



As per latest JKBOSE Syllabus-2025

STUDY MATERIAL

For Class 11th

COMPUTER SCIENCE

By

MR. JAVEED HASSAN SOFI (PGDCA/MCA/BED)



About the Author

Mr. Javeed Hassan Sofi (PGDCA/MCA/BEd) is a dedicated educator, computer science teacher, and IT expert from district Bandipora UT J&K, known for his innovative approach to teaching. He has been actively involved in training teachers and creating high-quality digital learning content. His work is widely appreciated across India through the **DIKSHA platform**, where his e-content has benefited thousands of students and teachers.

He is also a proud member of the **State Resource Group (SRG)**, a **cyber safety mentor with NCERT New Delhi**, and a **resource person at SCERT J&K and DIET Bandipora**. His research paper titled "*Multimedia and Cognitive Load*" was published in an international journal, highlighting his deep interest in blending education with technology.

This book is his sincere effort to provide **simple, clear, and exam-oriented study material** for Class 11 Computer Science students as per latest syllabus of JKBOSE. His goal is to help learners understand every concept with ease and grow in confidence.

Warm Regards
Mr. Javeed Hassan Sofi

Content

S. No.	Topic / Chapter Title	Page No.
1	Computer Fundamentals	3-8
2	Software Concepts	9-12
3	Number System	13-17
4	Programming Methodology	18-20
5	Introduction to Python	21-22
6	Data Types and Operators	23-25
7	Strings in Python	26-27
8	Practicals	28-29

Unit-1 (Computer Fundamentals)

History of Computers:

The history of computers began with the **Abacus** (3000 BC), the first counting tool used in China. In 1617, **Napier's Bones** helped with multiplication and division. **Pascaline**, invented by Blaise Pascal in 1642, was the first mechanical calculator, followed by the **Leibniz Calculator** in 1671, which could perform all basic math operations. In 1822, **Charles Babbage** designed the **Difference Engine** to solve math tables and later proposed the **Analytical Engine** (1837), the first idea of a modern general-purpose computer. In 1944, the **Mark I**, an electromechanical computer, was developed by IBM and Harvard. The **ENIAC** (1946) was the first electronic general-purpose computer, and the **EDVAC** introduced the concept of stored programs. The **UNIVAC** (1951) became the first computer used for business. Today's modern computers use microprocessors and artificial intelligence, making them fast, small, and useful in many areas like education, banking, and healthcare.

Generations of Computers

Generation	Time Period	Technology Used	Example
1st	1940-1956	Vacuum Tubes	ENIAC
2nd	1956-1963	Transistors	IBM 1401
3rd	1964-1971	Integrated Circuits	IBM 360
4th	1971–Present	Microprocessors	Desktop PCs
5th	Present+	AI & Super Computers	Siri, Alexa

Functions of a Computer

1 Input

What it means: Input is the process of entering data or instructions into the computer.

How it works:

- The user types or provides data using input devices.
- The data can be numbers, text, images, sound, etc.

Examples of input devices:

- **Keyboard:** For typing text or numbers.
- **Mouse:** For selecting or clicking items on the screen.
- **Scanner:** For scanning documents and pictures.
- **Microphone:** For giving voice input.

2 Processing

What it means: After data is entered, the computer works on it to convert it into meaningful information.

How it works:

- The **CPU (Central Processing Unit)** processes the data.
- It includes:
 - **ALU (Arithmetic Logic Unit)**: Performs calculations.
 - **CU (Control Unit)**: Controls all parts of the computer.

Example: When you solve a math problem on a computer, it processes your input to give the answer.

3 Storage

What it means: Computers store data for current use and for future use.

Types of storage:

- **Primary Storage:**
 - **RAM (Random Access Memory)** – Temporary storage; data is lost when the computer is turned off.
 - **ROM (Read Only Memory)** – Permanent memory; stores essential programs.
- **Secondary Storage:**
 - **Hard Disk, Pen Drive, CD/DVD** – Used to store data permanently.

Example: Saving a file or document on your computer.

4 Output

What it means: After processing, the computer shows or gives the result to the user.

How it works:

- Processed data (information) is sent to **output devices**.

Examples of output devices:

- **Monitor:** Displays the result visually.
- **Printer:** Prints the result on paper.
- **Speakers:** Give sound output.

Example: Seeing a typed letter on screen or getting a printout.

5 Control

What it means: The **Control Unit** inside the CPU manages and controls all the other functions.

