Part 34
-
Running Python Scripts

Version: 2018-03-18

Python Secure FTP module

One of the most important skills you need to build as a Python developer is to be able to run Python scripts and code. This is going to be the only way for you to know if your code works as you planned. It's even the only way of knowing if your code works at all!
This step-by-step tutorial will guide you through a series of ways to run Python scripts, depending on your environment, platform, needs, and skills as a programmer.

You'll have the opportunity to learn how to run Python scripts by using:

- The operating system command-line or terminal
- The Python interactive mode
- The IDE or text editor you like best
- The file manager of your system, by double-clicking on the icon of your script

This way, you'll get the knowledge and skills you'll need to make your development cycle more productive and flexible.

1. Scripts vs Modules

In computing, the word script is used to refer to a file containing a logical sequence of orders or a batch processing file. This is usually a simple program, stored in a plain text file.
Scripts are always processed by some kind of interpreter, which is responsible for executing each command sequentially.
A plain text file containing Python code that is intended to be directly executed by the user is usually called **script**, which is an informal term that means **top-level program file**.
On the other hand, a plain text file, which contains Python code that is designed to be imported and used from another Python file, is called **module**.
So, the main difference between a module and a script is that **modules are meant to be imported**, while **scripts are made to be directly executed**.
In either case, the important thing is to know how to run the Python code you write into your modules and scripts.

2. What's the Python Interpreter?

Python is an excellent programming language that allows you to be productive in a wide variety of fields.
Python is also a piece of software called an **interpreter**. The interpreter is the program you'll need to run Python code and scripts. Technically, the interpreter is a layer of software that works between your program and your computer hardware to get your code running.
Depending on the Python implementation you use, the interpreter can be:

- A program written in C, like CPython, which is the core implementation of the language
- A program written in Java, like Jython
- A program written in Python itself, like PyPy
- A program implemented in .NET, like IronPython

Whatever form the interpreter takes, the code you write will always be run by this program. Therefore, the first condition to be able to run Python scripts is to have the interpreter correctly installed on your system.
The interpreter is able to run Python code in two different ways:

- As a script or module
- As a piece of code typed into an interactive session

3. How to Run Python Code Interactively

A widely used way to run Python code is through an interactive session. To start a Python interactive session, just open a command-line or terminal and then type in python, or python3 depending on your Python installation, and then hit <Enter>.
Here's an example of how to do this on Linux:

```
$ python3
Python 3.6.7 (default, Oct 22 2018, 11:32:17)
[GCC 8.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

The standard prompt for the interactive mode is >>>, so as soon as you see these characters, you'll know you are in.
Now, you can write and run Python code as you wish, with the only drawback being that when you close the session, your code will be gone.
When you work interactively, every expression and statement you type in is evaluated and executed immediately:

```
>>> 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 you write, which makes it an awesome development tool and an excellent place to experiment with the language and test Python code on the fly.
To exit interactive mode, you can use one of the following options:

- quit() or exit(), which are built-in functions
- The Ctrl + Z and Enter key combination on Windows, or just Ctrl + D on Unix-like systems

Note: The first rule of thumb to remember when using Python is that if you're in doubt about what a piece of Python code does, then launch an interactive session and try it out to see what happens.

If you've never worked with the command-line or terminal, then you can try this:

- On Windows, the command-line is usually known as command prompt or MS-DOS console, and it is a program called cmd.exe. The path to this program can vary significantly from one system version to another.
  A quick way to get access to it is by pressing the Win + R key combination, which will take you to the Run dialog. Once you're there, type in cmd and press Enter.
- On GNU/Linux (and other Unixes), there are several applications that give you access to the system command-line. Some of the most popular are xterm, Gnome Terminal, and Konsole. These are tools that run a shell or terminal like Bash, ksh, csh, and so on.
  In this case, the path to these applications is much more varied and depends on the distribution and even on the desktop environment you use. So, you'll need to read your system documentation.
- On Mac OS X, you can access the system terminal from Applications → Utilities → Terminal.

