base 8; and hexadecimal is a system of base 16. What are these varying bases? The answer lies in what happens when we count up to the maximum number that the numbering system allows. In base 10, we can count from 0 to 9, that is,10 digits.

Number System	Base	Symbols used
Binary	2	0,1
Octal	8	0,1,2,3,4,5,6,7
Decimal	10	0,1,2,3,4,5,6,7,8,9
Hexadecimal	16	0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F where A = 10; B = 11; C = 12; D = 13; E = 14; F = 15

Converting a number from one Base to another:

Binary to Decimal

Method to convert Binary to Decimal:

1. Start at the rightmost bit.
2. Take that bit and multiply by 2^n where n is the current position beginning at 0 and increasing by 1 each time. This represents a power of two.
3. Sum each terms of product until all bits have been used.

Example

Convert the Binary number 101011 to its Decimal equivalent.

$1 * 2^5 + 0 * 2^4 + 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 1 * 2^0$

$32 + 0 + 8 + 0 + 2 + 1 = (43)_{10}$

Example

Convert the Binary number 1001 to its Decimal equivalent.

$1 * 2^3 + 0 * 2^2 + 0 * 2^1 + 1 * 2^0$

$8 + 0 + 0 + 1 = (9)_{10}$

Binary fraction to decimal

Example

Convert $(11011.101)_2$ to decimal

2^4	2^3	2^2	2^1	.	2^0	2^{-1}	2^{-2}	2^{-3}
1	1	0	1		1	1	0	1

$= (1 \times 2^4) + (1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) + (1 \times 2^{-1}) + (0 \times 2^{-2}) + (1 \times 2^{-3})$

$= 16+8+0+2+1+0.5+0+0.125$

$= (27.625)_{10}$

Decimal to Binary

Method to convert a Decimal number into its Binary equivalent:

1. Divide the decimal number by 2.
2. Take the remainder and record it on the side.
3. Divide the quotient by 2.
4. REPEAT UNTIL the decimal number cannot be divided further.
5. Record the remainders in reverse order and you get the resultant binary number.

Example

Convert the Decimal number 125 into its Binary equivalent.

125 / 2 = 62	1
62 / 2 = 31	0
31 / 2 = 15	1
15 / 2 = 7	1
7 / 2 = 3	1
3 / 2 = 1	1
1 / 2 = 0	1

Answer: $(1111101)_2$

Converting Decimal fraction to Binary

Example

Convert $(0.75)_{10}$ to binary

Multiply the given fraction by 2. Keep the integer in the product as it is and multiply the new fraction in the product by 2. Continue the process till the required number of decimal places or till you get zero in the fraction part. Record the integers in the products from top to bottom.

Given fraction	0.75
Multiply 0.75 by 2	1.50
Multiply 0.50 by 2	1.00

Reading the integers from top to bottom 0.75 in decimal number system is 0.11 in binary number system.

Example

Convert $(105.15)_{10}$ to binary

Let us convert 105 first.

$(105)_{10} = (1101001)_2$

Let us convert $(0.15)_{10}$

Multiply 0.15 by 2	0.30
Multiply 0.30 by 2	0.60
Multiply 0.60 by 2	1.20
Multiply 0.20 by 2	0.40
Multiply 0.40 by 2	0.80
Multiply 0.80 by 2	1.60

Reading the integers from top to bottom $(0.15)_{10} = (0.001001)_2$

Final result $(105.15)_{10} = (1101001.001001)_2$

Decimal to Octal

The method to convert a decimal number into its octal equivalent:

1. Divide the decimal number by 8.
2. Take the remainder and record it on the side.
3. Divide the quotient by 8.
4. REPEAT UNTIL the decimal number cannot be divided further.
5. Record the remainders in reverse order and you get the resultant binary

Example

Convert the Decimal number 125 into its Octal equivalent.

125 / 8 = 15	5
15/ 8 = 1	7
1/8 =0	1

Answer: $(175)_8$

Converting Decimal fraction to Octal

Example

Convert $(0.75)_{10}$ to Octal

Multiply the given fraction by 8. Keep the integer in the product as it is and multiply the new fraction in the product by 8. Continue the process and read the integers in the products from top to bottom.

Given fraction	0.75
Multiply 0.75 by 8	6.00

Reading the integers from top to bottom 0.75 in decimal number system is 0.6 in octal number system.

Octal to Decimal

Method to convert Octal to Decimal:

1. Start at the rightmost bit.
2. Take that bit and multiply by 8^n where n is the current position beginning at 0 and increasing by 1 each time. This represents the power of 8.
3. Sum each of the product terms until all bits have been used.

Example

Convert the Octal number 321 to its Decimal equivalent.

$3 * 8^2 + 2 * 8^1 + 1 * 8^0$

$192+16+ 1 = (209)_{10}$

Octal fraction to decimal

Example

Convert $(23.25)_8$ to decimal

8^1	8^0 .	8^{-1}	8^{-2}
2	3	2	5

= $(2 \times 8^1) + (3 \times 8^0) + (2 \times 8^{-1}) + (5 \times 8^{-2})$

= 16+3+0.25+0.07812

= $(19.32812)_{10}$

Decimal to Hexadecimal

Method to convert a Decimal number into its Hexadecimal equivalent:

1. Divide the decimal number by 16.
2. Take the remainder and record it on the side.
3. REPEAT UNTIL the decimal number cannot be divided further.
4. Record the remainders in reverse order and you get the equivalent hexadecimal number.

Example

Convert the Decimal number 300 into its hexadecimal equivalent.

300 / 16 = 18	12-(C)
18 / 16 = 1	2
1 / 16 = 0	1

Answer: $(12C)_{16}$

Converting Decimal fraction to Hexadecimal

Example

Convert $(0.75)_{10}$ to hexadecimal

Multiply the given fraction by 16. Keep the integer in the product as it is and multiply the new fraction in the product by 16. Continue the process and read the integers in the products from top to bottom.

Given fraction	0.75
Multiply 0.75 by 16	12.00 - C

Reading the integers from top to bottom 0.75 in decimal number system is 0C in Hexadecimal number system.

Hexadecimal to Decimal

Method to convert Hexadecimal to Decimal:

1. Start at the rightmost bit.
2. Take that bit and multiply by 16^n where n is the current position beginning at 0 and increasing by 1 each time. This represents a power of 16.
3. Sum each terms of product until all bits have been used.

Example

Convert the Hexadecimal number AB to its Decimal equivalent.

$=A * 16^1 + B * 16^0$

$=10 * 16^1 + 11 * 16^0$

$=160+11 = (171)_{16}$

Hexadecimal fraction to decimal

Example

Convert $(1E.8C)_{16}$ to decimal

16^1	16^0	.	16^{-1}	16^{-2}
1	E		8	C

$= (1 \times 16^1)+ (14 \times 16^0)+ (8 \times 16^{-1})+ (12 \times 16^{-2})$

$= 16+14+0.5+0.04688$

$= (30.54688)_{10}$

Binary to Hexadecimal

The hexadecimal number system uses the digits 0 to 9 and A, B, C, D, E, F.

Method to convert a Binary number to its Hexadecimal equivalent is:

We take a binary number in groups of 4 and use the appropriate hexadecimal digit in it's place. We begin at the rightmost 4 bits. If we are not able to form a group of four, insert 0s to the left until we get all groups of 4 bits each. Write the hexadecimal equivalent of each group. Repeat the steps until all groups have been converted.

Example

Convert the binary number 1000101 to its Hexadecimal equivalent.

0100   0101   Note that we needed to insert a 0 to the left of 100.

4   5

Answer: $\mathbf{(45)_{16}}$

In case of a fractional binary number form groups of four bits on each side of decimal point. Then replace each group by its corresponding hexadecimal number.

Example

Convert $(11100.1010)_2$ to hexadecimal equivalent.

$\longleftarrow$ $\longrightarrow$

0001 1100   .   1010

1   C   .   A

Answer : $\mathbf{(1C.A)_{16}}$

Hexadecimal to Binary

Method to convert a Hexadecimal number to its Binary equivalent is:

Convert each digit of Hexadecimal Number to it's binary equivalent and write them in 4 bits. Then, combine each 4 bit binary number and that is the resulting answer.

Example

Convert the Hexadecimal number $(10AF)_{16}$ to its Binary equivalent.

  1   0   A   F

0001 | 0000 | 1010 | 1111

Answer: $\mathbf{(0001000010101111)_2}$

Example

Convert the Hexadecimal number $(A2F)_{16}$ to its Binary equivalent.

  A   2   F

1010 | 0010 | 1111

Answer: $\mathbf{(1010\ 0010\ 1111)_2}$

Binary to Octal and Octal to Binary

To convert Binary to Octal, as the octal system is a power of two (2^3), we can take the bits into groups of 3 and represent each group as an octal digit. The steps are the same for the binary to hexadecimal conversions except we are dealing with the octal base now.

To convert from octal to binary, we simply represent each octal digit in it's three bit binary form.

Example

Convert the Octal number $(742)_8$ to its Binary equivalent.

```
  7 |   4 |   2
111 | 100 | 010
```

Answer: $(111100010)_2$

Hexadecimal to Octal and Octal to Hexadecimal

To convert Hexadecimal to Octal, Convert each digit of Hexadecimal Number to it's binary equivalent and write them in 4 bits. Then, combine each 3 bit binary number and that is converted into octal.

Example

Convert the Hexadecimal number $(A42)_{16}$ to its Octal equivalent.

```
A | 4 | 2
1010 | 0100 | 0010
101 | 001 | 000 | 010
```

Answer: $(5102)_8$

To convert Octal to hexadecimal, convert each digit of Octal Number to it's binary equivalent and write them in 3 bits. Then, combine each 4 bit binary number and that is converted into hexadecimal.

Example

Convert the Octal number $(762)_8$ to its hexadecimal equivalent.

7 | 6 | 2

101 | 110 | 010

0001 | 0111 | 0010

Answer: $(172)_{16}$

The following table summarizes the number representation in decimal, binary, octal and hexadecimal number system:

Decimal	Binary	Octal	Hexadecimal
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Fig: Number Representation

Binary Representation of Integers

Binary number can be represented only by using 0's and 1's, but can not use the sign (-) to denote the negative number or sign (+) to denote the positive number. So it must be either 0 or 1. There are three methods to represent binary number. They are

(i) Sign and magnitude method

(ii) One's complement method

(iii) Two's complement method

Sign and magnitude method

In this method, first bit is considered as a sign bit. Here positive number starts with 0 and negative number starts with 1.

Example

```
              25
     25/2 = 12      1
      12/2 = 6      0
      6/2  = 3      0
     3/2   = 1      1
     1/2   = 0      1
```

So the binary number is $(11001)_2$. If we take the size of the word is 1 byte, then the number 25 will be represented as

00011001

Suppose, if the number is -25, and then it will be represented as

10011001

One's Complement Method

In this method, the positive number is represented as same as the binary number. If the number is negative, then we need to find one's complement of a binary number. The one's complement of a binary number will replace every 0 with 1 and vice- versa.

Example

(i) Represent 86 in one's complement method (I byte representation)

86/2	=43	0
43/2	=21	1
21/2	=10	1
10/2	=5	0
5/2	=2	1
2/2	=1	0
1/2	=0	1

The binary number is 1010110

1 byte representation of number 86 is 01010110

(ii) Represent -55 in one's complement method (I byte representation)

55/2	=27	1
27/2	=13	1
13/2	=6	1
6/2	=3	0
3/2	=1	1
1/2	=0	1

The binary number is 110111

1 byte representation is 00110111

The given number is negative; hence we need to calculate one's complement

One's complement of 00110111 is 11001000 (convert 1 into 0 and 0 into 1)

Thus, the 1 byte representation of number -55 is 11001000.

Two’s Complement method

In this method, the positive number is represented as the binary number. If the number is negative, then we need to calculate two’s complement of a binary number. The two’s complement of a binary number is calculated by adding 1 to its one’s complement.

Example

(i) Represent 87 in two’s complement method (I byte representation)

87/2	=43	1
43/2	=21	1
21/2	=10	1
10/2	=5	0
5/2	=2	1
2/2	=1	0
1/2	=0	1

The binary number is 1010111

Hence, the 1 byte representation of number 86 is 01010111

(ii) Represent -54 two’s complement method (I byte representation)

54/2	=27	0
27/2	=13	1
13/2	=6	1
6/2	=3	0
3/2	=1	1
1/2	=0	1

The binary number is 110110

Hence, the 1 byte representation is 00110110

The given number is negative; hence we need to calculate two’s complement.

One’s complement of 00110110 is 11001001 (convert 1 into 0 and 0 into 1)

Add 1 to one's complement

```
                 1
          11001001        (1+1=2, binary equivalent =11)
               + 1
          --------
          11001010
```

Thus, 1 byte representation of number -54 is 11001010

(iii) Represent -77 two's complement method (I byte representation)

77/2	=38	1
38/2	=19	0
19/2	=9	1
9/2	=4	1
4/2	=2	0
2/2	=1	0
1/2	=0	1

The binary number is 1001101

Hence, 1 byte representation is 01001101.

Since the given number is negative, we need to calculate two's complement.

One's complement of 01001101 is 10110010

```
10110010
     + 1
--------
10110011
```

Thus, 1 byte representation of number -77 is 10110011

Representing Characters

A computer can handle numeric and non numeric data like letters, punctuation marks and other special characters. Some predefined codes are used to represent numeric and non numeric characters. Some of the standards are discussed below:

ASCII

ASCII stands for American Standard Code for Information Interchange. ASCII-7 can represent 128 characters. Out of 7 bits, 3 are zone bits and 4 are numeric bits. ASCII-8 can represent 256 characters. It is an extended form of ASCII-7.

Character	ASCII-7	Code	ASCII-8	Code
	Zone	Digit	Zone	Digit
0	011	0000	0101	0000
1	011	0004	0101	0001
2	011	0010	0101	0010
3	011	0011	0101	0011
4	011	0100	0101	0100
5	011	0101	0101	0101
6	011	0110	0101	0110
7	011	0111	0101	0111
8	011	1000	0101	1000
9	011	1000	0101	1001
A	100	0001	1010	0001
B	100	0010	1010	0010
C	100	0011	1010	0011
D	100	0100	1010	0100
E	100	0101	1010	0101
F	100	0110	1010	0110
G	200	0111	1010	0111
H	100	1000	1010	1000
I	100	1001	1010	1001
J	100	1010	1010	1010
K	100	1011	1010	1011

L	100	1100	1010	1100
M	100	1101	1010	1101
N	100	1110	1010	1110
O	100	1111	1010	1111
P	101	0000	1011	0000
Q	101	0001	1011	0000
R	101	0010	1011	0010
S	101	0011	1011	0011
T	101	0100	1011	0100
U	101	0101	1011	0101
V	101	0110	1011	0110
W	101	0111	1011	0111
X	101	1000	1011	1000
Y	101	1001	1011	1001
Z	101	1010	1011	1010

ISCII: (Indian Standard Code for Information Interchange)

A lot of efforts have gone into facilitating the use of Indian languages on computers. In 1991, the Bureau of Indian Standards adopted the ISCII. It is an 8 bit code which allows English and Indian Scripts alphabets to be used simultaneously. Characters coded in ISCII need 8 bits for each character.

Unicode

Unicode is a new universal coding standard adopted by all new platforms. It is promoted by Unicode Consortium which is a non profit organization. Unicode provides a unique number for every character irrespective of the platform, program and the language. It is a character coding system designed to support the worldwide interchange, processing, and display of the written texts of the diverse languages.

EXERCISE

a) What does ASCII stand for?

b) What does the base of a Number system mean?

c) What is the base of Decimal, Binary, Octal and Hexadecimal number systems?

d) How many digits are there in a Binary number system?

e) Which digits are used in Hexadecimal number system?

f) What is Unicode? How is it useful?

g) Distinguish between ASCII and ISCII.

h) Do as directed :

- ✡ Convert the Decimal number 781 to its Binary equivalent.
- ✡ Convert Binary number 101101.001 to its decimal equivalent
- ✡ Convert Octal number 321.7 into its Binary equivalent
- ✡ Convert the Hexadecimal number 3BC into its Binary equivalent
- ✡ Convert the Binary number 10011010.010101 to its Hexadecimal equivalent
- ✡ Convert the Decimal number 345 into Octal number.
- ✡ Convert the Decimal number 736 into Hexadecimal number.
- ✡ Convert the Octal number 246.45 into Hexadecimal number.
- ✡ Convert the Hexadecimal number ABF.C into Octal number.
- ✡ Convert the Octal number 576 to Decimal.
- ✡ Convert the Hexadecimal number A5C1 to Decimal.

Chapter 4

Microprocessors and Memory Concepts

After studying this session students will be able to:

**Learn about a microprocessor and its characteristics*

**Classify the microprocessors – RISC, CISC and EPIC*

**Learn the various units of memory*

**Understand the memory types- Primary memory and Secondary memory*

**Learn about various input- output ports /connections*

Microprocessor

We studied that the Central processing unit processes data inside the computer. It interprets all the instructions given to it and carries out these instructions. A microprocessor is a Central Processing Unit (CPU) on a single chip. It is a multipurpose programmable device constructed using Metal Oxide Semiconductor (MOS) technology. In 1971 Intel Corporation fabricated the first microprocessor- 4004. It could do only add and subtract operations and that too it could process only 4 bits at a time. But Intel 4004 powered one of the first portable electronic calculators and since then the microprocessor technology has come a long way. Today we have processors with which can process upto 128 bits at a time at the speed of billion instructions per second.

Let us now have a look at the characteristics of microprocessors:

- **Instruction Set:** It is the set of instructions that the microprocessor executes
- **Word Length:** The number of bits processed in a single instruction is called word length or word size. The word size is directly proportional to the processing power of the CPU. During the processing, the internal general purpose registers hold data. So if internal registers can hold data upto 8 bits, the word length is 8 bits. If it can process 16 bits at a time, then the internal registers can hold upto 16 bits at a time and so on. Hence a 32 bit processor is about 4 times faster than an 8 bit processor. Examples of word lengths are 16 bit, 32 bit, 64 bit. The terms 16-bit CPU, 32-bit CPU, 64-bit CPU are used very often while talking about CPUs. Now

we know that these terms mean the maximum number of bits a given CPU can handle at a time.

- **System Clock Speed:** The microprocessor's pace is controlled by the System Clock. The System Clock is an electronic circuit that generates pulses which are measured in million of cycles per second (MHz). The number of pulses generated by the clock per unit of time is its Clock speed. Each microprocessor is characterized by its clock speed. Nowadays microprocessors have clock speed of several GHz. The CPU uses this clock speed to control sequencing and execution of various operations in the computer.

Classification of Microprocessors

Apart from the width of data (word length) that the microprocessors can process at a time, the classification is also based on the architecture i.e. Instruction Set of the microprocessor. While studying about CPUs, we come across two abbreviations CISC and RISC.

- **RISC:** It stands for Reduced Instruction Set Computer. It is a type of microprocessor architecture that uses a small set of instructions of uniform length. These are simple but primitive instructions which execute in one clock cycle. For this reason, RISC chips are less complex and also less expensive to produce. The instructions are of uniform length which interface with about 32-36 registers. The program size in case of RISC architecture is more but more memory cycles are needed to access data. To reduce the number of memory cycles, RISC keeps the necessary data in the processor itself. The drawback of RISC design is that the computer must combine or repeat operations to complete a large program consisting of many processing operations. Since instructions are simple , RISC processors are relatively simple to design. Examples of RISC processor is SPARC, POWER PC etc.

- **CISC:** It stands for Complex Instruction Set Computer. A CISC chip such as Intel Pentium provides programmers with hundreds of instructions of variable sizes, and the processing circuitry includes many special purpose circuits that carry out these instructions at high speeds. These instructions interface with memory in multiple mechanisms with complex addressing modes. In this case the program size is reduced and hence lesser number of memory cycles are required to execute the instruction. So fewer general purpose registers(8-12) are present in CISC processors. Also less number of memory cycles result in faster execution of the program.

- **EPIC:** It stands for Explicitly Parallel Instruction Computing. It is a computer architecture that combines the best feature of both RISC and CISC. It does not use instructions of any fixed length but rather aims at parallel processing of instructions. It uses a bundle of complex instructions that in addition to basic instruction also contain information on how to run the instruction in parallel with other instructions. This greatly increases the efficiency of an EPIC processor. IA-64 (Intel Architecture-64) is Intel's first 64 bit processor based on EPIC.

Memory Concepts

Memory is one of the most important components of a computer system as it stores data and instructions. Every memory chip contains thousands of memory locations. In the computer, the data is stored in the form of bits and bytes. A **bit** is the smallest storage unit of memory. A **nibble** is a collection of 4 bits. Eight bits combined together to form a single **byte**, which in turn represents a single character. Other units of memory are KB (Kilobyte), MB (Megabyte), GB (Gigabyte) ,TB(Terabytes), PB (Petabyte), EB (Exabytes), ZB (Zettabytes) and YB (Yottabytes). Every higher unit is equal to 2^{10} of the previous unit. The following table shows various units of computer memory:

Memory unit	Relationship with earlier memory unit	In equivalent Bytes
Kilo Byte (KB)	1 Kilo Byte = 1024 Bytes(or 2^{10} Bytes)	1024
Mega Byte (MB)	1 Mega Byte = 1024 Kilo Byte(or 2^{10} KB)	1024x1024
Giga Byte (GB)	1 Giga Byte = 1024 Mega Byte(or 2^{10} MB)	1024x1024x1024
Tera Byte (TB)	1 Tera Byte = 1024 Giga Byte(or 2^{10} GB)	1024x1024x1024x1024
Peta Byte (PB)	1 Peta Byte = 1024 Tera Byte(or 2^{10} TB)	1024x1024x1024x1024x 1024
Exa Byte(EB)	1 Exa Byte = 1024 Peta Byte(or 2^{10} PB)	1024x1024x1024x1024x 1024x1024

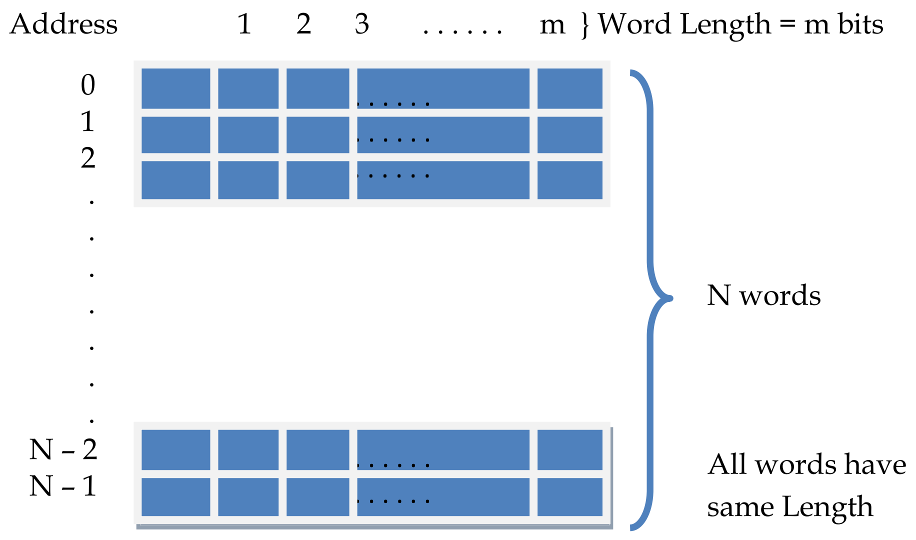
Zetta Byte(ZB)	1 Zetta Byte = 1024 Exa Byte(or 2^{10} EB)	1024x1024x1024x1024x 1024x1024x 1024
Yotta Byte(YB)	1 Yotta Byte = 1024 Zetta Byte(or 2^{10} ZB)	1024x1024x1024x1024x 1024x1024x 1024x1024

The computer memories can be divided into following categories:

- Primary Memory
- Secondary memory
- Cache Memory

Primary Memory

Primary memory or main memory is a Metal Oxide Semiconductor (MOS) memory used for storing program and data during the execution of the program. It is directly accessible to CPU.



Address
1 2 3 ...... m } Word Length = m bits
0
1
2
N – 2
N – 1
......
N words
All words have same Length

Fig: Organization of Main memory

The figure above shows a high speed main memory that is organised into words of fixed lengths. A given memory is divided into N words where each word is assigned an address in the memory. A word is generally more than 8 bits in length and the number of bits in a word is termed as **word length.** Computers with 8, 16, 24 and 32 and 64 bit word length are available. The higher the word length, the more powerful a computer is. Each word is assigned an address starting from 0 to the largest number that the computer can support. Each address uniquely specifies the memory location of a particular word. The total number of memory cells that can be uniquely addressed by CPU depends on the total number of address lines in an address bus. If there are n lines in the address bus then there are 2^n addressable locations in the memory.

Memories can be both read from and written to are called **read/write memories**. On the other hand memories that have data or program permanently stored onto them and hence can be only read from are called **Read Only memories**. Broadly primary memory can be of two types – RAM (Random Access Memory) and ROM (Read only memory).

Random Access Memory (RAM)

In case of RAM, the memory can be accessed from any desired location randomly. That means without searching the entire memory, any location can be accessed in the same amount of time. The instructions and data that we input into the computer are stored in the RAM of the Computer. It is a read/write memory, so data can be both read from and written to the RAM. It is a volatile memory and loses its contents when the power is switched off or interrupted. Nowadays RAMs are available in gigabytes. The normal memory access time of a RAM is 20-80 ns. RAM can be broadly classified into two categories: Dynamic RAM (DRAM) and Static RAM (SRAM).

Dynamic RAM (DRAM): It consists of a transistor and a capacitor that stores electric charge. The DRAMs are physically smaller, cheaper and slower memories. They are slower because the data stored in them needs to continuously refreshed and this consumes lot of processor time. Each refresh operation takes several CPU cycles to complete. This is because a capacitor tends to loose charge over a period of time which needs to be refreshed again and again. DRAM is used in primary storage areas and is available in various forms as EDORAM (**E**xtended **D**ata **O**utput RAM), SDRAM (**S**ynchronous DRAM) and DDR SDRAM.

Static RAM: This type of RAM is large in physical size but is an expensive and faster memory. It is faster because it is made up of flip flops to store data and these flip flops do not require any refreshing. Static RAM is also volatile and is easier to use as compared to dynamic RAM. These are used in specialized applications.

Read only memory (ROM)

As the name suggests, a ROM is a type of memory that can perform read operation only. The contents of ROM are written by the manufacturer and come along with the computer. We cannot change its contents or write something else on it. Data is written on to the ROM at the time of its manufacture and it cannot be changed thereafter. It is a non-volatile memory, which means that contents stored in it are not lost even when the power to the computer is switched off. ROMs are used in applications where the information once written, need not be altered. They hold certain essential instructions such as interrupt service routines or a monitor program controlling the machine. Instructions that are needed to start the computer are also stored in the ROM. ROMs are slower as compared to RAMs and are available in various types –

- **Programmable Read Only Memory (PROM):** This type of ROM can be programmed even after its manufacture using a PROM programmer circuit. But once a PROM is programmed, it becomes just like ROM i.e. it cannot be changed.
- **Erasable Programmable Read Only Memory (EPROM):** In this type of ROM, the contents can be erased and the memory can be reprogrammed. To erase the data, an EPROM is exposed to ultraviolet light and then it can be reprogrammed using a PROM programmer circuit. When the EPROM is in use, then it behaves like a ROM, that means the information can only be read.
- **Electrically Erasable Programmable Read Only Memory (EEPROM):** The contents of this type of ROM can be erased and then reprogrammed using electric signals. This makes it an excellent back up for RAM whose contents are lost when the power is switched off. When the power is returned, the contents of EEPROM are copied back into the RAM and the computer continues working without any data loss. Nowadays RAMs and EEPROMs are integrated in a single chip.

