Turiba University

LAUNCE MADNAWAT

# HISTORY OF PROGRAMMING

PROFESSIONAL BACHELOR THESIS

computer systems

| | |
|---|---|
| Author: | LAUNCE MADNAWAT |
| *Thesis* Advisor: | (Jānis  Pekša  Dean of IT |

Riga, 2025

# HISTORY OF PROGRAMMING

## A Story of Innovation: How We Taught Computers to Think

The history of programming isn't just a list of languages; it's a story of humanity constantly trying to simplify the impossible: talking to a machine.

### The Mechanical Whisper (Before Computers)

Our story begins not with electronics, but with gears. In the 1840s, long before the first actual computer, a brilliant mind named Ada Lovelace imagined a massive mechanical calculator called the Analytical Engine. She wrote the very first *algorithm*—a set of instructions—for this machine to crunch numbers. Because she essentially wrote the first "program," she is widely considered the world's first computer programmer.

## Generation 1 & 2: The Language Barrier

When the first electronic computers finally arrived in the 1940s, talking to them was a nightmare.

- 1st Generation (Machine Code)**:** You had to speak the computer's native tongue: binary code (pure 0s and 1s). Imagine writing an entire novel using only dots and dashes—it was incredibly slow, tedious, and error-prone.
- 2nd Generation (Assembly): Things got slightly better with Assembly Language in the early 1950s. Instead of 0s and 1s, programmers used simple English abbreviations . This was still difficult, but a piece of software called an *assembler* could finally translate the abbreviations into machine code, offering the first taste of simplification.

### The Great Leap: High-Level Languages (1950s - 1970s)

The real breakthrough came when programmers realized they needed languages closer to human speech and mathematics. This was the "High-Level Revolution."

- FORTRAN (1957): Scientists and mathematicians rejoiced! This was the first widely used high-level language, designed specifically for crunching formulas. It proved that a human-readable language could be fast and powerful.
- COBOL (1959): Thanks in part to the visionary Dr. Grace Murray Hopper, COBOL was created for business and government, designed to handle large amounts of data and be easy to read, almost like structured English.
- BASIC (1964): The goal was to make computing accessible. BASIC was created for beginners and students, helping to foster a whole new generation of programmers outside of big research labs.
- C (1972): Developed by Dennis Ritchie at Bell Labs, C was a game-changer. It was powerful enough to write operating systems (like Unix), yet high-level enough for humans. It became the foundation for almost every major language that followed.

## The Age of Objects and the Internet (1980s - 1990s)

Two major shifts defined this era: Object-Oriented Programming (OOP) and the rise of the Web.

- C++ (1983): Taking C's power, C++ added the revolutionary concept of OOP. This allowed developers to organize code into logical "objects" that mirrored the real world, making complex software much easier to manage.
- The 1990s Explosion: The internet arrived, demanding new, fast, and flexible languages:
    - Python (1991): Created by Guido van Rossum, Python focused on extreme readability. It’s now the go-to language for data science, AI, and web backend because it lets you write powerful code very quickly.

    - Java (1995): Sun Microsystems promised "Write once, run anywhere." Java became the backbone of enterprise systems and later, Android mobile apps.
    - JavaScript (1995): Initially created to make web pages *do stuff* in the browser (like having a button react to a click), JavaScript unexpectedly became an absolutely essential technology for the entire modern web.

## The Modern Era: Safety, Speed, and Scale

Today, the challenges are building massive systems, running things in the cloud, and ensuring security. Modern languages are built with these concerns in mind.

- Go (Golang, 2009): Google developed Go to handle huge, distributed systems efficiently, focusing on simplicity and making it easy to manage concurrent (simultaneous) tasks.
- Swift (2014): Apple created Swift as a safer, faster replacement for its older iOS language, specifically designed for a smooth mobile development experience.
- Rust (2015): The ultimate challenger to C and C++, Rust’s core feature is guaranteed memory safety. It lets developers write extremely fast, low-level code without the memory bugs that plague older systems.