01-How-to-Run-Python-Code

January 24, 2017

1 Finance 5350 - Fall 2016

Computational Financial Modeling

These notes are based off the material by Jake VanderPlas on his GitHub repository. A big thanks goes to Dr. VanderPlas for making these notebooks available under the "No Rights Reserved" CC0 license!

"No Rights Reserved" CC0

These notes form essentially the first chapter of his forthcoming book, which looks fantastic:

> A Whirlwind Tour of Python by Jake VanderPlas (O'Reilly). Copyright 2016 O'Reilly Media, Inc., 978-1-491-96465-1.

2 How to Run Python Code

Python is a flexible language, and there are several ways to use it depending on your particular task. One thing that distinguishes Python from other programming languages is that it is *interpreted* rather than *compiled*. This means that it is executed line by line, which allows programming to be interactive in a way that is not directly possible with compiled languages like Fortran, C, or Java. This section will describe four primary ways you can run Python code: the *Python interpreter*, the *IPython interpreter*, via *Self-contained Scripts*, or in the *Jupyter notebook*.

2.0.1 The Python Interpreter

The most basic way to execute Python code is line by line within the *Python interpreter*. The Python interpreter can be started by installing the Python language (see the previous section) and typing `python` at the command prompt (look for the Terminal on Mac OS X and Unix/Linux systems, or the Command Prompt application in Windows):

```
$ python
Python 3.5.1 |Continuum Analytics, Inc.| (default, Dec  7 2015, 11:24:55)
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

With the interpreter running, you can begin to type and execute code snippets. Here we'll use the interpreter as a simple calculator, performing calculations and assigning values to variables:

```
>>> 1 + 1
2
>>> x = 5
>>> x * 3
15
```

The interpreter makes it very convenient to try out small snippets of Python code and to experiment with short sequences of operations.

2.0.2 The IPython interpreter

If you spend much time with the basic Python interpreter, you'll find that it lacks many of the features of a full-fledged interactive development environment. An alternative interpreter called *IPython* (for Interactive Python) is bundled with the Anaconda distribution, and includes a host of convenient enhancements to the basic Python interpreter. It can be started by typing `ipython` at the command prompt:

```
$ ipython
Python 3.5.1 |Continuum Analytics, Inc.| (default, Dec  7 2015, 11:24:55)
Type "copyright", "credits" or "license" for more information.

IPython 4.0.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]:
```

The main aesthetic difference between the Python interpreter and the enhanced IPython interpreter lies in the command prompt: Python uses >>> by default, while IPython uses numbered commands (e.g. `In [1]:`). Regardless, we can execute code line by line just as we did before:

```
In [1]: 1 + 1
Out[1]: 2

In [2]: x = 5

In [3]: x * 3
Out[3]: 15
```

Note that just as the input is numbered, the output of each command is numbered as well. IPython makes available a wide array of useful features; for some suggestions on where to read more, see Resources for Further Learning.

2.0.3 Self-contained Python scripts

Running Python snippets line by line is useful in some cases, but for more complicated programs it is more convenient to save code to file, and execute it all at once. By convention, Python scripts are saved in files with a *.py* extension. For example, let's create a script called *test.py* which contains the following:

```
# file: test.py
print("Running test.py")
x = 5
print("Result is", 3 * x)
```

To run this file, we make sure it is in the current directory and type `python` *`filename`* at the command prompt:

```
$ python test.py
Running test.py
Result is 15
```

For more complicated programs, creating self-contained scripts like this one is a must.

2.0.4 The Jupyter notebook

A useful hybrid of the interactive terminal and the self-contained script is the *Jupyter notebook*, a document format that allows executable code, formatted text, graphics, and even interactive features to be combined into a single document. Though the notebook began as a Python-only format, it has since been made compatible with a large number of programming languages, and is now an essential part of the *Jupyter Project*. The notebook is useful both as a development environment, and as a means of sharing work via rich computational and data-driven narratives that mix together code, figures, data, and text.