Cache Memory

Cache memory is a special high speed memory made up of high speed static RAMs. It is used to hold frequently accessed data and instructions. We know that the processing speed of CPU is much more than the main memory access time of the computer. This means the CPU has to wait for a substantial amount of time. Alternatively we have the cache memory which is a small, expensive but fast memory that is placed between the CPU and the main memory. Whenever some data is required, the CPU first looks into cache. If data is found, we call it a **cache hit** and the information is transferred to the CPU. In case of a miss, the main memory is accessed. Memory caching proves to be efficient because most programs repeatedly access the same data and instructions, so access of frequently used data becomes very fast with cache memory. There are two types of cache memory:

- **L1 cache:** It is small and is built inside the CPU. It is fast as compared to L2 cache
- **L2 cache:** It is large but slower and is mounted on the motherboard

Secondary Memory

The major limitation of primary memory is that it has limited storage capacity and is volatile. To overcome this limitation we have secondary memory storage devices. This type of memory is also called external memory. It refers to the various storage media on which a computer can store data and programs. It is an additional storage, not part of the main computer.

The Secondary storage media can be fixed or removable. *Fixed Storage media* is an internal storage medium like hard disk that is fixed inside the computer. *Removable storage media* is a data storage medium that is portable and can be taken outside the computer.

Why do we need Secondary Memory?

Secondary memory is needed because of the following reasons:

a. **For permanence:** As the RAM is volatile, i.e. it loses all information when the electricity is turned off, something is needed for permanence. Secondary storage devices serve this purpose. They do not lose data when electricity is turned off.

b. **For portability:** Secondary storage, like the CDs, flash drives can be used to transport data from one computer to another.

Secondary Storage Media

There are the following main types of storage media.

a. Magnetic b. Optical c. Solid State

Magnetic storage media: Examples of magnetic storage media are hard disks, floppy disks and magnetic tapes. Magnetic media is coated with a magnetic sensitive layer and this layer is magnetized in clockwise or anticlockwise directions, which then are interpreted as binary 1s and 0s at reading.

Floppy Disk (Diskette): A floppy disk is a flexible disk made up of mylar with a magnetic coating on it. It is packaged inside a protective plastic envelope. These were one of the oldest type of portable storage devices that could store up to 1.44 MB of data but now they are no longer in use.

Hard disk: A hard disk consists of one or more circular disks called platters which are mounted on a common spindle. Each surface of a platter is coated with a magnetic material. Both surfaces of each disk are capable of storing data except the top and bottom disk where only the inner surface is used. The information is recorded on the surface of the rotating disk by magnetic read/write heads. These heads are joined to a common arm known as access arm. This arm moves over the surface of the rotating disk as shown in the figure below.

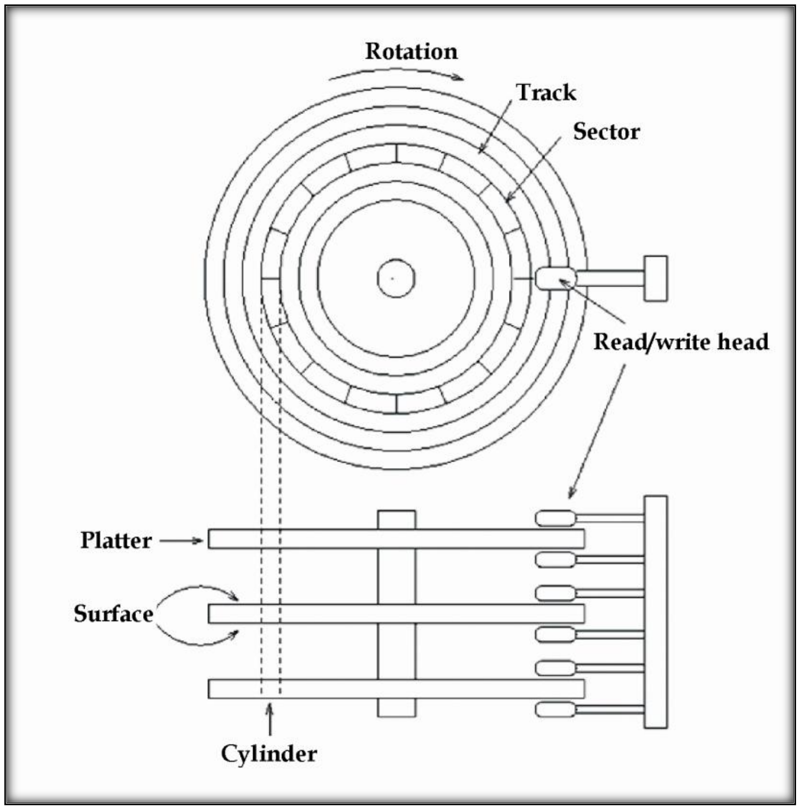


Fig: Hard Disk

Information is recorded on each of these disks in the form of concentric circles called tracks which are further divided into sectors. Hard drives however, are not very portable and are primarily used internally in a computer system. But external hard disks are also available as a substitute for portable storage. Today the hard disks have the storage capacity of several gigabytes to terabytes.

Optical storage media

On an optical storage media information is stored and read using a laser beam. The data is stored as a spiral pattern of pits and ridges denoting binary 0 and binary 1.

Examples of optical media are CDs, DVDs etc.

Compact Disk: A compact disk or CD can store approximately 650 to 700 megabytes (MB) of data. We must have a CD drive in our computer to read them.

Fig: A Compact Disk

The bits (0 and 1) are encoded as transitions between raised ridges and etched pits, which are lined up in a spiral like pattern. This pattern is then stamped into a 1.2-mm clear polycarbonate disc (a CD), which is then covered with a super thin coating of reflective metal (usually aluminum or gold) and a label.

To read the data, an infrared laser is beamed through the CD's polycarbonate substrate. The wavelength of light that bounces off the mirror-like reflective backing is then measured. A pit scatters the light and the ridge reflects the light. Since pits and ridges pass different amounts of light, the fluctuations in the reflected beam are then translated back into the original ones and zeros.

There are three types of CDs:

CD- ROM: It stands for Compact Disk - Read Only Memory and data is written on these disks at the time of manufacture. Thereafter this data cannot be changed but can only be read by a laser beam in the form of a continuous spiral. CD- ROMs are used for text, audio and video distribution like games, encyclopedias and application softwares.

CD-R: It stands for Compact Disk- Recordable. Data can be recorded on these disks but only once. So we can write data on these disks through a read/write CD drive but after that the disk cannot be erased/modified.

CD-RW: It stands for Compact Disk-Rewritable. It can be read or written multiple times. But a CD-RW drive needs to be installed on your computer.

DVD: It stands for Digital Versatile Disk or Digital Video Disk. It looks just like a CD and use a similar technology as that of the CDs discussed above but employ a shorter-wavelength red laser that permits a narrower beam. This allows tracks to be spaced closely enough to store data that is more than six times the CD's 700MB capacity. It is a significant advancement in portable storage technology. DVDs consist of two half-thickness (0.6-mm) CD-like discs glued back-to-back. This protects the delicate reflective coating as it is on the inside of the disc. Also it makes possible to have double-sided DVDs—where data can be stored on each half disc.

A DVD holds 4.7 GB to 17 GB of data. That means a complete movie can be stored on one side of a DVD. Like CDs DVDs also come in three varieties –

- DVD- ROM
- DVD- R
- DVD-RW

Blue Ray Disk: This is the latest optical storage media to store high definition audio and video. It looks like a CD or DVD but can store up to 27 GB of data on a single layer disk and up to 54 GB of data on a dual layer disk. Where CDs or DVDs use red laser beam, the blue ray disk uses a blue laser to read/write data on a disk.

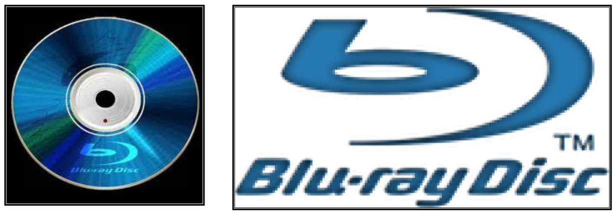


Fig: Blue ray Disk

As the wavelength of the blue ray is shorter, more data per unit area can be stored on the disk. This is because due to shorter wavelength, it is possible to focus the laser spot with greater precision. Hence data can be packed more tightly. Blue-ray Disc (BD) was developed to enable recording, rewriting and playback of high-definition video (HD), as well as storing large amounts of data.

Solid State Memories

The term 'solid-state' essentially means 'no moving parts'. Hence Solid-state storage devices are based on **electronic circuits** with **no moving parts** (no reels of tape, no spinning discs, no laser beams, etc.) Solid-state storage devices store data using a special type of memory called flash memory. SSD, Solid-state drive (or flash memory) is used mainly in digital cameras, pen drives or USB flash drives.

Pen Drives: Pen Drives or Thumb drives or Flash drives are the recently emerged portable storage media. It is an EEPROM based flash memory which can be repeatedly erased and written using electric signals. This memory is coupled with a USB connector through which it can be plugged into the computer. They have a capacity smaller than a hard disk but greater than a CD.

Fig : A pen Drive

Input Output Ports and Connections

Let us look at the back of a CPU. Computer ports are the points where external devices or peripherals connect to a computer. These ports are available at the rear or front of the computer. We connect peripherals to the computer with a cable that attaches to one of the ports.

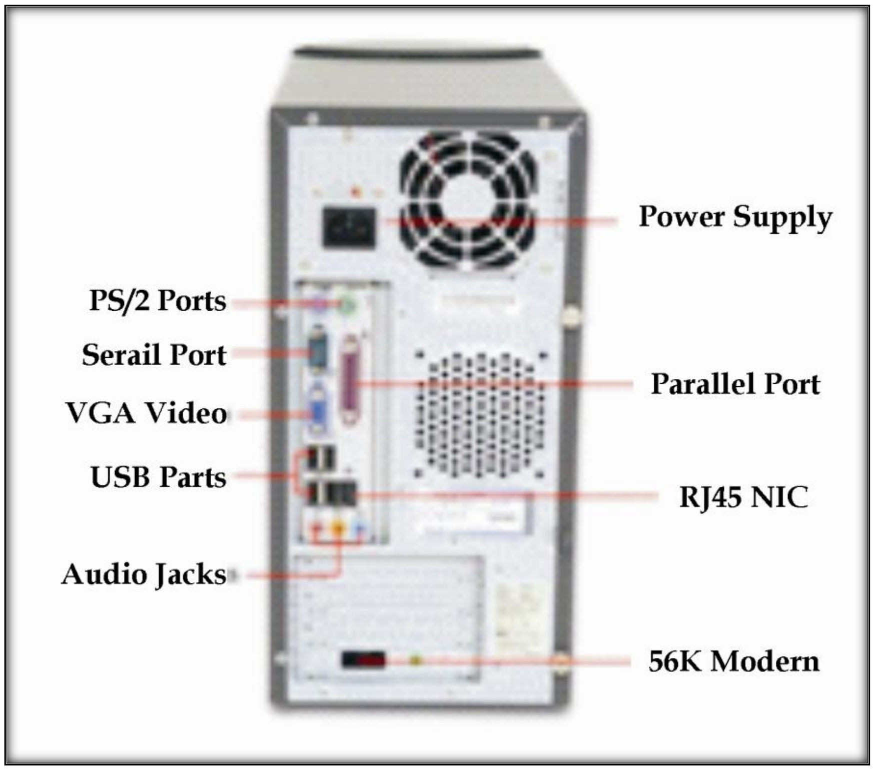


Fig: Ports at the back of CPU

A port's main function is to act as a point of attachment where the cable from the peripheral device plugs into the system unit, allowing data to flow from the peripheral device. Some of the common sockets/ports are power socket for connecting power cable, PS2 ports for connecting Mouse & Keyboard, USB Port for connecting USB devices such as mouse, keyboard, printer, pen drive etc. and VGA port for connecting Monitor/Screen.

Some ports are discussed in detail below:

Serial Port

Through a serial port data is transmitted travels one bit at a time through a single wire. The data transmission speed is quite slow. Serial ports are commonly known as communication (COM) ports or RS232C ports and connect devices like mouse and modem. These ports are rarely used these days.

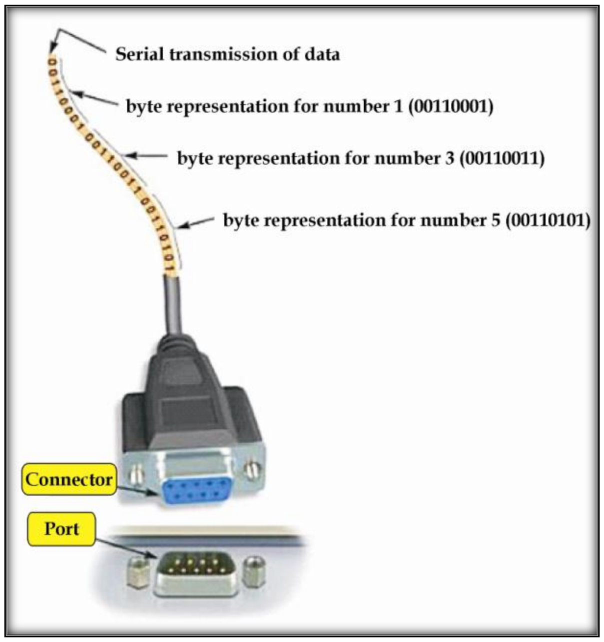

Serial transmission of data
byte representation for number 1 (00110001)
byte representation for number 3 (00110011)
byte representation for number 5 (00110101)
Connector
Port

Fig: Serial Port

Parallel port

These ports were earlier used to connect printers to the computer system. A parallel port can send 8 bits (1 byte) at a time simultaneously (in parallel). Hence data transmission is faster through these ports. Parallel ports are used to connect printers, scanners, CD writers etc.

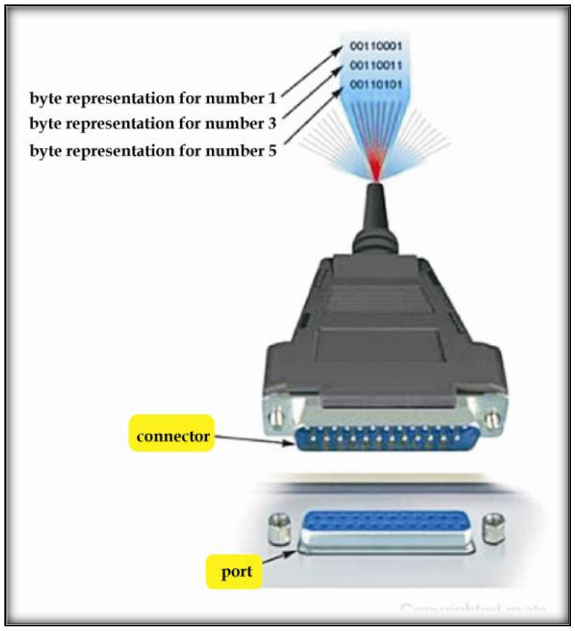

00110001
00110011
00110101
byte representation for number 1
byte representation for number 3
byte representation for number 5
connector
port

Fig: Parallel Port

PS/2 Port

This is a round port for plugging in keyboard or mouse. It has a PS/2 cable with a mini DIN connector. These ports are becoming obsolete now. In fact some systems these days do not have PS/2 ports.

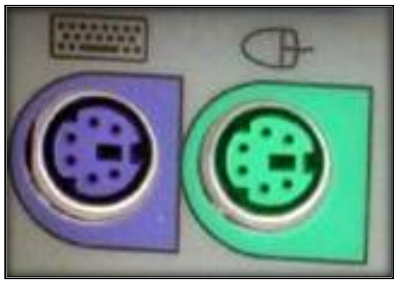


Fig: PS/2 port

USB Port

Fig: USB Port

A USB (Universal Serial Bus) port is a standard cable connection interface available on personal computers and some other electronic devices for data communication. It is a single, low cost, plug n play connector. The operating system automatically detects the device connected through the USB port. USB ports have become very popular these days as they connect many different devices to the computer these days. Most computers have USB ports on front, back and/or sides of system unit. Flash drives, digital cameras, printers, scanners are some of the devices that often connect through the USB port.

Infrared Port

In this type of port, data is transmitted through Infrared waves. For infrared transmission the device and the computer both must have infrared ports. These allow computers and peripherals to communicate serially over an Infrared link rather than over cables. The remote control of our TV sets uses the same technology. For wireless data communication between computers and various peripheral devices we use infrared ports.

Bluetooth Port

Bluetooth is used to connect mobile phones, computers and PDAs using a short range wireless connection. This technology uses radio waves to transmit data between any two devices. The devices that are Bluetooth enabled contain a small transceiver chip that allows them to communicate with other Bluetooth enabled computer or device. Data can be exchanged at the rate of about 2 megabit per second.

Fire wire Port

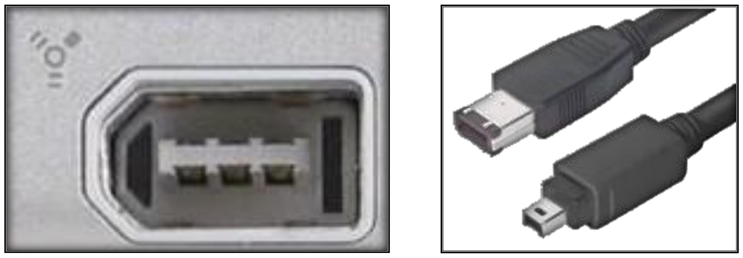


Fig: A Firewire port

FireWire® ports are forms of a serial port that make use of FireWire® technology to transfer data rapidly from one electronic device to another. The FireWire® port has the ability to interact with a number of different devices since it provides a single plug and socket connection for all devices. A FireWire® port can provide an ideal way to connect a scanner and digital camera/camcorder to a computer system as the data transfer is relatively faster than on USB and also results in excellent quality.

EXCERCISE

a) Define word length of a microprocessor.

b) Name the two types of Primary Memory.

c) What is the purpose of System Clock?

d) Differentiate between CISC and RISC processors.

e) Why do we use secondary storage? Name any two secondary storage devices.

f) How is Computer's internal memory important?

g) Why is it more appropriate to call RAM as Read-Write memory?

h) What is the purpose of Cache memory?

i) Explain in brief the different ports and their purposes.

j) Distinguish between the following pairs:

    a. Primary memory and Secondary memory

    c. RAM and ROM

    d. Bluetooth and Infrared port

UNIT 2

Programming Methodology

unit 2

Chapter 1

Algorithms and Flowcharts

After studying this lesson, the students will be able to

- ✡ *understand the need of Algorithm and Flowcharts;*
- ✡ *solve problems by using algorithms and flowcharts;*
- ✡ *get clear idea about sequential, selection and iteration construct; and*
- ✡ *understand the finite- and infinite- loop.*

Introduction

Algorithm is a step-by-step process of solving a well-defined computational problem. In practice, in order to solve any complex real life problems, first we have to define the problem and then, design algorithm to solve it. Writing and executing a simple program may be easy; however, for executing a bigger one, each part of the program must be well organized. In short, algorithms are used to simplify the program implementation. The next step is making the flowchart. It is a type of diagram that represents an algorithm or process, showing the steps as 'boxes' of various kinds and their order by connecting them with arrows. Then, the flowchart will be converted into program code.

Algorithm

An algorithm is an effective method expressed as a finite list of well defined instructions for calculating a function, starting from an initial state and initial input. The instructions describe a computation, which will eventually produce output, when executed. We can use algorithm to solve any kind of problems. However, before writing a program, we need to write the steps to solve the problem in simple English language. This step-by-step procedure to solve the problem is called algorithm.

Example

Let us take one simple day-to-day example by writing algorithm for making 'Maggi Noodles' as a food.

Step 1: Start

Step 2: Take pan with water

Step 3: Put pan on the burner

Step 4: Switch on the gas/burner

Step 5: Put magi and masala

Step 6: Give two minutes to boil

Step 7: Take off the pan

Step 8: Take out the magi with the help of fork/spoon

Step 9: Put the maggi on the plate and serve it

Step 10: Stop.

Further, the way of execution of the program shall be categorized into three ways: (i) sequence statements; (ii) selection statements; and (iii) iteration or looping statements. This is also called as 'control structure'.

Sequence statements: In this program**,** all the instructions are executed one after another.

Example

Write an algorithm to print 'Good Morning'.

Step 1: Start

Step 2: Print 'Good Morning'

Step 3: Stop

Example

Write an algorithm to find area of a rectangle.

Step 1: Start

Step 2: Take length and breadth and store them as L and B?

Step 3: Multiply by L and B and store it in area

```
Step 4: Print area

Step 5: Stop
```

In the above mentioned two examples (Example II and III), all the instructions are executed one after another. These examples are executed under sequential statement.

Selective Statements: In this program, some portion of the program is executed based upon the conditional test. If the conditional test is true, compiler will execute some part of the program, otherwise it will execute the other part of the program.

Example

Write an algorithm to check whether he is eligible to vote? (more than or equal to 18 years old).

```
Step 1: Start

Step 2: Take age and store it in age

Step 3: Check age value, if age >= 18 then go to step 4 else step 5

Step 4: Print "Eligible to vote" and go to step 6

Step 5: Print "Not eligible to vote"

Step 6: Stop
```

Example

Write an algorithm to check whether given number is +ve, -ve or zero.

```
Step 1: Start

Step 2: Take any number and store it in n.

Step 3: Check n value, if n > 0 then go to step 5 else go to step 4

Step 4: Check n value, if n < 0 then go to step 6 else go to step 7

Step 5: Print "Given number is +ve" and go to step 8

Step 6: Print "Given number is -ve" and go to step  8
```

Step 7: Print "Given number is zero"

Step 8: Stop

In the above mentioned examples IV and V, all the statements are not executed, but based upon the input, some portions of the algorithm are executed, because we have 'true' or 'false' situation in the program.

Iterative statements: In some programs, certain set of statements are executed again and again based upon conditional test. i.e. executed more than one time. This type of execution is called 'looping or iteration'.

Example

Write an algorithm to print all natural numbers up to 'n'.

Step 1: Start

Step 2: Take any number and store it in n.

Step 3: Store 1 in I

Step 4: Check I value, if I<=n then go to step 5 else go to step 8

Step 5: Print I

Step 6: Increment I value by 1

Step 5: Go to step 4

Step 8: Stop

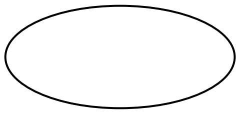
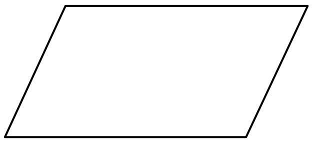
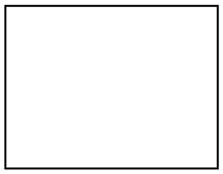
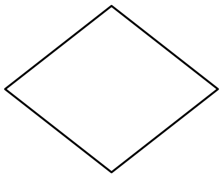
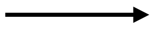
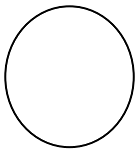
In the above example, steps 4, 5, 6 and 7 are executed more than one time.

Flowchart

In the previous section of this chapter, we have learnt to write algorithms, i.e. step-by-step process of solving a problem. We can also show these steps in graphical form by using some symbols. This is called flowcharting.

Flowchart Symbols

Some of the standard symbols along with respective function(s) that are used for making flowchart are as follows:

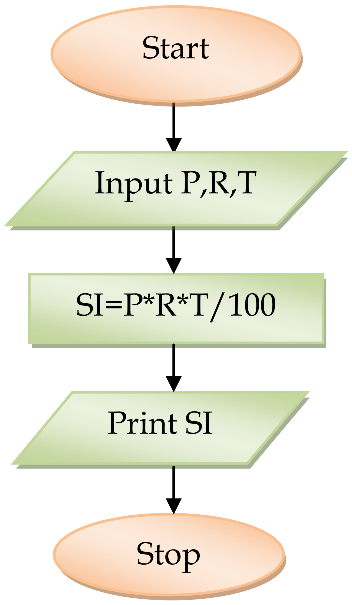
Symbols	Functions
1. 	Start/stop
2. 	Input/output
3. 	Processing
4. 	Decision Box
5. 	Flow of control
6. 	Connector

The following flowchart is an example of a sequential execution.

Example

Draw a flowchart to find the simple interest. (Sequence)

Solution:

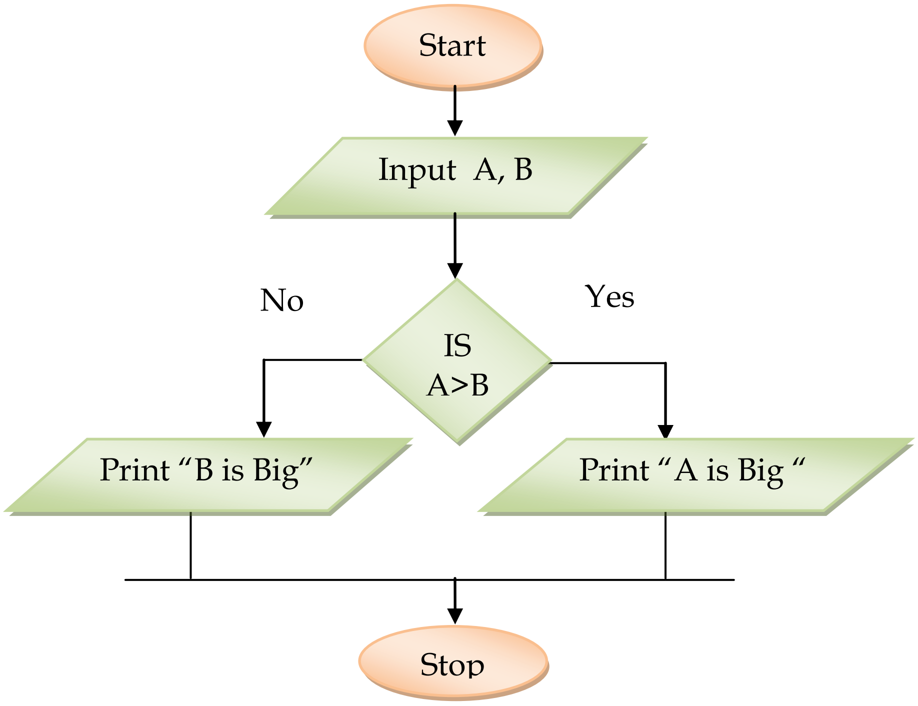


The following flowchart is an example of a selective execution.

Example

Draw a flowchart to find bigger number among two numbers (selective)

Solution:

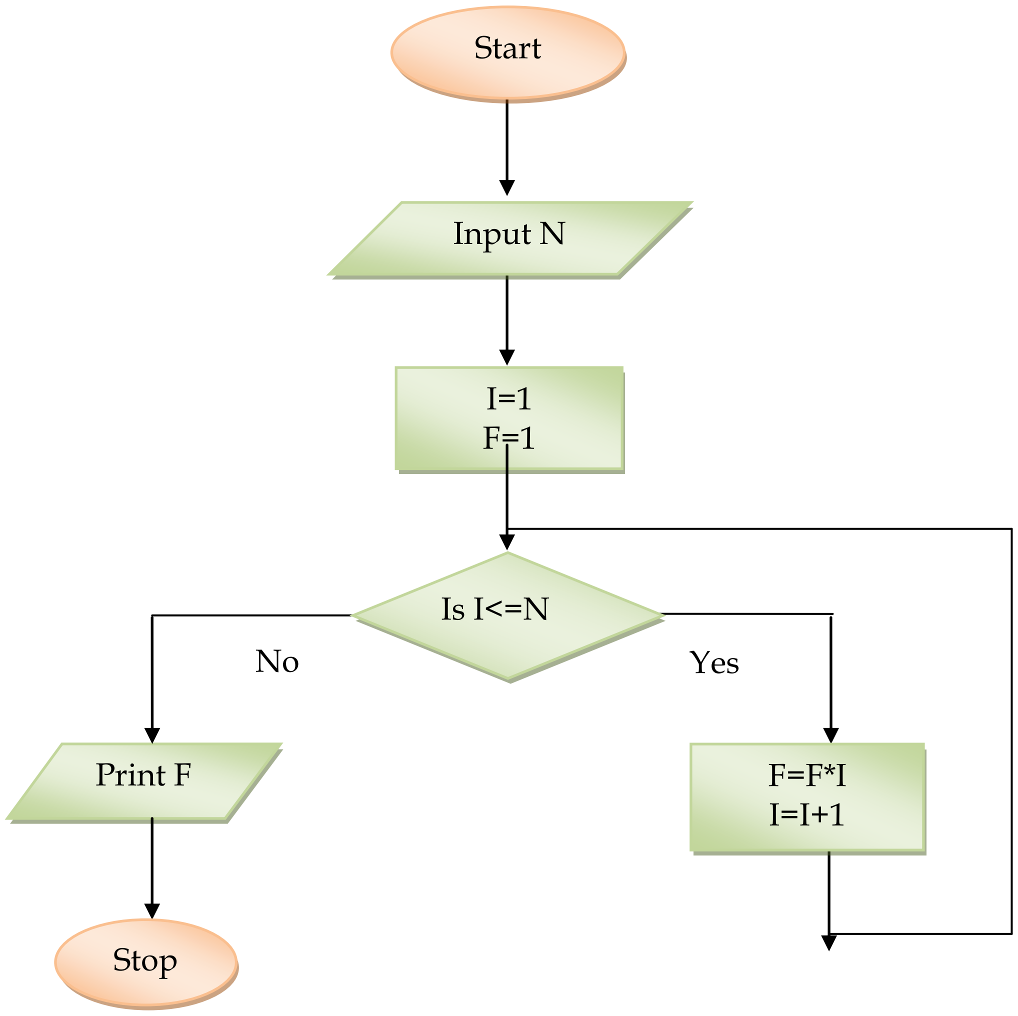


The following are the examples (VIII & IX) of an iterative execution.