4. How Does the Interpreter Run Python Scripts?

When you try to run Python scripts, a multi-step process begins. In this process the interpreter will:

1. **Process the statements of your script in a sequential fashion**
2. **Compile the source code to an intermediate format known as bytecode**
   This bytecode is a translation of the code into a lower-level language that's platform-independent. Its purpose is to optimize code execution. So, the next time the interpreter runs your code, it'll bypass this compilation step.

Strictly speaking, this code optimization is only for modules (imported files), not for executable scripts.

3. **Ship off the code for execution**
   At this point, something known as a Python Virtual Machine (PVM) comes into action. The PVM is the runtime engine of Python. It is a cycle that iterates over the instructions of your bytecode to run them one by one.
   The PVM is not an isolated component of Python. It's just part of the Python system you've installed on your machine. Technically, the PVM is the last step of what is called the Python interpreter.

The whole process to run Python scripts is known as the **Python Execution Model**.

Note: This description of the Python Execution Model corresponds to the core implementation of the language, that is, CPython. As this is not a language requirement, it may be subject to future changes.

5. How to Run Python Scripts Using the Command-Line

A Python interactive session will allow you to write a lot of lines of code, but once you close the session, you lose everything you've written. That's why the usual way of writing Python programs is by using plain text files. By convention, those files will use the .py extension. (On Windows systems the extension can also be .pyw.)
Python code files can be created with any plain text editor. You can use any editor you like.
To keep moving forward in this tutorial, you'll need to create a test script. Open your favorite text editor and write the following code:

```
1 #!/usr/bin/env python3
2
3 print('Hello World!')
```

Save the file in your working directory with the name hello.py. With the test script ready, you can continue reading.

6. Using the python Command

To run Python scripts with the python command, you need to open a command-line and type in the word python, or python3 if you have both versions, followed by the path to your script, just like this:

```
$ python3 hello.py
Hello World!
```

If everything works okay, after you press Enter, you'll see the phrase Hello World! on your screen. That's it! You've just run your first Python script!
If this doesn't work right, maybe you'll need to check your system PATH, your Python installation, the way you created the hello.py script, the place where you saved it, and so on. This is the most basic and practical way to run Python scripts.

7. Redirecting the Output

Sometimes it's useful to save the output of a script for later analysis. Here's how you can do that:

```
$ python3 hello.py > output.txt
```

This operation redirects the output of your script to output.txt, rather than to the standard system output (stdout). The process is commonly known as stream redirection and is available on both Windows and Unix-like systems.

If output.txt doesn't exist, then it's automatically created. On the other hand, if the file already exists, then its contents will be replaced with the new output.
Finally, if you want to add the output of consecutive executions to the end of output.txt, then you must use two angle brackets (>>) instead of one, just like this:

```
$ python3 hello.py >> output.txt
```

Now, the output will be appended to the end of output.txt.

8. Running Modules With the -m Option

Python offers a series of command-line options that you can use according to your needs. For example, if you want to run a Python module, you can use the command python -m <module-name>.
The -m option searches sys.path for the module name and runs its content as __main__:

```
$ python3 -m hello
Hello World!
```

Note: module-name needs to be the name of a module object, not a string.

9. Using the Script Filename

On recent versions of Windows, it is possible to run Python scripts by simply entering the name of the file containing the code at the command prompt:

```
C:\devspace> hello.py
Hello World!
```

This is possible because Windows uses the system registry and the file association to determine which program to use for running a particular file.
On Unix-like systems, such as GNU/Linux, you can achieve something similar. You'll only have to add a first line with the text #!/usr/bin/env python, just as you did with hello.py.
For Python, this is a simple comment, but for the operating system, this line indicates what program must be used to run the file.
This line begins with the #! character combination, which is commonly called **hash bang** or **shebang**, and continues with the path to the interpreter.
There are two ways to specify the path to the interpreter:

- **#!/usr/bin/python:** writing the absolute path
- **#!/usr/bin/env python:** using the operating system env command, which locates and executes Python by searching the PATH environment variable

This last option is useful if you bear in mind that not all Unix-like systems locate the interpreter in the same place.
Finally, to execute a script like this one, you need to assign execution permissions to it and then type in the filename at the command-line.
Here's an example of how to do this:

```
$ # Assign execution permissions
$ chmod +x hello.py
$ # Run the script by using its filename
$ ./hello.py
Hello World!
```

With execution permissions and the shebang line properly configured, you can run the script by simply typing its filename at the command-line.
Finally, you need to note that if your script isn't located at your current working directory, you'll have to use the file path for this method to work correctly.

