Compiled vs. Interpreted Programming Languages

Last modified: October 13, 2021

by [baeldung](https://www.baeldung.com/cs/author/baeldung" \o "Posts by baeldung)

* [**Core Concepts**](https://www.baeldung.com/cs/category/core-concepts)
* [**Programming**](https://www.baeldung.com/cs/category/core-concepts/programming)

If you have a few years of experience in Computer Science or research, and you’re interested in sharing that experience with the community, have a look at our [**Contribution Guidelines**](https://www.baeldung.com/cs/contribution-guidelines).

**1. Introduction**

In this tutorial, we’ll present the difference between compiled and interpreted programming languages.

**2. Why Do We Need Compilers and Interpreters?**

Computers can understand and run only binary code. Programmers use high–level programming languages, such as C, Python, or Java. Those languages are easier to work with since they resemble human languages and mathematical notation. However, computers cannot run the code written in a high–level language. **We first have to translate it into binary code**. To do so, we use [compilers](https://en.wikipedia.org/wiki/Compiler) and [interpreters](https://en.wikipedia.org/wiki/Interpreter_(computing)).

Languages whose programs we usually compile are called compiled languages. Similarly, those we usually interpret are called interpreted languages.

**3. Compilers**

Compilers take a whole program as input and translate it to an executable binary code [in several steps](https://www.baeldung.com/cs/how-compilers-work).

We can run the binary code only on the machine on which we compiled it. That’s because the binary code depends on the hardware and is not portable.

**The compilation step is required only once. Afterward, we can run the binary code as many times as we want.**

Because compilers process whole programs, [they can catch some errors and warn us to correct them](https://www.baeldung.com/cs/runtime-vs-compile-time). Those are syntax and type errors. Compilation fails if they are present.

[C](https://en.wikipedia.org/wiki/C_(programming_language)) is an example of a compiled language.

**4. Interpreters**

Interpreters read and execute the program at hand instruction by instruction. After being read, each instruction is translated into the machine’s binary code and run.

Unlike compilers, the interpreters do not produce a binary executable file. Each time we run a program, we invoke the interpreter. It then reads and executes the program one instruction at a time.

**That’s why it must be present in the computer’s RAM whenever we run a program. In contrast to interpreters, we need compilers only during compilation.**

On the other hand, unlike the compilers, the interpreters catch all the errors at runtime.

[Python](https://www.baeldung.com/java-working-with-python) is an example of an interpreted language.

**5. Example of Compilation and Interpretation**

Imagine a code in an unnamed language:

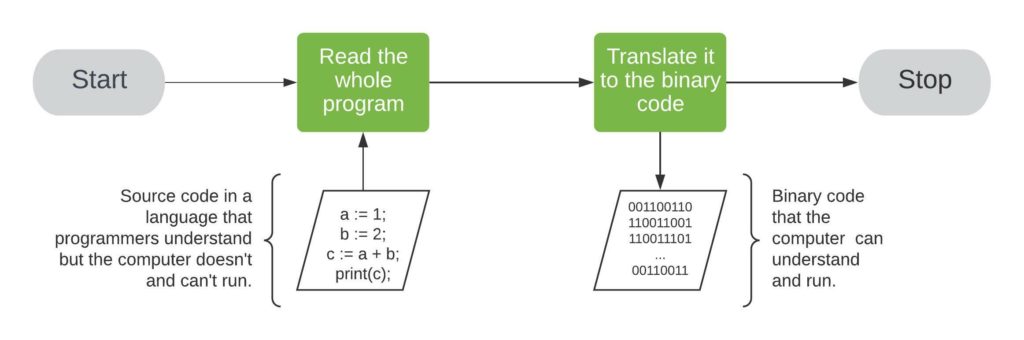
a := 1;

b := 2;

c := a + b;

print(c);