Example

Draw a flow chart to find factorial of any number.

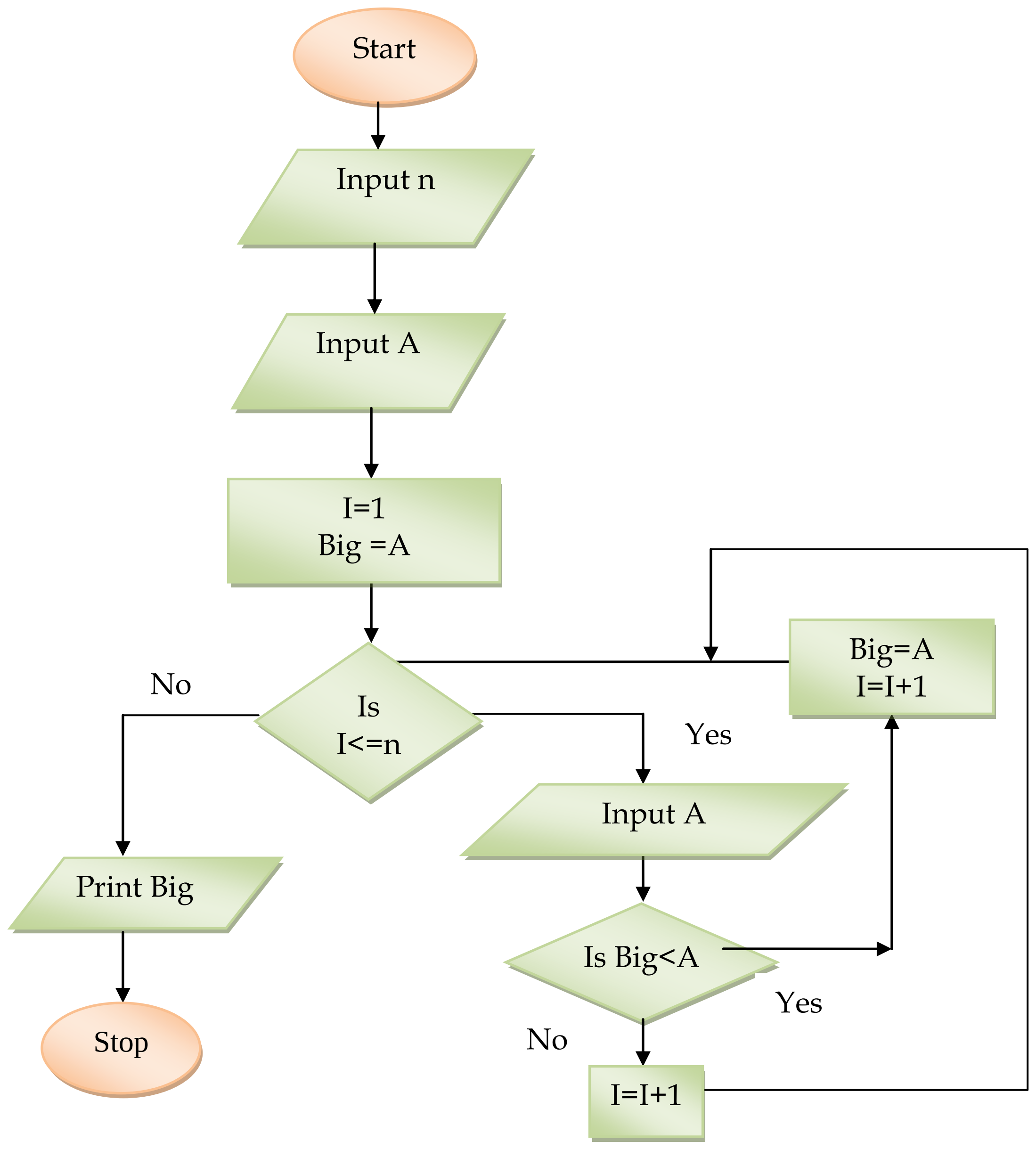
Solution:



Example

Draw a flow chart to find biggest number among 'n' numbers.

Solution:



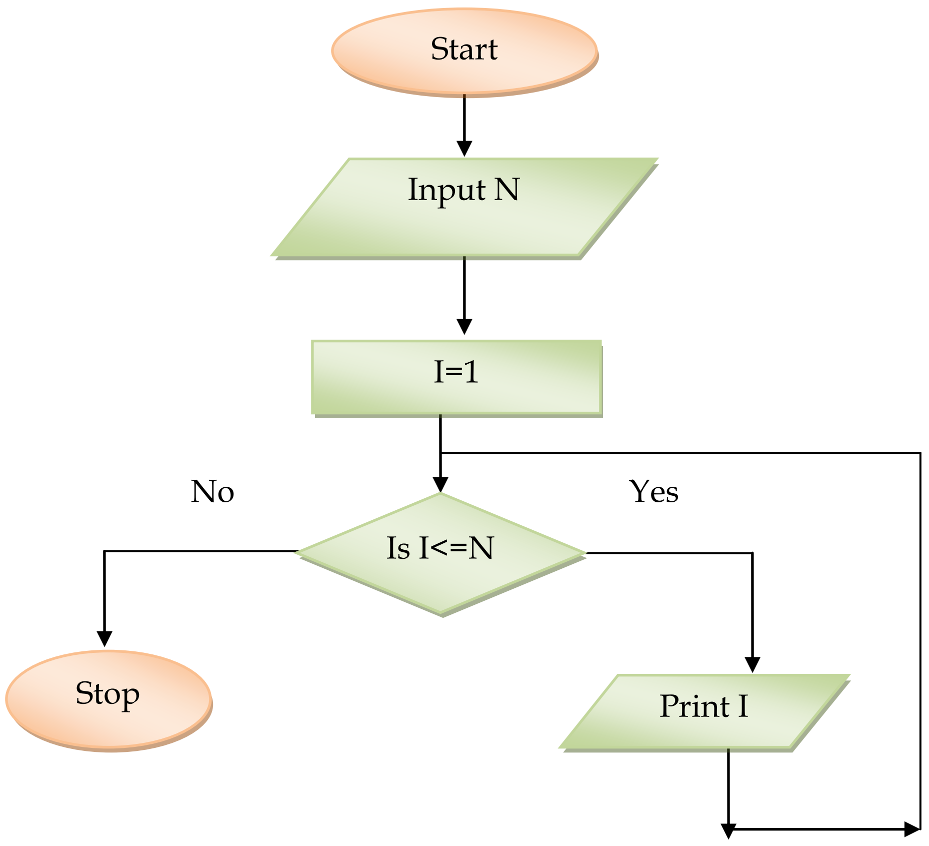
Finite and Infinite loop

In looping statements, if some set of statements are executed ‘n’ times (fixed number of times), then it is called ‘finite loop’. At the same time, if some set of statements are executed again and again without any end (infinite times), then it is called ‘infinite loop’. For example (X), if we are not incrementing ‘I’ (index) value, then we will get endless (infinite) loop. The following is an example of infinite loop.

Example

Draw a flow chart to print the number from 1 to ∞.

Solution:



In the above example "I" value is not at all incremented, so it will create endless loop. This is also called infinite loop.

Note: Set of statements is executed again and again without any end is called infinite loop.

EXERCISE

Multiple choice questions:

1. A step by step method for solving a problem using English Language

   (a) program (b) Flowchart

   (c) statement (d) Algorithm

2. Set of statements is executed based upon conditional test.

   (a) Looping (b) Selective

   (c) Sequence (d) None

3. Set of statements is executed again and again based upon conditional test.

   (a) Looping (b) Selective

   (c) Sequence (d) None

4. The graphical representation of algorithm is

   (a) program (b) Flowchart

   (c) statement (d) Algorithm

5. All instructions are executed one after other.

   (a) Looping (b) Selective

   (c) Sequence (d) None

Answer the following questions.

1. Define Algorithm.
2. Define Flowchart.
3. Write an algorithm to find the sum of two numbers.
4. Write an algorithm to find the area of a triangle.

5. Write an algorithm to find whether given number is odd or even.
6. Write an algorithm to find the sum of all even number up to given number.
7. Draw a flowchart to find the area of a circle.
9. Draw a flowchart to find the smallest number among n numbers.
10. Draw a flowchart to find the sum of all multiples of 5 up to given number.
11. Mona is confused about finite loop and infinite loop, explain her with the help of example.
12. Write an algorithm and a flowchart to find sum of n numbers.

Chapter 2

Programming Methodology

After studying this lesson, the students will be able to

- *understand the need for good programs;*
- *understand how to solve problems using different ways;*
- *get clear idea about problem solving methodology; and*
- *understand the types of errors normally occur while writing programs.*

Introduction

Learning to write computer program is very much like learning any skill. First, we should understand the problems well and then try to solve it in a logical manner. For example: We have read many books available in the market for describing the car driving methods. However, we can learn driving once we actually get into the car and start driving it. The same logic is applied in computer programming also. Computer programming is the process of writing, testing, troubleshooting, debugging and maintaining of a computer program.

An effective program is that which gives result of all different inputs, including wrong input also. While creating program, we need to follow certain systematic approach. This systematic approach comprises two steps/things, viz., program structure and program representation. The program structure is implemented by using top-down or bottom-up approach and is known as 'popular approach', while the program representation plays an important role in making the program more readable and understandable.

What is a Good Program?

A Good Program means that it should produce correct and faster results, taking into account all the memory constraints. While making good program, we need to follow certain guidelines of programming language for creating a successful program. The following is the list of good programming habits that most people agree.

Clarity and Simplicity of Expression

Expressions are used to implement a particular task. It is a combination of Operators, Operands and Constants. Any expression used in the program should be understood by the user. The followings are some of the points to be kept in mind while using expressions in a program.

(i) Use library functions to make programs more powerful

Example

To find output = x^6

Output = X *X * X * X * X * X

We can use output = power (X, 6)

(ii) Follow simplicity to maintain the clarity of expression

Example

$$X = \frac{A+B}{A-B} - \frac{U+VY}{X+Y}$$

Then, we can write

X1 = (A+B) / (A-B)

X2 = (U+V*Y) / (X +Y)

X = X1 –X2

(iii) Avoid program tricks usage, whose meaning is difficult to understand by the user.

Use of proper names for identifiers

Identifiers are user defined names. They are used to name things. A name is associated with a function or data object (constants and variables) and used to refer to that function or data object. Identifiers are made up of letters (A-Z, a-z), digits (0-9), and the underscore character (_). They, however, must begin with a letter or underscore and not with a digit.

(i) Give meaningful name for variable (data – object) and function.

Example

To calculate Area of a Square

We use the variable names are Area and Side

Area = Side * Side.

(ii) Use proper names for constants.

Example

¶ = 3.14

Give Pi = 3.14

(iii) Do not use same name like custom, customer or account, accountant.

(iv) Do not use one letter identifiers.

Comments

A comment is a programming language construct, which is used to embed programmer-readable annotations in the source code of a computer program. Those annotations are potentially significant to programmers but typically ignorable to compilers and interpreters. Comments are usually added with the purpose of making the source code easy to understand. Hence, add comments to your code in simple English language that describes the function of the code and the reason for your decision to do it in a particular way as well. They are generally categorized as either 'block comment' or 'line comment'. Block comment is implemented in python by """ and """ and line comment is implemented by #.

Example

"Write a program to print all numbers from 1 to 100 using while loop in python"

```
    A = 1
while (a<100):    # While statement
    print a
    a = a+1
```

Indentation

Leading white space (spaces and taps) at the beginning of each statement, which is used to determine the group of statement, is known as 'indentation'.

Example

```
If  A > B :
        print  'A is  Big'        # Block1
else:
        print 'B is Big'       # Block2
```

In the above example, **if statements** are a type of code block. If the 'if' expression evaluates to true, then Block1 is executed, otherwise, it executes Block2. Obviously, blocks can have multiple lines. As long as they are all indented with the same amount of spaces, they constitute one block.

Characteristics of good programming

Every computer needs proper instruction set (programs) to perform the required/assigned task. The quality of the program depends upon the instructions given to it. However, it is required to feed/provide the proper and correct instructions to the computer in order to yield/provide a correct and desired output. Hence, a program should be developed to ensure proper functionality of the computer and also should be easy to understand. A computer program should have some important characteristics, which are as follows:

Flexibility

A program should be flexible enough to handle most of the changes without having to rewrite the entire program. A flexible program is used to serve many purposes. For example, CAD (Computer Aided Design) software is used for different purposes such as; engineering drafting, printing circuit board layout and design, architectural design, technical drawing, industrial art, etc. Most of the programs are being developed for certain period and they need updation during the course of time.

User Friendly

A program that can be easily understood by a beginner is called 'user friendly'. It must interact with user through understandable messages. In addition, the proper message for the user to input data and to display the result, besides making the program easily understandable and modifiable.

Portability

Portability refers to the ability of an application to run on different platforms (operating systems) with or without minimal changes. Since the change of platform is a common phenomenon nowadays, due to the developments in hardware and the software, portability has to be taken care of it. In case, a program is developed for a particular platform, it would become obsolete after a certain period of time. At the same time, if a program that is developed does have the ability to work on different platforms, it makes software more useable. High language programs are often more portable than assembly language programs.

Reliability

It is the ability of a program to do its intended function accurately even if there are even small changes in the computer system. Moreover, the program must be able to handle unexpected situation like wrong input or no input. The programs, which save such ability are known as 'reliable'. For example, if the user does/gives wrong information to input, it should display a proper error message.

Self-Documenting Code

The source code, which uses suitable name for the identifiers (variables and methods), is called self-documenting code. Also, giving proper name for variables and methods would tell the reader of your code clearly -what is it doing? Hence, a good program must have a self-documenting code.

Problem solving process

The problem solving process starts with the problem specifications and ends with a concrete (and correct) program. Programming means a problem solving activity, which consists of four steps. They are;

(i) Understanding the problem;

(ii) Devising a plan;

(iii) Executing the plan; and

(iv) Evaluation

Understanding the problem

The first step is to understand the problem well. It may be very difficult to understand the problem but it is crucial. In general, one must find out the output from the given data (input data) and assess the relationship between input and output data. It is also important to verify whether the given information is sufficient to solve the problem or not.

Devising a plan

It means drawing an action plan to solve the problem, once understood. A plan is devised from data processing to the result according to the relationship that links both of them. If the problem is trivial, this step will not require much thinking.

Executing the plan

Once the plan is defined, it should follow the plan of action completely and each element of the plan should be checked as it is applied. In the course of execution, if any part of the plan is found to be unsatisfactory, the plan should be revised.

Evaluation

Finally, the result should be examined in order to make sure that it is valid and that the problem has been solved completely.

Problem solving methodology

As we all know, there are many methods/approaches available to solve a particular problem. However, the efficient way is to adopt a systematic method of problem solving. The use of systematic method of problem solving is crucial when we use a computer to solve a problem. We introduce here a seven steps problem solving method, which is closely related to the ***software life cycle*** (the various stages in the life of a program), that can be adapted by each person to solve the problem in their own style. They are given as under:

1. Problem Definition
2. Problem Analysis
3. Design the problem
4. Coding
5. Program Testing and Debugging
6. Documentation
7. Program Maintenance

Problem Definition/Specification (Theme)

Computer programs are written to solve problems posed by humankind. Prior to writing a program, one has to understand a description of the problem to solve. This description may be very precise or vague, but nevertheless, it is necessary/present. For instance, if you want to write a program to "Find the average of five numbers", you should ask yourself:

"What does average mean exactly?"

"How to calculate average value?"

Posing such questions compels you to define the problem very precisely. Once you are sure of what the problem entails, you must write down a list of specifications. *Specifications* are precise definitions of what the program must do. It must include the following at least:

- **Input:** what data must be included as input and in which form?
- **Output:** what data must the program produce and in which form? (in order to solve the problem)

Note: At the end of the problem definition step, you should have a list of specifications.

Problem Analysis

In this step, the problem has to be fragmented into smaller and manageable parts. The original problem has to be analyzed and divided into a number of sub-problems as these sub-problems are easier to solve and their solutions would become the components of the final program. Each sub-problem is divided into further smaller ones and this fragmentation has to be continued to achieve simple solutions. The use of modular programming is to get proper solution.

Modular Programming: Modular Programming is the act of designing and writing programs as functions (a large program is divided into the small individual components) that each one performs, a single well-defined function, which has minimal interaction between the sub-programs. It means that the content of each function is cohesive and there is low coupling between them. There are two methods available for modular programming. They are: top-down design and bottom-up design.

Top-Down design: The principles of top-down design dictate that a program should be divided into a main module and its related module. Each module should also be divided into sub modules according to software engineering and programming style. The division continues till the module consists only of an elementary process that is intrinsically understood and cannot be further sub-divided.

Bottom-up design: Bottom-up design is just the opposite of top-down design. It refers to a style of programming, in which, an application is constructed with existing primitives of the programming language and then gradually more and more complicated features are added till applications are written. In other words, initiating the design with simple modules and then build them into more complex structures ending at the top is bottom-up design.

Designing the problem

Designing the problem can be expressed in the form of

- Algorithm
- Flowchart

Algorithm: An algorithm is a set of instructions that describe a method for solving a problem. It is normally given in mix of computer code and English language. This is often called 'pseudo-code'.

Flowchart: The algorithm is represented in the form of a diagram with action boxes linked by lines showing the order in which they are executed. This is known as 'the flow of control'. It is the diagrammatic representation of an algorithm.

Coding

The process of translating the algorithm into syntax of a given language is known as 'Coding'. Since algorithm cannot be executed directly by the computer, it has to be translated into a *programming language.*

Program Testing and Debugging

Program Testing means running the program, executing all its instructions/ functions and testing the logic by entering sample data in order to check the output. Debugging is the process of finding and correcting the errors in the program code.

Type of errors: There are three types of errors generally occur during compilation and running a program. They are (i) Syntax error; (ii) Logical error; and (iii) Runtime error.

Syntax error: Every programming language has its own rules and regulations (syntax). If we overcome the particular language rules and regulations, the syntax error will appear (i.e. an error of language resulting from code that does not conform to the syntax of the programming language). It can be recognized during compilation time.

Example

```
a = 0
while a < 10
    a = a + 1
    print a
```

In the above statement, the second line is not correct. Since the while statement does not end with ':'. This will flash a syntax error.

Logical error: Programmer makes errors while writing program that is called 'logical error'. It is an error in a program's source code that results in incorrect or unexpected result. It is a type of runtime error that may simply produce the wrong output or may cause a program to crash while running. The logical error might only be noticed during runtime, because it is often hidden in the source code and are typically harder to find and debug.

```
a = 100
while a < 10:
    a = a + 1
    print a
```

In the above example, the while loop will not execute even a single time, because the initial value of 'a' is 100.

Runtime error: A runtime error is an error that causes abnormal termination of program during running time. In general, the dividend is not a constant but might be a number typed by you at runtime. In this case, division by zero is illogical. Computers check for a "division by zero" error during program execution, so that you can get a "division by zero" error message at runtime, which will stop your program abnormally. This type of error is called runtime error.

Example

(a)
```
A=10
B=0
print A/B
```

(b) During running time, if we try to open a file that does not exist in the hard disk, then it will create runtime error.

Documentation

The documentation includes the problem definition, design documents, a description of the test perform, a history of the program development and its different versions and a user's manual. Such a manual is designed for a naive user and illustrates the preparation of input data, running the program and obtaining & interpreting the results.

Program maintenance

It is not directly part of the original implementation process, but needs special emphasis. All activities that occur after a program operation are part of the program maintenance. Many large programs have long life span that often exceed the lifetime of the hardware they run on. Usually, the expenditure for the program maintenance will be more than the developmental cost of the program. The program maintenance includes the following:

- Finding and eliminating previously undetected program errors;
- Modifying the current program, often to improve its performance, or to adapt to new laws or government regulations, or to adapt to a new hardware, or to a new operating system;
- Adding new features or a better user interface, or new capabilities to the program; and
- Updating the documentation.

Maintenance is an important part of the life cycle of a program. It is also important as far as documentation is concerned, since any change pertaining to a program will require updating of internal as well as external documentation. Maintenance documentation will include results of the program development steps, design documents, program code and test information.

EXERCISE

Multiple choice questions:

1. User Define name.

   (a) Identifier (b) constant

   (c) syntax (d) expression

2. If we overcome the rules of the programming language, we get

   (a) Runtime error (b) Syntax error

   (c) logical error (d) None of the above.

3. Correcting the program code:

   (a) Testing (b) Syntax error

   (c) Runtime error (d) Debugging

4. Designing the problem

   (a) Testing (b) Debugging

   (c) logical error (d) Algorithm

5. Algorithm when translated into a programming language is called

   (a) Flowchart (b) Identifier

   (c) Code (d) Debugging

6. The program must be able to handle unexpected situation like wrong input or no input.

   (a) Error (b) Expression

   (c) Portability (d) Reliability

7. Leading white space at the beginning of each statement, which is used to determine the group of statement.

   (a) Testing (b) Indentation

   (c) Debugging (d) None of the above

8. It refers to the ability of an application to run on different platforms with or without minimal changes.

   (a) Error (b) Flexibility

   (c) Portability (d) Reliability

9. It is a combination of Operators, Operands and Constants.

   (a) Identifier (b) Expression

   (c) Syntax (d) Task

10. Each module should also be divided into sub modules according to software engineering and programming style.

    (a) Top down method (b) Bottom up method

    (c) Coding (d) None of the above

Answer the following questions.

1. What is a good program?
2. What is an identifier?
3. How to write comments in a program?
4. What is the purpose of expression? Explain with an example.
5. Write and explain all steps of programming methodology.
6. Differentiate between runtime errors and logical errors.
7. Define documentation.
8. What is program maintenance?
9. Define modular programming.
10. Differentiate between top down and bottom up methods of modular programming.
11. Explain types of errors with examples.
12. How to maintain programs?

13. Write all steps of program methodology?
14. What do you mean by clarity and simplicity of expression?
15. What do you mean by flexibility?
16. Explain all steps of problem solving process.
17. What is indentation? Explain with an example.
18. What do you mean by debugging?
19. What is the use of self documenting code in programming?
20. What is the purpose of giving meaningful name for identifiers?

UNIT 3

Introduction to Python

Chapter 1

Getting Started

After studying this lesson, students will be able to:

- *Appreciate the use of Graphical User interface and Integrated Development Environment for creating Python programs.*
- *Work in interactive & Script mode for programming.*
- *Create and assign values to variables.*
- *Understand the concept and usage of different data types in python.*
- *Appreciate the importance and usage of different types of operator (arithmetic, Relation and logical)*
- *Create Python expression(s) and statement(s).*

Introduction

In order to tell the computer 'what you want to do', we write a program in a language which computer can understand. Though there are many different programming languages such as BASIC, Pascal, C, C++, Java, Haskell, Ruby, Python, etc. but we will study Python in this course.

Before learning the technicalities of Python, let's get familiar with it.

Python was created by Guido Van Rossum when he was working at CWI (Centrum Wiskunde & Informatica) which is a National Research Institute for Mathematics and Computer Science in Netherlands. The language was released in I991. Python got its name from a BBC comedy series from seventies- "Monty Python's Flying Circus". Python can be used to follow both Procedural approach and Object Oriented approach of programming. It is free to use.

Some of the features which make Python so popular are as follows:

- It is a general purpose programming language which can be used for both scientific and non scientific programming.
- It is a platform independent programming language.

- It is a very simple high level language with vast library of add-on modules.
- It is excellent for beginners as the language is interpreted, hence gives immediate results.
- The programs written in Python are easily readable and understandable.
- It is suitable as an extension language for customizable applications.
- It is easy to learn and use.

The language is used by companies in real revenue generating products, such as:

- In operations of Google search engine, youtube, etc.
- Bit Torrent peer to peer file sharing is written using Python
- Intel, Cisco, HP, IBM, etc use Python for hardware testing.
- Maya provides a Python scripting API
- i–Robot uses Python to develop commercial Robot.
- NASA and others use Python for their scientific programming task.

First Step with Python

We are continuously saying that Python is a programming language but don't know what a program is? Therefore, let's start Python by understanding Program.

A program is a sequence of instructions that specifies how to perform a Computation. The Computation might be mathematical or working with text.

To write and run Python program, we need to have Python interpreter installed in our computer. IDLE (GUI integrated) is the standard, most popular Python development environment. IDLE is an acronym of Integrated Development Environment. It lets edit, run, browse and debug Python Programs from a single interface. This environment makes it easy to write programs.

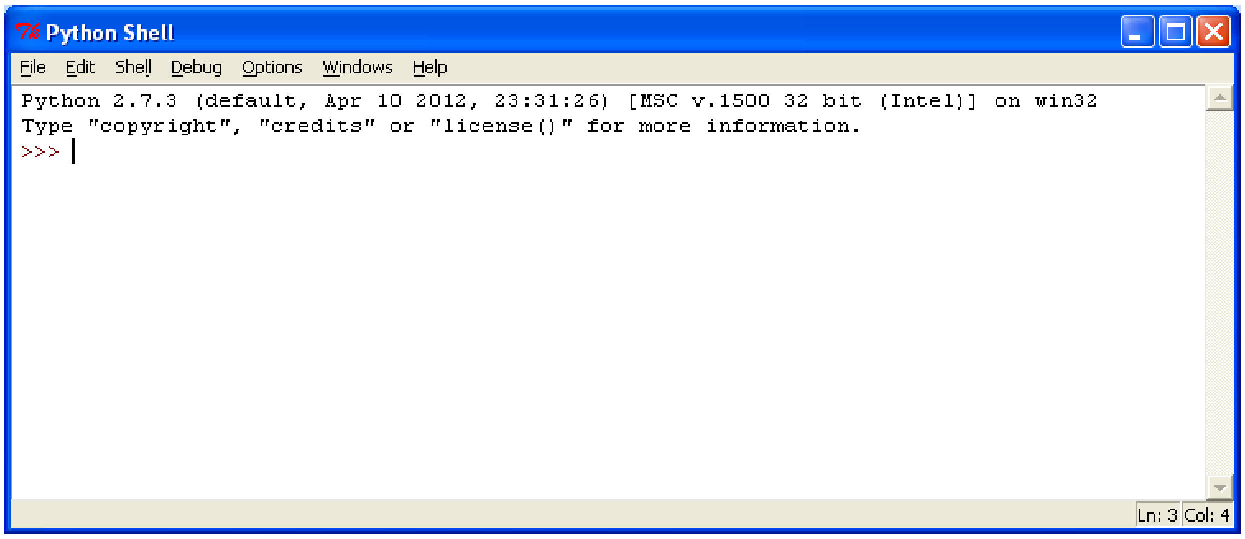
We will be using version 2.7 of Python IDLE to develop and run Python code, in this course. It can be downloaded from www.python.org

Python shell can be used in two ways, viz., interactive mode and script mode. Where Interactive Mode, as the name suggests, allows us to interact with OS; script mode let us create and edit python source file. Now, we will first start with interactive mode. Here, we type a Python statement and the interpreter displays the result(s) immediately.

