UNIT-I

Problem-Solving

- **Meaning**:
  The process of identifying a problem, analyzing it, and finding the most efficient solution.
- **Why important?**
  Programming is not just about writing code – it's about solving real-world problems step by step.

Steps in Problem-Solving

1. **Understanding the Problem** → Carefully read and figure out what is being asked.
2. **Breaking Down** → Divide the big problem into smaller, manageable parts (sub-problems).
3. **Designing a Solution (Algorithm)** → Plan a sequence of steps to solve each part.
4. **Implementing** → Convert the algorithm into a program (e.g., Python code).
5. **Testing and Debugging** → Check for errors and fix them.
6. **Optimization** → Improve the solution to make it faster or simpler.

Computational Thinking

- **Meaning**:
  A way of thinking like a computer scientist when solving problems.
- **Purpose**:
  Helps in creating solutions that can be implemented effectively with computers.

Key Concepts in Computational Thinking

1. **Decomposition** → Breaking a complex problem into smaller parts.
   - Example: To build a game → split into characters, rules, scoring, graphics.
2. **Pattern Recognition** → Identifying similarities or repeating elements.
   - Example: In math, recognizing that multiplication is repeated addition.
3. **Abstraction** → Focusing only on important details, ignoring unnecessary ones.
   - Example: A map shows roads and landmarks but ignores trees or small objects.

4. **Algorithms** → Writing step-by-step instructions to solve a problem.
    - Example: Recipe for cooking → step-by-step method to reach the final dish.

Computer Fundamentals

- A **computer** is an electronic device that takes input, processes it, stores it, and gives output.
- To work, it needs **two main parts**:
    - **Hardware** → The physical components.
    - **Software** → The set of instructions that tell the hardware what to do.

Computer Hardware

- **Meaning**: The physical, tangible parts of a computer that you can touch.
- **Main Components**:
    1. **Input Devices** → Used to enter data (keyboard, mouse, scanner, microphone).
    2. **Output Devices** → Show results (monitor, printer, speakers).
    3. **Storage Devices** → Store data permanently or temporarily (hard disk, SSD, USB, CD).
    4. **Processor (CPU)** → The "brain" of the computer that processes instructions.
        - Control Unit (CU) → Directs flow of data.
        - Arithmetic Logic Unit (ALU) → Performs calculations and logic operations.
    5. **Memory**:
        - **Primary memory** (RAM, ROM) → Fast, temporary storage.
        - **Secondary memory** (hard disk, SSD) → Permanent storage.

6. **Motherboard** → Main circuit board that connects all components.

Without hardware, a computer cannot exist physically.

Computer Software

- **Meaning**: A set of instructions/programs that tell hardware how to perform tasks.
- **Types of Software**:
    1. **System Software** → Controls and manages hardware.
        - Operating System (Windows, Linux, macOS).
        - Utility software (antivirus, file manager, compression tools).
    2. **Application Software** → Helps users do specific tasks.
        - Word processors (MS Word), browsers (Chrome), games, media players.
    3. **Programming Software** → Provides tools to create new software.
        - Compilers, interpreters, IDEs (Python, Java compilers).

Without software, hardware is useless because it won't know what to do.

Hardware + Software Relationship

- Hardware = Body of the computer.
- Software = Mind of the computer.
- Both are dependent on each other.
    - Hardware executes what software instructs.
    - Software cannot run without hardware.

Syntax

- **Meaning**: The *grammar rules* of a programming language.
- Defines **how statements must be written** so the computer can understand them.
- If syntax is wrong → program will not run (syntax error).

- **Example (conceptual)**: In Python, indentation is part of syntax; if missing, it causes an error.

Semantics

- **Meaning**: The *meaning* of code or statements written in correct syntax.
- Even if the syntax is correct, the program may not do what the programmer intends → this is a **semantic error** (logic error).
- **Example (conceptual)**: Writing code to divide by zero → syntax is fine, but meaning is invalid.