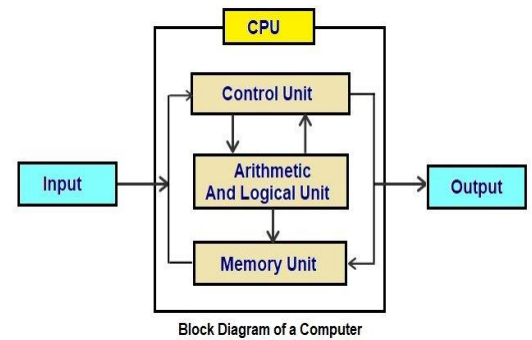
How it works:

- It tells the computer how to carry out input, processing, storage, and output.
- It ensures everything runs in the correct order without mistakes.
- **Example:** Like a teacher controlling all classroom activities, the control unit manages the computer's activities.

Block Diagram of a Computer System

Input Devices → CPU (ALU + CU) → Memory → Output Devices

- **Input:** Keyboard, Mouse
- **CPU:** ALU (calculations), CU (controls operations)
- **Memory:** RAM, ROM, Cache
- **Output:** Monitor, Printer



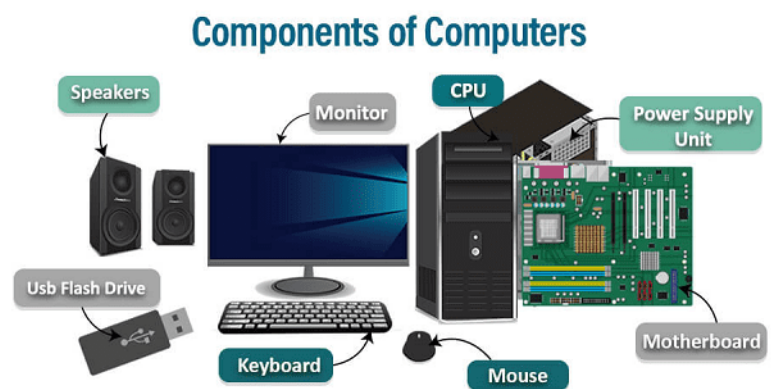
Components of a Computer System

◇ Input Devices

- **Keyboard** – For typing text
- **Mouse** – For pointing and clicking
- **Scanner** – Converts paper to digital
- **Barcode Reader** – Reads barcodes

◇ Output Devices

- **Monitor** – Displays text/images
- **Printer** – Prints documents



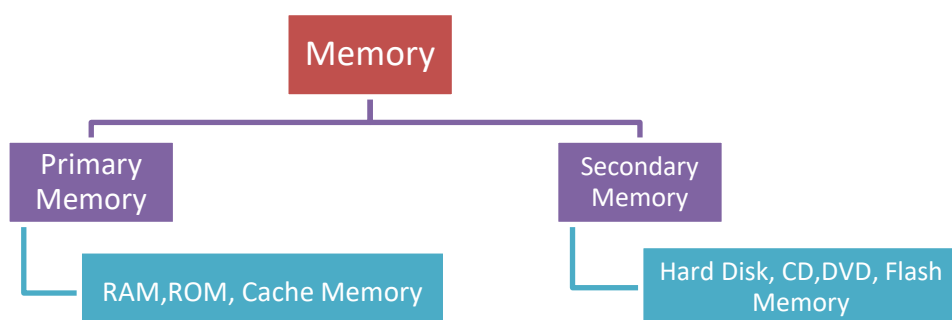
6. CPU (Central Processing Unit)

- **ALU (Arithmetic Logic Unit):** Performs calculations and comparisons.
- **CU (Control Unit):** Controls the flow of data and instructions.

7. Memory Unit

A **Memory Unit** is used to store data and instructions in a computer. It is like the brain's memory – the computer uses it to remember and recall data when needed.

The memory of a computer is mainly divided into **two types**:



1. Primary Memory (Main Memory)

Primary memory is directly accessed by the CPU. It stores data that is currently being used. It is fast but limited in size.

• RAM (Random Access Memory)

- **Temporary memory:** Data is lost when the computer is turned off.
- **Volatile:** Needs constant power to hold data.
- Stores data and instructions that are in use.
- Allows fast read/write access for the CPU.

◇ *Example:* When you open a game or file, it runs in RAM.

• ROM (Read Only Memory)

- **Permanent memory:** Data is not lost even when the power is off.
- Stores important programs like the **booting process** (startup instructions).
- Data is written once and mostly not changed.
- Non-volatile memory.

◇ *Example:* The instructions that start your computer (BIOS) are in ROM.

• Cache Memory

- **Very fast memory** placed between RAM and CPU.
- Stores frequently used data or instructions.
- Helps the CPU access data faster than fetching it from RAM.

◇ *Example:* When the CPU performs the same task repeatedly, it quickly gets the data from the cache.

◇ 2. Secondary Memory (Storage Devices)

Secondary memory is used for **long-term data storage**. It is slower than primary memory but can store much more data.