Interactive Mode

For working in the interactive mode, we will start Python on our computer. You can take the help of your Teacher.

When we start up the IDLE following window will appear:



```
Python 2.7.3 (default, Apr 10 2012, 23:31:26) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> |
```

What we see is a welcome message of Python interpreter with revision details and the Python prompt, i.e., '>>>'. This is a primary prompt indicating that the interpreter is expecting a python command. There is secondary prompt also which is '…' indicating that interpreter is waiting for additional input to complete the current statement.

Interpreter uses prompt to indicate that it is ready for instruction. Therefore, we can say, if there is prompt on screen, it means IDLE is working in interactive mode.

We type Python expression / statement / command after the prompt and Python immediately responds with the output of it. Let's start with typing print **"How are you"** after the prompt.

>>>print "How are you?"

How are you?

What we get is Python's response. We may try the following and check the response:

i) print 5+7

ii) 5+7

iii) 6*250/9

iv) print 5-7

It is also possible to get a sequence of instructions executed through interpreter.

Example 1	Example 2

>>> x=2
>>> y=6
>>> z = x+y
>>> print z
8
	>>> a=3
	>>> a+1, a-1
	(4,2) #result is tuple of 2 values

#result is tuple of 2 values, is a comment statement. We will talk about it in the later part of chapter.

Now we are good to write a small code on our own in Python. While writing in Python, remember Python is case sensitive. That means x & X are different in Python.

Note: If we want to repeat prior command in interactive window, you can use '↑' key to scroll backward through commands history and '↓' key to scroll forward. Use Enter key to select it. Using these keys, your prior commands will be recalled and displayed, and we may edit or rerun them also.

^D (Ctrl+D) or quit () is used to leave the interpreter.

^F6 will restart the shell.

Help of IDLE can be explored to know about the various menu options available for Programmer.

Apart from writing simple commands, let's explore the interpreter more.

Type **Credits** after the prompt and what we get is information about the organization involved in Python development. Similarly, **Copyright** and **Licenses** command can be used to know more about Python. Help command provides **help** on Python. It can be used as….. help() with nothing in parenthesis will allow us to enter an interactive help mode. And with a name (predefined) in bracket will give us details of the referred word.

To leave the help mode and return back to interactive mode, quit command can be used.

Script Mode

In script mode, we type Python program in a file and then use the interpreter to execute the content from the file. Working in interactive mode is convenient for beginners and for testing small pieces of code, as we can test them immediately. But for coding more than few lines, we should always save our code so that we may modify and reuse the code.

Note: Result produced by Interpreter in both the modes, viz., Interactive and script mode is exactly same.

Python, in interactive mode, is good enough to learn, experiment or explore, but its only drawback is that we cannot save the statements for further use and we have to retype all the statements to re-run them.

To create and run a Python script, we will use following steps in IDLE, if the script mode is not made available by default with IDLE environment.

1. File>Open OR File>New Window (for creating a new script file)
2. Write the Python code as function i.e. script
3. Save it (^S)
4. Execute it in interactive mode- by using RUN option (^F5)

   Otherwise (if script mode is available) start from Step 2

Note: For every updation of script file, we need to repeat step 3 & step 4

If we write Example 1 in script mode, it will be written in the following way:

Step 1: File> New Window

Step 2:

```
def test():
x=2
y=6
z = x+y
print z
```

Step 3:

Use File > Save or File > Save As - option for saving the file

(*By convention all Python program files have names which end with .py*)

Step 4:

For execution, press ^F5, and we will go to Python prompt (in other window)

```
>>> test()
8
```

Alternatively we can execute the script directly by choosing the RUN option.

Note: While working in script mode, we add 'print' statement in our program to see the results which otherwise were displayed on screen in interactive mode without typing such statements.

Variables and Types

When we create a program, we often like to store values so that it can be used later. We use objects to capture data, which then can be manipulated by computer to provide information. By now we know that object/ variable is a name which refers to a value.

Every object has:

A. An Identity, - can be known using id (object)

B. A type – can be checked using type (object) and

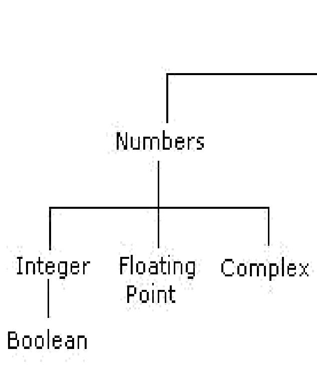
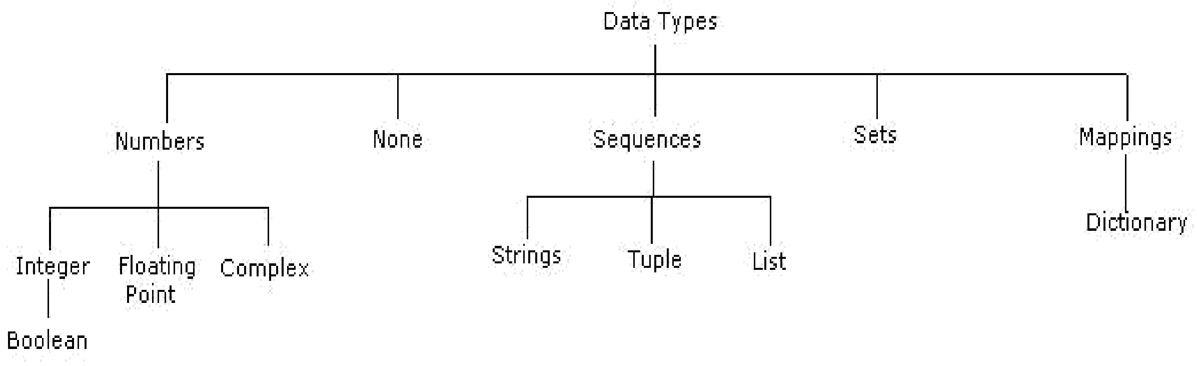
C. A value

Let us study all these in detail

A. Identity of the object: It is the object's address in memory and does not change once it has been created.

(We would be referring to objects as variable for now)

B. Type (i.e data type): It is a set of values, and the allowable operations on those values. It can be one of the following:

1. Number

Number data type stores Numerical Values. This data type is immutable i.e. value of its object cannot be changed (we will talk about this aspect later). These are of three different types:

a) Integer & Long

b) Float/floating point

c) Complex

Range of an integer in Python can be from -2147483648 to 2147483647, and long integer has unlimited range subject to available memory.

1.1 Integers are the whole numbers consisting of + or – sign with decimal digits like 100000, -99, 0, 17. While writing a large integer value, don't use commas to separate digits. Also integers should not have leading zeros.

When we are working with integers, we need not to worry about the size of integer as a very big integer value is automatically handled by Python. When we want a value to be treated as very long integer value append **L** to the value. Such values are treated as long integers by python.

```
>>> a = 10
>>> b = 5192L          #example of supplying a very long value to a variable
>>> c= 4298114
>>> type(c)            # type ( ) is used to check data type of value
<type 'int'>
>>> c = c * 5669
>>> type(c)
<type 'long'>
```

We can know the largest integer in our version of Python by following the given set of commands:

```
>>> import sys
>>> print sys.maxint
```

Integers contain Boolean Type which is a unique data type, consisting of two constants, **True** & **False**. A Boolean True value is Non-Zero, Non-Null and Non-empty.

Example

```
    >>> flag = True
    >>> type(flag)
    <type 'bool'>
```

1.2 **Floating Point:** Numbers with fractions or decimal point are called floating point numbers.

A floating point number will consist of sign (+,-) sequence of decimals digits and a dot such as 0.0, -21.9, 0.98333328, 15.2963. These numbers can also be used to represent a number in engineering/ scientific notation.

-2.0X 10^5 will be represented as -2.0e5

2.0X10^{-5} will be 2.0E-5

Example

```
y= 12.36
```

A value when stored as floating point in Python will have 53 bits of precision.

1.3 **Complex:** Complex number in python is made up of two floating point values, one each for real and imaginary part. For accessing different parts of variable (object) x; we will use x.real and x.image. Imaginary part of the number is represented by 'j' instead of 'i', so 1+0j denotes zero imaginary part.

Example

```
>>> x = 1+0j
>>> print x.real,x.imag
1.0 0.0
```

Example

```
>>> y = 9-5j
>>> print y.real, y.imag
9.0 -5.0
```

2. None

This is special data type with single value. It is used to signify the absence of value/false in a situation. It is represented by **None**.

3. Sequence

A sequence is an ordered collection of items, indexed by positive integers. It is combination of mutable and non mutable data types. Three types of sequence data type available in Python are Strings, Lists & Tuples.

3.1 **String:** is an ordered sequence of letters/characters. They are enclosed in single quotes (' ') or double ('' "). The quotes are not part of string. They only tell the computer where the string constant begins and ends. They can have any character or sign, including space in them. These are immutable data types. We will learn about immutable data types while dealing with third aspect of object i.e. value of object.

Example

```
>>> a = 'Ram'
```

A string with length 1 represents a character in Python.

Conversion from one type to another

If we are not sure, what is the data type of a value, Python interpreter can tell us:

```
>>> type ('Good Morning')
<type 'str'>
>>> type ('3.2')
<type 'str'>
```

It is possible to change one type of value/ variable to another type. It is known as type conversion or type casting. The conversion can be done explicitly (programmer specifies the conversions) or implicitly (Interpreter automatically converts the data type).

For explicit type casting, we use functions (constructors):

```
int ()
float ()
str ()
bool ()
```

Example

```
>>> a= 12.34
>>> b= int(a)
>>> print b
12
```

Example

```
>>>a=25
>>>y=float(a)
>>>print y
25.0
```

3.2 **Lists:** List is also a sequence of values of any type. Values in the list are called elements / items. These are mutable and indexed/ordered. List is enclosed in square brackets.

Example

```
l = ['spam', 20.5,5]
```

3.3 **Tuples:** Tuples are a sequence of values of any type, and are indexed by integers. They are immutable. Tuples are enclosed in (). We have already seen a tuple, in Example 2 (4, 2).

4. Sets

Set is an unordered collection of values, of any type, with no duplicate entry. Sets are immutable.

Example

```
s = set ([1,2,34])
```

5. Mapping

This data type is unordered and mutable. Dictionaries fall under Mappings.

5.1 **Dictionaries:** Can store any number of python objects. What they store is a key – value pairs, which are accessed using key. Dictionary is enclosed in curly brackets.

Example

```
d = {1:'a',2:'b',3:'c'}
```

C. **Value of Object (variable) –** to bind value to a variable, we use assignment operator (=). This is also known as building of a variable.

Example

```
>>> pi = 31415
```

Here, value on RHS of '=' is assigned to newly created 'pi' variable.

Mutable and Immutable Variables

A mutable variable is one whose value may change in place, whereas in an immutable variable change of value will not happen in place. Modifying an immutable variable will rebuild the same variable.

Example

```
>>>x=5
```

Will create a value 5 referenced by x

$x \rightarrow 5$

```
>>>y=x
```

This statement will make y refer to 5 of x

$$\begin{tikzcd} x \arrow[dr] & \\ & 5 \\ y \arrow[ur] & \end{tikzcd}$$

```
>>> x=x+y
```

As x being integer (immutable type) has been rebuild.

In the statement, expression on RHS will result into value 10 and when this is assigned to LHS (x), x will rebuild to 10. So now

$$\begin{tikzcd} x \arrow[r] & 10 \text{ and} \\ y \arrow[r] & 5 \end{tikzcd}$$

After learning about what a variable can incorporate, let's move on with naming them. Programmers choose the names of the variable that are meaningful. A variable name:

1. Can be of any size
2. Have allowed characters, which are a-z, A-Z, 0-9 and underscore (_)
3. should begin with an alphabet or underscore
4. should not be a keyword

It is a good practice to follow these identifier naming conventions:

1. Variable name should be meaningful and short
2. Generally, they are written in lower case letters

Keywords

They are the words used by Python interpreter to recognize the structure of program. As these words have specific meaning for interpreter, they cannot be used for any other purpose.

A partial list of keywords in Python 2.7 is

and	del	from	not
while	as	elif	global
or	with	assert	else
if	pass	yield	break
except	import	print	class
exec	in	raise	continue
finally	is	return	def
for	lambda	try	

Remember:

✡ Variables are created when they are first assigned a value.

✡ Variables must be assigned a value before using them in expression,

✡ Variables refer to an object and are never declared ahead of time.

Operators and Operands

Operators are special symbols which represents computation. They are applied on operand(s), which can be values or variables. Same operator can behave differently on different data types. Operators when applied on operands form an expression. Operators are categorized as Arithmetic, Relational, Logical and Assignment. Value and variables when used with operator are known as operands.

Following is the partial list of operators:

Mathematical/Arithmetic Operators

Symbol	Description	Example 1	Example 2
+	Addition	>>>55+45 100	>>> 'Good' + 'Morning' GoodMorning
-	Subtraction	>>>55-45 10	>>>30-80 -50
*	Multiplication	>>>55*45 2475	>>> 'Good'* 3 GoodGoodGood
/	Division	>>>17/5 3 >>>17/5.0 3.4 >>> 17.0/5 3.4	>>>28/3 9

%	Remainder/ Modulo	>>>17%5 2	>>> 23%2 1
**	Exponentiation	>>>2**3 8 >>>16**.5 4.0	>>>2**8 256
//	Integer Division	>>>7.0//2 3.0	>>>3/ / 2 1

Note: Division is Implementation Dependent

Relational Operators

Symbol	Description	Example 1	Example 2

<	Less than	>>>7<10
		True
		>>> 7<5
		False
		>>> 7<10<15
		True
		>>>7<10 and 10<15
		True
			>>>‘Hello’< ’Goodbye’
			False
			>>>'Goodbye'< 'Hello'
			True

>	Greater than	>>>7>5 True >>>10<10 False	>>>‘Hello’> ‘Goodbye’ True >>>'Goodbye'> 'Hello' False
<=	less than equal to	>>> 2<=5	>>>‘Hello’<= ‘Goodbye’

		True >>> 7<=4 False	False >>>'Goodbye' <= 'Hello' True
>=	greater than equal to	>>>10>=10 True >>>10>=12 False	>>>’Hello’>= ‘Goodbye’ True >>>'Goodbye' >= 'Hello' False
! =, <>	not equal to	>>>10!=11 True >>>10!=10 False	>>>’Hello’!= ‘HELLO’ True >>> ‘Hello’ != ‘Hello’ False
==	equal to	>>>10==10 True >>>10==11 False	>>>’Hello’ == ‘Hello’ True >>>’Hello’ == ‘Good Bye’ False

Note: Two values that are of different data type will never be equal to each other.

Logical Operators

Symbol	Description
or	If any one of the operand is true, then the condition becomes true.
and	If both the operands are true, then the condition becomes true.
not	Reverses the state of operand/condition.

Assignment Operators

Assignment Operator combines the effect of arithmetic and assignment operator

Symbol	Description	Example	Explanation
=	Assigned values from right side operands to left variable	>>>x=12* >>>y='greetings'	

*(*we will use it as initial value of x for following examples)*

+=	added and assign back the result to left operand	>>>x+=2	The operand/ expression/ constant written on RHS of operator is will change the value of x to 14
-=	subtracted and assign back the result to left operand	x-=2	x will become 10
*=	multiplied and assign back the result to left operand	x*=2	x will become 24
/=	divided and assign back the result to left operand	x/=2	x will become 6
%=	taken modulus using two operands and assign the result to left operand	x%=2	x will become 0
**=	performed exponential (power) calculation on operators and assign value to the left operand	x**=2	x will become 144
//=	performed floor division on operators and assign value to the left operand	x / /= 2	x will become 6

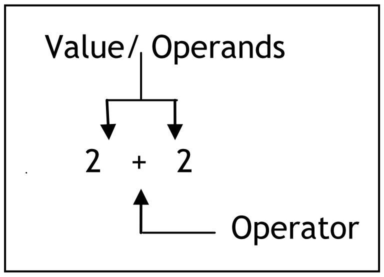
Note:

1. Same operator may perform a different function depending on the data type of the value to which it is applied.
2. Division operator '/' behaves differently on integer and float values.

Expression and Statements

An expression is a combination of value(s) (i.e. constant), variable and operators. It generates a single value, which by itself is an expression.

Example



The expression is solved by Computer and gets it value. In the above example, it will be 4, and we say the expression is evaluated.

Note: Expression values in turn can act as, Operands for Operators

We have seen many such expressions (with list of operator as example). 10+5 and 9+4+2 are two expressions which will result into value 15. Taking another example, 5.0/4+ (6-3.0) is an expression in which values of different data types are used. These type of expressions are also known as mixed type expressions.

When mixed type expressions are evaluated, Python promotes the result of lower data type to higher data type, i.e. to float in the above example. This is known as implicit type casting. So the result of above expression will be 4.25. Expression can also contain another expression. As we have already seen in 9+4+2. When we have an expression consisting of sub expression(s), how does Python decide the order of operations?

It is done based on precedence of operator. Higher precedence operator is worked on before lower precedence operator. Operator associativity determines the order of evaluation when they are of same precedence, and are not grouped by parenthesis. An operator may be Left-associative or Right –associative. In left associative, the operator falling on left side will be evaluated first, while in right assosiative operator falling on right will be evaluated first.

Note: In python '=' and '**' are Right Associative.

Precedence of operator - Listed from high precedence to low precedence.

Operator	**Description**
**	Exponentiation (raise to the power)
+ , -	unary plus and minus
* , /, %, //	Multiply, divide, modulo and floor division
+ , -	Addition and subtraction
<, <=, >, >=	Comparison operators
==, !=	Equality operators
% =, / =, // = , -=, + =, * =	Assignment operators
not and or	Logical operators

Using the above table, we know that 9+4 itself is an expression which evaluates to 13 and then 13+2 is evaluated to 15 by computer. Similarly, 5.0/4 + (6-3.0) will be evaluated as 5.0/4+3.0 and then to 1.25 + 3.0, and then 4.25.

If we just type 10+, we will get an error message. This happens because 10+ is not a complete expression. A complete expression will always have appropriate number of value (Operands) with each operator. '+' needs two operands and we have given just one.

Note: Remember precedence of operators is applied to find out which sub expression should be evaluated first.

Expression can be combined together to form large expressions but no matter how big the expression is, it always evaluate into a single value.

A Python statement is a unit of code that the Python interpreter can execute.

Example of statement are:

```
>>> x=5
>>> area=x**2          #assignment statement
>>>print x             #print statement
5
>>>print area
25
>>> print x, area
5  25
```

Note: To print multiple items in same line, separate them with comma.

Statements normally go to the end of a line.

```
X= "good morning"          #comment
```

What we have seen as an example till now were simple statements, i.e. they do not contain a nested block. In Python, there are compound/ group statements also. They are sometimes called nested block. Statement belonging to a block are indented (usually by 4 spaces). Leading whitespace at the beginning of logical line is used to determine the indentation level of line. That means statement(s) which go together must have same indentation level.

# Example of Compound Statement

Example

```
if i<0:
    print "i is negative"
else:
    print "i is non-negative"
```

Example

```
if i>0:
    print "i is positive"
else:
    print "i is equal to 0"
```

While writing Python statements, keep the following points in mind:

1. Write one python statement per line (Physical Line). Although it is possible to write two statements in a line separated by semicolon.

2. Comment starts with '#' outside a quoted string and ends at the end of a line. Comments are not part of statement. They may occur on the line by themselves or at the end of the statement. They are not executed by interpreter.

3. For a long statement, spanning multiple physical lines, we can use '/' at the end of physical line to logically join it with next physical line. Use of the '/' for joining lines is not required with expression consists of ( ), [ ], { }

4. When entering statement(s) in interactive mode, an extra blank line is treated as the end of the indented block.

5. Indentation is used to represent the embedded statement(s) in a compound/ Grouped statement. All statement(s) of a compound statement must be indented by a consistent no. of spaces (usually 4)

6. White space in the beginning of line is part of indentation, elsewhere it is not significant.

Note:

- ✡ Wrong indentation can give rise to syntax error(s).
- ✡ Most Python editor will automatically indent the statements.
- ✡ A physical line is what you see as a line when you write a program and a logical line is what Python sees as a single statement.

Input and Output

A Program needs to interact with end user to accomplish the desired task, this is done using Input-Output facility. Input means the data entered by the user (end user) of the program. While writing algorithm(s), getting input from user was represented by Take/Input. In python, we have raw-input() and input ( ) function available for Input.

raw_input()

Syntax of raw_input() is:

raw_input $\underbrace{\textbf{([prompt])}}_{\textbf{Optional}}$

If prompt is present, it is displayed on the monitor after which user can provide the data from keyboard. The function takes exactly what is typed from keyboard, convert it to string and then return it to the variable on LHS of '='.

Example (in interactive mode)

```
>>>x=raw_input ('Enter your name: ')
```

Enter your name: ABC

x is a variable which will get the string (ABC), typed by user during the execution of program. Typing of data for the raw_input function is terminated by 'enter' key.

We can use raw_input() to enter numeric data also. In that case we typecast, i.e., change the datatype using function, the string data accepted from user to appropriate Numeric type.

Example

    y=int(raw_input("enter your roll no"))

enter your roll no. 5

will convert the accepted string i.e. 5 to integer before assigning it to 'y'.

input()

Syntax for input() is:

$$\textbf{Input } \underbrace{\textbf{([prompt])}}_{\textbf{Optional}}$$

If prompt is present, it is displayed on monitor, after which the user can provide data from keyboard. Input takes whatever is typed from the keyboard and evaluates it. As the input provided is evaluated, it expects valid python expression. If the input provided is not correct then either syntax error or exception is raised by python.

Example

```
x= input ('enter data:')
Enter data: 2+1/2.0
Will supply 2.5 to x
```

input ( ), is not so popular with python programmers as:

i) Exceptions are raised for non-well formed expressions.

ii) Sometimes well formed expression can wreak havoc.

Output is what program produces. In algorithm, it was represented by print. For output in Python we use print. We have already seen its usage in previous examples. Let's learn more about it.

Print Statement

Syntax:

print expression/constant/variable

Print evaluates the expression before printing it on the monitor. Print statement outputs an entire (complete) line and then goes to next line for subsequent output (s). To print more than one item on a single line, comma (,) may be used.

Example

```
>>> print "Hello"
Hello
>>> print 5.5
5.5
>>> print 4+6
10
```

Try this on the computer and evaluate the output generated

```
>>>print 3.14159* 7**2
>>>print "I", "am" + "class XI", "student"
>>>print "I'm",
>>>print "class XI student"
>>>print "I'm ", 16, "years old"
```

Comments

As the program gets bigger, it becomes difficult to read it, and to make out what it is doing by just looking at it. So it is good to add notes to the code, while writing it. These notes are known as comments. In Python, comment start with '#' symbol. Anything written after # in a line is ignored by interpreter, i.e. it will not have any effect on the program.

A comment can appear on a line by itself or they can also be at the end of line.

Example

```
# Calculating area of a square
>>> area = side **2
```

or

```
>>>area= side**2                    #calculating area of a square
```

For adding multi-line comment in a program, we can:

i) Place '#' in front of each line, or

ii) Use triple quoted string. They will only work as comment, when they are not being used as docstring. (A docstring is the first thing in a class/function /module, and will be taken up in details when we study functions).

The comment line "#calculating area of a rectangle" can also be written as following using triple quote:

1. """ Calculating area of a rectangle """

2. """ Calculating area

   of a rectangle """

We should use as many useful comments as we can, to explain

*Any assumptions made

*important details or decisions made in the program. This will make program more readable. We already know the importance of comments (documented in the program).

EXERCISE

1. Create following Variables

   i) 'mystring' to contain 'hello'

   ii) 'myfloat' to contain '2.5'

   iii) 'myint' to contain '10'

2. Write the value justification

   i) 2*(3+4)

   ii) 2*3+4

   iii) 2+3*4

3. What is the type of the following result:

   i) 1+2.0+3

4. Which of the following is the valid variable name:

   i) global

   ii) 99flag

   iii) sum

   iv) an$wer

5. True or False

   i) Character Data type values should be delimited by using the single quote.

   ii) None is one of the data type in python

   iii) The += operator is used to add the right hand side value to the left hand side variable.

   iv) The data type double is not a valid python data type.

   v) Python does not have any keywords

   vi) The equal to condition is written by using the == operator

6. Check all syntactically correct statements

a) Which input statements are correct

   i) a = raw_input ( )

   ii) a = raw_input (“enter a number”)

   iii) a = raw_imput (enter your name)

b) Which print statements are correct?

   i) _print “9” + “9”

   ii) _print int(“nine”)

   iii) _print 9+9

   iv) print 9

c) Which are correct arithmetical operations?

   i) a = 1*2

   ii) 2 = 1+1

   iii) 5 + 6 = y

   iv) Seven = 3 * 4

d) Which are correct type conversions?

   i) int (7.0+0.1)

   ii) str (1.2 * 3.4)

   iii) float (“77”+“.0”)

   iv) str (9 / 0)

e) Which operations result in 8?

   i) 65 // 8

   ii) 17 % 9

   iii) 2 * * 4

   iv) 64 * * 0.5

f) Which lines are commented?

i) “””This is a comment”””

ii) # This is a comment

iii) // this is a comment

iv) ‘ ‘ ‘ This is a comment’ ‘ ‘

g) Find the matching pairs of expressions and values.

i)	1023	boolean
ii)	None	int
iii)	[2, 4, 8, 16]	tuple
iv)	True	list
v)	17.54	str
vi)	(‘Roger’, 1952)	NoneType
vii)	“my fat cat”	float

7. MCQ

i) The __________ data type allows only True/False values

a) bool b) boolean c) Boolean d) None

ii) If the value of a = 20 and b = 20, then a+=b will assign ________ to a

a) 40 b) 30 c) 20 d) 10

iii) The ____________ operator is used to find out if division of two number yields any remainder

a) / b) + c) % d) //

8. When will following statement in interpreter result into error:

```
>>> B+4
```

9. How can we change the value of 6*1-2 to -6 from 4?

10. Is python case sensitive?

11. What does 'immutable' mean; which data type in python are immutable.

12. Name four of Python's Basic data types? Why are they called so?

13. What are relational operators? Explain with the help of examples.

14. What is an integer?

15. What is a variable? What names may variable have?

16. How are keywords different from variable names?

17. Why are data types important?

18. How can you convert a string to integer and when can it be used?

19. How can text be read from the keyboard?

20. How are comments written in a program?

LAB EXERCISE

1. Record what happens when following statements are executed:

    a) print n=7

    b) print 5+7

    c) print 5.2, "this", 4-2, "that", 5/2.0

2. Use IDLE to calculate:

    a) 6+4*10

    b) (6+4)*10

3. Type following mathematical expression and record your observations:

    a) 2**500

    b) 1/0

4. What will be the output of the following code:

```
a = 3 - 4 + 10
b = 5 * 6
```

```
c = 7.0/8.0

print "These are the values:", a, b, c
```

5. Write a code to show the use of all 6 math function.

6. Write a code that prints your full name and your Birthday as separate strings.

7. Write a program that asks two people for their names; stores the names in variables called name1 and name2; says hello to both of them.

8. Calculate root of the following equation:

   a) $34x^2 + 68x - 510$

   b) $2x^2 - x - 3 = 0$

Chapter 2

Functions

After studying this lesson, students will be able to:

- *Understand and apply the concept of module programming*
- *Write functions*
- *Identify and invoke appropriate predefined functions*
- *Create Python functions and work in script mode.*
- *Understand arguments and parameters of functions*
- *Work with different types of parameters and arguments*
- *Develop small scripts involving simple calculations*

Introduction

Remember, we earlier talked about working in script mode in chapter-1 of this unit to retain our work for future usage. For working in script mode, we need to *write a function* in the Python and save it in the file having **.py** extension.

A function is a named sequence of statement(s) that performs a computation. It contains line of code(s) that are executed sequentially from top to bottom by Python interpreter. They are the most important building blocks for any software in Python.