Algorithms

- **Meaning**: A step-by-step procedure or set of rules to solve a problem.
- Should be **clear, finite, and effective**.
- **Properties**:
    1. **Input** → Takes some data.
    2. **Output** → Produces a result.
    3. **Definiteness** → Steps must be clear.
    4. **Finiteness** → Must end after a finite number of steps.
    5. **Effectiveness** → Each step should be doable.
- Used as a blueprint before writing actual code.

Flowcharts

- **Meaning**: A **diagrammatic representation of an algorithm** using symbols.
- Helps visualize the flow of a program.
- **Common Symbols**:
    - **Oval** → Start / End.
    - **Rectangle** → Process / Task.
    - **Diamond** → Decision (Yes/No).
    - **Parallelogram** → Input / Output.
    - **Arrows** → Show flow of control.
- Makes it easier to explain logic to non-programmers and helps in debugging.

History of Python

- **Creator**: Guido van Rossum.
- **Year**: Developed in the late 1980s, first released in **1991**.
- **Inspiration**:
    - Based on ABC language (a teaching language).
    - Guido wanted a simple, easy-to-read language.
- **Name Origin**: Not named after the snake but after the British comedy show *"Monty Python's Flying Circus"*.
- **Evolution**:
    - Python 2 (2000) → Introduced many features but later became outdated.
    - Python 3 (2008 onwards) → Major redesign, current standard version.
- **Current Use**: Popular in web development, data science, AI, machine learning, automation, scripting, and more.

Features of Python

1. **Simple and Easy to Learn**
    - Syntax is close to English, beginner-friendly.
    - Reduces complexity in coding.
2. **High-Level Language**
    - Programmer does not need to manage memory or hardware details.
    - Focuses only on problem-solving logic.
3. **Interpreted Language**
    - Executes line by line (no need to compile before running).
    - Makes debugging easier.
4. **Dynamically Typed**
    - No need to declare variable types explicitly.
    - Type is decided at runtime (e.g., a variable can hold int, then later a string).
5. **Object-Oriented**

    - Supports classes, objects, inheritance, encapsulation, and polymorphism.
    - Promotes reusability and modular programming.
6. **Supports Multiple Paradigms**
    - Procedural programming.
    - Object-oriented programming.
    - Functional programming.
7. **Extensive Standard Library**
    - Comes with built-in modules for math, file handling, networking, databases, etc.
    - Saves time for developers.
8. **Portable and Cross-Platform**
    - Runs on Windows, Mac, Linux, and more without major changes.
9. **Large Community Support**
    - Huge online community, forums, and documentation.
    - Rich third-party libraries (e.g., NumPy, Pandas, Django, TensorFlow).
10. **Used Everywhere**
    - Web development (Django, Flask).
    - Data science and AI.
    - Automation and scripting.
    - Game development, IoT, cyber security.

Downloading Python

- Official source: **python.org** → Downloads section.
- Choose the correct version (usually Python 3.x).

- Available for **Windows, macOS, Linux**.

Installing on Windows

1. Download the installer (.exe file).
2. Run the installer.
3. **Important**: Check the box **"Add Python to PATH"** before proceeding → allows Python to run from Command Prompt.
4. Choose **Install Now** (default settings) or **Customize Installation** if needed.
5. Once installed, confirm by opening **Command Prompt** and typing:
    - python --version (to check Python version).

Installing on macOS

1. macOS usually has Python pre-installed (older version).
2. To install the latest version:
    - Download the .pkg installer from python.org.
    - Run the package installer and follow instructions.
3. Verify installation in **Terminal**:
    - python3 --version.

Installing on Linux

1. Most Linux distributions include Python by default.
2. To install/update manually:
    - For Ubuntu/Debian:
    - sudo apt update
    - sudo apt install python3
3. Verify installation:
    - python3 --version.

