CHAPTER TWO

Interpreted Language vs Compiled Languages

An interpreter is present during the course of the application's execution in interpreted languages. The interpreter is in control at this point in the execution. The interpreter, in essence, creates a virtual machine whose "machine language" is the high-level programming language. More or less, one at a time, the interpreter reads and executes statements from the language. A good source-level debugger could also be a function of the interpreter because the source code is being executed directly. It can also work with languages whose basic program properties, such as variable sizes and types, or even names that designate which variables are meant to be referenced, can vary depending on the input data. **All scripting languages are interpreted Languages.**

Compiler-based languages, as opposed to interpreter-based ones, first translate a high-level source program into a target program that is equivalent (usually in machine language), and then they goes away. The user specifies a random moment in the future for operating system required to run the intended program. The source of control during compilation is the compiler, and the target. When a program is being executed, it is where control is located. So, the compiler is a machine language program, produced through the compilation of another high-level application.

Scripting Languages vs Other Kind of Programming Languages (JavaScript vs C++)

A good example of a scripting language is JavaScript, which is designed for a runtime system to automate task execution. JavaScript does not require an explicit compilation step. Due to its great level of abstraction, it is also known as very high-level programming languages. A "script," or a brief program created for a particular runtime environment, is supported by scripting languages. In place of being compiled, these are interpreted live. Scripting languages therefore utilize an interpreter rather than a compiler to translate source code into machine code. Thus, a scripting language can automate various settings like application software, websites, text editors, operating system shells, video games, etc.

C++, on the other hand, is a good example of a non-interpreted programming language (compiled language). The generated program is extremely efficient because it is typically translated into machine language, which the system can understand directly. In compilation, the source code is first preprocessed using a preprocessor, followed by compilation using a compiler, assembly of the compiled source code, linking an object code file, and lastly creation of an executable file.

Types of Scripting Languages

1. Client-side Scripting Languages

Languages for client-side scripting are executed by the user's browser. It is typically carried out in the front-end, where it is visible to visitors and less prone to vulnerabilities and leaks. As a result, it is frequently employed to create user interfaces and other types of lightweight functionality. Since they operate locally, they typically offer superior performance and do not tax your server. More interactivity is possible, some tasks can be completed without the user's involvement, and database connections to web server-based databases are not supported. The file system in the web browser is inaccessible to these programs. Based on the user's preferences, pages are changed. It can also be used to make "cookies" that the user's computer uses to store information. Examples include JavaScript, jQuery, CSS, and HTML.

2. Server-side Scripting Languages

Languages that run on a web server are referred to as server-side scripting languages. The visitor cannot see the script because it operates on the back-end. It is a more secure method as a result. They are frequently employed to build interactive websites and platforms, respond to user inquiries, develop and give data, among other things. The use of PHP in WordPress is a well-known instance of server-side scripting. There are also Python, Node.js, Perl, and Ruby as examples.