Functions can be categorized as belonging to

i. Modules

ii. Built in

iii. User Defined

Module

A module is a file containing Python definitions (i.e. functions) and statements. Standard library of Python is extended as module(s) to a programmer. Definitions from the module can be used within the code of a program. To use these modules in the program, a programmer needs to import the module. Once you import a module, you can reference (use), any of its functions or variables in your code. There are many ways to import a module in your program, the one's which you should know are:

i. import

ii. from

Import

It is simplest and most common way to use modules in our code. Its syntax is:

import modulename1 [,modulename2, ---------]

Example

```
>>> import math
```

On execution of this statement, Python will

(i) search for the file '**math.py**'.

(ii) Create space where modules definition & variable will be created,

(iii) then execute the statements in the module.

Now the definitions of the module will become part of the code in which the module was imported.

To use/ access/invoke a function, you will specify the module name and name of the function- separated by dot (.). This format is also known as *dot notation*.

Example

```
>>> value= math.sqrt (25)          # dot notation
```

The example uses sqrt( ) function of module **math** to calculate square root of the value provided in parenthesis, and returns the result which is inserted in the ***value***. The expression (variable) written in parenthesis is known as argument (actual argument). It is common to say that the function takes arguments and return the result.

This statement invokes the sqrt ( ) function. We have already seen many function invoke statement(s), such as

```
>>> type ( )

>>> int ( ), etc.
```

From Statement

It is used to get a specific function in the code instead of the complete module file. If we know beforehand which function(s), we will be needing, then we may use **from**. For modules having large no. of functions, it is recommended to use **from** instead of import. Its syntax is

>>> from modulename import functionname [, functionname…..]

Example

```
>>> from math import sqrt

value = sqrt (25)
```

Here, we are importing sqrt function only, instead of the complete math module. Now sqrt( ) function will be directly referenced to. These two statements are equivalent to previous example.

```
from modulename import *
```

will import everything from the file.

Note: You normally put all import statement(s) at the beginning of the Python file but technically they can be anywhere in program.

Lets explore some more functions available in **math module:**

Name of the function	Description	Example

ceil(x)	It returns the smallest integer not less than x, where *x is a numeric expression.*	math.ceil(-45.17)
		-45.0
		math.ceil(100.12)
		101.0
		math.ceil(100.72)
		101.0

floor(x)	It returns the largest integer not greater than x, where *x is a numeric expression.*	math.floor(-45.17)
		-46.0
		math.floor(100.12)
		100.0
		math.floor(100.72)
		100.0

fabs(x)	It returns the absolute value of x, *where x is a numeric value.*	math.fabs(-45.17)
		45.17
		math.fabs(100.12)
		100.12
		math.fabs(100.72)
		100.72

exp(x)	It returns exponential of x: e^x, where x is a numeric expression.	math.exp(-45.17)
		2.41500621326e-20
		math.exp(100.12)
		3.03084361407e+43
		math.exp(100.72)
		5.52255713025e+43

log(x)	It returns natural logarithm of x, for x > 0, where *x is a numeric expression.*	math.log(100.12)
		4.60636946656
		math.log(100.72)
		4.61234438974

log10(x)	It returns base-10 logarithm of x for x > 0, where *x is a numeric expression.*	math.log10(100.12)
		2.00052084094
		math.log10(100.72)
		2.0031157171

pow(x, y)	It returns the value of x^y, *where x and **y** are numeric expressions.*	math.pow(100, 2)
		10000.0
		math.pow(100, -2)

		0.0001 math.pow(2, 4) **16.0** math.pow(3, 0) **1.0**

sqrt (x)	It returns the square root of x for x > 0, where x is a numeric expression.	math.sqrt(100)
		10.0
		math.sqrt(7)
		2.64575131106

cos (x)	It returns the cosine of x in radians, *where x is a numeric expression*	math.cos(3)
		-0.9899924966
		math.cos(-3)
		-0.9899924966
		math.cos(0)
		1.0
		math.cos(math.pi)
		-1.0

sin (x)	It returns the sine of x, in radians, *where x must be a numeric value.*	math.sin(3)
		0.14112000806
		math.sin(-3)
		-0.14112000806
		math.sin(0)
		0.0

tan (x)	It returns the tangent of x in radians, *where x must be a numeric value.*	math.tan(3)
		-0.142546543074
		math.tan(-3)
		0.142546543074
		math.tan(0)
		0.0

degrees (x)	It converts angle x from radians to degrees, *where x must be a numeric value.*	math.degrees(3)
		171.887338539
		math.degrees(-3)
		-171.887338539
		math.degrees(0)
		0.0

radians(x)	It converts angle x from degrees to radians, *where x must be a numeric value.*	math.radians(3)
		0.0523598775598
		math.radians(-3)
		-0.0523598775598
		math.radians(0)
		0.0

Some functions from **random module** are:

Name of the function	**Description**	**Example**
random ( )	It returns a random float x, such that 0 ≤ x<1	>>>random.random ( ) **0.281954791393** >>>random.random ( ) **0.309090465205**

randint (a, b)	It returns a int x between a & b such that
	a ≤ x ≤ b
		>>> random.randint (1,10)
		5
		>>> random.randint (-2,20)
		-1

uniform (a,b)	It returns a floating point number x, such that a <= x < b	>>>random.uniform (5, 10) **5.52615217015**

randrange ([start,] stop [,step])	It returns a random item from the given range	>>>random.randrange(100,1000,3) **150**

Some of the other modules, which you can explore, are: string, time, date

Built in Function

Built in functions are the function(s) that are built into Python and can be accessed by a programmer. These are always available and for using them, we don't have to import any module (file). Python has a small set of built-in functions as most of the functions have been partitioned to modules. This was done to keep core language precise.

Name	Description	Example
abs (x)	It returns distance between x and zero, where *x is a numeric expression.*	>>>abs(-45) **45** >>>abs(119L) **119**
max(x, y, z, ...)	It returns the largest of its arguments: where x, y and z are numeric variable/expression.	>>>max(80, 100, 1000) **1000** >>>max(-80, -20, -10) **-10**
min(x, y, z, ...)	It returns the smallest of its arguments; where x, y, and z are numeric variable/expression.	>>> min(80, 100, 1000) **80** >>> min(-80, -20, -10) **-80**
cmp(x, y)	It returns the sign of the difference of two numbers: -1 if x < y, 0 if x == y, or 1 if x > y, *where x and y are numeric variable/expression.*	>>>cmp(80, 100) **-1** >>>cmp(180, 100) **1**

divmod (x,y)	Returns both quotient and remainder by division through a tuple, when x is divided by y; where x & y are variable/expression.	>>> divmod (14,5)
		(2,4)
		>>> divmod (2.7, 1.5)
		(1.0, 1.20000)

len (s)	Return the length (the number of items) of an object. The argument may be a sequence (string, tuple or list) or a mapping (dictionary).	>>> a= [1,2,3]
		>>>len (a)
		3
		>>> b= 'Hello'
		>>> len (b)
		5

range (*start*, *stop*[, *step*])	This is a versatile function to create lists containing arithmetic progressions. It is most often used in for loops. The arguments must be plain integers. If the *step* argument is omitted, it defaults to 1. If the *start* argument is omitted, it defaults to 0. The full form returns a list of plain integers [start, start + step, start + 2 * step, ...]. If *step* is positive, the last element is the largest start + i * step less than *stop*; if *step* is negative, the last element is the smallest start + i * step greater than *stop*. *step* must not be zero (or else Value Error is raised).	>>> range(10)
		[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
		>>> range(1, 11)
		[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
		>>> range(0, 30, 5)
		[0, 5, 10, 15, 20, 25]
		>>> range(0, 10, 3)
		[0, 3, 6, 9]
		>>> range(0, -10, -1)
		[0, -1, -2, -3, -4, -5, -6, -7, -8, -9]
		>>> range(0)
		[]
		>>> range(1, 0)
		[]

round(x [, n])	It returns float x rounded to n digits from the decimal point, *where x and n are numeric expressions.* If n is not provided then x is rounded to 0 decimal digits.	>>>round(80.23456, 2) **80.23** >>>round(-100.000056, 3) **-100.0** >>> round (80.23456) **80.0**

Apart from these functions, you have already seen the use of the following functions:

bool ( ), chr ( ), float ( ), int ( ), long (), str ( ), type ( ), id ( ), tuple ( )

Composition

Composition is an art of combining simple function(s) to build more complicated ones, i.e., result of one function is used as the input to another.

Example

Suppose we have two functions fn1 & fn2, such that

```
a= fn2 (x)

b= fn1 (a)
```

then call to the two functions can be combined as

```
b= fn1 (fn2 (x))
```

Similarly, we can have statement composed of more than two functions. In that result of one function is passed as argument to next and result of the last one is the final result.

Example

```
math.exp (math.log (a+1))
```

Example

```
degrees=270

math.sin (degrees/360.0 *2*math.pi)
```

Composition is used to package the code into modules, which may be used in many different unrelated places and situations. Also it is easy to maintain the code.

Note: Python also allow us to take elements of program and compose them.

User Defined Functions

So far we have only seen the functions which come with Python either in some file (module) or in interpreter itself (built in), but it is also possible for programmer to write their own function(s). These functions can then be combined to form a module which can then be used in other programs by importing them.

To define a function keyword **def** is used. After the keyword comes an identifier i.e. name of the function, followed by parenthesized list of parameters and the colon which ends up the line. Next follows the block of statement(s) that are the part of function.

Before learning about Function header & its body, lets explore block of statements, which become part of function body.

Block of statements

A block is one or more lines of code, grouped together so that they are treated as one big sequence of statements while executing. In Python, statements in a block are written with indentation. Usually, a block begins when a line is indented (by four spaces) and all the statements of the block should be at same indent level. A block within block begins when its first statement is indented by four space, i.e., in total eight spaces. To end a block, write the next statement with the same indentation before the block started.

Now, lets move back to function- the **Syntax** of function is:

```
def NAME ([PARAMETER1, PARAMETER2, …..]): #Square brackets include
statement(s)                              #optional part of statement
```

Let's write a function to greet the world:

```
def sayHello ():             # Line No. 1
    print "Hello World!"     # Line No.2
```

The first line of function definition, i.e., Line No. 1 is called **header** and the rest, i.e. Line No. 2 in our example, is known as **body**. Name of the function is ***sayHello***, and empty parenthesis indicates no parameters. Body of the function contains one Python statement, which displays a string constant on screen. So the general structure of any function is

Function Header

It begins with the keyword def and ends with colon and contains the function identification details. As it ends with colon, we can say that what follows next is, block of statements.

Function Body

Consisting of sequence of indented (4 space) Python statement(s), to perform a task.

Defining a function will create a variable with same name, but does not generate any result. The body of the function gets executed only when the function is called/invoked. Function **call** contains the name of the function (being executed) followed by the list of values (i.e. arguments) in parenthesis. These arguments are assigned to parameters from LHS.

```
    >>> sayHello ()          # Call/invoke statement of this function
```

Will produce following on screen

```
Hello World!
```

Apart from this, you have already seen many examples of invoking of functions in Modules & Built-in Functions.

Let's know more about **def.** It is an executable statement. At the time of execution a function is created and a name (name of the function) is assigned to it. Because it is a statement, **def** can appear anywhere in the program. It can even be nested.

Example

```
    if condition:
        def fun ( ):                # function definition one way
    .
```

```
.
.
else:
        def fun ( ):                    # function definition other way
.
.
.
fun ( )                         #  calls the function selected.
```

This way we can provide an alternative definition to the function. This is possible because def is evaluated when it is reached and executed.

```
def fun (a):
```

Let's explore Function body

The first statement of the function body can optionally be a string constant, **docstring,** enclosed in triple quotes. It contains the essential information that someone might need about the function, such as

- What function does (**without How it does**) i.e. summary of its purpose
- Type of parameters it takes
- Effect of parameter on behavior of functions, etc.

DocString is an important tool to document the program better, and makes it easier to understand. We can actually access docstring of a function using __ doc__ (function name). Also, when you used help(), then Python will provide you with docstring of that function on screen. So it is strongly recommended to use docstring … when you write functions.

Example

```
def area (radius):
        """ calculates area of a circle.                      docstring begins
        require an integer or float value to calculate area.
        returns the calculated value to calling function """  docstring ends
```

```
    a=radius**2

    return a
```

Function is pretty simple and its objective is pretty much clear from the docString added to the body.

The last statement of the function, i.e. return statement returns a value from the function. Return statement may contain a constant/literal, variable, expression or function, if return is used without anything, it will return **None.** In our example value of a variable **area** is returned.

Instead of writing two statements in the function, i.e.

```
    a = radius **2

    return a
```

We could have written

```
    return radius **2
```

Here the function will first calculate and then return the value of the expression.

It is possible that a function might not return a value, as sayHello( ) was not returning a value. sayHello( ) prints a message on screen and does not contain a return statement, such functions are called **void functions**.

Void functions might display something on the screen or have some other effect, but they don't have a return value. If you try to assign the result of such function to a variable, you get a special value called **None**.

Example

```
    def check (num):

        if (num%2==0):

        print "True"

   else:

        print "False"

>>> result = check (29)
```

False

```
>>> print result
```

None

DocString Conventions:

- The first line of a docstring starts with capital letter and ends with a period (.)
- Second line is left blank (it visually separates summary from other description).
- Other details of docstring start from 3rd line.

Parameters and Arguments

Parameters are the value(s) provided in the parenthesis when we write function header. These are the values required by function to work. Let's understand this with the help of function written for calculating area of circle.

radius is a parameter to function area.

If there is more than one value required by the function to work on, then, all of them will be listed in parameter list separated by comma.

Arguments are the value(s) provided in function call/invoke statement. List of arguments should be supplied in same way as parameters are listed. Bounding of parameters to arguments is done 1:1, and so there should be same number and type of arguments as mentioned in parameter list.

Example

of argument in function call

```
>>> area (5)
```

5 is an argument. An argument can be constant, variable, or expression.

Scope of Variables

Scope of variable refers to the part of the program, where it is visible, i.e., area where you can refer (use) it. We can say that scope holds the current set of variables and their values. We will study two types of scope of variables- global scope or local scope.

Global Scope

A variable, with global scope can be used anywhere in the program. It can be created by defining a variable outside the scope of any function/block.

Example

```
x=50
def test ( ):
    print "Inside test x is" , x
print "Value of x is" , x
```

on execution the above code will produce

```
Inside test x is 50
Value of x is 50
```

Any modification to global is permanent and visible to all the functions written in the file.

Example

```
x=50
def test ( ):
    x+= 10
    print "Inside test x is", x
print "Value of x is", x
```

will produce

```
Inside test x is 60
Value of x is 60
```

Local Scope

A variable with local scope can be accessed only within the function/block that it is created in. When a variable is created inside the function/block, the variable becomes local to it. A local variable only exists while the function is executing.

Example

```
X=50
def test ( ):
          y = 20
          print 'Value of x is ', X, ';  y is ' , y
print 'Value of x is ', X, ' y is ' , y
```

On executing the code we will get

Value of x is 50; y is 20

The next print statement will produce an error, because the variable **y** is not accessible outside the function body.

A global variable remains global, till it is not recreated inside the function/block.

Example

```
x=50
def test ( ):
          x=5
          y=2
          print 'Value of x & y inside the function are ' , x , y
print 'Value of x outside the function is ' , x
```

This code will produce following output:

Value of x & y inside the function are 5 2

Value of x outside the function is 50

If we want to refer to global variable inside the function then keyword global will be prefixed with it.

Example

```
x=50
```

```
def test ( ):
        global x
        x =2
        y = 2
        print 'Value of x & y inside the function are ' , x , y
print 'Value of x outside function is ' , x
```

This code will produce following output:

Value of x & y inside the function are 2 2

Value of x outside the function is 2

More on defining Functions

It is possible to provide parameters of function with some default value. In case the user does not want to provide values (argument) for all of them at the time of calling, we can provide default argument values.

Example

Default value to parameter

```
def   greet   (message,
times=1):
        print message * times
>>> greet ('Welcome')        # calling function with one argument value
>>> greet ('Hello', 2)       # calling function with both the argument values.
```

Will result in:

Welcome

HelloHello

The function **greet ()** is used to print a message (string) given number of times. If the second argument value, is not specified, then parameter **times** work with the default value provided to it. In the first call to greet ( ), only one argument value is provided, which is passed on to the first parameter from LHS and the **string is printed only once**

as the variable times take default value 1. In the second call to greet ( ), we supply both the argument values a **string** and **2**, saying that we want to print the message twice. So now, parameter **times** get the value **2** instead of default 1 and the message is printed twice.

As we have seen functions with default argument values, they can be called in with fewer arguments, then it is designed to allow.

Note:

- The default value assigned to the parameter should be a constant only.
- Only those parameters which are at the end of the list can be given default value. You cannot have a parameter on left with default argument value, without assigning default values to parameters lying on its right side.
- The default value is evaluated only once, at the point of function definition.

If there is a function with many parameters and we want to specify only some of them in function call, then value for such parameters can be provided by using their **name**, instead of the position (order)- this is called **keyword arguments**.

```
def fun(a, b=1, c=5):
        print 'a is ', a, 'b is ', b, 'c is ', c
```

The function fun can be invoked in many ways

1. >>>fun (3)

   a is 3 b is 1 c is 5

2. >>>fun (3, 7, 10)

   a is 3 b is 7 c is 10

3. >>>fun (25, c = 20)

   a is 25 b is 1 c is 20

4. >>>fun (c = 20, a = 10)

   a is 10 b is 1 c is 20

1st and 2nd call to function is based on default argument value, and the 3rd and 4th call are using **keyword arguments.**

In the first usage, value 3 is passed on to **a, b** & **c** works with default values. In second call, all the three parameters get values in function call statement. In third usage, variable **a** gets the first value 25, due to the position of the argument. And parameter **c** gets the value 20 due to naming, i.e., keyword arguments. The parameter **b** uses the default value.

In the fourth usage, we use keyword argument for all specified value, as we have specified the value for **c** before **a**; although **a** is defined before **c** in parameter list.

Note: The function named fun ( ) have three parameters out of which first one is without default value and other two have default values. So any call to the function should have at least one argument.

While using keyword arguments, following should be kept in mind:

- ✡ An argument list must have any positional arguments followed by any keywords arguments.
- ✡ Keywords in argument list should be from the list of parameters name only.
- ✡ No parameter should receive value more than once.
- ✡ Parameter names corresponding to positional arguments cannot be used as keywords in the same calls.

Following calls to fun () would be invalid

```
fun ()                # required argument missing
fun (5, a=5, 6)  # non keyword argument (6) following keyword argument
fun (6, a=5)          # duplicate value for argument a
fun (d=5)             # unknown parameter
```

Advantages of writing functions with keyword arguments are:

- Using the function is easier as we do not need to remember about the order of the arguments.
- We can specify values of only those parameters to which we want to, as - other parameters have default argument values.

In python, as function definition happens at run time, so functions can be bound to other names. This allow us to

(i) Pass function as parameter

(ii) Use/invoke function by two names

Example

```
def x ( ):
    print 20
    >>> y=x
    >>>x ( )
    >>>y ( )
    20
```

Example

```
def x ( ):
    print 20
    def test (fn):
    for I in range (4):
    fn( )
    >>> test (x)
    20
    20
```

```
20

20
```

Flow of Execution of program containing Function call

Execution always begins at the first statement of the program. Statements are executed one at a time, in order from top to bottom. Function definition does not alter the flow of execution of program, as the statement inside the function is not executed until the function is called.

On a function call, instead of going to the next statement of program, the control jumps to the body of the function; executes all statements of the function in the order from top to bottom and then comes back to the point where it left off. This remains simple, till a function does not call another function. Simillarly, in the middle of a function, program might have to execute statements of the other function and so on.

Don't worry; Python is good at keeping track of execution, so each time a function completes, the program picks up from the place it left last, until it gets to end of program, where it terminates.

Note:

- ✡ Python does not allow you to call a function before the function is declared.
- ✡ When you write the name of a function without parenthesis, it is interpreted as the reference, when you write the function name with parenthesis, the interpreter invoke the function (object).

EXERCISE

1. The place where a variable can be used is called its

   a) area b) block

   c) function d) Scope

2. True or False

   i. Every variable has a scope associated with it.

   ii. ! (p or q) is same as !p or !q

3. What will be the output of the following? Explain:

```
def f1 ( ):
    n = 44
def f2( ):
    n=77
    print "value of n", n
    print "value of n", n
```

4. For each of the following functions. Specify the type of its **output**. You can assume each function is called with an appropriate argument, as specified by its docstrings.

a)
```
def a (x):
'''
x: int or float.
'''
return x+1
```

b)
```
def b (x):
'''
x: int or float.
'''
```

```
        return x+1.0
c)      def c (x, y):
            '''
            x: int or float.
            y: int or float.
            '''
            return x+y
d)      def e (x, y,z):
            '''
            x: can be of any type.
            y: can be of any type.
            z: can be of any type
            '''
            return x >= y and x <= z
e)      def d (x,y):
            '''
            x: can be of any type.
            y: can be of any type.
            '''
            return x > y
```

5. Below is a transcript of a session with the Python shell. Assume the functions in previous question (Q 4) have been defined. Provide the type and value of the expressions being evaluated.

i)	a (6)	ii)	a (-5. 3)
iii)	a (a(a(6)))	iv)	c (a(1), b(1))
v)	d ('apple', 11.1)		

6. Define a function **get Bigger Number (x,y)** to take in two numbers and return the bigger of them.

7. What is the difference between methods, functions & user defined functions.

8. Open help for math module

   i. How many functions are there in the module?

   ii. Describe how square root of a value may be calculated without using a math module

   iii. What are the two data constants available in math module.

9. Generate a random number *n* such that

   i. $0 \leq n < 6$

   ii. $2 \leq n < 37$ and *n* is even

LAB EXERCISE

1. Write a program to ask for following as input

   Enter your first name: Rahul

   Enter your last name: Kumar

   Enter your date of birth

   Month? March

   Day? 10

   Year? 1992

   And display following on screen

   Rahul Kumar was born on March 10, 1992.

2. Consider the following function definition:

```
def intDiv (x, a):
    """
    x: a non-negative integer argument
```

```
        a: a positive integer argument
        returns: integer, the integer division of x divided by a.
        """
        while x>=a:
        count +=1
        x = x-a
        return count
```

when we call

print intDiv (5, 3)

We get an error message. Modify the code so that error does not occur.

3. Write a script that asks a user for a number. Then adds 3 to that number, and then multiplies the result by 2, subtracts twice the original number, then prints the result.

4. In analogy to the example, write a script that asks users for the temperature in F and prints the temperature in C. (Conversion: Celsius = (F - 32) * 5/9).

5. Write a Python function, odd, that takes in one number and returns True when the number is odd and False otherwise. You should use the % (mod) operator, not **if**.

6. Define a function 'SubtractNumber(x,y)' which takes in two numbers and returns the difference of the two.

7. Write a Python function, fourthPower( ), that takes in one number and returns that value raised to the fourth power.

8. Write a program that takes a number and calculate and display the log, square, sin and cosine of it.

9. a) Write a program, to display a tic-tac-toe board on screen, using print statement.

   b) Write a program to display a tic-tac-toe board on screen using variables, so that you do not need to write many print statements?

10. Write a function roll_D ( ), that takes 2 parameters- the no. of sides (with default value 6) of a dice, and the number of dice to roll-and generate random roll values for each dice rolled. Print out each roll and then return one string "That's all".

    Example roll_D (6, 3)

    ```
    4
    1
    6
    That's all
    ```

Chapter 3

Conditional and Looping Construct

After studying this lesson, students will be able to:

- *Understand the concept and usage of selection and iteration statements.*
- *Know various types of loops available in Python.*
- *Analyze the problem, decide and evaluate conditions.*
- *Will be able to analyze and decide for an appropriate combination of constructs.*
- *Write code that employ decision structures, including those that employ sequences of decision and nested decisions.*
- *Design simple applications having iterative nature.*

Control Flow Structure

Such as depending on time of the day you wish Good Morning or Good Night to people. Similarly while writing program(s), we almost always need the ability to check the condition and then change the course of program, the simplest way to do so is using **if** statement

```
if x > 0:

   print 'x is positive'
```

Here, the Boolean expression written after **if** is known as condition, and if Condition is **True,** then the statement written after, is executed. Let's see the syntax of **if** statement

Option 1	**Option 2**

if condition:
   STATEMENTs- **BLOCK 1**
[else:
	if condition-1:
	   STATEMENTs- **BLOCK 1**
	[elif condition-2:

STATEMENTs- **BLOCK 2**]	STATEMENTs- **BLOCK 2** else: STATEMENTs- **BLOCK N**]

Statement with in [ ] bracket are optional.

Let us understand the syntax, in Option 1- if the condition is **True** (i.e. satisfied), the statement(s) written after **if** (i.e. STATEMENT-BLOCK 1) is executed, otherwise statement(s) written after else (i.e. STATEMENT-BLOCK 2) is executed. Remember **else** clause is optional. If provided, in any situation, one of the two blocks get executed not both.

We can say that, 'if' with 'else' provides an alternative execution, as there are two possibilities and the condition determines which one gets executed. If there are more than two possibilities, such as based on percentage print grade of the student.

Percentage Range	**Grade**
> 85	A
> 70 to <=85	B
> 60 to <=70	C
> 45 to <=60	D

Then we need to chain the **if** statement(s). This is done using the 2nd option of **if** statement. Here, we have used **'elif'** clause instead of 'else'. **elif** combines **if else- if else** statements to one **if elif …else**. You may consider elif to be an abbreviation of *else if*. There is no limit to the number of 'elif' clause used, but if there is an 'else' clause also it has to be at the end.

Example for combining more than one condition:

```
if perc > 85:
      print 'A'
elif perc >70 and perc <=85:          #alternative to this is if 70 <perc<85
```

```
        print 'B'
    elif perc > 60 and perc <=70:          #if 60 <perc <=70
        print 'C'
    elif perc >45 and perc <=60:
        print 'D'
```

In the chained conditions, each condition is checked in order– if previous is F**alse** then next is checked, and so on. If one of them is **True** then corresponding block of statement(s) are executed and the statement ends i.e., control moves out of 'if statement'. If none is true, then else block gets executed if provided. *If more than one condition is true, then only the first true option block gets executed.*

If you look at the conditional construct, you will find that it has same structure as function definition, terminated by a colon. Statements like this are called compound statements. In any compound statement, there is no limit on how many statements can appear inside the body, but there has to be at least one. Indentation level is used to tell Python which statement (s) belongs to which block.

There is another way of writing a simple if else statement in Python. The complete simple if, can be written as:

Variable= variable 1 if condition else variable 2.

In above statement, on evaluation, if condition results into True then variable 1 is assigned to Variable otherwise variable 2 is assigned to Variable.

Example

```
>>> a =5
>>> b=10
>>> x = True
>>> y = False
>>>result = x if a <b else y
```

Will assign **True** to result

Sometimes, it is useful to have a body with no statements, in that case you can use **pass** statement. Pass statement does nothing.

Example

```
if condition:
pass
```

It is possible to have a condition within another condition. Such conditions are known as **Nested Condition.**

Example

```
if x==y:
      print x, ' and ', y, ' are equal'
else:
if x<y:
      print x, ' is less than ', y          } Nested if
else:
      print x, ' is greater than ', y
```

Here a complete **if… else** statement belongs to else part of outer if statement.

Note: The condition can be any Python expression (i.e. something that returns a value). Following values, when returned through expression are considered to be False:

None, Number Zero, A string of length zero, an empty collection

Looping Constructs

We know that computers are often used to automate the repetitive tasks. One of the advantages of using computer to repeatedly perform an identical task is that –it is done without making any mistake. Loops are used to repeatedly execute the same code in a program. Python provides two types of looping constructs:

1) While statement

2) For statement

While Statements

Its syntax is:

```
while condition:                # condition is Boolean expression returning True or False
   STATEMENTs BLOCK 1
[else:                          # optional part of while
   STATEMENTs BLOCK 2]
```

We can see that while looks like if statement. The statement bExampleins with keyword **while** followed by boolean condition followed by colon (:). What follows next is block of statement(s).

