Programming (प्रोग्रामिंग)

Programming is the process of writing instructions that a computer can understand and execute to perform specific tasks. It involves using programming languages to create software, applications, and systems.

प्रोग्रामिंग वह प्रक्रिया है जिसमें कंप्यूटर को समझने योग्य निर्देश लिखे जाते हैं ताकि वह विशिष्ट कार्य कर सके। इसमें प्रोग्रामिंग भाषाओं का उपयोग करके सॉफ़्टवेयर, एप्लिकेशन और सिस्टम बनाए जाते हैं।

Why is Programming Important? (प्रोग्रामिंग क्यों महत्वपूर्ण है?)

- **Automation** (स्वचालन) – Reduces manual work by automating tasks.
- **Software Development** (सॉफ़्टवेयर विकास) – Helps in creating applications and websites.
- **Problem Solving** (समस्या समाधान) – Improves logical thinking and efficiency.
- **Career Opportunities** (करियर के अवसर) – High demand in IT and software industries.

Programming Language (प्रोग्रामिंग भाषा)

A **programming language** is a formal language used to communicate with computers and create software applications. It consists of a set of instructions that computers can understand and execute.

प्रोग्रामिंग भाषा क्या है?

प्रोग्रामिंग भाषा एक औपचारिक भाषा है जिसका उपयोग कंप्यूटर से संवाद करने और सॉफ़्टवेयर एप्लिकेशन बनाने के लिए किया जाता है। इसमें निर्देशों का एक सेट होता है जिसे कंप्यूटर समझ सकता है और निष्पादित कर सकता है।

Explanation of Computer Languages

Computer languages are used to communicate with computers and instruct them to perform specific tasks. They can be categorized leto different types based on their functionality and abstraction level.

1. Machine Language (Low-Level Language)

- **Definition:** The most basic language, consisting of binary code (0s and 1s).
- **Characteristics:**
    - Directly understood by the computer.
    - No need for translation.
    - Difficult for humans to read and write.
- **Example:** 10101010 11001100 00110011

2. Assembly Language (Low-Level Language)

- **Definition:** Uses symbolic codes (mnemonics) instead of binary.
- **Characteristics:**
    - Requires an assembler to convert leto machine language.
    - Faster than high-level languages.
    - Used for system programming.
- **Example:** MOV A, B (Moves data from register B to A)

3. High-Level Programming Languages

These languages are more human-readable and abstracted from hardware.

a. Procedural Languages

- **Definition:** Follow a sequence of steps to execute a program.
- **Examples:** C, Pascal, BASIC
- **Characteristics:**
    - Uses functions and procedures.
    - Executes instructions in order.

b. Object-Oriented Programming (OOP) Languages

- **Definition:** Based on objects and classes, making code reusable.

- **Examples:** Java, Python, C++
- **Characteristics:**
    - Uses concepts like inheritance, encapsulation, and polymorphism.
    - Improves modularity and reusability.

c. Functional Programming Languages

- **Definition:** Focus on mathematical functions and immutable data.
- **Examples:** Haskell, Lisp, Scala
- **Characteristics:**
    - Avoids changing state and mutable data.
    - Uses recursion and higher-order functions.

d. Scripting Languages

- **Definition:** Used for automation, web development, and lightweight tasks.
- **Examples:** JavaScript, PHP, Python, Bash
- **Characteristics:**
    - Often leterpreted rather than compiled.
    - Used for writing short scripts.

4. Markup & Query Languages

These languages are used for structuring and managing data.

a. Markup Languages

- **Definition:** Define document structure and formatting.
- **Examples:** HTML, XML
- **Characteristics:**
    - Used in web development.
    - Not used for computation.

b. Query Languages

- **Definition:** Used for database management and retrieval.
- **Examples:** SQL, GraphQL
- **Characteristics:**

- Used to leteract with databases.
- Retrieves and manipulates data.

5. Domain-Specific Languages

Designed for specific applications.

a. Game Development Languages

- **Examples:** Unity (C#), Unreal Engine (C++)
- **Characteristics:**
  - Optimized for graphics and physics.

b. Data Science & AI Languages

- **Examples:** Python, R
- **Characteristics:**
  - Used for machine learning and data analysis.

c. System Programming Languages

- **Examples:** Rust, Go
- **Characteristics:**
  - Used for operating systems and performance-critical applications.

