It may seem odd to software developers working today, but the Python programming language was once considered a gap-filler, a way to write scripts that “automate the boring stuff" (as one popular book on learning Python put it) or to rapidly prototype applications that would then be implemented in other languages.

Python has since emerged from such humble beginnings and is now a first-class citizen in modern software development, infrastructure management, and data analysis. It is no longer seen as a back-room utility language but a major force in web application creation and systems management, and a key driver of the explosion in data science, machine learning, and, increasingly, generative AI.

Python’s key advantages

Some key advantages drive Python’s success with beginners and expert programmers alike, so let's start with an overview.

Python is easy to learn and use

Python encompasses a relatively modest number of features, so it requires a fairly minimal investment of time and effort to produce your first programs. Python's syntax is designed to be readable and straightforward. This simplicity makes it an ideal teaching language, which newcomers can pick up quickly. As a result, developers can spend more time thinking about the problem they’re trying to solve, rather than worrying about syntactic complexities or deciphering legacy code.

Python is broadly adopted and supported

Python is both popular and widely used, as the high rankings in surveys like the Tiobe Index and the large number of GitHub projects using Python attest. Python runs on every major operating system and platform, and most minor ones, too. Many major libraries and API-powered services have Python bindings or wrappers, so Python interfaces freely with them.

Python is not a toy language

Even though scripting and automation cover a large chunk of Python’s use cases (more on that later), Python is also used to build professional-quality software, both as standalone applications and as web services. Python may not be the fastest language, but what it lacks in speed, it makes up for in versatility. There are also major efforts underway to make Python faster.

Python keeps moving forward

Each revision of the Python language adds useful new features to keep pace with modern software development practices. Asynchronous operations and coroutines, for instance, are now standard parts of the language, making it easier to write Python applications that are capable of concurrent processing. Type hints allow linting tools to analyze program logic and reduce the complexity that comes with a dynamic language. And the CPython runtime, the default implementation of Python, is being incrementally redesigned to allow faster execution and better parallelism.

What is Python used for?

Python's most basic use case is as a scripting and automation language. Python isn’t just a replacement for shell scripts or batch files; it is also used to automate interactions with web browsers and application GUIs, or to do system provisioning and configuration in tools such as Ansible and Salt. But scripting and automation represent only the tip of the iceberg with Python.

General application programming

You can create both command-line and cross-platform GUI applications with Python and deploy them as self-contained executables. Python doesn’t have the native ability to generate a standalone binary from a script, but you can use third-party packages like PyInstaller and Nuitka for that purpose.

Data science and machine learning

Sophisticated data analysis has become one of the fastest-moving areas of IT and one of Python’s star use cases. The vast majority of libraries used for data science or machine learning have Python interfaces, making the language the most popular high-level command interface to for machine learning libraries and other numerical algorithms.

Web services and RESTful APIs

Python’s native libraries and third-party web frameworks provide fast and convenient ways to create everything from simple REST APIs in a few lines of code to full-blown, data-driven sites. Python’s latest versions have strong support for asynchronous operations, letting sites handle tens of thousands of requests per second with the right libraries.

Metaprogramming and code generation

In Python, everything in the language is an object, including Python modules and libraries themselves. This lets Python work as a highly efficient code generator, making it possible to write applications that manipulate their own functions and have the kind of extensibility that would be difficult or impossible to pull off in other languages.

Python can also be used to drive code-generation systems, such as LLVM, to efficiently create code in other languages.

Writing glue code

Python is often described as a “glue language,” meaning it can let disparate code (typically libraries with C language interfaces) interoperate. Its use in data science and machine learning is in this vein, but that’s just one incarnation of the general idea. If you have applications or program domains that cannot talk to each other directly, you can use Python to connect them.

What Python does not do well

Also worth noting are the sorts of tasks Python is not well-suited for.

Python is a high-level language, so it’s not suitable for system-level programming—device drivers or OS kernels are out of the picture.

It’s also not ideal for situations that call for cross-platform standalone binaries. You could build a standalone Python app for Windows, macOS, and Linux, but not elegantly or simply. Mobile-native applications are also not easy to make in Python, at least not compared to languages with native toolchains for mobile platforms, like Swift or Kotlin.

Finally, Python is not the best choice when speed is an absolute priority in every aspect of the application. For that, you’re better off with C/C++, Rust, or another language of that caliber. That said, you can often wrap libraries written in those languages to get Python to speeds within striking distance of them.