The statement(s) in BLOCK 1 keeps on executing till condition in while remains **True;** once the condition becomes **False** and if the else clause is written in while, then else will get executed. While loop may not execute even once, if the condition evaluates to false, initially, as the condition is tested before entering the loop.

Example

a loop to print nos. from 1 to 10

```
i=1
while (i <=10):
    print i,
    i = i+1          #could be written as i+=1
```

You can almost read the statement like English sentence. The first statement initialized the variable (controlling loop) and then **while** evaluates the condition, which is True so the block of statements written next will be executed.

Last statement in the block ensures that, with every execution of loop, **loop control variable** moves near to the termination point. If this does not happen then the loop will keep on executing infinitely.

As soon as **i becomes 11**, condition in while will evaluate to **False** and this will terminate the loop. Result produced by the loop will be:

1 2 3 4 5 6 7 8 9 10

As there is ',' after print i all the values will be printed in the same line

Example

```
i=1
while (i <=10):
    print i,
    i+ =1
else:
    print               # will bring print control to next printing line
    print "coming out of loop"
```

Will result into

1 2 3 4 5 6 7 8 9 10

coming out of loop

Nested loops

Block of statement belonging to while can have another while statement, i.e. a while can contain another while.

Example

```
i=1
while i<=3:
    j=1
    while j<=i:  }
    print j,     } # inner while loop
    j=j+1        }
```

```
        print
        i=i+1
```

will result into

```
1
1 2
1 2 3
```

For Statement

Its Syntax is

```
for TARGET- LIST in EXPRESSION-LIST:
   STATEMENT BLOCK 1
[else:                                # optional block
   STATEMENT BLOCK 2]
```

Example

```
# loop to print value 1 to 10
for i in range (1, 11, 1):
        print i,
```

Execution of the loop will result into

```
1 2 3 4 5 6 7 8 9 10
```

Let's understand the flow of execution of the statement:

The statement introduces a function range ( ), its syntax is

```
range(start, stop, [step])                # step is optional
```

range( ) generates a list of values starting from **start** till **stop-1**. Step if given is added to the value generated, to get next value in the list. *You have already learnt about it in built-in functions.*

Let's move back to the for statement: **i** is the variable, which keeps on getting a value generated by range ( ) function, and the block of statement (s) are worked on for each value of **i**. As the last value is assigned to **i,** the loop block is executed last time and control is returned to next statement. If **else** is specified in **for** statement, then next statement executed will be else. Now we can easily understand the result of **for** statement. range( ) generates a list from 1, 2, 3, 4, 5, …., 10 as the step mentioned is 1, **i** keeps on getting a value at a time, which is then printed on screen.

Apart from range( ) i (loop control variable) can take values from string, list, dictionary, etc.

Example

```
for letter in 'Python':
    print 'Current Letter', letter
else:
    print 'Coming out of loop'
```

On execution, will produce the following:

```
Current Letter: P
Current Letter: y
Current Letter: t
Current Letter: h
Current Letter: o
Current Letter: n
Coming out of loop
```

A **for** statement can contain another **for** statement or **while** statement. We know such statement form nested loop.

Example

```
# to print table starting from 1 to specified no.
n=2
for i in range (1, n+1):
    j=1
```

```
          print “Table to “, i, “is as follows”
          while j <6:
    print i, “*”, j “=”, i*j
          j = j+1
          print
```

Will produce the result

```
Table to 1 is as follows
1 * 1 = 1
1 * 2 = 2
1 * 3 = 3
1 * 4 = 4
1 * 5 = 5
Table to  2 is as follows
2 * 1 = 2
2 * 2 = 4
2 * 3 = 6
2 * 4 = 8
2 * 5 = 10
```

Nesting a **for loop** within **while loop** can be seen in following example :

Example

```
i = 6
while i >= 0:
      for j in range (1, i):
            print j,
      print
```

```
        i=i-1
```

will result into

1 2 3 4 5

1 2 3 4

1 2 3

1 2

1

By now, you must have realized that, Syntax of **for statement** is also same as **if statement** or **while statement.**

Let's look at the equivalence of the two looping construct:

While	**For**
i= initial value	for i in range (initial value, limit, step):
while (i <limit):	statement(s)
statement(s)	
i+=step	

Break Statement

Break can be used to unconditionally jump out of the loop. It terminates the execution of the loop. Break can be used in while loop and for loop. Break is mostly required, when because of some external condition, we need to exit from a loop.

Example

```
for letter in 'Python':
if letter = = 'h':
        break
        print letter
```

will result into

```
P
y
t
```

Continue Statement

This statement is used to tell Python to skip the rest of the statements of the current loop block and to move to next iteration, of the loop. Continue will return back the control to the bExampleinning of the loop. This can also be used with both while and for statement.

Example

```
for letter in 'Python':
if letter == 'h':
    continue
    print letter
will result into
P
y
t
o
n
```

EXERCISE

1) Mark True/False:

   (i) While statements gets executed at least once

   (ii) The break statement allows us to come out of a loop

   (iii) The continue and break statement have same effect

   (iv) We can nest loops

   (v) We cannot write a loop that can execute forever.

   (vi) Checking condition in python can be done by using the if-else statement

2) What is the difference between the following two statements:

   (i)

```
if n>2:
if n <6 :
        print 'OK'
    else:
        print 'NG'
```

   (ii)

```
if n>2:
    if n<6:
        print 'OK'
    else:
        print 'NG'
```

3) Mark the correct Option(s)

   (i) If there are two or more options, then we can use

a)	Simple if statement	b)	If elif statement
c)	While	d)	None of these

(ii) A loop that never ends is called a:

a) Continue loop b) Infinite loop

c) Circle loop d) None of these

4) Construct a logical expression to represent each of the following conditions:

(i) Score is greater than or equal to 80 but less than 90

(ii) Answer is either 'N' or 'n'

(iii) N is between 0 and 7 but not equal to 3

5) Which of the following loop will continue infinitely:

(i) a) while O: b) while 1:

c) while :1: d) while False:

(ii) We can go back to the start of the loop by using __________

a) loop b) back

c) start d) continue

6) What is the difference between selection and repetition?

7) Explain use of if statement with example.

LAB EXERCISE

1)
```
answer = raw_input("Do you like Python? ")
if answer == "yes":
print "That is great!"
else:
    print "That is disappointing!"
```

Modify the program so that it answers "That is great!" if the answer was "yes", "That is disappointing" if the answer was "no" and "That is not an answer to my question." otherwise.

2) Write a function to find whether given number is odd or even.

3) Print all multiples of 13 that are smaller than 100. Use the range function in the following manner: range (start, end, step) where "start" is the starting value of the counter, "end" is the end value and "step" is the amount by which the counter is increased each time.

4) Write a program using while loop that asks the user for a number, and prints a countdown from that number to zero. **Note:** Decide on what your program will do, if user enters a nExampleative number.

5) Using for loop, write program that prints out the decimal equivalent of ½, $^1/_3$, ¼, ------, $^1/_{10}$

6) Write a function to print the Fibonacci Series up to an Input Limit.

7) Write a function to generate and print factorial numbers up to *n* (provided by user).

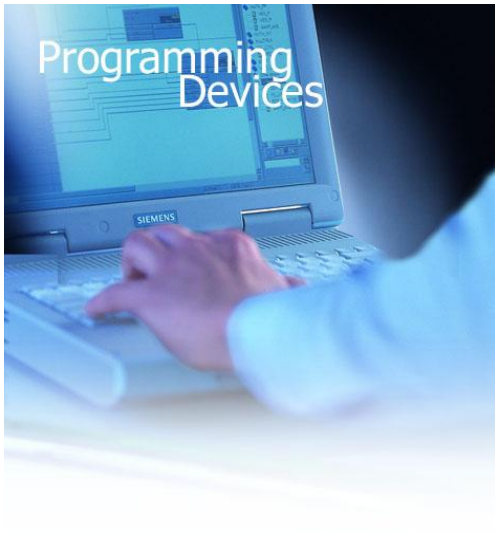
8) Write a program using a for loop, that calculates exponentials. Your program should ask for base and exp. value form user. **Note:** Do not use ** operator and math module.

9) Write a program using loop that asks the user to enter an even number. If the number entered is not even then display an appropriate message and ask them to enter a number again. Do not stop until an even number is entered. Print a Congratulatory message at end.

10) Using random module, Simulate tossing a coin N times. Hint: you can use zero for head and 1 for tails.

UNIT 4

Programming with Python



Chapter 1

Strings

After studying this lesson, students will be able to:

- *Learn how Python inputs strings*
- *Understand how Python stores and uses strings*
- *Perform slicing operations on strings*
- *Traverse strings with a loop*
- *Compare strings and substrings*
- *Understand the concept of immutable strings*
- *Understanding string functions.*
- *Understanding string constants*

Introduction

In python, consecutive sequence of characters is known as a string. An individual character in a string is accessed using a subscript (index). The subscript should always be an integer (positive or negative). A subscript starts from 0.

Example

```
# Declaring a string in python
>>>myfirst="Save Earth"
>>>print myfirst
Save Earth
```

Let's play with subscripts

To access the **first character** of the string

```
>>>print myfirst[0]
S
```

To access the **fourth character** of the string

```
>>>print myfirst[3]
e
```

To access the **last character** of the string

```
>>>print myfirst[-1]
>>h
```

To access the **third last character** of the string

```
>>>print myfirst[-3]
r
```

Consider the given figure

String A	H	E	L	L	O
Positive Index	0	1	2	3	4
Negative Index	-5	-4	-3	-2	-1

Important points about accessing elements in the strings using subscripts

- ✡ Positive subscript helps in accessing the string from the beginning
- ✡ Negative subscript helps in accessing the string from the end.
- ✡ Subscript 0 or –ve n(where n is length of the string) displays the first element.

  Example : A[0] or A[-5] will display 'H'
- ✡ Subscript 1 or –ve (n-1) displays the second element.

Note: Python does not support character data type. A string of size 1 can be treated as characters.

Creating and initializing strings

A literal/constant value to a string can be assigned using a single quotes, double quotes or triple quotes.

✡ Enclosing the string in single quotes

Example

```
>>>print (‘A friend in need is a friend indeed’)
A friend in need is a friend indeed
```

Example

```
>>>print(‘ This book belongs to Raghav\’s sister’)
This book belongs to Raghav’s sister
```

As shown in example 2, to include the single quote within the string it should be preceded by a backslash.

✡ Enclosing the string in double quotes

Example

```
>>>print(“A room without books is like a body without a soul.”)
A room without books is like a body without a soul.
```

✡ Enclosing the string in triple quote

Example

```
>>>life=”””\” Live as if you were to die tomorrow.
      Learn as if you were to live forever.\”
      ---- Mahatma Gandhi “””
>>> print life
      ” Live as if you were to die tomorrow.
    Learn as if you were to live forever.”
    ---- Mahatma Gandhi “””
```

Triple quotes are used when the text is multiline.

In the above example, backslash (\) is used as an escape sequence. An escape sequences is nothing but a special character that has a specific function. As shown above, backslash (\) is used to escape the double quote.

Escape sequence	Meaning	Example
\n	New line	>>> print "Hot\nCold" Hot Cold
	Tab space	>>>print "Hot\tCold" Hot Cold

✡ By invoking raw_input() method

Let's understand the working of raw input() function

Example

```
>>>raw_input()
Right to education
'Right to education'
```

As soon as the interpreter encounters raw_input method, it waits for the user to key in the input from a standard input device (keyboard) and press Enter key. The input is converted to a string and displayed on the screen.

Note: raw_input( ) method has been already discussed in previous chapter in detail.

✡ By invoking input() method

Example

```
>>>str=input("Enter the string")
Enter the string hello
NameError: name 'hello' is not defined
```

Python interpreter was not able associate appropriate data type with the entered data. So a NameError is shown. The error can be rectified by enclosing the given input i.e. hello in quotes as shown below

>>>str=input("Enter the String") Enter the String "hello" >>> print str Hello	>>>str=input("Enter the String") Enter the String'hello' >>> print str hello

Strings are immutable

Strings are immutable means that the contents of the string cannot be changed after it is created.

Let us understand the concept of immutability with help of an example.

Example

```
>>>str='honesty'

>>>str[2]='p'

TypeError: 'str' object does not support item assignment
```

Python does not allowthe programmer to change a character in a string. As shown in the above example, str has the value 'honesty'. An attempt to replace 'n' in the string by 'p' displays a TypeError.

Traversing a string

Traversing a string means accessing all the elements of the string one after the other by using the subscript. A string can be traversed using: for loop or while loop.

String traversal using for loop	String traversal using while loop

A='Welcome'
>>>for i in A:
        print i
W
	A='Welcome'
	>>>i=0
	>>>while i<len(A)
	        print A[i]

e l c o m e	i=i+1 W e l c o m e
A is assigned a string literal 'Welcome'. On execution of the for loop, the characters in the string are printed till the end of the string is not reached.	A is assigned a string literal 'Welcome' i is assigned value 0 The len() function calculates the length of the string. On entering the while loop, the interpreter checks the condition. If the condition is true, it enters the loop. The first character in the string is displayed. The value i is incremented by 1. The loop continues till value i is less than len-1. The loop finishes as soon as the value of I becomes equal to len-1, the loop

Strings Operations

Operator	Description	Example
+ (Concatenation)	The + operator joins the text on both sides of the operator	>>> 'Save'+'Earth' 'Save Earth' To give a white space between the two words, insert a space before the closing single quote of the first literal.
* (Repetition)	The * operator repeats the	>>>3*'Save Earth '

	string on the left hand side times the value on right hand side.	'Save Earth Save Earth Save Earth '
in (Membership)	The operator displays 1 if the string contains the given character or the sequence of characters.	>>>A='Save Earth' >>> 'S' in A True >>>'Save' in A True >>'SE' in A False
not in	The operator displays 1 if the string does not contain the given character or the sequence of characters. (working of this operator is the reverse of **in** operator discussed above)	>>>'SE' not in 'Save Earth' True >>>'Save ' not in 'Save Earth' False
range (*start*, *stop*[, *step*])	**This function is already discussed in previous chapter.**	
Slice[n:m]	The Slice[n : m] operator extracts sub parts from the strings.	>>>A='Save Earth' >>> print A[1:3] av The print statement prints the substring starting from subscript 1 and ending at subscript 3 but not including subscript 3

More on string Slicing

Consider the given figure

String A	S	A	V	E		E	A	R	T	H
Positive Index	0	1	2	3	4	5	6	7	8	9
Negative Index	-10	-9	-9	-7	-6	-5	-4	-3	-2	-1

Let's understand Slicing in strings with the help of few examples.

Example

```
>>>A='Save Earth'
>>> print A[1:3]
av
```

The print statement prints the substring starting from subscript 1 and ending at subscript 3 .

Example

```
>>>print A[3:]
'e Earth'
```

Omitting the second index, directs the python interpreter to extract the substring till the end of the string

Example

```
>>>print A[:3]
Sav
```

Omitting the first index, directs the python interpreter to extract the substring before the second index starting from the beginning.

Example

```
>>>print A[:]
'Save Earth'
```

Omitting both the indices, directs the python interpreter to extract the entire string starting from 0 till the last index

Example

```
>>>print A[-2:]
'th'
```

For negative indices the python interpreter counts from the right side (also shown above). So the last two letters are printed.

Example

```
>>>Print A[:-2]
'Save Ear'
```

Omitting the first index, directs the python interpreter to start extracting the substring form the beginning. Since the negative index indicates slicing from the end of the string. So the entire string except the last two letters is printed.

Note: Comparing strings using relational operators has already been discussed in the previous chapter

String methods & built in functions

Syntax	Description	Example
len()	Returns the length of the string.	>>>A='Save Earth' >>> print len(A) >>>10
capitalize()	Returns the exact copy of the string with the first letter in	>>>str='welcome'

	upper case	>>>print str.capitalize() Welcome
find(*sub*[, *start*[, *end*]])	The function is used to search the first occurrence of the substring in the given string. It returns the index at which the substring starts. It returns -1 if the substring does occur in the string.	>>>str='mammals' >>>str.find('ma') 0 **On omitting the start parameters, the function starts the search from the beginning.** >>>str.find('ma',2) 3 >>>str.find('ma',2,4) -1 **Displays -1 because the substring could not be found between the index 2 and 4-1** >>>str.find('ma',2,5) 3
isalnum()	Returns True if the string contains only letters and digit. It returns False ,If the string contains any special character like _ , @,#,* etc.	>>>str='Save Earth' >>>str.isalnum() False The function returns False as space is an alphanumeric character. >>>'Save1Earth'.isalnum() True

isalpha()	Returns True if the string contains only letters. Otherwise return False.	>>> 'Click123'.isalpha()
		False
		>>> 'python'.isalpha()
		True

isdigit()	Returns True if the string	>>>print str.isdigit()

	contains only numbers. Otherwise it returns False.	false
lower()	Returns the exact copy of the string with all the letters in lowercase.	>>>print str.lower() 'save earth'
islower()	Returns True if the string is in lowercase.	>>>print str.islower() True
isupper()	Returns True if the string is in uppercase.	>>>print str.isupper() False
upper()	Returns the exact copy of the string with all letters in uppercase.	>>>print str.upper() WELCOME

lstrip()	Returns the string after removing the space(s) on the left of the string.	>>> print str
		 Save Earth
		>>>str.lstrip()
		'Save Earth'
		>>>str='Teach India Movement'
		>>> print str.lstrip("T")
		each India Movement
		>>> print str.lstrip("Te")
		ach India Movement
		>>> print str.lstrip("Pt")
		Teach India Movement
		If a string is passed as argument to the lstrip() function, it removes those characters from the left of the string.

rstrip()	Returns the string after removing the space(s) on the	>>>str='Teach India Movement' >>> print str.rstrip()

	right of the string.	Teach India Movement

isspace()	Returns True if the string contains only white spaces and False even if it contains one character.	>>> str=' '
		
		>>> print str.isspace()
		
		True
		
		>>> str='p'
		
		>>> print str.isspace()
		
		False

istitle()	Returns True if the string is title cased. Otherwise returns False	>>> str='The Green Revolution'
		
		>>> str.istitle()
		
		True
		
		>>> str='The green revolution'
		
		>>> str.istitle()
		
		False

replace(old, new)	The function replaces all the occurrences of the old string with the new string	>>>str='hello'
		
		>>> print str.replace('l','%')
		
		He%%o
		
		>>> print str.replace('l','%%')
		
		he%%%%o

join ()	Returns a string in which the string elements have been joined by a separator.	>>> str1=('jan', 'feb' ,'mar')
		
		>>>str='&"
		
		>>> str.join(str1)
		
		'jan&feb&mar'

swapcase()	Returns the string with case changes	>>> str='UPPER'
		
		>>> print str.swapcase()
		
		upper
		
		>>> str='lower'
		
		>>> print str.swapcase()

		LOWER

partition(sep)	The function partitions the strings at the first occurrence of separator, and returns the strings partition in three parts i.e. before the separator, the separator itself, and the part after the separator. If the separator is not found, returns the string itself, followed by two empty strings	>>> str='The Green Revolution'
		>>> str.partition('Rev')
		('The Green ', 'Rev', 'olution')
		>>> str.partition('pe')
		('The Green Revolution', '', '')
		>>> str.partition('e')
		('Th', 'e', ' Green Revolution')

split([sep[, maxsplit]])	The function splits the string into substrings using the separator. The second argument is optional and its default value is zero. If an integer value N is given for the second argument, the string is split in N+1 strings.	>>>str='The$earth$is$what$we$all $have$in$common.'
		>>> str.split($,3)
		SyntaxError: invalid syntax
		>>> str.split('$',3)
		['The', 'earth', 'is', 'what$we$all$have$in$common.']
		>>> str.split('$')
		['The', 'earth', 'is', 'what', 'we', 'all', 'have', 'in', 'common.']
		>>> str.split('e')
		['Th', ' Gr', '', 'n R', 'volution']
		>>> str.split('e',2)
		['Th', ' Gr', 'en Revolution']

Note: In the table given above, len( ) is a built in function and so we don't need import the string module. For all other functions ***import string*** statement is required for their successful execution.

Let's discuss some interesting strings constants defined in string module:

string.ascii_uppercase

The command displays a string containing uppercase characters.

Example

```
>>> string.ascii_uppercase
'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
```

string.ascii_lowercase

The command displays a string containing all lowercase characters.

Example

```
>>> string.ascii_lowercase
'abcdefghijklmnopqrstuvwxyz'
```

string.ascii_letters

The command displays a string containing both uppercase and lowercase characters.

```
>>> string.ascii_letters
'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'
```

string.digits

The command displays a string containing digits.

```
>>> string.digits
'0123456789'
```

string.hexdigits

The command displays a string containing hexadecimal characters.

```
>>> string.hexdigits
'0123456789abcdefABCDEF'
```

string.octdigits

The command displays a string containing octal characters.

```
>>> string.octdigits
'01234567'
```

string.punctuations

The command displays a string containing all the punctuation characters.

```
>>> string.punctuations
'!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}-'
```

string.whitespace

The command displays a string containing all ASCII characters that are considered whitespace. This includes the characters space, tab, linefeed, return, formfeed, and vertical tab.

```
>>> string.whitespace
'\t\n\x0b\x0c\r '
```

string.printable

The command displays a string containing all characters which are considered printable like letters, digits, punctuations and whitespaces.

```
>>> string.printable
'0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!
"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}- \t\n\r\x0b\x0c'
```

Note: Import string module to get the desired results with the commands mentioned above.

Programs using string functions and operators

1. Program to check whether the string is a palindrome or not.

```
defpalin():
    str=input("Enter the String")
    l=len(str)
    p=l-1
    index=0
    while (index<p):
        if(str[index]==str[p]):
            index=index+1
            p=p-1
    else:
        print "String is not a palidrome"
        break
    else:
    print "String is a Palidrome"
```

2. Program to count no of 'p' in the string pineapple.

```
def lettercount():
    word = 'pineapple'
    count = 0
    for letter in word:
        if letter == 'p':
        count = count + 1
    print(count)
```

Regular expressions and Pattern matching

A regular expression is a sequence of letters and some special characters (also called meta characters). These special characters have symbolic meaning. The sequence formed by using meta characters and letters can be used to represent a group of patterns.

Let's start by understanding some meta characters.

For example

str= "Ram$"

The pattern "Ram$" is known as a regular expression. The expression has the meta character '$'. Meta character '$' is used to match the given regular expression at the end of the string. So the regular expression would match the string 'SitaRam' or 'HeyRam' but will not match the string 'Raman'.

Consider the following codes:

```
def find():
    import re
    string1='SitaRam'
    if re.search('Ram$',string1):
        print "String Found"
    else :
        print" No Match"
Output:
    String Found
```

```
def find():
    import re
    string1='SitaRam'
    if re.search('Sita$',string1):
        print "String Found"
    else :
        print" No Match"
Output
    No Match
```

As shown in the above examples, Regular expressions can be used in python for matching a particular pattern by importing the re module.

Note: re module includes functions for working on regular expression.

Now let's learn how the meta characters are used to form regular expressions.

S.No	Meta character	Usage	Example
1	[ ]	Used to match a set of characters.	[ram] The regular expression would match any of the characters r, a, or m. [a-z] The regular expression would match only lowercase characters.
2	^	Used to complementing a set of characters	[^ram] The regular expression would match any other characters than r, a or m.
3	$	Used to match the end of string only	Ram$ The regular expression would match Ram in SitaRam but will not match Ram in Raman
4	*	Used to specify that the previous character can be matched zero or more times.	wate*r The regular expression would match strings like watr, wateer, wateeer and so on.
5	+	Used to specify that the previous character can be matched one or more times.	wate+r The regular expression would match strings like water, wateer, wateeer and so on.

6	?	Used to specify that the previous character can be matched either once or zero times	wate?r The regular expression would only match strings like watr or water
7	{ }	The curly brackets accept two integer value s. The first value specifies the minimum no of occurrences and second value specifies the maximum of occurrences	wate{1,4}r The regular expression would match only strings water, wateer, wateeer or wateeeer

Let's learn about few functions from re module

re.compile()

The re.compile( ) function will compile the pattern into pattern objects. After the compilation the pattern objects will be able to access methods for various operations like searching and subsitutions

Example

```
import re

p=re.compile('hell*o')
```

re.match()

The match function is used to determine if the regular expression (RE) matches at the beginning of the string.

re.group()

The group function is used to return the string matched the RE

Example

```
>>>P=re.compile('hell*o')

>>>m=re.match('hell*o', ' hellooooo world')

>>>m.group()

'hello'
```

re.start()

The start function returns the starting position of the match.

re.end()

The end function returns the end position of the match.

re.span()

The span function returns the tuple containing the (start, end) positions of the match

Example

```
>>> import re
>>> P=re.compile('hell*o')
>>> m=re.match('hell*o', 'hellooooo world')
>>> m.start()
0
>>> m.end()
5
>>> m.span()
(0, 5)
```

re.search()

The search function traverses through the string and determines the position where the RE matches the string

Example

```
>>> m=re.search('hell*o', 'favorite words hellooooo world')
>>> m.start()
15
>>> m.end()
```

```
20
>>> m.group()
'hello'
>>> m.span()
(15, 20)
```

Re.findall()

The function determines all substrings where the RE matches, and returns them as a list.

Example

```
>>> m=re.findall('hell*o', 'hello my favorite words hellooooo world')
>>> m
['hello', 'hello']
```

re.finditer()

The function determines all substrings where the RE matches, and returns them as an iterator.

Example

```
>>> m=re.finditer('hell*o', 'hello my favorite words hellooooo world')
>>> m
<callable-iterator object at 0x0000000002E4ACF8>
>>> for match in m:
print match.span()
(0, 5)
(24, 29)
```

As shown in the above example, m is a iterator. So m is used in the for loop.

Script 1: Write a script to determine if the given substring is present in the string.

```
def search_string():
        import re
        substring='water'
        search1=re.search(substring,'Water water everywhere but not a drop to drink')
        if search1:
                position=search1.start()
                print "matched", substring, "at position", position
        else:
                print "No match found"
```

Script 2: Write a script to determine if the given substring (defined using meta characters) is present in the given string

```
def metasearch():
    import re
    p=re.compile('sing+')
    search1=re.search(p,'Some singers sing well')
    if search1:
        match=search1.group()
        index=search1.start()
        lindex=search1.end()
        print "matched", match, "at index", index ,"ending at" ,lindex
    else:
        print "No match found"
```

EXERCISE

1. Input a string "Green Revolution". Write a script to print the string in reverse.

2. Input the string "Success". Write a script of check if the string is a palindrome or not

3. Input the string "Successor". Write a script to split the string at every occurrence of the letter s.

4. Input the string "Successor". Write a script to partition the string at the occurrence of the letter s. Also Explain the difference between the function split( ) and partition().

5. Write a program to print the pyramid.

```
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
```

6. What will be the output of the following statement? Also justify for answer.

```
>>> print 'I like Gita\'s pink colour dress'.
```

7. Give the output of the following statements

```
>>> str='Honesty is the best policy'
>>> str.replace('o','*')
```

8. Give the output of the following statements

```
>>> str='Hello World'
>>>str.istiltle()
```

9. Give the output of the following statements.

```
>>> str="Group Discussion"
>>> print str.lstrip("Gro")
```

10. Write a program to print alternate characters in a string. Input a string of your own choice.

11. Input a string 'Python'. Write a program to print all the letters except the letter'y'.

12. Consider the string str="Global Warming"

    Write statements in python to implement the following

    a) To display the last four characters.

    b) To display the substring starting from index 4 and ending at index 8.

    c) To check whether string has alphanumeric characters or not.

    d) To trim the last four characters from the string.

    e) To trim the first four characters from the string.

    f) To display the starting index for the substring 'Wa'.

    g) To change the case of the given string.

    h) To check if the string is in title case.

    i) To replace all the occurrences of letter 'a' in the string with '*'

13. Study the given script

```
def metasearch():
    import re
    p=re.compile('sing+')
    search1=re.search(p,'Some singers sing well')
    if search1:
        match=search1.group()
        index=search1.start()
        lindex=search1.end()
        print "matched", match, "at index", index ,"ending at", lindex
else:
        print "No match found"
```

What will be the output of the above script if search() from the re module is replaced by match () of the re module. Justify your answer

14. What will be the output of the script mentioned below? Justify your answer.

```
def find():
        import re
        p=re.compile('sing+')
        search1=p.findall('Some singer sing well')
        print search1
```

15. Rectify the error (if any) in the given statements.

```
>>> str="Hello World"
>>> str[5]='p'
```

Chapter 2

Lists

After studying this lesson, students will be able to:

- *Understand the concept of mutable sequence types in Python.*
- *Appreciate the use of list to conveniently store a large amount of data in memory.*
- *Create, access & manipulate list objects*
- *Use various functions & methods to work with list*
- *Appreciate the use of index for accessing an element from a sequence.*

Introduction

Like a String, list also is sequence data type. It is an ordered set of values enclosed in square brackets []. Values in the list can be modified, i.e. it is mutable. As it is set of values, we can use index in square brackets [] to identify a value belonging to it. The values that make up a list are called its elements, and they can be of any type.