10. How to Run Python Scripts Interactively

It is also possible to run Python scripts and modules from an interactive session. This option offers you a variety of possibilities.

11. Taking Advantage of import

When you import a module, what really happens is that you load its contents for later access and use. The interesting thing about this process is that import runs the code as its final step. When the module contains only classes, functions, variables, and constants definitions, you probably won't be aware that the code was actually run, but when the module includes calls to functions, methods, or other statements that generate visible results, then you'll witness its execution.
This provides you with another option to run Python scripts:

```
>>> import hello
Hello World!
```

You'll have to note that this option works only once per session. After the first import, successive import executions do nothing, even if you modify the content of the module. This is because import operations are expensive and therefore run only once. Here's an example:

```
>>> import hello  # Do nothing
>>> import hello  # Do nothing again
```

These two import operations do nothing, because Python knows that hello has already been imported.
There are some requirements for this method to work:

- The file with the Python code must be located in your current working directory.
- The file must be in the Python Module Search Path (PMSP), where Python looks for the modules and packages you import.

To know what's in your current PMSP, you can run the following code:

```
>>> import sys
>>> for path in sys.path:
...     print(path)
...
/usr/lib/python36.zip
/usr/lib/python3.6
/usr/lib/python3.6/lib-dynload
/usr/local/lib/python3.6/dist-packages
/usr/lib/python3/dist-packages
```

Running this code, you'll get the list of directories and .zip files where Python searches the modules you import.

12. Using importlib and imp

In the Python Standard Library, you can find importlib, which is a module that provides import_module().
With import_module(), you can emulate an import operation and, therefore, execute any module or script. Take a look at this example:

```
>>> import importlib
>>> importlib.import_module('hello')
Hello World!
<module 'hello' from '/home/username/hello.py'>
```

Once you've imported a module for the first time, you won't be able to continue using import to run it. In this case, you can use importlib.reload(), which will force the interpreter to re-import the module again, just like in the following code:

```
>>> import hello  # First import
Hello World!
>>> import hello  # Second import, which does nothing
>>> import importlib
>>> importlib.reload(hello)
Hello World!
<module 'hello' from '/home/username/hello.py'>
```

An important point to note here is that the argument of reload() has to be the name of a module object, not a string:

```
>>> importlib.reload('hello')
Traceback (most recent call last):
    ...
TypeError: reload() argument must be a module
```

If you use a string as an argument, then reload() will raise a TypeError exception.

Note: The output of the previous code has been abbreviated (...) in order to save space.

importlib.reload() comes in handy when you are modifying a module and want to test if your changes work, without leaving the current interactive session.
Finally, if you are using Python 2.x, then you'll have imp, which is a module that provides a function called reload(). imp.reload() works similarly to importlib.reload(). Here's an example:

```
>>> import hello  # First import
Hello World!
>>> import hello  # Second import, which does nothing
>>> import imp
>>> imp.reload(hello)
Hello World!
<module 'hello' from '/home/username/hello.py'>
```

In Python 2.x, reload() is a built-in function. In versions 2.6 and 2.7, it is also included in imp, to aid the transition to 3.x.

Note: imp has been deprecated since version 3.4 of the language. The imp package is pending deprecation in favor of importlib.

