

Introduction to Programming

Introduction to Programming – Study Notes

1. What Is Programming?

Aspect	Description
Definition	The process of designing, writing, testing, and maintaining code that tells a computer how to perform tasks.
Goal	Transform a problem or requirement into a set of precise, executable instructions.
Key Elements	Algorithms, data structures, syntax, semantics, and debugging.

2. Core Concepts

Concept	Explanation	Example
Algorithm	Step-by-step procedure for solving a problem.	Sorting a list of numbers.
Variable	Symbolic name that stores a value which can change.	<code>int age = 25;</code>
Data Type	Classification of data that determines the kind of values a variable can hold.	<code>int, float, char, bool.</code>
Control Flow	Directs the order in which statements are executed.	<code>if/else, for, while.</code>
Function / Procedure	Reusable block of code that performs a specific task.	<code>def add(a, b): return a + b</code>
Input / Output (I/O)	Mechanisms for receiving data from the user or external sources and presenting results.	<code>print(), scanf().</code>
Error / Exception	Unexpected conditions that disrupt normal execution.	Division by zero, file not found.
Debugging	Systematic process of locating and fixing defects.	Using breakpoints, print statements, or debuggers.

Introduction to Programming

3. Programming Languages – A Quick Overview

Language	Paradigm(s)	Typical Use Cases	Notable Feature
Python	Imperative, Object? Oriented, Functional	Data science, web, scripting	Readable syntax, extensive libraries
Java	Object?Oriented	Enterprise apps, Android	Strong typing, JVM portability
C	Procedural	Systems programming, embedded	Low?level memory control
C++	Object?Oriented, Generic	Game engines, high? performance apps	RAII, templates
JavaScript	Event?driven, Functional	Web front?end, Node.js back?end	Runs in browsers, async programming
Ruby	Object?Oriented	Web development (Rails)	Elegant DSLs
SQL	Declarative	Database querying	Set?based operations

Choosing a language depends on the problem domain, performance needs, and ecosystem support.

4. The Programming Workflow

1. Understand the Problem

- Clarify requirements.
- Identify inputs, expected outputs, and constraints.

2. Design an Algorithm

- Pseudocode or flowchart.
- Choose appropriate data structures.

3. Write Code

- Follow language syntax.

Introduction to Programming

- Keep code modular (functions, classes).

4. Compile / Interpret

- Translate source code into executable form (if required).

5. Test

- Unit tests for individual components.
- Integration tests for combined parts.

6. Debug

- Locate bugs using tools or print statements.
- Fix and retest.

7. Document & Refactor

- Add comments, README, and usage examples.
- Improve readability and performance.

8. Maintain

- Update for new requirements or bug reports.

5. Basic Syntax