Compiled and Interpreted Programming Languages

Complied Languages:

Compiled languages are converted directly into machine code that the processor can execute.

As a result, they tend to be faster and more efficient to execute than interpreted languages. They also give the developer more control over hardware aspects, like memory management and CPU usage.

They need to be compiled manually and every time you make a change you should rebuild the program.

There are some disadvantages:

- A compiler takes an entire program and a lot of time to analyze the source code.
- Platform dependence of the generated binary code.
- All changes need recompilation.

Examples:

C, C++, C#, CLEO, COBOL

Interpreted Languages:

Interpreters run through a program line by line and execute each command.

Interpreted languages were once significantly slower than compiled languages. But, with the development of just-in-time compilation, that gap is shrinking.

Just-in-time compilation:

Just-in-time compilation is a method for improving the performance of interpreted programs. During execution the program may be compiled into native code to improve its performance. It is also known as dynamic compilation.

Advantages:

- Interpreted languages tend to be more flexible.
- It does not have to be compiled for each hardware target separately.
- Small program size and dynamic typing.
- platform independence

Disadvantages:

- Slowe execution than compiled languages.
- No control as compiled languages on memory management.
- the need for the interpreter to always be present before execution can occur.

Examples:

PHP, Ruby, Python, and JavaScript.