13. Using runpy.run_module() and runpy.run_path()

The Standard Library includes a module called runpy. In this module, you can find run_module(), which is a function that allows you to run modules without importing them first. This function returns the globals dictionary of the executed module.
Here's an example of how you can use it:

```
>>> runpy.run_module(mod_name='hello')
Hello World!
{'__name__': 'hello',
    ...
'_': None}}
```

The module is located using a standard import mechanism and then executed on a fresh module namespace.
The first argument of run_module() must be a string with the absolute name of the module (without the .py extension).
On the other hand, runpy also provides run_path(), which will allow you to run a module by providing its location in the filesystem:

```
>>>
>>> import runpy
>>> runpy.run_path(file_path='hello.py')
Hello World!
{'__name__': '<run_path>',
    ...
'_': None}}
```

Like run_module(), run_path() returns the globals dictionary of the executed module.
The file_path parameter must be a string and can refer to the following:

- The location of a Python source file
- The location of a compiled bytecode file
- The value of a valid entry in the sys.path, containing a __main__ module (__main__.pyfile)

14. Hacking exec()

So far, you've seen the most commonly used ways to run Python scripts. In this section, you'll see how to do that by using exec(), which is a built-in function that supports the dynamic execution of Python code.
exec() provides an alternative way for running your scripts:

```
>>> exec(open('hello.py').read())
'Hello World!'
```

This statement opens hello.py, reads its content, and sends it to exec(), which finally runs the code.
The above example is a little bit out there. It's just a "hack" that shows you how versatile and flexible Python can be.

15. Using execfile() (Python 2.x Only)

If you prefer to use Python 2.x, you can use a built-in function called execfile(), which is able to run Python scripts.
The first argument of execfile() has to be a string containing the path to the file you want to run. Here's an example:

```
>>> execfile('hello.py')
Hello World!
```

Here, hello.py is parsed and evaluated as a sequence of Python statements.

16. How to Run Python Scripts From an IDE or a Text Editor

When developing larger and more complex applications, it is recommended that you use an integrated development environment (IDE) or an advanced text editor.
Most of these programs offer the possibility of running your scripts from inside the environment itself. It is common for them to include a Run or Build command, which is usually available from the tool bar or from the main menu.
Python's standard distribution includes IDLE as the default IDE, and you can use it to write, debug, modify, and run your modules and scripts.
Other IDEs such as Eclipse-PyDev, PyCharm, Eric, and NetBeans also allow you to run Python scripts from inside the environment.

Advanced text editors like Sublime Text and Visual Studio Code also allow you to run your scripts.
To grasp the details of how to run Python scripts from your preferred IDE or editor, you can take a look at its documentation.

17. How to Run Python Scripts From a File Manager

Running a script by double-clicking on its icon in a file manager is another possible way to run your Python scripts. This option may not be widely used in the development stage, but it may be used when you release your code for production.
In order to be able to run your scripts with a double-click, you must satisfy some conditions that will depend on your operating system.
Windows, for example, associates the extensions .py and .pyw with the programs python.exeand pythonw.exe respectively. This allows you to run your scripts by double-clicking on them.
When you have a script with a command-line interface, it is likely that you only see the flash of a black window on your screen. To avoid this annoying situation, you can add a statement like input('Press Enter to Continue...') at the end of the script. This way, the program will stop until you press `Enter`.
This trick has its drawbacks, though. For example, if your script has any error, the execution will be aborted before reaching the input() statement, and you still won't be able to see the result.
On Unix-like systems, you'll probably be able to run your scripts by double-clicking on them in your file manager. To achieve this, your script must have execution permissions, and you'll need to use the shebang trick you've already seen. Likewise, you may not see any results on screen when it comes to command-line interface scripts.
Because the execution of scripts through double-click has several limitations and depends on many factors (such as the operating system, the file manager, execution permissions, file associations), it is recommended that you see it as a viable option for scripts already debugged and ready to go into production.

18. Conclusion

With the reading of this tutorial, you have acquired the knowledge and skills you need to be able to run Python scripts and code in several ways and in a variety of situations and development environments.

You are now able to run Python scripts from:

- The operating system command-line or terminal
- The Python interactive mode
- The IDE or text editor you like best
- The file manager of your system, by double-clicking on the icon of your script

These skills will make your development process much faster, as well as more productive and flexible.