INTRODUCTION TO PROGRAMMING

Programming is the process of designing, writing, testing, and maintaining instructions (code) that a computer follows to solve problems or perform tasks.

Importance of Programming

- Automates tasks
- Solves complex problems efficiently
- Forms the foundation of software, web, mobile, IoT, and AI systems
- Enables innovation and digital transformation

Computer Programs and Languages

A computer program is a set of instructions written in a programming language that tells a computer what to do.

Types of Programming Languages

Low-level languages: Machine language, Assembly
High-level languages: Python, Java, C++, PHP, JavaScript
Scripting languages: Python, JavaScript, PHP
Domain-specific languages: SQL, HTML, MATLAB

Program Development Process

The basic steps in developing a program include:

1. Problem Definition – Understand the problem clearly
2. Algorithm Design – Develop a step-by-step solution
3. Flowchart/Pseudocode – Represent logic visually or textually
4. Coding – Write the program
5. Testing & Debugging – Identify and fix errors
6. Documentation & Maintenance

Algorithms

An algorithm is a finite sequence of clear steps for solving a problem.

Characteristics of a Good Algorithm

- Clear and unambiguous
- Finite (must terminate)
- Input and output clearly defined
- Efficient and effective

Example Algorithm (Add Two Numbers)

Start
Input A and B
Compute Sum = A + B
Output Sum
End

Flowcharts and Pseudocode
Flowcharts
Graphical representation of an algorithm using symbols:

Oval → Start/End
Parallelogram → Input/Output
Rectangle → Process
Diamond → Decision
Pseudocode

English-like representation of program logic.
Example:

```
BEGIN
  INPUT age
  IF age >= 18 THEN
     PRINT "Eligible"
  ELSE
     PRINT "Not Eligible"
  ENDIF
END
```

Variables and Constants

Variables
A variable is a named memory location that stores data whose value can change during program execution.

Example:
age = 20

Constants
A constant is a value that does not change.
Example:
PI = 3.14

Data Types
Data types specify the kind of data a variable can store.

Common Data Types
Integer – whole numbers (10, -5)
Float/Double – decimal numbers (3.14)
Character – single characters ('A')
String – sequence of characters ("Hello")
Boolean – true or false

Operators
Operators perform operations on variables and values.

Types of Operators
Arithmetic: +, -, *, /, %
Relational: >, <, >=, <=, ==, !=
Logical: AND, OR, NOT
Assignment: =, +=, -=
Increment/Decrement: ++, --

Control Structures
Control structures determine the flow of program execution.

a) Sequence
Statements executed one after another.

b) Selection (Decision Making)

if
if–else
switch
Example:

```
if (marks >= 50) {
  System.out.println("Pass");
} else {
  System.out.println("Fail");
}
```

c) Iteration (Loops)
for
while
do–while
Example:

```
for i in range(1, 6):
   print(i)
```

Functions and Modular Programming

Functions
A function is a reusable block of code that performs a specific task.
Example:

```
def add(a, b):
   return a + b
```

Benefits of Modular Programming

- Code reuse
- Easier debugging
- Better organization
- Improved maintainability

Input and Output Operations
Input allows data entry, output displays results.

Example (Python):
name = input("Enter name: ")
print("Hello", name)

Arrays and Data Structures (Basic)

Arrays
Used to store multiple values of the same type.

Example:
int numbers[] = {1, 2, 3, 4};

Common Data Structures

Arrays
An array is a data structure that stores a fixed-size collection of elements of the same data type in contiguous memory locations.

marks = [60, 70, 80, 90]
print(marks[0]) # Output: 60

Lists
A list is a dynamic data structure that stores an ordered collection of elements, which may be of different data types.
Example
items = ["Pen", 10, True]
items.append("Book")

Stacks
A stack is a linear data structure that follows the LIFO (Last In, First Out) principle.

Basic Operations

- **Push** – add element
- **Pop** – remove element

- **Peek** – view top element
- **isEmpty**

```
stack = []
stack.append(10)   # Push
stack.pop()        # Pop
```

Queues
Dictionaries/Maps
A queue is a linear data structure that follows the FIFO (First In, First Out) principle.
Basic Operations
Enqueue – add element
Dequeue – remove element
Front/Peek

Example
```
from collections import deque
queue = deque()
queue.append(1)     # Enqueue
queue.popleft()     # Dequeue
```

Errors and Debugging
Types of Errors
Syntax errors – grammar mistakes
Runtime errors – occur during execution
Logical errors – incorrect output due to wrong logic

Debugging
Process of finding and fixing errors in a program.

Documentation and Coding Standards
Importance
- Improves readability
- Helps team collaboration
- Simplifies maintenance

Examples
Meaningful variable names
Comments
Consistent indentation

Importance of Programming Fundamentals
- Foundation for advanced topics (OOP, Data Structures, Web, AI)
- Enhances problem-solving skills

- Improves logical and analytical thinking
- Enables efficient software development