Generations of programming language

[Programming languages](https://www.includehelp.com/basics/computer-programming-languages.aspx) have been developed over the year in a phased manner. Each phase of developed has made the programming language more user-friendly, easier to use and more powerful. Each phase of improved made in the development of the programming languages can be referred to as a generation. The programming language in terms of their performance reliability and robustness can be grouped into five **different generations**,

1. First generation languages (1GL)
2. Second generation languages (2GL)
3. Third generation languages (3GL)
4. Fourth generation languages (4GL)
5. Fifth generation languages (5GL)

1. First Generation Language (Machine language)

The first generation programming language is also called low-level programming language because they were used to program the computer system at a very low level of abstraction. i.e. at the machine level. The machine language also referred to as the native language of the computer system is the first generation programming language. In the machine language, a programmer only deals with a binary number.

**Advantages of first generation language**

* They are translation free and can be directly executed by the computers.
* The programs written in these languages are executed very speedily and efficiently by the CPU of the computer system.
* The programs written in these languages utilize the memory in an efficient manner because it is possible to keep track of each bit of data.

2. Second Generation language (Assembly Language)

The second generation programming language also belongs to the category of low-level- programming language. The second generation language comprises assembly languages that use the concept of mnemonics for the writing program. In the assembly language, symbolic names are used to represent the opcode and the operand part of the instruction.

**Advantages of second generation language**

* It is easy to develop understand and modify the program developed in these languages are compared to those developed in the first generation programming language.
* The programs written in these languages are less prone to errors and therefore can be maintained with a great case.

3. Third Generation languages (High-Level Languages)

The third generation programming languages were designed to overcome the various limitations of the first and second generation programming languages. The languages of the third and later generation are considered as a high-level language because they enable the programmer to concentrate only on the logic of the programs without considering the internal architecture of the computer system.

**Advantages of third generation programming language**

* It is easy to develop, learn and understand the program.
* As the program written in these languages are less prone to errors they are easy to maintain.
* The program written in these languages can be developed in very less time as compared to the first and second generation language.

**Examples:** FORTRAN, ALGOL, COBOL, C++, C

4. Fourth generation language (Very High-level Languages)

The languages of this generation were considered as very high-level programming languages required a lot of time and effort that affected the productivity of a programmer. The fourth generation programming languages were designed and developed to reduce the time, cost and effort needed to develop different types of software applications.

**Advantages of fourth generation languages**

* These programming languages allow the efficient use of data by implementing the various database.
* They require less time, cost and effort to develop different types of software applications.
* The program developed in these languages are highly portable as compared to the programs developed in the languages of other generation.

**Examples:** SOL, CSS, coldfusion

5. Fifth generation language (Artificial Intelligence Language)

The programming languages of this generation mainly focus on constraint programming. The major fields in which the fifth generation programming language are employed are Artificial Intelligence and Artificial Neural Networks

**Advantages of fifth generation languages**

* These languages can be used to query the database in a fast and efficient manner.
* In this generation of language, the user can communicate with the computer system in a simple and an easy manner.

**Examples:** mercury, prolog, OPS5

**Computer Programming**

* **Why Programming?**
You may already have used software, perhaps for word processing or spreadsheets, to solve problems. Perhaps now you are curious to learn how programmers write software. A *program*is a set of step-by-step instructions that directs the computer to do the tasks you want it to do and produce the results you want.

There are at least three good reasons for learning programming:
	+ Programming helps you understand computers. The computer is only a tool. If you learn how to write simple programs, you will gain more knowledge about how a computer works.
	+ Writing a few simple programs increases your confidence level. Many people find great personal satisfaction in creating a set of instructions that solve a problem.
	+ Learning programming lets you find out quickly whether you like programming and whether you have the analytical turn of mind programmers need. Even if you decide that programming is not for you, understanding the process certainly will increase your appreciation of what programmers and computers can do.

