**DevOps With Python**

# How to Run Your Python Scripts and Code

* Run Python scripts from your operating system’s **command line** or **terminal**
* Execute Python code and scripts in **interactive mode** using the standard REPL
* Use your favorite **IDE** or **code editor** to run your Python scripts
* Fire up your scripts and programs from your operating system’s **file manager**

## What Scripts and Modules Are

Scripting languages are usually [interpreted](https://en.wikipedia.org/wiki/Interpreter_(computing)) at [runtime](https://en.wikipedia.org/wiki/Runtime_(program_lifecycle_phase)) rather than [compiled](https://en.wikipedia.org/wiki/Compiler). So, scripts are typically run by some kind of interpreter, which is responsible for executing each order in a sequence.

Python is an interpreted language. Because of that, Python programs are commonly called scripts. However, this terminology isn’t completely accurate because Python programs can be way more complex than a simple, sequential script.

On the other hand, a file containing Python code that’s designed to be imported and used from another Python file is called a **module**.

So, the main difference between a [module](https://realpython.com/python-modules-packages/) and a script is that modules store **importable code** while scripts hold **executable code**.

## How to Run Python Scripts From the Command Line

PS> python .\hello.py

Hello, World!

PS> py .\hello.py

Hello, World!

A cool feature of a terminal or [shell application](https://en.wikipedia.org/wiki/Unix_shell) is that you can redirect the output of your commands using a straightforward syntax. This feature may be useful in those situations where you have a Python program that can generate a long output, and you’d like to save it to a file for later analysis.

$ python hello.py > output.txt

Finally, if you want to add the output of consecutive executions to the end of output.txt, then you can use two angle brackets (>>) instead of one:

$ python hello.py >> output.txt

### **Using the Script’s Filename Directly**

On Windows, you can also run Python scripts by simply entering the name of the file containing the executable code at the command line:

PS> .\hello.py

Once you’ve written the path to your script and pressed Enter, you’ll note that a new terminal window appears on your screen for a few seconds, showing the script output. This is possible because Windows associates .py and .pyw files to python.exe and pythonw.exe, respectively.

This way of running Python scripts on Windows may be annoying because the code runs in a new terminal window that automatically closes after the execution ends. In most cases, you won’t be able to check the program’s output.

$ ./hello.py

bash: ./hello.py: Permission denied

Unix systems prioritize security, which means that you can’t go around executing any file as a program. So, you get a permission denied error when you try to run hello.py directly. To fix this issue, you need to explicitly tell the system that the file is executable. To do this, you can use the chmod command:

$ chmod +x hello.py

After running this command, your hello.py file will be executable. However, that’s not enough for the script to run properly:

$ ./hello.py

./hello.py: line 1: syntax error near unexpected token `"Hello, World!"'

./hello.py: line 1: `print("Hello, World!")'

Why are you getting another error now? The problem is that your operating system (OS) doesn’t know which program to use for running your script and is trying to run it with the shell itself. You can fix that by making a small addition to your hello.py file:

#!/usr/bin/env python3

print("Hello, World!")

You’ve added a new line at the beginning of hello.py. It now starts with a Unix-style [shebang](https://realpython.com/python-shebang/), which is a special kind of [comment](https://realpython.com/python-comments-guide/) that you can include in your scripts to tell the operating system which program to use for running the content of this file.

**#!/usr/bin/python3**

**#!/usr/bin/env python3**

### **Running Modules With the -m Option**

The python command has a series of [command-line options](https://realpython.com/python-command-line-arguments/) that can be useful in specific situations. For example, if you want to run a Python module, then you can use the command python -m <module-name>. The -m option searches Python’s [module search path](https://docs.python.org/3/tutorial/modules.html#the-module-search-path), sys.path, for the module name and runs its content:

**$ python -m hello**

**Hello, World!**

In this example, you run the hello.py file as a module. In other words, you don’t include the .py suffix.

## How to Run Python Code Interactively

### **Getting to Know the Python Interpreter**

The name Python also refers to a piece of software called the **interpreter**, which is the program that allows you to run Python code.

The interpreter is a layer of software that works between your program and your computer hardware to get your code running.

