History of programming languages

A programming language is a formal language designed for writing computer programs.
The programming language defines a set of lexical, syntactic and semantic rules that determine the appearance of the program and the actions that the performer (usually a computer) will perform under its control.
Initially, programming had an extremely primitive look and practically had no differences from the ordered binary code with a formalized approach. In fact, at the origin of the sphere, there were few differences between a programming language and computer code. There were no obvious and natural conveniences for the programmer, he had to have knowledge of numerical codes for each machine command. Even the allocation of memory for executing commands fell on the specialist.
To simplify the handling of computers, people began to actively develop languages, Assembler was one of the first. Symbolic names were used to display variables. Instead of numerical operations, it is enough for a person to know mnemonic names, their memorization was much easier. Already at this stage, programming languages have become more close to a human-understandable language
To simplify the handling of computers, people began to actively develop languages, Assembler was one of the first. Symbolic names were used to display variables. Instead of numerical operations, it is enough for a person to know mnemonic names, their memorization was much easier. Already at this stage, programming languages have become more close to a human-understandable language

first-generation programming language, 1GL

- The first generation includes machine languages — programming languages at the level of processor commands of a specific machine. No translator was used for programming, program commands were entered directly in the machine code by switches on the front panel of the machine. Such languages were good for a detailed understanding of the functioning of a particular machine, but difficult to study and solve applied problems. (Fortran)

second-generation programming language, 2GL

- Second-generation languages (2 GL) were created in order to facilitate the hard work of programming, moving in language expressions from low-level machine concepts closer to how a programmer usually thinks. These languages appeared in the 1950s, in particular, languages such as Fortran and Algol. The most important problem faced by developers of second-generation languages was the task of convincing customers that the code created by the compiler performs well enough to justify abandoning assembly programming. Skepticism about the possibility of creating effective programs using automatic compilers was quite common, so the developers of such systems had to demonstrate that they could indeed generate almost as effective code as with manual coding, and for almost any initial task. (Assembler)

third-generation programming language, 3GL

- The third generation (3GL) originally meant all languages at a higher level than assembler. The main distinguishing feature of third-generation languages has become hardware independence, that is, the expression of the algorithm in a form that does not depend on the specific characteristics of the machine on which it will be executed. Code written in a thirdgeneration language is translated either directly into machine commands or into assembler code before execution and then assembled. When compiling, unlike previous generations, there is no longer a one-to-one correspondence between the program instructions and the generated code.
- Program interpretation has become widely used — at the same time, program instructions are not converted into machine code, but are executed directly one after the other. Independence from hardware is achieved by using an interpreter compiled for a specific hardware platform. (Fortran 2, Algol 60, Cobol, Pascal, Basic)