• Hard Disk

- Main storage device of a computer.
- Stores operating system, software, documents, videos, etc.
- Can store data in **GBs or TBs**.
- Available as **internal (inside CPU)** and **external (portable)**.

◇ *Example:* All your saved files and installed apps are stored on the hard disk.

• CD/DVD (Compact Disc / Digital Versatile Disc)

- Optical storage device.
- Data is read/written using a laser.
- CDs store up to 700 MB, DVDs up to 4.7 GB or more.
- Used for music, movies, software, etc.

◇ *Example:* Installing games or software from a CD.

• Pen Drive (USB Flash Drive)

- Small, portable, and easy to use.
- Connected via USB port.
- Can store 4GB, 8GB, 16GB, 32GB or more.
- Used to transfer files between computers.

◇ *Example:* Carrying your project or files to another computer.

Units of Memory

Unit	Symbol	Size
Bit	b	0 or 1
Byte	B	1 character (8 bits)
Kilobyte	KB	1,024 Bytes
Megabyte	MB	1,024 KB
Gigabyte	GB	1,024 MB
Terabyte	TB	1,024 GB
Petabyte	PB	1,024 TB

ROM – Read Only Memory

☒ Definition:

ROM (Read Only Memory) is a **permanent memory** of the computer that **stores important instructions** needed to start the computer. The data in ROM is **written by the manufacturer** and **cannot be changed easily**.

☒ Function of ROM:

- **Stores startup instructions** (also called BIOS – Basic Input Output System).
- Helps the computer **boot up** or start when it is powered on.
- Data in ROM is **non-volatile**, meaning it **remains even when the computer is turned off**.
- It **does not allow changes** during normal use (Read-only).

Concept of PROM, EPROM, EEPROM (Types of ROM)

Memory Type	Full Form	Description
PROM	Programmable Read-Only Memory	Can be written once, then read only
EPROM	Erasable Programmable ROM	Can be erased using UV light and reprogrammed
EEPROM	Electrically Erasable PROM	Can be erased and rewritten using electricity

Unit-2 (Software Concepts)

What is Software?

Software is a collection of instructions, data, or programs. It is not a physical part of the computer. You cannot touch software, but it works inside the system to control hardware and perform actions like typing, drawing, calculating, etc.

Imagine software as the brain and ideas behind the actions. Hardware is like the body, and software tells the body what to do.

Example: When you use a calculator app on your phone, the calculator is software, and your phone is the hardware.

Types of Software

Software can be divided into three main types:

a) System Software

This software manages and controls the internal operations of a computer. It helps the hardware and software to work together. You do not use this software directly; it works in the background.

Examples include operating systems like Windows, Linux, and Android.

b) Application Software

This is the software which is used to perform specific tasks. It helps you write, draw, calculate, play music or videos, etc.

Some common examples include MS Word for typing, MS Excel for calculations, VLC Player for playing videos, and web browsers like Chrome or Firefox.

c) Utility Software

Utility software performs maintenance and support tasks. It keeps your computer safe, clean, and running smoothly.

Examples of utility software include antivirus software to protect from viruses, file compression tools, backup software, and disk cleanup tools.

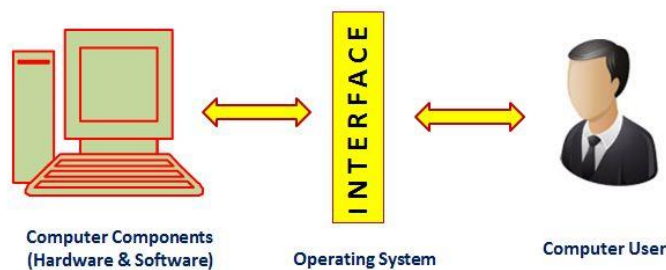
Why Do We Need an Operating System?

An operating system is the most important software in a computer. It is the first program that loads when the computer starts. It acts as a bridge (or Interface) between the user and the hardware.

The operating system helps in running applications, managing memory, processing input and output, and securing files.

Without an operating system, you would not be able to use a computer properly. It allows you to open apps, browse files, print documents, and much more.

Some common operating systems are Windows, macOS, Linux, and Android.



Functions of an Operating System

The operating system performs many functions to manage the computer. Some of the main functions are:

- **Processor Management:** It decides which task should be handled by the CPU and in what order. It also switches between tasks smoothly.
- **Memory Management:** It keeps track of how memory (RAM) is being used and makes sure each program gets enough memory to run properly.
- **File Management:** It organizes the data into files and folders, allows saving and deleting files, and manages storage devices like hard disks or pen drives.
- **Device Management:** It controls the input and output devices such as the keyboard, mouse, printer, and scanner. It uses device drivers to help the operating system communicate with these devices.
- **User Interface:** It provides a way for users to interact with the computer, either through a graphical interface (like icons and windows) or a command-line interface (typing commands).
- **Security:** It protects your files and data using passwords and permissions, and it can prevent unauthorized access.

