

JANUARY 10, 2020 / #PROGRAMMING LANGUAGES

Interpreted vs Compiled Programming Languages: What's the Difference?

Every program is a set of instructions, whether it's to add two numbers or send a request over the internet. Compilers and interpreters take human-readable code and convert it to computer-readable machine code.

In a compiled language, the target machine directly translates the program. In an interpreted language, the source code is not directly translated by the target machine. Instead, a *different* program, aka the interpreter, reads and executes the code.

Okay... but what does that *actually* mean?

Imagine you have a hummus recipe that you want to make, but it's written in ancient Greek. There are two ways you, a non-ancient-Greek speaker, could follow its directions.

The first is if someone had already translated it into English for you. You (and anyone else who can speak English) could read the English version of the recipe and make hummus. Think of this translated recipe as the *compiled* version.

The second way is if you have a friend who knows ancient Greek. When you're ready to make hummus, your friend sits next to you and translates the recipe into English as you go, line by line. In this case, your friend is the interpreter for the *interpreted* version of the recipe.

Compiled 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.

Compiled languages need a "build" step – they need to be manually compiled first. You need to "rebuild" the program every time you need to make a change. In our hummus example, the entire translation is written before it gets to you. If the original author decides that he wants to use a different kind of olive oil, the entire recipe would need to be translated again and resent to you.

Examples of pure compiled languages are C, C++, Erlang, Haskell, Rust, and Go.

Interpreted Languages

Interpreters run through a program line by line and execute each command. Here, if the author decides he wants to use a different kind of olive oil, he could scratch the old one

out and add the new one. Your translator friend can then convey that change to you as it happens.

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

Examples of common interpreted languages are PHP, Ruby, Python, and JavaScript.

A Small Caveat

Most programming languages can have both compiled and interpreted implementations – the language itself is not necessarily compiled or interpreted. However, for simplicity's sake, they're typically referred to as such.

Python, for example, can be executed as either a compiled program or as an interpreted language in interactive mode. On the other hand, most command line tools, CLIs, and shells can theoretically be classified as interpreted languages.

Advantages and disadvantages

Advantages of compiled languages

Programs that are compiled into native machine code tend to be faster than interpreted code. This is because the process of translating code at run time adds to the overhead, and can cause the program to be slower overall.

Disadvantages of compiled languages

The most notable disadvantages are:

- Additional time needed to complete the entire compilation step before testing
- Platform dependence of the generated binary code

Advantages of interpreted languages

Interpreted languages tend to be more flexible, and often offer features like dynamic typing and smaller program size. Also, because interpreters execute the source program code themselves, the code itself is platform independent.

Disadvantages of interpreted languages

The most notable disadvantage is typical execution speed compared to compiled languages.

If this article was helpful, [tweet it.](#)

Learn to code for free. freeCodeCamp's open source curriculum has helped more than 40,000 people get jobs as developers. [Get started](#)

freeCodeCamp is a donor-supported tax-exempt 501(c)(3) nonprofit organization (United States Federal Tax Identification Number: 82-0779546)

Our mission: to help people learn to code for free. We accomplish this by creating thousands of videos, articles, and interactive coding lessons - all freely available to the public. We also have thousands of freeCodeCamp study groups around the world.

Donations to freeCodeCamp go toward our education initiatives, and help pay for servers, services, and staff.

You can [make a tax-deductible donation here](#).

What is Docker?
TCP/IP Model
RTF File
CSS Transition
How to Use Instagram?
MBR VS GPT
FAT32 Format
Error 503 Code
Windows Hosts File
Mobi to PDF

Trending Guides

What is STEM?
JavaScript Void 0
SQL Delete Row
JavaScript Replace
Python JSON Parser
cmd Delete Folder
What is NFC?
Content Type JSON
Convert HEIC to JPG
Math Random Java
WordPress for Beginners
Qualitative VS Quantitative
JavaScript Split String
Accented Letters on Mac
Windows 10 Product Key
Google Docs Landscape
Antimalware Executable
Windows 10 Start Menu
Windows 10 Command Line
Google Account Recovery

[About](#) [Alumni Network](#) [Open Source](#) [Shop](#) [Support](#) [Sponsors](#) [Academic Honesty](#) [Code of Conduct](#) [Privacy Policy](#) [Terms of Service](#) [Copyright Policy](#)