* [C](https://realpython.com/c-for-python-programmers/), like [CPython](https://www.python.org/about/), which is the core implementation of the language
* Python itself, like [PyPy](https://realpython.com/pypy-faster-python/), which is a [fast](http://speed.pypy.org/) implementation with a [just-in-time (JIT) compiler](https://en.wikipedia.org/wiki/Just-in-time_compilation)
* [Java](https://realpython.com/oop-in-python-vs-java/), like [Jython](http://www.jython.org/index.html), which can take advantage of the Java ecosystem
* .NET, like [IronPython](http://ironpython.net/), which uses the .NET ecosystem

The Python interpreter can run code in two different modes:

1. [Script](https://realpython.com/run-python-scripts/), or program
2. [Interactive](https://realpython.com/interacting-with-python/), or REPL

In **script mode**, you use the interpreter to run a source file as an executable program

Alternatively, **interactive mode** is when you launch the interpreter and use it as a platform to run code that you type in directly. This mode is pretty useful for learning Python as well as for developing, testing, and debugging your applications.

### **Running Python Code Interactively**

Interactive sessions are a widely used tool for running Python code. To start a Python interactive session, or REPL, open a command-line window, type in the python command, and then press  enter

$ python

Python 3.11.5 (main, Sep 13 2023, 18:12:18) [GCC 11.4.0] on linux

Type "help", "copyright", "credits" or "license" for more information.

>>>

The Python interpreter is an interactive way to talk to your computer using the language. It’s like live chat

REPL **Reading Evaluating Printing Looping**

>>> print("Hello, World!")

Hello, World!

>>> 2 + 5

7

>>> print("Welcome to Real Python!")

Welcome to Real Python!

An interactive session will allow you to test every piece of code that you execute. That’s why this tool is an awesome development helper and an excellent space to experiment with the language and test ideas on the fly.

### **Taking Advantage of import Statements**

When you [import a module](https://realpython.com/absolute-vs-relative-python-imports/) from another module, script, or interactive session, what really happens is that Python loads its contents for later access and use. The interesting point is that the [import](https://realpython.com/absolute-vs-relative-python-imports/) statement runs any executable code in the imported module.

You’ll note that import runs the code only once per session. After you first import a module, successive imports do nothing, even if you modify the content of the module. This is because import operations are expensive, and Python takes some extra steps to optimize overall performance:

Python

>>> import hello # Do nothing

>>> import hello # Do nothing again

### **Using the importlib Standard-Library Module**

In the Python [standard library](https://docs.python.org/3/library/index.html), you can find the [importlib](https://docs.python.org/3/library/importlib.html) module. This module provides the [import\_module()](https://docs.python.org/3/library/importlib.html" \l "importlib.import_module) function, which allows you to programmatically import modules.

With import\_module(), you can emulate an import operation and, therefore, execute any module or script.

>>> import importlib

>>> importlib.import\_module("hello")

Hello, World!

<module 'hello' from '/home/username/hello.py'>

You already know that once you’ve imported a module for the first time, you won’t be able to import it again using another import statement. If you want to reload the module and run it once again, then you can use the [reload()](https://docs.python.org/3/library/importlib.html#importlib.reload) function, which forces the interpreter to import the module again:

>>> import hello

Hello World!

>>> import importlib

>>> importlib.reload(hello)

Hello World!

<module 'hello' from '/home/username/hello.py'>

### **Leveraging the Power of the Built-in exec() Function**

So far, you’ve learned about some handy ways to run Python scripts. In this section, you’ll learn how to do that by using the built-in [exec()](https://realpython.com/python-exec/) function, which supports the dynamic execution of Python code.

The exec() function provides an alternative way to run your scripts from inside your code:

>>> with open("hello.py") as hello:

... exec(hello.read())

...

Hello, World!

In this example, you use the [with statement](https://realpython.com/python-with-statement/) to open the hello.py file for reading. Then, you read the file’s content with the .read() method. This method returns a string that you pass to exec() for execution.

You must be careful when using the exec() function because it implies some important security risks, especially if you’re using it for running external code.

## How to Run Python Scripts on IDEs and Code Editors

Python’s standard distribution comes with [IDLE](https://realpython.com/python-idle/) as the default IDE. You can use this program to write, debug, modify, and run your modules and scripts.

 Other IDEs, such as [PyCharm](https://realpython.com/pycharm-guide/) and [Thonny](https://realpython.com/python-thonny/), also allow you to run scripts from inside the environment.