A set of rules that provides a way of telling a computer what operations to perform is called a programming language. There is not, however, just one programming language; there are many. In this chapter you will learn about controlling a computer through the process of programming. You may even discover that you might want to become a programmer.

An important point before we proceed: You will not be a programmer when you finish reading this chapter or even when you finish reading the final chapter. Programming proficiency takes practice and training beyond the scope of this book. However, you will become acquainted with how programmers develop solutions to a variety of problems.

* **What Programmers Do**
In general, the programmer's job is to convert problem solutions into instructions for the computer. That is, the programmer prepares the instructions of a computer program and runs those instructions on the computer, tests the program to see if it is working properly, and makes corrections to the program. The programmer also writes a report on the program. These activities are all done for the purpose of helping a user fill a need, such as paying employees, billing customers, or admitting students to college.

The programming activities just described could be done, perhaps, as solo activities, but a programmer typically interacts with a variety of people. For example, if a program is part of a system of several programs, the programmer coordinates with other programmers to make sure that the programs fit together well. If you were a programmer, you might also have coordination meetings with users, managers, systems analysts, and with peers who evaluate your work-just as you evaluate theirs.

Let us turn to the programming process.
* **The Programming Process**
Developing a program involves steps similar to any problem-solving task. There are five main ingredients in the programming process:
	+ Defining the problem
	+ Planning the solution
	+ Coding the program
	+ Testing the program
	+ Documenting the program

Let us discuss each of these in turn.

**Defining the Problem**
Suppose that, as a programmer, you are contacted because your services are needed. You meet with users from the client organization to analyze the problem, or you meet with a systems analyst who outlines the project. Specifically, the task of defining the problem consists of identifying what it is you know (input-given data), and what it is you want to obtain (output-the result). Eventually, you produce a written agreement that, among other things, specifies the kind of input, processing, and output required. This is not a simple process.

**Planning the Solution**

Two common ways of planning the solution to a problem are to draw a flowchart and to write pseudocode, or possibly both. Essentially, a flowchart is a pictorial representation of a step-by-step solution to a problem. It consists of arrows representing the direction the program takes and boxes and other symbols representing actions. It is a map of what your program is going to do and how it is going to do it. The American National Standards Institute (ANSI) has developed a standard set of flowchart symbols. Figure 1 shows the symbols and how they might be used in a simple flowchart of a common everyday act-preparing a letter for mailing.

Pseudocode is an English-like nonstandard language that lets you state your solution with more precision than you can in plain English but with less precision than is required when using a formal programming language. Pseudocode permits you to focus on the program logic without having to be concerned just yet about the precise syntax of a particular programming language. However, pseudocode is not executable on the computer. We will illustrate these later in this chapter, when we focus on language examples.

**Coding the Program**
As the programmer, your next step is to code the program-that is, to express your solution in a programming language. You will translate the logic from the flowchart or pseudocode-or some other tool-to a programming language. As we have already noted, a programming language is a set of rules that provides a way of instructing the computer what operations to perform. There are many programming languages: BASIC, COBOL, Pascal, FORTRAN, and C are some examples. You may find yourself working with one or more of these. We will discuss the different types of languages in detail later in this chapter.

Although programming languages operate grammatically, somewhat like the English language, they are much more precise. To get your program to work, you have to follow exactly the rules-the syntax-of the language you are using. Of course, using the language correctly is no guarantee that your program will work, any more than speaking grammatically correct English means you know what you are talking about. The point is that correct use of the language is the required first step. Then your coded program must be keyed, probably using a terminal or personal computer, in a form the computer can understand.

One more note here: Programmers usually use a text editor, which is somewhat like a word processing program, to create a file that contains the program. However, as a beginner, you will probably want to write your program code on paper first.

**Testing the Program**
Some experts insist that a well-designed program can be written correctly the first time. In fact, they assert that there are mathematical ways to prove that a program is correct. However, the imperfections of the world are still with us, so most programmers get used to the idea that their newly written programs probably have a few errors. This is a bit discouraging at first, since programmers tend to be precise, careful, detail-oriented people who take pride in their work. Still, there are many opportunities to introduce mistakes into programs, and you, just as those who have gone before you, will probably find several of them.

