

Introduction to Programming

Introduction to Programming – Quick Reference Notes

1. What Is Programming?

Aspect	Description
Definition	The process of creating a set of instructions (code) that a computer can execute to perform a task.
Goal	Transform a problem or requirement into an automated solution.
Key Elements	Algorithms, data structures, syntax, semantics, and debugging.

2. Core Concepts

Concept	Explanation	Example
Algorithm	Step-by-step procedure to solve a problem.	Sorting a list of numbers.
Variable	Named storage location for data that can change.	<code>int age = 25;</code>
Data Types	Define the kind of data a variable can hold.	<code>int, float, char, bool, String.</code>
Control Structures	Direct the flow of execution.	<code>if/else, for, while, switch.</code>
Function / Procedure	Reusable block of code that performs a specific task.	<code>int add(int a, int b) { return a + b; }</code>
Input / Output (I/O)	Interaction with the user or external systems.	<code>cin >> name; / cout << "Hello";</code>
Error / Exception Handling	Manage unexpected conditions gracefully.	<code>try { ... } catch (Exception e) { ... }</code>
Comment	Non-executable text for documentation.	<code>// Calculate area</code>

Introduction to Programming

3. Programming Languages – Overview

Language	Paradigm(s)	Typical Use Cases	Learning Curve
Python	Imperative, Object?Oriented, Functional	Scripting, data science, web, automation	Low
Java	Object?Oriented	Enterprise apps, Android, large systems	Medium
C	Procedural	Systems programming, embedded	Medium?High
C++	Procedural, Object?Oriented, Generic	Game engines, performance? critical apps	High
JavaScript	Event?driven, Functional	Web front?end, Node.js back?end	Low?Medium
Ruby	Object?Oriented, Dynamic	Web (Rails), rapid prototyping	Low
Go	Procedural, Concurrent	Cloud services, networking	Medium

Tip: Choose a language that matches your project goals and community support.

4. The Typical Development Workflow

1. Problem Analysis

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

2. Algorithm Design

- Pseudocode or flowcharts.
- Choose appropriate data structures.

3. Coding

- Write source files following language syntax.
- Use meaningful names and consistent style.

Introduction to Programming

4. Compilation / Interpretation

- Translate code to executable form (compile) **or** run directly (interpret).

5. Testing

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

6. Debugging

- Locate and fix logical or runtime errors.
- Tools: breakpoints, watch variables, stack traces.

7. Documentation

- Inline comments, README, API docs.

8. Deployment

- Package and deliver the program to its target environment.

9. Maintenance

- Update, refactor, and improve over time.

5. Basic Syntax (Python Example)

1. Variables