Introduction to Programming

Md. Aminul Islam Shazid

Outline

1. What is Programming?
2. Types of Programming Languages
3. Development Tools

What is Programming?

What is Programming?

- Process of giving instructions to a computer
- Computers follow step-by-step instructions precisely
- A program is a sequence of instructions that accomplishes a task
- Algorithms (logic) vs Programs (implemented in code)

History & Evolution of Programming Languages

- 1940s: Machine code (binary)
- 1950s: Assembly language (mnemonics)
- 1960s–70s: Structured languages (C, Pascal)
- 1990s: Object-oriented (Java, C++)
- 2000s–Now: Multi-paradigm, scripting (Python, JavaScript)

Flow of Program Development

1. Define the problem
2. Design algorithm
3. Write pseudocode or flowchart
4. Translate into source code
5. Compile and run
6. Debug and test
7. Update and maintain

Program Structure

- Typical structure of a program:
  - Input (data from user or file)
  - Processing (calculations, logic)
  - Output (display results)
- Uses variables, control flow (if/loops), and functions

Types of Programming Languages

Levels of Programming Languages

- Machine code (binary: 0s and 1s)
  - The lowest-level representation of a program
  - Directly executed by the CPU
- Assembly language (x86 assembly, ARM assembly)
  - Low-level language using mnemonics instead of binary
  - Easier for humans to read than machine code
- Higher-level languages (C, Python, Java etc.)
  - Abstract away hardware details
  - Easier to write, read, and maintain
- Domain-specific languages (DSL)
  - Designed for a particular purpose or domain
  - Examples: SQL for databases, HTML for web, R for data science

Pseudocode

- Informal description of a program's logic.
- Bridges the gap between human thought and actual code.
- Example:
  - Start
  - Input: integer a, integer b
  - Let c = a + b
  - Print c
  - End

Compiled vs Interpreted

- **Compiled**: C, C++, Rust etc.
  - Source code compiled into machine code by a compiler producing an execuatble file
  - Faster execution
  - Platform-specific
- **Interpreted:** Python, Javascript etc.
  - Code executed line by line by an interpreter
  - No separate compilation step
  - Easier to test and debug, but slower

Programming Paradigms

- **Procedural:** Step-by-step instructions (C)
- **Object-Oriented:** Organize into classes/objects (Java, Python)
- **Functional:** Emphasis on *pure* functions, immutability (Haskell, Lisp)

Open Source vs Proprietary

- **Open Source:** Free to use, modify, share (Python, GCC)
- **Proprietary:** Owned, restricted licenses (MATLAB)
- Ecosystem and libraries often drive language adoption

Performance vs Productivity

- **Performance:** How fast a program runs (C, C++)
- **Productivity:** How quickly developers can write programs (Python, Ruby)
- Trade-off: High performance vs faster development

Portable Languages

- Code can run on multiple platforms with little/no modification
- Example: Java – Write Once, Run Anywhere
- Achieved via virtual machines or interpreters

Why So Many Languages?

- Different goals:
  - Performance: C, C++
  - Safety and reliability: Go, Rust
  - Ease of use and productivity: Python
  - Domain focus: SQL, R
- No single language is best for all problems.

Development Tools

Development Tools: Editors & IDEs

- **Text Editors:** lightweight coding (Notepad++, Sublime)
- **IDEs:** full-featured, include debugging, auto-completion (VS Code, PyCharm)

Development Tools: Debugging, Profiling, VCS

- **Debugger:** Step through code, inspect variables
- **Profiler:** Find performance bottlenecks
- **Version Control System, VCS:** Track changes, collaborate (Git)

Terminal and Command Line Interface (CLI)

- **Terminal:** Text-based interface to interact with the OS
- **CLI:** Execute commands directly by typing them
- Used for compiling, running, and managing projects
- Common tools: `gcc`, `gdb`, `python`, `git`