**Documenting the Program**
Documenting is an ongoing, necessary process, although, as many programmers are, you may be eager to pursue more exciting computer-centered activities. Documentation is a written detailed description of the programming cycle and specific facts about the program. Typical program documentation materials include the origin and nature of the problem, a brief narrative description of the program, logic tools such as flowcharts and pseudocode, data-record descriptions, program listings, and testing results. Comments in the program itself are also considered an essential part of documentation. Many programmers document as they code. In a broader sense, program documentation can be part of the documentation for an entire system.

The wise programmer continues to document the program throughout its design, development, and testing. Documentation is needed to supplement human memory and to help organize program planning. Also, documentation is critical to communicate with others who have an interest in the program, especially other programmers who may be part of a programming team. And, since turnover is high in the computer industry, written documentation is needed so that those who come after you can make any necessary modifications in the program or track down any errors that you missed.

**Six-Step:- Problem Solving Model**

1. **Define the problem:** Obviously, you aren’t going to get far if you don’t know what the problem is that you are trying to solve. Take some time at this first step to understand the problem on a deeper level so you will be able to take logical action later on.
2. **Determine the root cause:** This is an important step that is frequently missed when individuals or teams are trying to solve problems. What is it that is causing you to wind up in this situation? Is there just one root cause, or is it a combination of issues coming together negatively?
3. **Develop alternative solutions:** At this point, you aren’t trying to find the single answer to the problem – instead, you are trying to find all possible answers. Anything which you believe may be able to help you solve this issue should be on the table at this point.
4. **Select a solution:** Now that you have a list of possible solutions to consider, you can pare them down and wind up picking out the one which you believe will resolve the matter.
5. **Implement the solution:** With your choice made, it’s now time to put that choice into action and see what happens. Of course, implementation itself can be a complex process which may require its own methods and tools to work through successfully.
6. **Evaluate the outcome:** This is important. Once you implement a solution, you need to follow through with that solution to make sure it actually worked. If not, you may need to go back and consider one of your other potential solutions that was identified in step three.

**Program Control Structures**

Top of Form

Bottom of Form

**Overview of Programming Control Structures**

The theoretical control structures are:

* sequence
* procedure (method) call
* selection
* iteration (optional if recursion is permitted)

The above can represent the minimal set that a programming language must support.  This has been mathematically proven that any algorithm can be expressed with these control structures and without "goto's" or unconditional branches.

Combine with data structures (arrays, hashtables, structs, classes) and object oriented programming.

**Flavors of control structures**

**Selection structures**

* if/then
* if/then/else
* if/then/else if ...
* switch

**Iteration structures**

* while
* for (foreach)
* do/while

Addional flow control within iteration structures

* + break
	+ continue
	+ goto
	+ return (exit loop and function)

Programming Methodologies

**Background**

There are various programming methodologies for solving real-time problems programmatically. This article just provides an overview of some programming methodologies and we will see OOP concepts in greater details.

Let us now see the programming methodologies.

**What a Programming Methodology is**

A programming methodology is nothing but a type of technique to solve some given requirements using programming languages.

For example, I want to add the number 10 to itself.

Then the example above is an example of a problem because it is very difficult to calculate these types of calculations manually again and again so to solve the preceding given requirement programmatically with a permanent solution, each programming language has its own methodology and style of coding to solve the given problem.

The following are some of the common types of programming methodologies.



**Unstructured Programming**

A programming language in which the entire logic of the program is written as a single continuous (nonstop or unbroken) block is called *unstructured programming*.