We can also say that list data type is a container that holds a number of elements in a given order. For accessing an element of the list, indexing is used.

Its syntax is:

Variable name [index] (variable name is name of the list).

It will provide the value at 'index+1' in the list. Index here, has to be an integer value-which can be positive or negative. Positive value of index means counting forward from beginning of the list and negative value means counting backward from end of the list. Remember the result of indexing a list is the value of type accessed from the list.

Index value	**Element of the list**
0, -size	1st
1, -size +1	2nd

2, -size +2	3rd
. . .	
size -2, -2	2nd last
size -1, -1	last

Please note that in the above example size is the total number of elements in the list.

Let's look at some example of simple list:

```
i)    >>>L1 = [1, 2, 3, 4]                      # list of 4 integer elements.
ii)   >>>L2 = ["Delhi", "Chennai", "Mumbai"] #list of 3 string elements.
iii)  >>>L3 = [ ]                               # empty list i.e. list with no element
iv)   >>>L4 = ["abc", 10, 20]                   # list with different types of elements
v)    >>>L5 = [1, 2, [6, 7, 8], 3]              # A list containing another list known as
      nested list
```

You will study about Nested lists in later parts of the chapter.

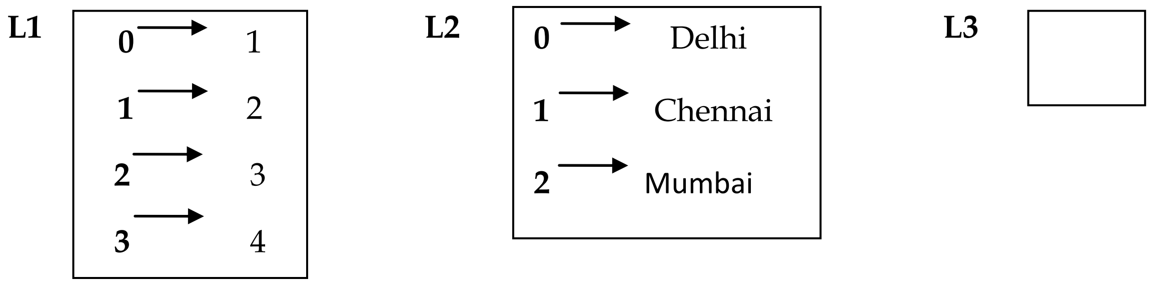
To change the value of element of list, we access the element & assign the new value.

Example

```
>>>print L1          # let's get the values of list before change
>>> L1 [2] = 5
>>> print L1         # modified list
 [1, 2, 5, 4]
```

Here, 3rd element of the list (accessed using index value **2**) is given a new value, so instead of **3** it will be **5.**

State diagram for the list looks like:



Note: List index works the same way as String index, which is:

- ✡ An integer value/expression can be used as index.
- ✡ An Index Error appears, if you try and access element that does not exist in the list.
- ✡ An index can have a negative value, in that case counting happens from the end of the list.

Creating a list

List can be created in many ways:

i) By enclosing elements in [ ], as we have done in above examples.

ii) Using other Lists

Example

```
L5=L1 [:]
```

Here L5 is created as a copy of L1.

```
>>>print L5
L6 = L1 [0:2]
>>>print L6
```

will create L6 having first two elements of L1.

iii) List comprehension

Example

```
>>>n = 5
```

```
>>>l = range(n)

>>>print l

[0, 1, 2, 3, 4]
```

Example

```
>>> S= [x**2 for x in range (10)]

>>> print S

[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

In mathematical terms, S can be defined as S = {x^2 for: x in (0.....9)}. So, we can say that list comprehension is short-hand for creating list.

Example

```
>>> A = [3, 4, 5]

>>> B = [value *3 for value in A]
```

Here B will be created with the help of A and its each element will be thrice of element of A.

```
>>> print B

[9, 12, 15]
```

Comprehensions are functionally equivalent to wrting as:

```
>>>B = [ ]

>>>for i in A

B. append (i*3)
```

Similarly, other comprehensions can be expended.

Example

```
>>> print B

[9, 12, 15]
```

Let's create a list of even numbers belonging to '**S**' list:

```
>>>C = [i for i in S if i % 2 = = 0]
```

```
>>>print C
[0, 4, 16, 36, 64]
```

iv) Using built-in object

L = list ( ) will create an empty list

Example

```
>>>l = list ( )
>>>print l
[ ] # empty list
```

Or

L = list (sequence)

Example

```
>>>L = list [(1, 2, 3, 4)]
>>>print L
[1, 2, 3, 4]
```

A single new list is created every time, you execute [ ]. We have created many different lists each using [ ]. But if a list is assigned to another variable, a new list is not created.

i) A=B=[ ]

Creates one list mapped to both A & B

Example

```
>>>A = B = [10, 20, 30]
>>> print A, B
[10, 20, 30] [10, 20, 30]
```

ii) A = [ ]

B = A

Will also create one list mapped to both

Example

```
>>> A = [1, 2, 3]

>>> B = A

>>> print A, B

[1, 2, 3] [1, 2, 3]
```

Accessing an element of list

For accessing an element, we use index and we have already seen example doing so. To access an element of list containing another list, we use pair of index. Lets access elements of L5 list. Also a sub-list of list can be accessed using list slice.

List Slices

Slice operator works on list also. We know that a slice of a list is its sub-list. For creating a list slice, we use

[n:m] operator.

```
>>>print L5 [0]

1

>>>print L5 [2]

[6, 7, 8]
```

as the 3rd element of this list is a list. To access a value from this sub-list, we will use

```
>>>print L5 [2] [0]

6

>>>print L5 [2] [2]

8
```

This will return the part of the list from nth element to mth element, including the first element but excluding the last element. So the resultant list will have m-n elements in it.

```
>>> L1 [1:2]
```

will give

```
[2]
```

Slices are treated as boundaries, and the result will contain all the elements between boundaries.

Its Syntax is:

seq = L [start: stop: step]

Where start, stop & step- all three are optional. If you omit first index, slice starts from '0' and omitting of stop will take it to end. Default value of step is 1.

Example

For list L2 containing ["Delhi", "Chennai", "Mumbai"]

```
>>>L2 [0:2]

["Delhi", "Chennai"]
```

Example

```
>>>list = [10, 20, 30, 40, 50, 60]

>>> list [::2]    # produce a list with every alternate element

[10, 30, 50]

>>>list [4:]      # will produce a list containing all the elements from 5th position
                  till end

[50, 60]
```

Example

```
>>>list [:3]

[10, 20, 30]

>>>list [:]

[10, 20, 30, 40, 50, 60]
```

Example

```
>>> list [-1]    # '-1' refers to last elements of list

60
```

will produce a list with every other element

Note: Since lists are mutable, it is often recommended to make a copy of it before performing operation that change a list.

Traversing a List

Let us visit each element (traverse the list) of the list to display them on screen. This can be done in many ways:

(i)
```
i = 0
while i < 4:
print L1 [i],
i + = 1
```
will produce following output
```
1 2 5 4
```

(ii)
```
for i in L1:
print i,
```
will also produce the same output

(iii)
```
i=0
while i < len [L1]:
print L1 [i],
i + = 1
```
OR
```
i= 0
L = len (L1)
while i < L :
print L1 [i],
i + = 1
```
will also produce the same output.

Here len( ) function is used to get the length of list L1. As length of L1 is 4, i will take value from 0 to 3.

(iv) for i in range (len (L1)):

print L1 [i],

Using 2nd way for transversal will only allow us to print the list, but other ways can also be used to write or update the element of the list.

In 4th way, range ( ) function is used to generate, indices from 0 to len -1; with each iteration i gets the index of next element and values of list are printed.

Note: for loop in empty list is never executed:

Example

```
for i in [ ]:
        print i
        Accessing list with negative index
        i = 1
        while i < len (L1):
        print L1 [-i],
        i += 1
```

In this case, Python will add the length of the list to index and then return the index value and accesses the desired element. In this loop execution for a positive value of 'i' L1 [-i] will result into L1 [len (L1)-i] for i=1, L1 [4-1] will be printed. So resultant of the loop will be 4 5 2.

Appending in the list

Appending a list is adding more element(s) at the end of the list. To add new elements at the end of the list, Python provides a method append ( ).

Its Syntax is:

List. append (item)

L1. append (70)

This will add 70 to the list at the end, so now 70 will be the 5th element of the list, as it already have 4 elements.

```
>>> print L1
```

will produce following on screen

```
[1, 2, 5, 4, 70]
```

Example

```
>>>L4.append (30)          # will add 30 at the end of the list
>>>print L4
['abc', 10, 20, 30]
```

Using append ( ), only one element at a time can be added. For adding more than one element, extend ( ) method can be used, this can also be used to add elements of another list to the existing one.

Example

```
>>>A = [100, 90, 80, 50]
>>> L1. extend (A)
>>> print L1
```

will add all the elements of list 'A' at the end of the list 'L1'.

```
[1, 2, 5, 4, 70, 100, 90, 80, 50]
>>>print A
[100, 90, 80, 50]
```

Example

```
>>>B=[2009, 2011, 'abc']
>>>C=['xyz', 'pqr', 'mn']
>>>B.extend (c)
>>>print B
```

```
[2009, 2011, 'abc', 'xyz', 'pqr', 'mn']
```

Remember: 'A' remains unchanged

Updating array elements

Updating an element of list is, accomplished by accessing the element & modifying its value in place. It is possible to modify a single element or a part of list. For first type, we use index to access single element and for second type, list slice is used. We have seen examples of updations of an element of list. Lets update a slice.

Example

```
>>> L1 [1:2] = [10, 20]

>>> print L1

will produce

[1, 10, 20, 4, 70, 100, 90, 80, 50]
```

Example

```
>>>A=[10, 20, 30, 40]

>>>A [1:4] = [100]

>>>print A

will produce

[10, 100]
```

As lists are sequences, they support many operations of strings. For example, operator **+** & * results in concatenation & repetition of lists. Use of these operators generate a new list.

Example

```
>>> a= L1+L2
```

will produce a 3rd list **a** containing elements from L1 & then L2. **a** will contain

```
[1, 10, 20, 4, 70, 100, 90, 80, 50, "Delhi", "Chennai", "Mumbai"]
```

Example

```
>>> [1, 2, 3] + [4, 5, 6]
[1, 2, 3, 4, 5, 6]
```

Example

```
>>> b = L1*2
>>> print b
[[1, 10, 20, 4, 70, 100, 90, 80, 50, 1, 10, 20, 4, 70, 100, 90, 80, 50]
```

Example

```
>>> ['Hi!']* 3
['Hi!', 'Hi!', 'Hi!']
```

It is important to know that **'+'** operator in lists expects the same type of sequence on both the sides otherwise you get a type error.

If you want to concatenate a list and string, either you have to convert the list to string or string to list**.**

Example

```
>>> str([11, 12]) + "34"    or  >>>"[11,12]" + "34"
'[11, 12] 34'
>>> [11, 12] + list ("34")    or >>>[11, 12] + ["3", "4"]
  [11, 12, '3', '4']
```

Deleting Elements

It is possible to delete/remove element(s) from the list. There are many ways of doing so:

(i) If index is known, we can use pop ( ) or del

(ii) If the element is known, not the index, remove ( ) can be used.

(iii) To remove more than one element, del ( ) with list slice can be used.

(iv) Using assignment operator

Let us study all the above methods in details:

Pop ( )

It removes the element from the specified index, and also return the element which was removed.

Its syntax is:

List.pop ([index])

Example

```
>>> L1 = [1, 2, 5, 4, 70, 10, 90, 80, 50]

>>> a= L1.pop (1)                  # here the element deleted will be returned to 'a'

>>> print L1

[1, 5, 4, 70, 10, 90, 80, 50]

>>> print a

2
```

If no index value is provided in pop ( ), then last element is deleted.

```
>>>L1.pop ( )

50
```

del removes the specified element from the list, but does not return the deleted value.

```
>>> del L1 [4]

>>> print L1

[1, 5, 4, 70, 90, 80]
```

remove ( )

In case, we know the element to be deleted not the index, of the element, then remove ( ) can be used.

```
>>> L1. remove (90)
```

will remove the value 90 from the list

```
>>> print L1
[1, 5, 4, 70, 80]
```

del () with slicing

Consider the following example:

Examples

```
>>> del L1 [2:4]
>>>print L1
[1, 5, 80]
```

will remove 2nd and 3rd element from the list. As we know that slice selects all the elements up to 2nd index but not the 2nd index element. So **4th element** will remain in the list.

```
>>> L5 [1:2] = [ ]
```

Will delete the slice

```
>>>print L5
[1, [6, 7, 8], 3]
```

Note:

(i) All the methods, modify the list, after deletions.

(ii) If an out of range index is provided with del ( ) and pop ( ), the code will result in to run-time error.

(iii) del can be used with negative index value also.

Other functions & methods

insert ( )

This method allows us to insert an element, at the given position specified by its index, and the remaining elements are shifted to accommodate the new element. Insert (

() requires two arguments-**index value** and **item value**.

Its syntax is

list. insert (index, item)

Index specifies the position (starting from 0) where the element is to be inserted. Item is the element to be inserted in the list. Length of list changes after insert operation.

Example

```
>>> L1.insert (3,100)
>>>print L1
will produce
[1, 5, 80, 100]
```

Note: If the index specified is greater then len (list) the object is inserted in the last and if index is less than zero, the object is inserted at the beginning.

```
>>> print len(L1)
4
>>> L1.insert (6, 29)
>>> L1.insert (-2, 46)
>>>print L1
will produce
[46, 1, 5, 80, 100, 29]
```

reverse ( )

This method can be used to reverse the elements of the list in place

Its syntax is:

list.reverse ( )

Method does not return anything as the reversed list is stored in the same variable.

Example

```
>>> L1.reverse ( )
>>> print L1
```

will produce

```
[29, 100, 80, 5, 1, 46]
```

Following will also result into reversed list.

```
>>>L1 [: : -1]
```

As this slices the whole sequence with the step of -1 i.e. in reverse order.

sort ( )

For arranging elements in an order Python provides a method **sort ( )** and a function **sorted ( )**. sort ( ) modifies the list in place and sorted ( ) returns a new sorted list.

Its Syntax are:

sort ([cmp [, key [, reverse]]])

sorted (list [, cmp [, key [, reverse]]])

Parameters mentioned in [ ] are optional in both the cases. These parameters allow us to customize the function/method.

cmp, argument allow us to override the default way of comparing elements of list. By default, sort determines the order of elements by comparing the elements in the list against each other. To overside this, we can use a user defined function which should take two values and return -1 for **'less than'**, 0 for **'equal to'** and 1 for 'greater than'.

'**Key'** argument is preferred over **'cmp'** as it produces list faster.

Example

The parameter **'key'** is for specifying a function that transforms each element of list before comparison. We can use predefined functions or a user defined function here. If its user defined then, the function should take a single argument and return a key which can be used for sorting purpose.

Reverse parameter can have a boolean value which is used to specify the order of arranging the elements of list. Value **'True'** for reverse will arrange the elements of list in descending order and value **'False'** for reverse will arrange the elements in ascending order. Default value of this parameter is False.

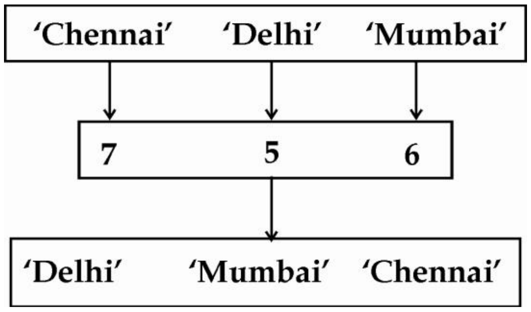
sorted ( ) function also behaves in similar manner except for it produce a new sorted list, so original is not changed. This function can also be used to sort any iterable collection. As sort ( ) method does not create a new list so it can be little faster.

Example

```
>>> L1.sort ( )

>>> print L1

will produce

[1, 5, 29, 46, 80, 100]

>>> L2.sort ( )

>>> print L2

will produce

['Chennai', 'Delhi', 'Mumbai']

>>> L2.sort (key=len)

will produce

['Delhi', 'Mumbai', 'Chennai']
```

Here we have specified len ( ) built in function, as key for sorting. So the list will get sorted by the length of the strings, i.e., from shorted to longest.

sort will call len ( ) function for each element of list and then these lengths will be used for arranging elements.



```
>>> L4.sort ( )

>>> print L4

will produce
```

```
[10, 20, 30, 'abc']
>>>L4.sort (reverse = True)
['abc', 30, 20, 10]
>>> def compare (str):
...         return len (str)
>>> L2.sort (key=compare)
>>> L2
['Delhi', 'Mumbai', 'Chennai']
```

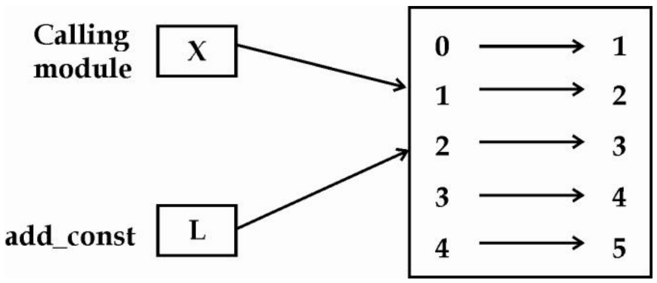
List as arguments

When a list is passed to the function, the function gets a reference to the list. So if the function makes any changes in the list, they will be reflected back in the list.

Example

```
def add_Const (L):
for i in range (len (l)):
L [i] += 10
>>> X = [1, 2, 3, 4, 5]
>>> add_Const (X)
>>> print X
[11, 12, 13, 14, 15]
```

Here parameter 'L' and argument 'X' are alias for same object. Its state diagram will look like

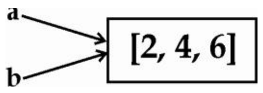


So any changes made in L will be reflected to X as lists as mutable.

Note: Here, it becomes important to distinguish between the operations which modifies a list and operation which creates a new list. Operations which create a new list will not affect the original (argument) list.

Let's look at some examples to see when we have different lists and when an alias is created.

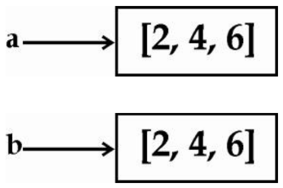
```
>>> a = [2, 4, 6]

>>> b = a
```



will map b to a. To check whether two variables refer to same object (i.e. having same value), we can use 'is' operator. So in our example:

```
>>> a is b

will return 'True'

>>> a = [2, 4, 6]

>>> b = [2, 4, 6]

>>> a is b
```

False



In first example, Python created one list, reference by a & b. So there are two references to the same object b. We can say that object [2, 4, 6] is aliased as it has more than one name, and since lists are mutable. So changes made using 'a' will affect 'b'.

```
>>> a [1] = 10

>>> print b

will print

[2, 10, 6]
```

Matrix implementation using list

We can implement matrix operation using list. Matrix operation can be implemented using nested list. List inside another list is called nested list.

Its syntax is:

a=[[random.random() for row in range(number of row)]for col in range(number of column)]

Here random function is used. So we need to import random file.

Example

Write a program to input any matrix with mXn, and print the number on the output screen in matrix format.

Matrix creation

Program 1

```
m=input ("Enter total number of rows")
n=input ("Enter total number of columns")
l=range (m*n)
k=0
print "Input all matrix elements one after other"
for i in range(m):
for j in range(n):
l[k]=input("Enter new element")
k=k+1
print "output is"
k=0
for i in range(m):
for j in range(n):
print l[k],'\t',
k=k+1
print
```

Output

```
>>>
Enter total number of rows3
Enter total number of columns3
Input all matrix elements one after other
Enter new element10
Enter new element20
Enter new element30
Enter new element40
Enter new element50
Enter new element60
Enter new element70
Enter new element80
Enter new element90
output is
10      20      30
40      50      60
70      80      90
>>>
```

Program 2

```
import random
m=input("Enter total number of rows in the first matrix")
n=input("Enter total number of columns in the first  matrix")
a=[[random.random()for row in range(m)]for col in range(n)]
print "Enter all elements one after other"
for i in range(m):
```

```
for j in range(n):
a[i][j]=input()
print "output is"
for i in range(m):
for j in range(n):
print a[i][j],'\t',
print
```

Output

```
>>>
Enter total number of rows in the first matrix3
Enter total number of columns in the first matrix3
Enter all elements one after other
1
2
3
4
5
6
7
8
9
output is
1       2       3
4       5       6
7       8       9
>>>
```

Matrix Addition

Write a program to input any two matrices and print sum of matrices.

```
import random
m1=input ("Enter total number of rows in the first matrix")
n1=input ("Enter total number of columns in the first matrix")
a=[[random.random()for row in range(m1)]for col in range(n1)]
for i in range(m1):
for j in range(n1):
a[i][j]=input()
m2=input("Enter total number of rows in the second matrix")
n2=input("Enter total number of columns in the second matrix")
b=[[random.random()for row in range(m1)]for col in range(n1)]
for i in range(2):
for j in range(2):
b[i][j]=input()
c=[[random.random()for row in range(m1)]for col in range(n1)]
if ((m1==m2) and (n1==n2)):
print "output is"
for i in range(m1):
for j in range(n1):
c[i][j]=a[i][j]+b[i][j]
print c[i][j],'\t',
print
else
print “Matrix addition not possible”
```

Output

```
>>>
Enter total number of rows in the first matrix2
Enter total number of columns in the first matrix2
1
1
1
1
Enter total number of rows in the second matrix2
Enter total number of columns in the second matrix2
2
2
2
2
output is
3        3
3       3
```

Example

Write a program to input any two matrices and print product of matrices.

```
import random
m1=input ("Enter total number of rows in the first matrix")
n1=input ("Enter total number of columns in the first  matrix")
a=[[random.random()for row in range(m1)]for col in range(n1)]
for i in range(m1):
for j in range(n1):
a[i][j]=input()
```

```
m2=input ("Enter total number of rows in the second matrix")
n2=input ("Enter total number of columns in the second matrix")
b=[[random.random()for row in range(m1)]for col in range(n1)]
for i in range(m2):
for j in range(n2):
b[i][j]=input()
c=[[random.random()for row in range(m1)]for col in range(n2)]
if (n1==m2):
for i in range(m1):
for j in range(n2):
c[i][j]=0
for k in range(n1):
c[i][j]+=a[i][k]*b[k][j]
print c[i][j],'\t',
print
else:
print "Multiplication not possible"
```

Output

```
>>>
Enter total number of rows in the first matrix2
Enter total number of columns in the first  matrix2
1
1
1
1
Enter total number of rows in the second matrix2
Enter total number of columns in the second matrix2
2
2
2
2
4        4
4        4
>>>
```

Example

Write a program to input any matrix and print both diagonal values of the matrix.

```
import random
m=input ("Enter total number of rows in the first matrix")
n=input ("Enter total number of columns in the first  matrix")
a=[[random.random()for row in range(m)] for col in range(n)]
if (m==n):
for i in range(m):
for j in range(n):
a[i][j]=input()
print "First diagonal"
for i in range(m):
print a[i][i],'\t',
print
k=m-1
print "Second diagonal"
for j in range(m):
print a[j][k],'\t',
k-=1
else:
print "Diagonal values are not possible"
```

Output

```
>>>
Enter total number of rows in the first matrix3
Enter total number of columns in the first matrix3
1
2
3
4
5
6
7
8
9
First diagonal
1    5    9
Second diagonal
3    5    7
>>>
```

Functions with list

We can pass list value to function. Whatever modification we are doing with in function will affect list.

Example

Write a program to pass any list and to arrange all numbers in descending order.