Computer Languages

Computers do not understand English or any human language. They only understand binary (0s and 1s). To make it easier for humans to communicate with computers, programming languages were developed.

There are three main types of computer languages:

a) Machine Language

Machine language is the lowest-level language that a computer understands. It is written entirely in binary (0s and 1s). It is very difficult for humans to read or write. Programs written in this language run very fast, but they are hard to develop.

Example: 10100010 11000011

b) Assembly Language

Assembly language uses short codes or symbols called mnemonics, like ADD, MOV, SUB, etc., instead of binary numbers. It is easier than machine language, but still considered low-level. An assembler is needed to convert assembly language into machine language.

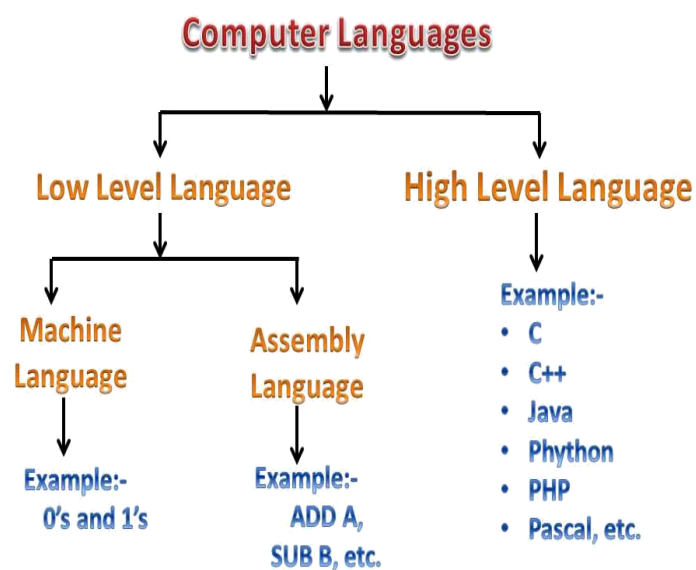
Example: MOV A, B (Move data from B to A)

c) High-Level Language

High-level languages are designed to be easy to read and write. They use words and symbols similar to English. They are much easier to understand and are portable, meaning the same code can run on different types of computers.

Examples of high-level languages include Python, C++, Java, BASIC, and Pascal.

Example in Python: `print("Hello World")`



Language Processors

Since the computer can only understand machine language, we need special programs to convert our high-level or assembly language code into machine code. These programs are called **language processors** or **translators**.

There are three main types of language processors:

a) Assembler

It converts assembly language into machine language. Without an assembler, the computer cannot understand assembly instructions.

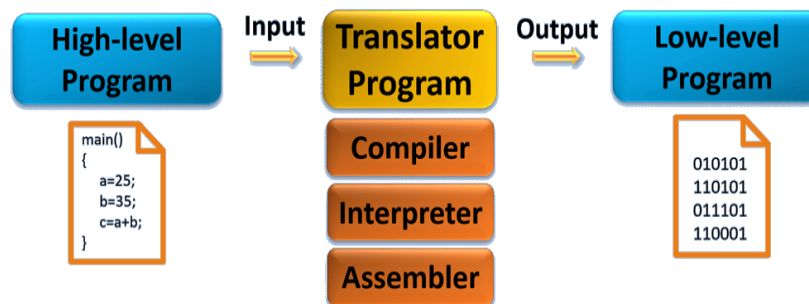
b) Compiler

A compiler translates the entire high-level program into machine language at once. If there are errors, it shows them after translation is complete. It is fast and used in languages like C and C++.

c) Interpreter

An interpreter translates high-level language code into machine language line by line. It executes one line, checks for errors, and then goes to the next. It is slower than a compiler but helpful in testing and learning.

Python is a popular interpreted language.



Unit-3 (Number System)

NUMBER SYSTEM

☒ Definition:

A **number system** is a way to **represent numbers** using **symbols or digits**.

In computers, number systems are used to **store, process, and display** all kinds of data like numbers, text, and images.

◇ Types of Number Systems

Number System	Base	Digits Used	Use in Computers
Binary	2	0, 1	Main language of computers (ON/OFF signals)
Decimal	10	0 to 9	Used by humans for input/output
Octal	8	0 to 7	Used as a shortcut for binary (in older systems)
Hexadecimal	16	0–9 and A–F	Used in programming, memory addresses, colours

A = 10, B = 11, C = 12, D = 13, E = 14, F = 15 in hexadecimal.