The following are the key points of unstructured programming languages:

* It is a type of problem-solving technique, in which we solve the problem in terms of lots of code. If the requirements increase then the code is also increased.
* There is no re-usability of existing code.
* Finding an error in a program is very difficult.
* Syntax and structure is very difficult to understand and remember.
* Modification is very difficult.
* Non-structured languages allow only basic data types, such as numbers, strings and arrays.
* Statements are usually one in each line.

For example:

[JOSS](http://en.wikipedia.org/wiki/JOSS), [FOCAL](http://en.wikipedia.org/wiki/FOCAL_%28programming_language%2529), [MUMPS](http://en.wikipedia.org/wiki/MUMPS), [TELCOMP](http://en.wikipedia.org/wiki/TELCOMP), [COBOL](http://en.wikipedia.org/wiki/COBOL), FORTRAN and so on.

2**. Structured Programming Language**

A programming language in which the entire logic of the program is written by dividing it into smaller units or modules is called a *Structured Programming Language*.

The following are the key points of Structured Programming languages:

* Finding an error in a program is easily.
* Syntax and structure is very simple to understand and remember.
* Modification is easy.
* There is no re-usability of existing code as much as expected.

For example,

       **C**and so on.

3**.  Object Oriented Programming**

Object Oriented Programming (OOP) is the programming methodology in which each entity is an **object**.

This provides the following key advantages.

* Re-usability of existing code.
* Extensibility of existing code.
* Security.
* Flexibility.
* Maintainability.

The following are the key points of OOP languages:

* Classes
* Objects
* Methods
* abstraction
* Encapsulation
* Inheritance
* Polymorphism

And so on.

For example:

C++, Java, C#, F#, VB.net .and so on.

Now in my next article we will learn details about the C#'s OOP features.

**Stages of program development process**

The various stages in the development of a computer program are :

1. Problem Definition
2. Program Design
3. Coding
4. Debugging
5. Testing
6. Documentation
7. Maintenance

**Problem Definition:**

* The first step in the process of program development is the thorough understanding and identification of the problem for which is the program or software is to be developed.
* In this step the problem has to be defined formally.
* All the factors like Input/output, processing requirement, memory requirements, error handling, interfacing with other programs have to be taken into consideration in this stage.

**Program Design:**

* The next stage is the program design. The software developer makes use of tools like algorithms and flowcharts to develop the design of the program.
	+ Algorithm
	+ Flowchart

**Coding:**

* Once the design process is complete, the actual computer program is written, i.e. the instructions are written in a computer language.
* Coding is generally a very small part of the entire program development process and also a less time consuming activity in reality.
* In this process all the syntax errors i.e. errors related to spelling, missing commas, undefined labels etc. are eliminated.
* For effective coding some of the guide lines which are applied are :
	+ Use of meaningful names and labels of variables,
	+ Simple and clear expressions,
	+ Modularity with emphasis on making modules generalized,
	+ Making use of comments and indenting the code properly,
	+ Avoiding jumps in the program to transfer control.

**Debugging:**

* At this stage the errors in the programs are detected and corrected.
* This stage of program development is an important process. Debugging is also known as program validation.
* Some common errors which might occur in the programs include:
	+ Un initialization of variables.
	+ Reversing of order of operands.
	+ Confusion of numbers and characters.
	+ Inverting of conditions eg jumping on zero instead of on not zero.

**Testing:**

* The program is tested on a number of suitable test cases.
* A test plan of the program has to be done at the stage of the program design itself.
* This ensures a thorough understanding of the specifications.
* The most trivial and the most special cases should be identified and tested.
* It is always useful to include the maximum and minimum values of all variables as test data.

**Documentation:**

* Documentation is a very essential step in the program development.
* Documentation help the users and the people who maintain the software.
* This ensures that future modification if required can be done easily. Also it is required during redesigning and maintenance.

**Maintenance:**

* Updating and correction of the program for changed conditions and field experience is accounted for in maintenance.
* Maintenance becomes essential in following situations:
	+ Change in specification,
	+ Change in equipment,
	+ Errors which are found during the actual execution of the program.