Lecture One: The History of Programming – From Counting to Artificial Intelligence

In a world where digital development accelerates, programming has become the language of our era and a central tool for understanding and shaping reality. Yet, to grasp programming as we know it today, we must return to its earliest roots—when humans counted on their fingers and devised simple tools to organize calculation and logic.

This historical journey is not merely a chronological narrative; it is an entry point to understanding how ideas evolved, transforming from philosophical and mathematical concepts into digital instructions executed by machines.

In this lecture, we begin by defining programming from both functional and conceptual perspectives, then pose a fundamental question: Why start with history? Because understanding the historical context enables us to perceive the major transformations—from tools of counting and logic in ancient times, to the first seeds of programming in the 19th century, to the birth of programming languages in the 20th century, culminating in artificial intelligence that reshapes our relationship with machines and information.

At the conclusion of this journey, we pause at a pivotal and unexpected point: the entry of programming into the social sciences and humanities. How and why did this happen? This is what we will begin to explore, to understand how programming became a tool for studying humanity, analyzing society, and interpreting cultural and political phenomena—in an unconventional encounter between technology and thought.

1. What is Programming?

The definitions of programming have varied according to the intellectual approaches of computer science pioneers, each offering a perspective shaped by their theoretical or practical background.

- **Alan Turing**, regarded as one of the founders of theoretical computing, did not provide a direct definition of programming. Instead, he laid the foundation for understanding it as a mental process that can be transformed into mechanical steps, through the Turing Machine, which executes symbolic instructions according to a defined logic.

- **Donald Knuth**, in his seminal work *The Art of Computer Programming*, considered programming an art of organizing and defining the operations performed by computers to solve problems, emphasizing its creative dimension.
- **Niklaus Wirth**, creator of the Pascal language, defined programming as the process of converting ideas into precise, executable instructions, stressing the importance of logical structure and clarity.
- **Dennis Ritchie**, creator of the C language, did not provide a philosophical definition, but his work demonstrates programming as a tool for building operating systems and scalable applications, making it a practical means of organizing computational functions.
- **Guido van Rossum**, creator of Python, viewed programming as a way to express logic in a simple and clear manner, reflecting the trend toward making programming accessible even to non-specialists.

From these diverse perspectives, a reference definition of programming in the academic context can be formulated:

"Programming is the process of designing and implementing a set of precise logical instructions used to direct the computer to perform specific tasks. It represents a means of translating ideas into executable operations, combining mathematical methodology with interpretive capacity."

So, Programming is not merely a set of technical instructions; it is a language of thought, a tool for organizing information, analyzing phenomena, and making decisions. In social and human contexts, programming can be used to understand individual behavior, analyze political discourse, trace social networks, or even model economic and administrative phenomena.

Programming closely resembles the construction of a conceptual model: we define variables, set conditions, execute operations, and interpret results. For this reason, learning programming enhances skills in logical reasoning, systematic analysis, and the ability to handle complex data.

To understand how programming has reached this vital role, it is useful to embark on a historical journey that shows how it evolved—from simple counting tools to artificial intelligence.

2. Introductory Approach: Why Start with History?

Before learning how to program, it is useful to understand how programming emerged and the intellectual and technical contexts that contributed to its development. Programming is not merely writing instructions for a computer; it is an extension of a long history of logical reasoning, symbolic organization, and the pursuit of understanding and improving the world. In the social sciences and humanities, this historical perspective helps us realize that programming is not foreign to our fields, but rather a powerful tool for analyzing and interpreting phenomena.

3. Before Programming: Tools of Counting and Logic

The idea of programming began indirectly when humans started using counting and calculation tools to organize daily life. In Sumerian civilization (Mesopotamia, 3500 BCE), the earliest known counting systems appeared. The Sumerians used a sexagesimal system (based on the number 60) to record trade transactions, document taxes, and organize the agricultural calendar. They inscribed numeric symbols on clay tablets—among the earliest forms of systematic encoding in history—functionally similar to the concept of "structured data" in modern programming.

In philosophy, **Aristotle** contributed to the foundation of formal logic by developing the syllogism, which expresses relationships between premises and conclusions in a systematic way. This conditional reasoning is the basis of programming's *if…then* statements, where decisions depend on certain conditions. Later, **Avicenna (Ibn Sina)** deepened and refined this logic in works such as *The Book of Healing* and *The Book of Salvation*, addressing symbolic logic and rational inference, distinguishing between types of propositions and logical connectors. These contributions laid intellectual groundwork for the concept of "algorithms" before the term emerged in mathematics.