Here are some examples:

- **Binary number:** 1010 in binary is written as **1010₂** (base 2).
- **Octal number:** 175 in octal is written as **175₈** (base 8).
- **Decimal number:** 125 in decimal is written as **125₁₀** (base 10).
- **Hexadecimal number:** 7D in hexadecimal is written as **7D₁₆** (base 16).

1. Conversion from Decimal number system to Binary, Octal and Hexadecimal number system

☒ 1. Decimal to Binary (Base 2)

Convert 125 to Binary

We divide by 2 until the quotient becomes 0:

```
2 | 125
2 | 62   → Remainder: 1
2 | 31   → Remainder: 0
2 | 15   → Remainder: 1
2 | 7    → Remainder: 1
2 | 3    → Remainder: 1
2 | 1    → Remainder: 1
0       → Remainder: 1
```

☒ Now write the remainders from bottom to top:

☒ **125₁₀ = 1111101₂**

☒ 2. Decimal to Octal (Base 8)

Convert 125 to Octal number

```
8 | 125
8 | 15   → Remainder: 5
8 | 1     → Remainder: 7
  0       → Remainder: 1
```

☒ Write the remainders from bottom to top:

☒ $125_{10} = 175_8$

☒ 3. Decimal to Hexadecimal (Base 16)

Convert 125 to Hexadecimal number

Remember:

Remainder 10 = A, 11 = B, 12 = C, 13 = D, 14 = E, 15 = F

```
16 | 125
16 | 7     → Remainder: 13 → D
  0       → Remainder: 7
```

☒ Write the remainders from bottom to top:

☒ $125_{10} = 7D_{16}$

2. Conversion from Binary, Octal and Hexadecimal number system to Decimal Number System

☒ 1. Binary to Decimal

👉 **Rule:** Multiply each binary digit by 2 raised to the power of its position, starting from **right to left** (position 0).

Example: Convert 1011_2 to Decimal

$$\begin{aligned} 1011_2 &= 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \\ &= 8 + 0 + 2 + 1 \\ &= 11_{10} \end{aligned}$$

☒ 2. Octal to Decimal

☞ **Rule:** Multiply each octal digit by **8 raised to the power of its position**, from **right to left**.

Example: Convert 157_8 to Decimal

$$157_8 = 1 \times 8^2 + 5 \times 8^1 + 7 \times 8^0$$

$$= 64 + 40 + 7$$

$$= 111_{10}$$

☒ 3. Hexadecimal to Decimal

☞ **Rule:** Multiply each digit by **16 raised to the power of its position**, right to left.

A, **B**, **C**... stand for numbers:

- A = 10, B = 11, C = 12, ..., F = 15

Example: Convert $2F_{16}$ to Decimal

$$2F_{16} = 2 \times 16^1 + 15 \times 16^0$$

$$= 32 + 15$$

$$= 47_{10}$$

3. Conversion from Binary to Octal and Hexadecimal number system using shortcut method

☒ 1. Binary to Octal (Shortcut Method)

☞ **Rule:**

Group binary digits in **sets of 3** (from right to left), and convert each group to its **octal value**.

◇ **Example: Convert 1101011_2 to Octal**

Step 1: Group binary digits in 3s (right to left):

$1101011 \rightarrow 001\ 101\ 011$ (add leading zeros if needed)

Step 2: Convert each group:

$$001 = 1$$

$$101 = 5$$

$$011 = 3$$

Step 3: Combine:

☒ $1101011_2 = 153_8$

☒ 2. Binary to Hexadecimal (Shortcut Method)

Rule:

Group binary digits in **sets of 4** (from right to left), and convert each group to **hex value**.

◇ Example: Convert 1101011_2 to Hexadecimal

Step 1: Group binary digits in 4s (right to left):

$1101011 \rightarrow 0110\ 1011$

(Simpler way is $\rightarrow 0110\ 1011$)

$0110\ 1011 \rightarrow$ Group: 0110 and 1011

Step 2: Convert:

$0110 = 6$

$1011 = B$

Step 3: Combine:

☒ $1101011_2 = 6B_{16}$

4. Conversion from Octal and Hexadecimal number system to Binary number system using shortcut method

☒ 1. Octal to Binary (Shortcut Method)

 Rule: Convert **each octal digit** into a **3-digit binary number**.

◇ Example: Convert 157_8 to Binary

Step-by-step:

$1 \rightarrow 001$

$5 \rightarrow 101$

$7 \rightarrow 111$

Combine:

$001\ 101\ 111 \rightarrow 1101111$

☒ Answer: $157_8 = 1101111_2$

✓ 2. Hexadecimal to Binary (Shortcut Method)

☞ **Rule:** Convert **each hex digit** into a **4-digit binary number**.