```
def arrange (l,n):
for i in range(n-1):
for j in range(n-i-1):
if l[j]>l[j+1]:
temp=l[j]
l[j]=l[j+1]
l[j+1]=temp
```

Output

```
>>>
>>> l=[7,5,8,2,9,10,3]
>>> arrange (l)
>>> print l
[10, 9, 8, 7, 5, 3, 2]
>>>
```

Function pass nested list also:

Example

Write a program to input nXm matrix and find sum of all numbers using function.

Function:

```
def summat(a,m,n):
s=0
for i in range(m):
for j in range(n):
s+=a[i][j]
return s
```

Note: This function is stored in mataddition.py

Function call

```
import random
import mataddition
m=input("Enter total number of rows in the first matrix")
n=input("Enter total number of columns in the first  matrix")
a=[[random.random()for row in range(m)]for col in range(n)]
for i in range(m):
for j in range(n):
a[i][j]=input()
s=mataddition.summat(a,m,n)
print s
```

Output

```
>>>
Enter total number of rows in the first matrix2
Enter total number of columns in the first matrix2
1
2
3
4
10
>>>
```

Example

```
# Accessing the elements of a sublist
a = [[1, 2, 3], [4, 5], [6, 7, 8]]
count = -1
for list in a:
```

```
count + = 1

print "elements of the list at index", count, "are:"

for item in list:

print item,

print
```

will produce the result

```
elements of the list at index 0 are

1 2 3

elements of the list at index 1 are

4 5

elements of the list at index 2 are

6 7 8
```

EXERCISE

1. Define list

2. What is the output of the following code:

   a) print type ([1,2])

      (i) <type 'complex'>

      (ii) <type 'int'>

      (iii) <type 'list'>

   b) a= [1, 2, 3, None, ( ), [ ]}

      print len(a)

      (i) Syntax error (ii) 4

      (iii) 5 (iv) 6

      (v) 7

3. Write the output from the following code:

```
A=[2,4,6,8,10]
L=len(L)
S=0
for I in range(1,L,2):
S+=A[I]
print "Sum=",S
```

4. Find the errors from the following program

```
n=input (Enter total number of elements)
l=range(n)
print l
for i in (n);
l[i]=input("enter element")
```

```
print "All elements in the list on the output screen"
for i on range(n):
print l[i]
```

5. Write a function group of list (list, size) that takes a list and splits into smaller list of given size.

6. Write a function to find all duplicates in the list.

7. For each of the expression below, specify its type and value. If it generates error, write error.

   Assume that expressions are evaluated in order.

   x= [1, 2, [3, 'abc', 4], 'Hi']

   (i) x[0]

   (ii) x[2]

   (iii) x[-1]

   (iv) x[0:1]

   (v) 2 in x

   (vi) x[0]=8

8. For each of the expression below, specify its type and value. If it generates error, write error:

   List A= [1, 4, 3, 0]

   List B= ['x', 'z', 't', 'q']

   (i) List A.sort ( )

   (ii) List A

   (iii) List A.insert (0, 100)

   (iv) List A.remove (3)

   (v) List A.append (7)

   (vi) List A+List B

(vii) List B.pop ( )

(viii) List A.extend ([4, 1, 6, 3])

LAB EXERCISE

1. We can use list to represent polynomial.

   For Example

   p (x) = -13.39 + 17.5 x + 3 x^2 + x^4

   can be stored as

   [-13.39, 17.5, 3, 1.0]

   Here 'index' is used to represent power of 'x' and value at the index used to represent the coefficient of the term.

   Write a function to evaluate the polynomial for a given 'x'.

2. Write a function that takes a list of numbers and returns the cumulative sum; that is, a new list where the its element is the sum of the first i+1 elements from the original list. For example, the cumulative sum of [1, 2, 3] is [1, 3, 6].

3. Write a function called *chop* that takes a list and modifies it, removing the first and last elements, and returns *None*. Then write a function called *middle* that takes a list and returns a new list that contains all but the first and last elements.

4. Write a function called *is_sorted* that takes a list as a parameter and returns *True* if the list is sorted in ascending order and *False* otherwise. You can assume (as a precondition) that the elements of the list can be compared with the relational operators <, >, etc.

   For example, *is_sorted ([1, 2, 2])* should return *True* and *is_sorted (['b', 'a'])* should return *False.*

5. Write a function called *remove_duplicates* that takes a list and returns a new list with only the unique elements from the original. Hint: they don't have to be in the same order.

6. Write a function that takes in two sorted lists and merges them. The lists may not be of same length and one or both may be empty. Don't use any Python built-in methods or functions.

7. Create a list that contains the names of 5 students of your class. (Do not ask for input to do so)

   (i) Print the list

   (ii) Ask the user to input one name and append it to the list

   (iii) Print the list

   (iv) Ask user to input a number. Print the name that has the number as index (Generate error message if the number provided is more than last index value).

   (v) Add "Kamal" and "Sanjana" at the beginning of the list by using '+'.

   (vi) Print the list

   (vii) Ask the user to type a name. Check whether that name is in the list. If exist, delete the name, otherwise append it at the end of the list.

   (viii) Create a copy of the list in reverse order

   (ix) Print the original list and the reversed list.

   (x) Remove the last element of the list.

8. Use the list of student names from the previous exercise. Create a for loop that asks the user for every name whether they would like to keep the name or delete it. Delete the names which the user no longer wants. Hint: you cannot go through a list using a for loop and delete elements from the same list simultaneously because in that way the for loop will not reach all elements. You can either use a second copy of the list for the loop condition or you can use a second empty list to which you append the elements that the user does not want to delete.

9. Write a function to find product of the element of a list. What happens when the function is called with list of strings?

10. Write a program to input NXM matrix and find sum of all even numbers in the matrix.

11. Write a program to print upper triangle matrix.

12. Write a program to print lower triangle matrix.

13. Write a program to find sum of rows and columns of the matrix.

Chapter 3

Dictionaries

After studying this lesson, the students will be able to

- *understand the need of dictionaries;*
- *solve problems by using dictionaries;*
- *get clear idea about dictionaries functions; and*
- *understand the difference between list and dictionary.*

What is dictionary?

A dictionary is like a list, but more in general. In a list, index value is an integer, while in a dictionary index value can be any other data type and are called keys. The key will be used as a string as it is easy to recall. A dictionary is an extremely useful data storage construct for storing and retrieving all key value pairs, where each element is accessed (or indexed) by a unique key. However, dictionary keys are not in sequences and hence maintain no left-to right order.

Key-value pair

We can refer to a dictionary as a mapping between a set of indices (which are called keys) and a set of values. Each key maps a value. The association of a key and a value is called a **key-value pair**.

Syntax:

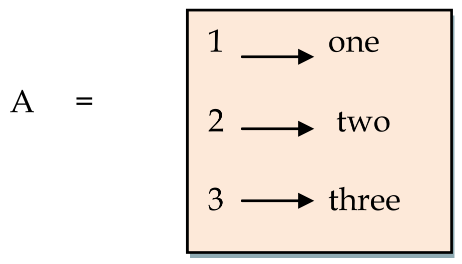
```
my_dict = {'key1': 'value1','key2': 'value2','key3': 'value3'…'keyn': 'valuen'}
```

Note: Dictionary is created by using curly brackets(ie. {}).

Example

```
>>> A={1:"one",2:"two",3:"three"}
>>> print A
{1: 'one', 2: 'two', 3: 'three'}
```

In the above example, we have created a list that maps from numbers to English words, so the keys values are in numbers and values are in strings.



Map between keys and values

Example

```
>>>computer={'input':'keybord','output':'mouse','language':'python','os':'windows-8',}

>>> print computer

{'input': 'keyboard', 'os': 'windows-8', 'language': 'python', 'output': 'mouse'}

>>>
```

In the above example, we have created a list that maps from computer related things with example, so here the keys and values are in strings. The order of the key-value pairs is not in same order (ie. input and output orders are not same). We can get different order of items in different computers. Thus, the order of items in a dictionary is unpredictable.

Example

```
>>>
D={'sun':'Sunday','mon':'Monday','tue':'Tuesday','wed':'Wednesday','thu':'Thursday','fri':'Friday','sat':'Saturday'}

>>> print D

{'wed': 'Wednesday', 'sun': 'Sunday', 'thu': 'Thursday', 'tue': 'Tuesday', 'mon': 'Monday', 'fri': 'Friday', 'sat': 'Saturday'}
```

Creation, initializing and accessing the elements in a Dictionary

The function dict ( ) is used to create a new dictionary with no items. This function is called built-in function. We can also create dictionary using {}.

```
>>> D=dict()
>>> print D
{}
```

{} represents empty string. To add an item to the dictionary (empty string), we can use square brackets for accessing and initializing dictionary values.

Example

```
>>> H=dict()
>>> H["one"]="keyboard"
>>> H["two"]="Mouse"
>>> H["three"]="printer"
>>> H["Four"]="scanner"
>>> print H
{'Four': 'scanner', 'three': 'printer', 'two': 'Mouse', 'one': 'keyboard'}
>>>
```

Traversing a dictionary

Let us visit each element of the dictionary to display its values on screen. This can be done by using 'for-loop'.

Example

Code

```
H={'Four': 'scanner', 'three': 'printer', 'two': 'Mouse', 'one': 'keyboard'}
for i in H:
print i,":", H[i],"   ",
```

Output

```
>>>
Four: scanner     one: keyboard     three: printer     two: Mouse
>>>
```

OR

Code

```
H = {'Four': 'scanner', 'three': 'printer', 'two': 'Mouse', 'one': 'keyboard'}
print "i value","\t","H[i] value"
for i in H:
print i,"\t", H[i]
```

Output

```
i value   H[i] value
Four      scanner
one       keyboard
three     printer
two       Mouse
```

As said previously, the order of items in a dictionary is unpredictable.

Creating, initializing values during run time (Dynamic allocation)

We can create a dictionary during run time also by using dict () function. This way of creation is called dynamic allocation. Because, during the run time, memory keys and values are added to the dictionary.

Example

Write a program to input total number of sections and class teachers' name in 11th class and display all information on the output screen.

Code

```
classxi=dict()
n=input("Enter total number of section in xi class")
i=1
while i<=n:
a=raw_input("enter section")
```

```
b=raw_input ("enter class teacher name")
classxi[a]=b
i=i+1
print "Class","\t","Section","\t","teacher name"
for i in classxi:
print "XI","\t",i,"\t",classxi[i]
```

Output

```
>>>
Enter total number of section in xi class3
enter sectionA
enter class teacher nameLeena
enter sectionB
enter class teacher nameMadhu
enter sectionC
enter class teacher nameSurpreeth
Class    Section    teacher name
XI       A              Leena
XI       C              Surpreeth
XI       B              Madhu
>>>
```

Appending values to the dictionary

We can add new elements to the existing dictionary, extend it with single pair of values or join two dictionaries into one. If we want to add only one element to the dictionary, then we should use the following method.

Syntax:

Dictionary name [key]=value

Example

```
>>> a={"mon":"monday","tue":"tuesday","wed":"wednesday"}
>>> a["thu"]="thursday"
>>> print a
{'thu': 'thursday', 'wed': 'wednesday', 'mon': 'monday', 'tue': 'tuesday'}
>>>
```

Merging dictionaries: An update ( )

Two dictionaries can be merged in to one by using update ( ) method. It merges the keys and values of one dictionary into another and overwrites values of the same key.

Syntax:

Dic_name1.update (dic_name2)

Using this dic_name2 is added with Dic_name1.

Example

```
>>> d1={1:10,2:20,3:30}
>>> d2={4:40,5:50}
>>> d1.update(d2)
>>> print d1
{1: 10, 2: 20, 3: 30, 4: 40, 5: 50}
```

Example

```
{1: 10, 2: 30, 3: 30, 5: 40, 6: 60}  # k>>> d1={1:10,2:20,3:30}   # key 2 value is 20
>>> d2={2:30,5:40,6:60}  #key 2 value is 30
>>> d1.update(d2)
>>> print d1
ey 2 value is replaced with 30 in d1
```

Removing an item from dictionary

We can remove item from the existing dictionary by using del key word.

Syntax:

del dicname[key]

Example

```
>>> A={"mon":"monday","tue":"tuesday","wed":"wednesday","thu":"thursday"}

>>> del A["tue"]

>>> print A

{'thu': 'thursday', 'wed': 'wednesday', 'mon': 'monday'}

>>>
```

Dictionary functions and methods

cmp ( )

This is used to check whether the given dictionaries are same or not. If both are same, it will return 'zero', otherwise return 1 or -1. If the first dictionary having more number of items, then it will return 1, otherwise return -1.

Syntax:

cmp(d1,d2) **#d1and d2 are dictionary.**

returns 0 or 1 or -1

Example

```
  >>>
D1={'sun':'Sunday','mon':'Monday','tue':'Tuesday','wed':'Wednesday','thu':'Thursd
ay','fri':'Friday','sat':'Saturday'}

  >>>
D2={'sun':'Sunday','mon':'Monday','tue':'Tuesday','wed':'Wednesday','thu':'Thursd
ay','fri':'Friday','sat':'Saturday'}

>>> D3={'mon':'Monday','tue':'Tuesday','wed':'Wednesday'}

>>> cmp(D1,D3)              #both are not equal
```

```
1
>>> cmp(D1,D2)        #both are  equal
0
>>> cmp(D3,D1)
-1
```

len( )

This method returns number of key-value pairs in the given dictionary.

Syntax:

```
len(d)      #d dictionary
```

returns number of items in the list.

Example

```
>>> H={'Four': 'scanner', 'three': 'printer', 'two': 'Mouse', 'one': 'keyboard'}
>>> len(H)
4
```

clear ( )

It removes all items from the particular dictionary.

Syntax:

```
d.clear( )    #d dictionary
```

Example

```
>>> D={'mon':'Monday','tue':'Tuesday','wed':'Wednesday'}
>>> print D
{'wed': 'Wednesday', 'mon': 'Monday', 'tue': 'Tuesday'}
>>> D.clear( )
>>> print D
{}
```

get(k, x)

There are two arguments (k, x) passed in 'get( )' method. The first argument is key value, while the second argument is corresponding value. If a dictionary has a given key (k), which is equal to given value (x), it returns the corresponding value (x) of given key (k). However, if the dictionary has no key-value pair for given key (k), this method returns the default values same as given key value. The second argument is optional. If omitted and the dictionary has no key equal to the given key value, then it returns None.

Syntax:

D.get (k, x)   #D dictionary, k key and x value

Example

```
>>>
D={'sun':'Sunday','mon':'Monday','tue':'Tuesday','wed':'Wednesday','thu':'Thursda
y','fri':'Friday','sat':'Saturday'}
>>> D.get('wed',"wednesday")        # corresponding value wed
'Wednesday'
>>> D.get("fri","monday")           # default value of fri
'Friday'
>>> D.get("mon")                    # default value of mon
'Monday'
>>> D.get("ttu")                    # None
>>>
```

has_key( )

This function returns 'True', if dictionary has a key, otherwise it returns 'False'.

Syntax:

D.has_key(k)   #D dictionary and k key

Example

```
>>>
```

```
D={'sun':'Sunday','mon':'Monday','tue':'Tuesday','wed':'Wednesday','thu':'Thursda
y','fri':'Friday','sat':'Saturday'}
>>> D.has_key("fri")
True
>>> D.has_key("aaa")
False
>>>
```

items( )

It returns the content of dictionary as a list of key and value. The key and value pair will be in the form of a tuple, which is not in any particular order.

Syntax:

D.items()   # D dictionary

Example

```
>>>
D={'sun':'Sunday','mon':'Monday','tue':'Tuesday','wed':'Wednesday','thu':'Thursda
y','fri':'Friday','sat':'Saturday'}
>>> D.items()
[('wed', 'Wednesday'), ('sun', 'Sunday'), ('thu', 'Thursday'), ('tue', 'Tuesday'), ('mon',
'Monday'), ('fri', 'Friday'), ('sat', 'Saturday')]
```

Note: items () is different from print command because, in print command dictionary values are written in {}

keys()

It returns a list of the key values in a dictionary, , which is not in any particular order.

Syntax:

D.keys( )   #D dictionary

Example

```
>>>
```

```
D={'sun':'Sunday','mon':'Monday','tue':'Tuesday','wed':'Wednesday','thu':'Thursda
y','fri':'Friday','sat':'Saturday'}

>>> D.keys()

['wed', 'sun', 'thu', 'tue', 'mon', 'fri', 'sat']

>>>
```

values()

It returns a list of values from key-value pairs in a dictionary, which is not in any particular order. However, if we call both the items () and values() method without changing the dictionary's contents between these two (items() and values()), Python guarantees that the order of the two results will be the same.

Syntax:

D.values()   #D values

Example

```
>>>

D={'sun':'Sunday','mon':'Monday','tue':'Tuesday','wed':'Wednesday','thu':'Thursda
y','fri':'Friday','sat':'Saturday'}

>>> D.values()

['Wednesday', 'Sunday', 'Thursday', 'Tuesday', 'Monday', 'Friday', 'Saturday']

>>> D.items()

[('wed', 'Wednesday'), ('sun', 'Sunday'), ('thu', 'Thursday'), ('tue', 'Tuesday'), ('mon',
'Monday'), ('fri', 'Friday'), ('sat', 'Saturday')]
```

Solved Examples

1. Write a python program to input 'n' names and phone numbers to store it in a dictionary and to input any name and to print the phone number of that particular name.

   Code

   ```
   phonebook=dict()
   ```

```
n=input("Enter total number of friends")
i=1
while i<=n:
a=raw_input("enter name")
b=raw_input("enter phone number")
phonebook[a]=b
i=i+1
name=raw_input("enter name")
f=0
l=phonebook.keys()
for i in l:
if (cmp(i,name)==0):
print "Phone number= ",phonebook[i]
f=1
if (f==0):
print "Given name not exist"
```

Output

```
>>>
Enter total number of friends3
enter nameMona
enter phone number23456745
enter nameSonu
enter phone number45678956
enter nameRohan
enter phone number25678934
enter nameSonu
```

```
Phone number=  45678956
>>>
```

2. Write a program to input 'n' employee number and name and to display all employee's information in ascending order based upon their number.

Code

```
empinfo=dict()
n=input("Enter total number of employees")
i=1
while i<=n:
a=raw_input("enter number")
b=raw_input("enter name")
empinfo[a]=b
i=i+1
l=empinfo.keys()
l.sort()
print "Employee Information"
print "Employee Number",'\t',"Employee Name"
for i in l:
print i,'\t',empinfo[i]
```

Output

```
>>>
Enter total number of employees5
enter number555
enter nameArpit
enter number333
enter nameShilpa
enter number777
enter nameKush
enter number222
enter nameAnkita
enter number666
enter nameArun
Employee Information
Employee Number        Employee Name
222                     Ankita
333                     Shilpa
555                     Arpit
666                      Arun
777                      Kush
>>>
```

3. Write the output for the following Python codes.

```
A={1:100,2:200,3:300,4:400,5:500}
print A.items()
print A.keys()
print A.values()
```

Output

```
[(1, 100), (2, 200), (3, 300), (4, 400), (5, 500)]
[1, 2, 3, 4, 5]
[100, 200, 300, 400, 500]
```

4. Write a program to create a phone book and delete particular phone number using name.

Code

```
phonebook=dict()
n=input("Enter total number of friends")
i=1
while i<=n:
a=raw_input("enter name")
b=raw_input("enter phone number")
phonebook[a]=b
i=i+1
name=raw_input("enter name")
del phonebook[name]
l=phonebook.keys()
print "Phonebook Information"
print "Name",'\t',"Phone number"
for i in l:
print i,'\t',phonebook[i]
```

Output

```
>>>
Enter total number of friends5
enter nameLeena
enter phone number 9868734523
enter nameMadhu
enter phone number 9934567890
enter nameSurpreeth
enter phone number 9678543245

enter nameDeepak

enter phone number 9877886644

enter nameAnuj

enter phone number 9655442345

enter nameDeepak

Phonebook Information

Name            Phone number

Leena           9868734523

Surpreeth       9678543245

Madhu           9934567890

Anuj            9655442345

>>>
```

EXERCISE

1. Write the code to input any 5 years and the population of any city and print it on the screen.

2. Write a code to input 'n' number of subject and head of the department and also display all information on the output screen.

3. Write the output for the following codes.

```
A={10:1000,20:2000,30:3000,40:4000,50:5000}
print A.items()
print A.keys()
print A.values()
```

4. Write a code to create customer's list with their number & name and delete any particular customer using his /her number.

5. Write a Python program to input 'n' names and phone numbers to store it in a dictionary and print the phone number of a particular name.

6. Find errors from the following codes:

```
c=dict()
n=input(Enter total number )
i=1
while i<=n
a=raw_input("enter place")
b=raw_input("enter number")
c(a)=b
i=i+1
print "place","\t","number"
for i in c:
print   i,"\t",cla[i]
```

Chapter 4

Tuples

After studying this lesson, the students will be able to

- *understand the need of Tuples;*
- *solve problems by using Tuples;*
- *get clear idea about Tuple functions; and*
- *understand the difference between list, dictionary and tuples.*

What is a Tuple?

A tuple is a sequence of values, which can be of any type and they are indexed by integer. Tuples are just like list, but we can't change values of tuples in place. Thus tuples are immutable. The index value of tuple starts from 0.

A tuple consists of a number of values separated by commas. For example:

```
>>> T=10, 20, 30, 40
>>> print T
(10, 20, 30, 40)
```

But in the result, same tuple is printed using parentheses. To create a tuple with single element, we have to use final comma. A value with in the parenthesis is not tuple.

Example

```
>>> T=(10)
>>> type(T)
<type 'int'>
```

Example

```
>>> t=10,
>>> print t
(10,)
```

Example

```
>>> T=(10,20)
>>> type(T)
<type 'tuple'>
```

Example

Tuple with string values

```
>>> T=('sun','mon','tue')
>>> print T
('sun', 'mon', 'tue')
```

Example

Tuples with single character

```
>>> T=('P','Y','T','H','O','N')
>>> print T
('P', 'Y', 'T', 'H', 'O', 'N')
```

Tuple Creation

If we need to create a tuple with a single element, we need to include a final comma.

Example

```
>>> t=10,
>>> print t
(10,)
```

Another way of creating tuple is built-in function tuple ().

Syntax:

T = tuple()

Example

```
>>> T=tuple()
>>> print T
()
```

Add new element to Tuple

We can add new element to tuple using + operator.

Example

```
>>> t=(10,20,30,40)
>>> t+(60,)     # this will not create modification of t.
(10, 20, 30, 40, 60)
>>> print t
(10, 20, 30, 40)
>>> t=t+(60,) # this will do modification of t.
>>> print t
(10, 20, 30, 40, 60)
```

Example

Write a program to input 'n' numbers and store it in tuple.

Code

```
t=tuple()
n=input("Enter any number")
print " enter all numbers one after other"
for i in range(n):
a=input("enter number")
t=t+(a,)
print "output is"
print t
```

Output

```
>>>
Enter any number3
enter all numbers one after other
enter number10
enter number20
enter number30
output is
(10, 20, 30)
>>>
```

Another version of the above program:

Code

```
t=tuple()
n=input("Enter any number")
print " enter all numbers one after other"
for i in range(n):
a=input("enter number")
t=t+(a,)
print "output is"
for i in range(n):
print t[i]
```

Output

```
>>>
Enter any number3
enter all numbers one after other
enter number10
enter number20
enter number30
output is
10
20
30
>>>
```

We can also add new element to tuple by using list. For that we have to convert the tuple into a list first and then use append() function to add new elements to the list. After completing the addition, convert the list into tuple. Following example illustrates how to add new elements to tuple using a list.

```
>>> T=tuple()  #create empty tuple
>>> print T
()
>>> l=list(T)           #convert tuple       into list
>>> l.append(10)        #Add new elements to list
>>> l.append(20)
>>> T=tuple(l)          #convert list into tuple
>>> print T
(10, 20)
```

Initializing tuple values:

```
>>> T=(0,)*10
>>> print T
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
```

Tuple Assignment

If we want to interchange (swap) any two variable values, we have to use temporary variable. For example;

```
>>> A=10
>>> B=20
>>> print A,B
10 20
>>> T=A
>>> A=B
>>> B=T
>>> print A,B
20 10
```

But in python, **tuple assignment** is more elegant:

Example

```
>>> T1=(10,20,30)
>>> T2=(100,200,300,400)
>>> print T1
(10, 20, 30)
>>> print T2
(100, 200, 300, 400)
>>> T1,T2=T2,T1      # swap T1 and T2
>>> print T1
(100, 200, 300, 400)
>>> print T2
(10, 20, 30)
```

The left side is a tuple of variables, while the right side is a tuple of expressions. Each value is assigned to its respective variable. All the expressions on the right side are evaluated before any of the assignments.

The number of variables on the left and the number of values on the right have to be the same:

Example

```
>>> T1=(10,20,30)

>>> T2=(100,200,300)

>>> t3=(1000,2000,3000)

>>> T1,T2=T2,T1,t3
Traceback (most recent call last):
    File "<pyshell#3>", line 1, in <module>
    T1,T2=T2,T1,t3
ValueError: too many values to unpack
```

Here, two tuples are in the left side and three tuples are in right side. That is why, we get errors. Thus, it is required to have same number of tuples in both sides to get the correct result.

Example

```
>>> T1,T2,t3=t3,T1,T2

>>> print T1

(1000, 2000, 3000)

>>> print T2

(10, 20, 30)

>>> print t3

(100, 200, 300)
```

Tuple Slices

Slice operator works on Tuple also. This is used to display more than one selected value on the output screen. Slices are treated as boundaries and the result will contain all the elements between boundaries.

Syntax is:

Seq = T [start: stop: step]

Where start, stop & step all three are optional. If we omit first index, slice starts from '0'. On omitting stop, slice will take it to end. Default value of step is 1.

Example

```
>>> T=(10,20,30,40,50)
>>> T1=T[2:4]
>>> print T1
(30, 40)
```

In the above example, starting position is 2 and ending position is 3(4-1), so the selected elements are 30 & 40.

```
>>> T[:]
(10, 20, 30, 40, 50)
```

Will produce a copy of the whole tuple.

```
>>> T[::2]
(10, 30, 50)
```

Will produce a Tuple with every alternate element.

```
>>> T[:3]
(10, 20, 30)
```

Will produce 0 to 2(3-1)

```
>>> T[2:]
(30, 40, 50)
```

Will produce from 2 to end.

Tuple Functions

cmp( )

This is used to check whether the given tuples are same or not. If both are same, it will return 'zero', otherwise return 1 or -1. If the first tuple is big, then it will return 1, otherwise return -1.

Syntax:

cmp(t1,t2)   **#t1and t2 are tuples.**

returns 0 or 1 or -1

Example

```
>>> T1=(10,20,30)
>>> T2=(100,200,300)
>>> T3=(10,20,30)
>>> cmp(T1,T2)
-1
>>> cmp(T1,T3)
0
>>> cmp(T2,T1)
1
```

len( )

It returns the number of items in a tuple.

Syntax:

len(t)   **#t tuples**

returns number of items in the tuple.

Example

```
>>> T2=(100,200,300,400,500)
>>> len(T2)
5
```

max( )

It returns its largest item in the tuple.

Syntax:

```
max(t)        #t tuples
```

returns maximum value among the given tuple.

Example

```
>>> T=(100,200,300,400,500)
>>> max(T)
500
```

min( )

It returns its smallest item in the tuple.

Syntax:

```
min(t)        #t tuples
```

returns minimum value among the given tuple.

Example

```
>>> T=(100,200,300,400,500)
>>> min(T)
100
```

tuple( )

It is used to create empty tuple.

Syntax:

```
T=tuple()        #t tuples
```

Create empty tuple.

Example

```
>>> t=tuple()
>>> print t
()
```

Solved Examples

1. Write a program to input 5 subject names and put it in tuple and display that tuple information on the output screen.

Code

```
t=tuple()
print " enter all subjects one after other";
for i in range(5):
a=raw_input("enter subject")
t=t+(a,)
print "output is"
print t
```

Output

```
>>>
enter all subjects one after other
enter subjectEnglish
enter subjectHindi
enter subjectMaths
enter subjectScience
enter subjectSocial Science
output is
('English', 'Hindi', 'Maths', 'Science', 'Social Science')
>>>
```

2. Write a program to input any two tuples and interchange the tuple values.

Code

```
t1=tuple()
n=input("Total number of values in first tuple")
```

```
for i in range(n):
a=input("enter elements")
t1=t1+(a,)
t2=tuple()
m=input("Total number of values in first tuple")
for i in range(m):
a=input("enter elements")
t2=t2+(a,)
print "First Tuple"
print t1
print "Second Tuple"
print t2
t1,t2=t2,t1
print "AFTER SWAPPING"
print "First Tuple"
print t1
print "Second Tuple"
print t2
```

Output

```
>>>
Total number of values in first tuple3
enter elements100
enter elements200
enter elements300
Total number of values in first tuple4
enter elements10
enter elements20
enter elements30
enter elements40
First Tuple
(100, 200, 300)
Second Tuple
(10, 20, 30, 40)
AFTER SWAPPING
First Tuple
(10, 20, 30, 40)
Second Tuple
(100, 200, 300)
>>>
```

3. Write a program to input 'n' numbers and store it in a tuple and find maximum & minimum values in the tuple.

Code

```
t=tuple()
n=input("Total number of values in  tuple")
for i in range(n):
a=input("enter elements")
t=t+(a,)
print "maximum value=",max(t)
print "minimum value=",min(t)
```

Output

```
>>>
Total number of values in  tuple3
```

```
enter elements40
enter elements50
enter elements10
maximum value= 50
minimum value= 10
>>>
```

4. Find the output from the following code:

```
T=(10,30,2,50,5,6,100,65)
print max(T)
print min(T)
```

Output

```
100
2
```

5. Find the output from the following code:

```
t=tuple()
t = t +(PYTHON,)
print t
print len(t)
t1=(10,20,30)
print len(t1)
```

Output

```
('PYTHON',)
1
3
```

EXERCISE

1. Write the output from the following codes;

(i)
```
t=(10,20,30,40,50)
print  len(t)
```

(ii)
```
t=('a','b','c','A','B')
max(t)
min(t)
```

(iii)
```
T1=(10,20,30,40,50)
T2 =(10,20,30,40,50)
T3 =(100,200,300)
cmp(T1,T2)
cmp(T2,T3)
cmp(T3,T1)
```

(iv)
```
t=tuple()
Len(t)
```

(v)
```
T1=(10,20,30,40,50)
T2=(100,200,300)
T3=T1+T2
print T3
```

2. Write a program to input two set values and store it in tuples and also do the comparison.

3. Write a program to input 'n' employees' salary and find minimum & maximum salary among 'n' employees.

4. Find the errors from the following code:

```
t=tuple{}
```

```
n=input(Total number of values in  tuple)
for i in range(n)
a=input("enter elements")
t=t+(a)
print "maximum value=",max(t)
print "minimum value=",min(t)
```

5. Write a program to input 'n' customers' name and store it in tuple and display all customers' names on the output screen.

6. Write a program to input 'n' numbers and separate the tuple in the following manner.

   Example

   T=(10,20,30,40,50,60)

   T1 =(10,30,50)

   T2=(20,40,60)

CBSE
CONTINUOUS AND COMPREHENSIVE EVALUATION
KNOW - AS YOU GROW

CBSE

CONTINUOUS AND COMPREHENSIVE EVALUATION

KNOW - AS YOU GROW

CENTRAL BOARD OF SECONDARY EDUCATION

Shiksha Kendra, 2, Community Centre, Preet Vihar, Delhi-110 092 India