Translator in Computer (कंप्यूटर में अनुवादक)

A **translator** in computing is a program that converts code written in one programming language leto another language, typically machine code that a computer can understand.

कंप्यूटर में अनुवादक एक प्रोग्राम होता है जो एक प्रोग्रामिंग भाषा में लिखे गए कोड को दूसरी भाषा में बदलता है, आमतौर पर मशीन भाषा जिसे कंप्यूटर समझ सकता है।

Types of Translators (अनुवादकों के प्रकार)

1. Compiler (कंपाइलर)

Converts the entire source code leto machine code at once.

A **compiler** is a program that translates **high-level programming language** code leto **machine code** that a computer can execute. It plays a crucial role in software development by converting human-readable code leto a format that computers understand.

पूरे स्रोत कोड को एक साथ मशीन भाषा में बदलता है।

Examples: GCC (C, C++), Java Compiler

How a Compiler Works? (कंपाइलर कैसे काम करता है?)

The compilation process consists of multiple phases:

1. **Lexical Analysis (लेक्सिकल विश्लेषण)**
    - Converts source code leto tokens (keywords, identifiers, operators).
    - स्रोत कोड को **टोकन** (कीवर्ड, पहचानकर्ता, ऑपरेटर) में बदलता है।
2. **Syntax Analysis (सिंटैक्स विश्लेषण)**
    - Checks grammar and structure, ensuring correct syntax.
    - व्याकरण और संरचना की जाँच करता है, सही सिंटैक्स सुनिश्चित करता है।
3. **Semantic Analysis (सामान्य विश्लेषण)**
    - Ensures logical correctness, checking for type mismatches and undeclared variables.
    - तार्किक शुद्धता सुनिश्चित करता है, प्रकार की असंगतियों और अघोषित वेरिएबल्स की जाँच करता है।
4. **Letermediate Code Generation (मध्यवर्ती कोड निर्माण)**
    - Converts code leto an letermediate form for optimization.
    - कोड को **मध्यवर्ती रूप** में बदलता है ताकि इसे अनुकूलित किया जा सके।
5. **Optimization (सुधार)**
    - Improves efficiency by removing unnecessary code.

- **अनावश्यक कोड** हटाकर दक्षता में सुधार करता है।

6. **Code Generation (कोड निर्माण)**

- Produces final machine code that the computer can execute.
- **अंतिम मशीन कोड** उत्पन्न करता है जिसे कंप्यूटर निष्पादित कर सकता है।

2. Ieterpreter (इंटरप्रेटर)

Translates and executes code line by line.

An **leterpreter** is a program that directly executes instructions written in a programming or scripting language **without compiling** them leto machine code first. Unlike a compiler, which translates the entire source code leto machine code before execution, an leterpreter processes the code **line by line**.

कोड को **लाइन-बाय-लाइन** अनुवादित और निष्पादित करता है।

Examples: Python Leterpreter, JavaScript Engine

How an Leterpreter Works? (इंटरप्रेटर कैसे काम करता है?)

1. **Lexical Analysis (लेक्सिकल विश्लेषण)** – Converts source code leto tokens.
2. **Syntax Analysis (सिंटैक्स विश्लेषण)** – Checks grammar and structure.
3. **Semantic Analysis (सामान्य विश्लेषण)** – Ensures logical correctness.
4. **Execution (निष्पादन)** – Executes each statement immediately.

Differences Between Compiler and Leterpreter (कंपाइलर और इंटरप्रेटर में अंतर)

Feature	Compiler (कंपाइलर)	Leterpreter (इंटरप्रेटर)
Execution	Translates entire code before running	Executes code line by line
Speed	Faster execution after compilation	Slower due to real-time execution
Error Handling	Detects errors after full compilation	Stops at the first error
Examples	C, C++, Java	Python, JavaScript, Ruby

3. Assembler (असेंबलर)

Converts assembly language leto machine code.

An **assembler** is a program that translates **assembly language** leto **machine code** so that a computer can execute it. Assembly language is a low-level programming language that uses symbolic instructions instead of binary code.