◇ **Example: Convert $2F_{16}$ to Binary**

Step-by-step:

$2 \rightarrow 0010$

$F \rightarrow 1111$

Combine:

$0010\ 1111 \rightarrow 101111$

✓ **Answer: $2F_{16} = 101111_2$**

Additional Information:

(Shortcut method from Binary to decimal and decimal to binary)

✓ 1. Binary to Decimal (Using 2 powers)

☞ Write powers of 2 under each digit from **right to left** and multiply.

◇ **Example: Convert 1011_2 to Decimal**

$$= 2^3 + 2^2 + 2^1 + 2^0$$

$$= 8 + 4 + 2 + 1$$

$$= 1 \quad 0 \quad 1 \quad 1$$

$$\text{Add: } 8+2+1 = 11$$

$$= 11_{10}$$

✓ **Answer: $1011_2 = 11_{10}$**

✓ 2. Decimal to Binary (Using 2 powers)

☞ Write powers of 2 under each digit from **right to left** and multiply.

◇ **Example: Convert 13 to Binary**

$$= 2^3 + 2^2 + 2^1 + 2^0$$

$$= 8 + 4 + 2 + 1$$

$$= 1 \quad 1 \quad 0 \quad 1$$

$$\text{Add: } 8+4+1 = 13$$

✓ **Answer: $13_{10} = 1101_2$**

Unit-4 (Programming Methodology)

1. Concept of a Program

A **program** is a set of instructions written in a specific programming language like Python, Java, or C++. These instructions are given to the computer to perform a particular task or solve a specific problem. For example, when you write a program to calculate the sum of two numbers, you are actually giving step-by-step instructions to the computer to take input, perform addition, and then display the result.

Programs can be small (like a calculator) or large (like an operating system). Programming is the process of designing, writing, testing, and maintaining these programs.

2. Characteristics of a Good Program

A good program should follow certain qualities to be considered well-written and useful:

- **Correctness:** The most important quality of a program is that it should produce the correct output for all valid inputs. If a program gives wrong results, it's not useful.
 - **Efficiency:** A good program should take less memory and time. It should be optimized to run fast and consume fewer resources.
 - **Simplicity and Clarity:** The program should be easy to read and understand. It should not be confusing. This can be done by using meaningful variable names and adding comments in the code.
 - **Modularity:** A good program should be divided into small parts or modules. Each module should handle a specific part of the task. This helps in better management and testing of code.
 - **Reusability:** If some part of a program (a function or module) can be used in another program, it saves time. Reusable code is an important feature of a good program.
 - **Maintainability:** The program should be easy to modify or update in the future if there are changes in requirements or technology.
-

3. Concept of Modular Approach

Modular programming is a technique in which a large program is divided into smaller, manageable parts called **modules** or **functions**. Each module handles a specific task and can be written, tested, and debugged independently.

For example, if you are writing a program for school management, you can divide it into different modules such as:

- A module for student admission
- A module for fee collection
- A module for attendance
- A module for report cards

This approach makes the program easier to manage, test, and reuse. If there's an error in one module, it can be fixed without disturbing the entire program.

4. Program Documentation

Documentation means writing down detailed information about a program so that others can understand how it works.

There are two types of documentation:

- **Internal Documentation:** This is written inside the program using comments. Comments are not executed by the computer. They help the programmer and others understand what a specific part of the code does. For example, you can write `# This function calculates the average` before a function to describe it.
- **External Documentation:** This is written separately from the code. It includes user manuals, system design documents, flowcharts, and installation guides. This type of documentation helps new users or developers to understand how to use or work with the software.

Good documentation is helpful when you need to update the program or when someone else wants to understand or work on your code.

5. Program Maintenance

Once a program is developed and used by people, it may need changes due to new requirements, errors, or better solutions. This process of updating, improving, or fixing the program is called **program maintenance**.

There are different types of maintenance:

- **Corrective maintenance:** Fixing errors or bugs in the program.
- **Adaptive maintenance:** Modifying the program to run in a new environment like a different operating system or device.
- **Perfective maintenance:** Making the program better by adding new features or improving performance.
- **Preventive maintenance:** Making changes to reduce the chances of future problems or bugs.

Maintenance is an important part of the software development life cycle because software needs to evolve over time.

6. Debugging a Program

Debugging is the process of finding and removing errors from a program. When a program doesn't run correctly, the programmer needs to examine the code, identify where things are going wrong, and fix those mistakes.

There are several methods of debugging:

- Using print statements to display variable values at different points in the program.
- Using debugging tools available in programming environments (like IDLE, PyCharm, or Visual Studio Code).
- Manually checking the logic and flow of the program line-by-line.

Debugging helps in improving the accuracy and reliability of the program.