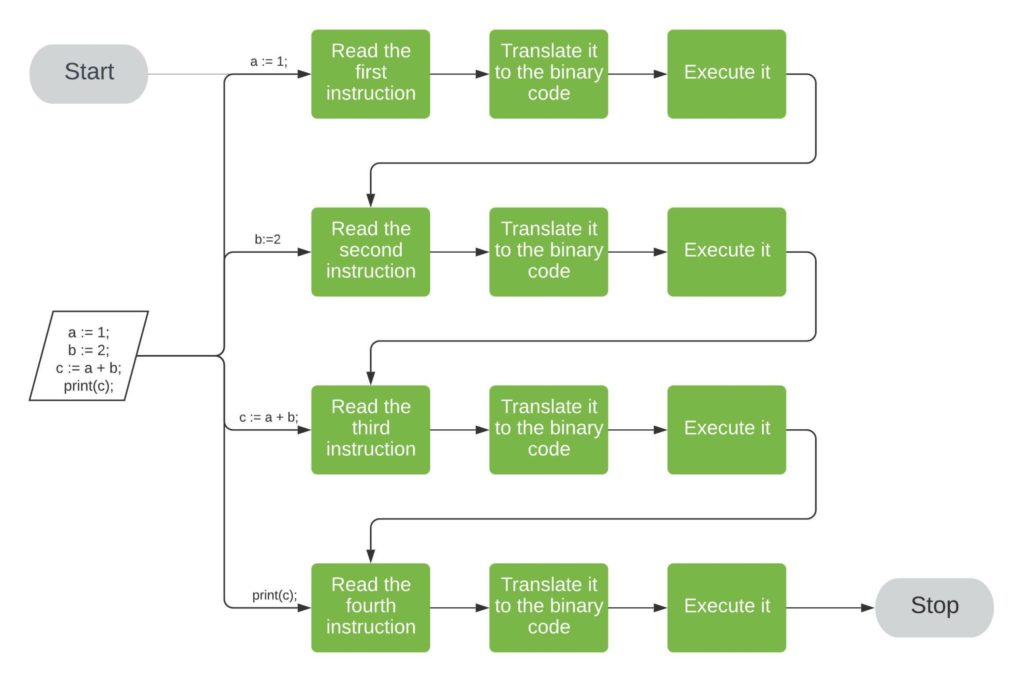
A compiler would translate it to the binary code, which we can run later:



**All those binary instructions are the commands for the computer at hand.** They instruct it to set the appropriate CPU register to 1. Afterward, they choose a free address in RAM to write the register’s content. That is how a compiler translates *a := 1;* to the binary code. Other binary instructions are translations of the rest of the code to the machine’s binary language.

Once we run the binary code, it gets loaded into RAM, and the CPUs execute it.

On the other hand, an interpreter would first read the instruction *a := 1;*. **It would load it into the computer’s RAM, translate it into binary code and let the CPU(s) execute it**. Then, the interpreter would do the same with instruction *b := 2;*. After the second instruction, it would move on to the third, and finally, to the last one:



**So, we see that translation and execution are separated if we use compilers. However, they are intertwined if we interpret our programs.** That’s why interpretation is usually slower than running compiled programs.

**6. Differences**

Let’s sum up the differences between compilers and interpreters:



Note that we usually think of programming languages as either compiled or interpreted. Still, we can make [interpreters for C](https://www.drdobbs.com/cpp/ch-a-cc-interpreter-for-script-computing/184402054) just as we can create [compilers for Python](https://nuitka.net/pages/overview.html).

**Furthermore, some languages mix both concepts.** [For example](https://www.baeldung.com/java-compiled-interpreted), we first compile Java programs to bytecode, the code of [Java Virtual Machine](https://www.baeldung.com/jvm-vs-jre-vs-jdk). Afterward, we interpret the bytecode.

**A compiled language compiles the code into machine instructions and runs it together. But, interpreted languages translate and execute each line of code one after the other.**

A common misconception is that a language is either compiled or interpreted. But that is not the case. Some programming languages can be both compiled and interpreted.

Interpreted languages can work in 3 different ways:

* Execute the code and produce the output directly.
* Convert the code to an intermediate code and execute the code.
* Use an internal compiler to get a precompiled code, then execute the precompiled code.

Let’s dive deep and understand the pros and cons of interpreted languages.

**Advantages of Interpreted Languages**

* An interpreted language is more flexible compared to a compiled language because interpreted language executes the program code itself, thus making the code platform-independent. A platform-independent code can run across multiple environments and hardware, which helps in less planning in case of translation across enterprises.
* It is portable and need not be recompiled for different platforms. The code can be downloaded as such and used if there is an interpreter in the machine.
* The programmer will be able to identify exactly the part of the program which has an error and simultaneously check if the remaining part is correct. When the program is executed line by line, it is easier to check errors in the program.
* An interpreter directly translates the code written in a high-level language to machine-level language. Hence, the size and memory of the program are smaller. In a compiled language, an executable file is created, which takes up additional space.
* Debugging in the program occurs at run-time, so even if the last part of the code has an error, the code executes till it reaches the part of the code that is incorrect.
* In an interpreted language, you can change the code while it is running and start testing the program even when the code has errors. In a compiled language, you can test the code only if the entire code is compiled into an executable file.

**Disadvantages of Interpreted Languages**

* An interpreted language is slower compared to a compiled language. The language analyzes and executes every part of the code, making it process the code every time. An interpreter is needed in the local machine to run the program.
* Executing the program in an interpreter is less efficient than regular program execution.
* An interpreted language is less secure. Unlike compiled languages, the interpreter does not have an executable file. So, while sharing the code, we need to share the entire source code, which can be a concern to the programmer if the company or organization is concerned about their privacy.

**Final Thoughts**

As a programmer, choosing which language you want to go about completely depends upon your use case. By understanding the pros and cons of the compiled and interpreter languages, you can choose the language which fits your application.