

# Introduction to IT Systems for Beginners

Maciej Gowin @ [CoderBrother](#)

All rights reserved

# Programming Language

A Programming Language is a set of rules, syntax, and keywords that allows humans to write instructions that computers can execute. It acts as a bridge between human ideas and machine operations.

## Purpose

- Enables programmers to create software, websites, apps, and control hardware by giving specific instructions.
- Programming languages allow us to create technology and control how computers work, forming the basis of all software and applications.

# Types of Programming Languages

- **High-Level Languages**

- Examples: Python, Java, C++.
- Human-Readable syntax, closer to natural language.
- Easier to write, but needs to be compiled or interpreted into machine code.

- **Low-Level Languages**

- Examples: Assembly, Machine Code.
- Directly related to the hardware, making them faster but harder to read and write.

# Compilation

Compilation is the process of **transforming source code** written in a high-level programming language (e.g., C++, Java) into **machine code** (binary) that a computer's CPU can execute.

Enables the execution of programs by converting human-readable code into a format that can be directly understood and executed by a computer.

Compilation helps **catch errors early** (during development) and optimizes code for better performance, making programs faster and more efficient.

# How Compilation Works

## 1. Source Code

The original code written by the programmer.

## 2. Compiler

A special program that analyzes the source code and translates it into machine code.

## 3. Executable File

The output of the compilation process, which can be run on the computer.

# Development example

## Source code: main.c

```
#include <stdio.h>
int main() {
    printf("Hello, world!\n");
    return 0;
}
```

# Development example

## Compilation

```
gcc main.c
```

## Execution

```
gowinm:~> ./a.out  
Hello, World!
```

# Stages of Compilation

- **Lexical Analysis:** Breaking the code into tokens (keywords, operators, identifiers).
- **Syntax Analysis:** Checking the code structure against the language rules.
- **Semantic Analysis:** Ensuring the code makes logical sense.
- **Code Generation:** Producing the machine code or intermediate code.
- **Optimization:** Improving the code for better performance.

# Interpreter

An Interpreter is a program that **executes high-level programming code** directly, translating it into **machine code line-by-line** or statement-by-statement, without producing a separate executable file.

Enables developers to run programs written in high-level languages (e.g., Python, Ruby, JavaScript) interactively and quickly, making it easier to test and debug code.

# How Interpreters Work

## 1. Source Code

The original code written by the programmer.

## 2. Interpreter

Reads and executes the source code line-by-line, translating it into machine code on-the-fly.

## 3. Output

The results of the executed code are immediately available, allowing for real-time feedback.

# Development example

Source code: main.php

```
<?php
    echo "Hello, World!\n";
?>
```

# Development example

## Interpretation and execution

```
gowinm:~> php ./main.php  
Hello, World!
```

# Interpreter Pros and Cons

## Advantages

- **Ease of Use:** Simple to run and modify code without needing a separate compilation step.
- **Immediate Execution:** Changes can be tested instantly, enhancing the development process.

## Disadvantages

- **Performance:** Generally slower than compiled code, as translation occurs during execution.
- **Error Detection:** Errors may only appear when the specific line of code is executed, which can make debugging more challenging.

# Examples of Programming Languages

- Compiled: Java, C, C++, Go
- Interpreted: Python, PHP, JavaScript, Ruby

# Compiled vs. Interpreted Summary

- Compiled
  - Source Code: `main.c`
  - Compilation: `gcc main.c`
  - Execution: `./a.out`
- Interpreted
  - Source Code: `main.php`
  - Interpretation and execution: `php main.php`

# Frontend vs. Backend Programming

Aspect	Frontend	Backend
<b>Definition</b>	The part of a website or application that users interact with directly.	The server-side logic, database interactions, and application functionality that users don't see.
<b>Languages</b>	HTML, CSS, JavaScript, TypeScript	Python, Java, Ruby, PHP, Node.js
<b>Responsibilities</b>	Designing user interfaces, ensuring responsiveness, and improving user experience.	Managing databases, server logic, user authentication, and application performance.
<b>Tools</b>	Web browsers, design tools	Server management, APIs, database management systems
<b>Interaction</b>	Communicates with the backend via APIs to fetch and send data.	Handles requests from the frontend, processes data, and sends responses.

# Programming Language Framework

A **framework** is a collection of pre-written code that provides a foundation to build applications. It offers a structure and set of guidelines for developers to streamline the development process.

# Framework Purpose

- **Save Time:** Reduce repetitive coding by providing reusable components.
- **Enhance Productivity:** Enable faster development with built-in functions and libraries.
- **Maintain Consistency:** Ensure that code follows best practices and standard patterns.

# Benefits of Using Frameworks

- **Community Support:** Many frameworks have large communities for troubleshooting and collaboration.
- **Security Features:** Built-in security features to help protect applications.
- **Scalability:** Frameworks are often designed to handle increased loads and user demands easily.

# Types of Frameworks

- **Web Frameworks:** Designed for building web applications (e.g., Django for Python, Ruby on Rails for Ruby).
- **Mobile Frameworks:** Used for developing mobile applications (e.g., React Native, Flutter).
- **Desktop Frameworks:** Built for desktop application development (e.g., Electron, .NET).

# Popular Frameworks by Language

Programming Language	Popular Frameworks
JavaScript	React, Angular, Vue.js
Python	Django, Flask
Python	Ruby on Rails
Java	Spring, Hibernate
PHP	Laravel, Symfony
C#	ASP.NET, Blazor
Go	Gin, Revel

# Open Source

**Open Source** refers to software whose source code is made **publicly available** for anyone to view, use, modify, and distribute. This fosters collaboration and transparency in software development.

Open source drives innovation, encourages collaboration, and allows users to maintain control over their software solutions.

# Open Source Key Characteristics

- **Accessibility:** Anyone can access the source code and contribute to its development.
- **Community Collaboration:** Encourages contributions from developers worldwide, enhancing innovation and improvements.
- **Licensing:** Open-source software is typically released under licenses (e.g., MIT, GNU GPL) that dictate how it can be used and shared.

# Open Source Advantages

- **Cost-Effective:** Often free to use, reducing software costs for individuals and organizations.
- **Flexibility:** Users can customize software to fit their specific needs.
- **Security:** Transparency allows for more scrutiny, potentially leading to faster identification and resolution of vulnerabilities.

# Examples of Open Source Software

Group	Solutions
<b>Frameworks</b>	Spring, Angular, Ruby on Rails, Django, Laravel
<b>Operating Systems</b>	Linux, FreeBSD
<b>Development Tools</b>	Git, VS Code
<b>Content Management Systems</b>	WordPress, Joomla
<b>Databases</b>	MySQL, PostgreSQL, Redis