7. Errors and Types of Errors

Errors are mistakes in a program that stop it from running properly. There are mainly three types of errors:

- **Syntax Errors:** These are mistakes in the spelling or grammar of the programming language. For example, forgetting to close a bracket, using the wrong punctuation, or writing `print` without parentheses in Python. The program will not run until syntax errors are fixed.
- **Logical Errors:** These occur when the program runs but gives wrong results. This happens because the logic used in the code is incorrect. For example, if you want to subtract two numbers but write addition by mistake, the program will run but give the wrong answer.
- **Runtime Errors:** These errors occur while the program is running. They may not appear every time. Examples include dividing a number by zero, accessing an element from an empty list, or trying to open a file that doesn't exist. These errors cause the program to crash during execution.

Unit-5 (Introduction to Python)

Introduction to Python

► Origin of Python

- **Python** was created by **Guido van Rossum** in the late 1980s.
- It was officially released in **1991**.
- Named after a TV show called "**Monty Python's Flying Circus**" (not the snake!).

► Unique Features of Python

- Easy to read and write.
- Free and open source.
- Interpreted language (runs line-by-line).
- Portable (works on Windows, Mac, Linux).
- Large standard library.
- Used in AI, ML, Web development, etc.

► Python Character Set

- **Letters:** A–Z, a–z
- **Digits:** 0–9
- **Special Symbols:** +, -, *, /, @, %, &, etc.
- **Whitespace:** Space, tab, newline
- **Other Characters:** {}, [], (), :, ., etc.

► Tokens in Python

Tokens are the smallest units of a program.

Types:

1. **Keywords:** Reserved words (e.g., if, else, for, while, def, class)
2. **Identifiers:** Names used for variables, functions, etc. (e.g., name, age)
3. **Literals:** Fixed values (e.g., 10, 'hello', True)
4. **Operators:** Symbols used for operations (e.g., +, -, *)
5. **Punctuators:** Symbols like (), {}, :, ,

► The `print()` and `input()` Statement

- `print()` is used to **display output**.
► Example: `print("Hello World")`
- `input()` is used to **take input from user**.
► Example: `name = input("Enter your name: ")`

► Comments in Python

- Used to explain code.
- Not executed by Python.
- Start with # for single-line comment.
► Example: `# This is a comment`

► Concept of Variables

- A **variable** stores data in memory.
- Example:

```
x = 10
name = "Ali"
a = 6.5
```

► Rules for Naming a Variable

- Can contain letters, digits, underscore.
- Must start with a **letter or underscore**.
- Cannot use **keywords**.
- Case-sensitive (Name and name are different).

☑ Do's and ✗ Don'ts of Using Variables in Python

☑ Do's	✗ Don'ts	💡 Explanation/Example
Use meaningful variable names	Don't use random or confusing names	age = 25 ☑ is better than a = 25 ✗
Use lowercase letters for variable names	Don't use capital letters unless needed	total_marks = 450 ☑, avoid Total_Marks = 450 ✗ (unless it's a constant or class name)
Use underscores (_) to separate words	Don't use spaces or special symbols	student_name = "Ali" ☑, student name = "Ali" ✗ (space not allowed)
Start variable names with a letter or underscore	Don't start with a number	_name = "Javeed" ☑, 1name = "Javeed" ✗
Assign value using = correctly	Don't confuse = with ==	x = 10 ☑ assigns value, but x == 10 ✗ checks equality
Use variables only after defining them	Don't use undefined variables	name = "Sara"; print(name) ☑, print(name) ✗ if not defined earlier
Follow naming rules and PEP8 conventions	Don't use Python reserved keywords	sum_value = 100 ☑, for = 100 ✗ (for is a keyword)

Unit-6 (Data Type & Operators)

Data Types and Operators

In Python, **data types** tell what kind of value a variable has. For example, numbers like 5 are **integers**, numbers with a point like 5.5 are **floats**, words or text like "Hello" are **strings**, and True or False are **Booleans**. Python also has other data types like **lists** and **dictionaries** to store many values. **Operators** are symbols like +, -, *, and / that are used to do different actions like adding, subtracting, comparing, or checking conditions. They help us write logic in our programs and work with the data easily.

► Built-in Data Types in Python

Type	Example
Text Type	str → "Hello"
Numeric Type	int → 10, float → 10.5
Sequence Type	list → [1, 2, 3]
Boolean Type	bool → True, False

► Displaying Data Type

Use `type ()` function:

```
a = 5
print(type(a))           # Output: <class 'int'>
```

► Type Conversion

Convert one type to another:

```
x = 5
y = float(x)             # x becomes 5.0

a = 3.7
b = int(a)               # a becomes 3
```

► Operators in Python

1. Arithmetic Operators

Operator	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Remainder
**	Power (Ex: 2**3 = 8)

2. Comparison Operators

Operator	Checks
==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal

3. Logical Operators

Operator	Meaning
and	Both conditions True
or	At least one True
not	Reverses condition

4. Assignment Operators

Operator	Use
=	Assign value
+=	Add and assign
-=	Subtract and assign
*=	Multiply and assign
/=	Divide and assign
%=	Modulus and assign
**=	Power and assign

- `is` → True if two variables point to the same object.