Setting Up an IDE (Optional but Recommended)

- While Python can be run from the terminal, an **IDE (Integrated Development Environment)** makes coding easier.
- Popular options:
    - **IDLE** → Comes bundled with Python.
    - **PyCharm** (by JetBrains).
    - **Visual Studio Code (VS Code)**.
    - **Jupyter Notebook** (for data science).

Python REPL (Interactive Mode)

- After installation, you can test Python using REPL (Read–Eval–Print Loop).
- Just type python or python3 in terminal/command prompt.
- Lets you execute Python commands line by line interactively.

Variables in Python

- **Definition**: A variable is a *named memory location* used to store data.
- **Dynamic Typing**: In Python, you don't need to declare variable types – they are assigned automatically at runtime.
- **Rules for Naming Variables**:
    1. Must start with a letter or underscore (_).
    2. Cannot start with a digit.
    3. Can only contain letters, digits, and underscores.
    4. Case-sensitive (Age ≠ age).
    5. Cannot use reserved keywords (e.g., if, while).

Example (conceptual): Storing student's name, age, and marks in variables.

Python Data Types

Python has several built-in data types, categorized as follows:

A. Numeric Types

- **int** → Whole numbers (e.g., 10, -5).
- **float** → Decimal numbers (e.g., 3.14, -2.7).
- **complex** → Numbers with real + imaginary part (e.g., 3 + 4j).

B. Sequence Types

- **str (String)** → Collection of characters inside quotes (e.g., "Hello").
- **list** → Ordered, mutable collection (e.g., [1, 2, 3]).
- **tuple** → Ordered, immutable collection (e.g., (1, 2, 3)).
- **range** → Sequence of numbers, often used in loops.

C. Set Types

- **set** → Unordered collection of unique elements (e.g., {1, 2, 3}).
- **frozenset** → Immutable version of set.

D. Mapping Type

- **dict (Dictionary)** → Key-value pairs (e.g., {"name": "Sam", "age": 20}).

E. Boolean Type

- **bool** → Stores True or False.

F. None Type

- **None** → Represents "no value" or "null".

Key Characteristics

- Variables can change type during execution (dynamic typing).
  Example: A variable can store a number first, and later store a string.
- Python provides functions like type() to check the type of a variable.
- Data types determine the kind of operations that can be performed.

Python Objects

- **Definition**: In Python, **everything is an object** (numbers, strings, functions, classes, etc.).
- Each object has three main properties:
    1. **Identity** → Unique memory address (like an ID card).
    2. **Type** → Defines what kind of object it is (int, str, list, etc.).
    3. **Value** → The actual data stored in the object.
- Objects = combination of **data (attributes)** + **behavior (methods)**.
- Example (conceptual):
    - A string object "Hello" has a value (Hello), a type (str), and methods like upper().

Key Point: Since Python is object-oriented, variables don't hold values directly – they hold *references* to objects in memory.

Python REPL and Basic Syntax

Python REPL

- **REPL** stands for **Read–Eval–Print Loop**.
- It is an **interactive environment** where you can type Python commands and get immediate results.
- Useful for testing small pieces of code, debugging, or learning Python.
- Example (conceptual): Type 2 + 3 in REPL → instantly shows 5.

Basic Syntax in Python

- **Case-sensitive** → Name and name are different variables.
- **Indentation** → Spaces or tabs define code blocks (instead of {} like in other languages).
- **Comments** → Start with # (ignored by the interpreter).

- **Statements**:
    - Single-line → written normally.
    - Multi-line → can be split using \ or parentheses.
- **Keywords** → Reserved words (like if, else, class) cannot be used as variable names.

Key Point: Correct syntax is essential; otherwise, Python raises a **SyntaxError**.

Basic Input and Output

- **Input**: Getting data from the user.
    - Done using the **input()** function.
    - Always takes data as a string, which can later be converted to other types if needed.
- **Output**: Displaying results to the user.
    - Done using the **print()** function.
    - Can display text, numbers, or variables.
    - Supports formatting for neat presentation.
- Importance: Input/Output makes a program interactive, connecting user and computer.