In the 17th century, **Gottfried Leibniz** (1646–1716) proposed the idea of a universal symbolic language reducible to simple symbols representing all ideas, based on logical principles and binary arithmetic (0 and 1). This binary system became the cornerstone of modern computer design, making Leibniz's ideas remarkably close to programming as we know it today, where instructions are translated into sequences of 0s and 1s executed by processors.

4. The 19th Century: The First Seeds of Programming

In 1843, **Ada Lovelace** wrote what is considered the first algorithm designed for execution on a mechanical device known as the *Analytical Engine*, conceived by mathematician **Charles Babbage**.

This machine was envisioned as a revolutionary mechanical computer capable of performing multiple calculations using punched cards to store instructions and data—a concept that preceded electronic computers by a century.

Although the Analytical Engine was never built during Babbage's lifetime, Lovelace's contribution was unprecedented. She wrote detailed notes on how the machine could execute a sequence of operations, formulating what we now call an "algorithm"—an ordered set of instructions to solve a problem. Importantly, she distinguished between data and instructions, and envisioned the machine processing symbols and text, not just numbers—a precursor to the idea of "general-purpose programming." Lovelace foresaw computers composing music, analyzing language, and generating images—visions realized only in the 20th century.

5. The 20th Century: The Birth of Programming Languages

During World War II, the urgent need to process massive amounts of data quickly—especially for cryptography, artillery calculations, and ballistic trajectories—led to the development of the first electronic digital computer, **ENIAC** (1945, University of Pennsylvania). ENIAC could perform 5,000 additions per second and was the first machine used for military computational tasks, though it was enormous and consumed vast amounts of energy.

At the same time, **Alan Turing** emerged as a foundational figure in the concept of the programmable machine. Working at Britain's codebreaking center, he contributed to the development of the *Bombe* machine to break Germany's Enigma code. More importantly, he formulated the theoretical model of the *Turing Machine*, a mathematical abstraction defining what computers can compute. This model remains central to theoretical computer science and understanding the limits of programming and machine cognition.

With computers entering civilian institutions after the war, the need for programming languages arose. In the 1950s and 1960s, the first high-level languages appeared:

- **Fortran (1957)**: Developed at IBM, aimed at scientists and engineers, simplifying mathematical programming.
- **COBOL (1959)**: Designed for administrative institutions like banks and governments, enabling business process automation.

- **LISP (1958)**: Created for artificial intelligence, used in symbolic data processing such as language and logic.

These languages enabled automation, data analysis, and the development of information systems in administration, planning, and research, paving the way for programming's spread into social and human sciences.

6. The Digital Transformation: From Machine to Human

With the rise of personal computers in the 1980s and the spread of the internet in the 1990s, programming entered a new phase. It was no longer confined to laboratories or military institutions but became accessible to the general public, used in education, media, administration, and the arts. This shift was accompanied by more flexible and user-friendly languages:

- **C**: A powerful language from the 1970s, standard for operating systems and foundational software, still used for high-performance applications.
- **Java**: Introduced in the mid-1990s, known for its cross-platform capability ("write once, run anywhere"), ideal for internet applications, mobile devices, and administrative systems.
- **Python**: Emerging in the early 1990s, widely adopted in recent decades for its simplicity, readability, and strong support for data analysis, AI, and social science applications.

Programming became a versatile tool employed by journalists (data analysis, interactive tools), social researchers (survey analysis, social modeling), administrators (automated reporting, resource management), and artists/designers (digital and generative art).

7. Programming and the Social Sciences: An Unexpected Encounter

In recent decades, the social sciences and humanities have undergone a profound transformation in research tools, with programming entering fields traditionally reliant on descriptive and qualitative methods. This shift reflects not only technical progress but also a reconfiguration of how we understand humans and societies—through quantitative tools capable of processing big data, extracting patterns, and building interpretive models.

Programming is no longer exclusive to engineers; it has become part of the intellectual toolkit of social and human researchers. Today, programming is used to analyze surveys, trace social networks, study political discourse, and automate administrative services. For example, languages like Python and R can analyze thousands of survey responses to extract precise indicators of public trends. Natural language processing techniques allow analysis of political and media discourse, tracking conceptual shifts over time. In administration, programming enables efficient resource management, scheduling, and automation, enhancing institutional transparency and efficiency.

Programming today is not just a technical skill but a **cognitive tool for understanding social reality**. By analyzing behavioral data, researchers can build models that explain individual and collective decisions and help predict future transformations. Programming bridges quantitative and qualitative approaches, opening new horizons for interactive, experimental, and interdisciplinary research—making it an essential knowledge tool in the age of data.