5. Identity Operators

- `not` → True if they don't.

6. Membership Operators

- `in` → True if value exists in a sequence.
- `not in` → True if value does not exist.

Unit-7(Strings in Python)

Strings in Python

In Python, a **string** is a type of data that is used to store **text**. It can be a word, a sentence, or even a paragraph, and is always written inside **quotes** — either single ('Hello') or double ("Hello"). You can do many things with strings like joining them ("Hello" + "World"), repeating them ("Hi" * 3), or checking their length using len(). Strings are very useful when working with names, messages, or any kind of text in a program.

► Defining String

A string is a set of characters inside quotes:

Program:
name = "Ali"

► Assigning String to Variable

Program:
greeting = "Good Morning"

► Multiline Strings

Use triple quotes:

```
msg = '''This is  
a multiline  
string'''
```

► String Functions

Function	Use
len()	Length of string
upper()	Converts to UPPERCASE
lower()	Converts to lowercase
replace(old, new)	Replaces part of string
in	Checks if substring is present
not in	Checks if substring is not present

Example:

<pre>msg = "Hello" print(len(msg)) print(msg.upper()) print(msg.replace("H", "J"))</pre>	<pre><u>Output</u> # 5 # HELLO # Jello</pre>
--	--

► Slicing Strings

Slicing in Python means cutting a part of a string using **index numbers**.

We use the format: `string[start:end]`

It includes the start position but **not** the end position.

✂ Slice from the Start

To get characters **from the beginning** to a certain position:

```
text = "Python"
print(text[:4])          # Output: Pyth
```

✂ Slice to the End

To get characters **from a position to the end**:

```
text = "Python"
print(text[2:])          # Output: thon
```

abc Use of Escape Characters in Strings

Escape characters are **special symbols** used to insert things like quotes, tabs, or new lines in a string. Here are some common ones:

- `\"` → for double quote inside string
- `\'` → for single quote
- `\n` → for new line
- `\t` → for tab space

☑ Sample Program

```
# Slicing and escape characters in Python
text = "Python Programming"
```

```
# Slicing examples
print("Slice from start to 6:", text[:6])          # Output: Python
print("Slice from index 7 to end:", text[7:])      # Output: Programming
```

```
# Escape characters
print("She said, \"Hello!\")                    # Using double quote inside string
print('It\'s a sunny day.')                     # Using single quote
print("Line1\nLine2")                           # New line
print("Name:\tJaved")                           # Tab space
```

Practicals

1. WAP (WRITE A PROGRAM) to display "hello world".

```
print("Hello World")
```

2. WAP (WRITE A PROGRAM)to add two numbers.

```
a = int(input("Enter first number: "))
b = int(input("Enter second number: "))
sum = a + b
print("The sum is:", sum)
```

3. WAP (WRITE A PROGRAM) to find the length of the string.

```
text = input("Enter a string: ")
print("Length of the string is:", len(text))
```

4. WAP (WRITE A PROGRAM) to convert uppercase string into lowercase and vice-versa.

```
text = input("Enter a string: ")
print("Lowercase:", text.lower())
print("Uppercase:", text.upper())
```

5. WAP (WRITE A PROGRAM) to convert temperature from Celsius to Fahrenheit.

```
celsius = float(input("Enter temperature in Celsius: "))
fahrenheit = (celsius * 9/5) + 32
print("Temperature in Fahrenheit:", fahrenheit)
```

6. WAP (WRITE A PROGRAM) for displaying multiline strings.

```
message = """This is a
multiline string
in Python."""
print(message)
```

7. WAP (WRITE A PROGRAM) to concatenate two strings.

```
str1 = input("Enter first string: ")
str2 = input("Enter second string: ")
result = str1 + str2
print("Concatenated string:", result)
```

8. WAP (WRITE A PROGRAM) to slice a string from start to a particular position in the string.

```
text = input("Enter a string: ")
pos = int(input("Enter position to slice up to: "))
print("Sliced string:", text[:pos])
```

9. WAP (WRITE A PROGRAM) to replace some characters of the string with new characters.

```
text = input("Enter a string: ")
old = input("Enter character to replace: ")
new = input("Enter new character: ")
new_text = text.replace(old, new)
print("Updated string:", new_text)
```

10. WAP (WRITE A PROGRAM) to find the area of a square.

```
side = float(input("Enter the length of a side: "))
area = side * side
print("Area of the square is:", area)
```