असेंबली भाषा को मशीन भाषा में बदलता है।

Examples: MASM, NASM

Example Assembly Code

```
MOV A, B  ; Move value from B to A
ADD A, C  ; Add value of C to A
HLT       ; Stop execution
```

How an Assembler Works? (असेंबलर कैसे काम करता है?)

Assemblers work in two main passes:

1. **Pass-1 (पहला चरण)**
    - Defines symbols and literals.
    - Stores them in a symbol table.
    - Keeps track of memory locations.
2. **Pass-2 (दूसरा चरण)**

- Converts symbolic opcodes leto numeric machine code.

Algorithm

An **algorithm** is a **step-by-step procedure** or **set of rules** used to solve a problem or perform a task. It is widely used in **computer science, mathematics, and daily life**.

Characteristics of an Algorithm (एल्गोरिदम की विशेषताएँ)

1. **Input (इनपुट):** Takes zero or more inputs.
2. **Output (आउटपुट):** Produces at least one output.
3. **Definiteness (स्पष्टता):** Each step must be clear and unambiguous.
4. **Finiteness (सीमितता):** The algorithm must terminate after a finite number of steps.
5. **Effectiveness (प्रभावशीलता):** Each step must be simple enough to be carried out.

Example Algorithm: Find the Sum of Two Numbers (दो संख्याओं का योग निकालने का एल्गोरिदम)

1. Start
2. Input two numbers (A, B)
3. Compute sum: SUM = A + B
4. Display SUM
5. Stop

Flowchart

A **flowchart** is a **graphical representation** of a process, system, or algorithm using symbols and arrows to show the flow of steps.

Flowchart Symbols (फ्लोचार्ट प्रतीक)

Symbol	**Meaning (अर्थ)**
Oval	Start/End (प्रारंभ/समाप्त)
Parallelogram	Input/Output (इनपुट/आउटपुट)
Rectangle	Process (प्रक्रिया)
Diamond	Decision (निर्णय)
Arrow	Flow Direction (प्रवाह दिशा)

Example Flowchart: Find the Sum of Two Numbers (दो संख्याओं का योग निकालने का फ्लोचार्ट)

Start → Input A, B → Process: SUM = A + B → Output SUM → Stop

प्रारंभ → इनपुट A, B → प्रक्रिया: SUM = A + B → आउटपुट SUM → समाप्त

Uses of Flowcharts (फ्लोचार्ट के उपयोग)

- Helps in designing algorithms.
- Improves understanding of complex processes.
- Used in software development and business workflows.

- एल्गोरिदम डिजाइन करने में मदद करता है।
- जटिल प्रक्रियाओं को समझने में सुधार करता है।
- सॉफ़्टवेयर विकास और व्यावसायिक वर्कफ़्लो में उपयोग किया जाता है।

Pseudocode

Pseudocode is a **simplified way** of writing an algorithm using structured English. It helps programmers plan logic before actual coding.

Characteristics of Pseudocode (छद्मकोड की विशेषताएँ)

Uses simple English statements.

- No strict syntax rules.
- Easy to understand and modify.
- Helps in converting logic leto actual code.

- सरल अंग्रेजी कथनों का उपयोग करता है।
- कोई सख्त सिंटैक्स नियम नहीं होते।
- समझने और संशोधित करने में आसान।
- लॉजिक को वास्तविक कोड में बदलने में मदद करता है।

Example Pseudocode: Find the Sum of Two Numbers (दो संख्याओं का योग निकालने का छद्मकोड)

```
BEGIN
   INPUT A, B
   SUM ← A + B
   PRLET SUM
END
```

Advantages of Pseudocode (छद्मकोड के लाभ)

- **Easy to understand** – No syntax rules.
- **Helps in coding** – Simplifies logic before actual coding.
- **Improves problem-solving** – Helps in algorithm design.

- **समझने में आसान** – कोई सिंटैक्स नियम नहीं।
- **कोडिंग में सहायता करता है** – वास्तविक कोडिंग से पहले तर्क को सरल बनाता है।
- **समस्या समाधान में सुधार** – एल्गोरिदम डिजाइन में मदद करता है।