CHAPTER

ONE

GETTING STARTED

1.1 Purpose of This Book

This book is written for those who want to use Python to perform scientific data analysis and create visually pleasing, publication-quality graphic plots of their data. It assumes familiarity with computer programming, though not necessarily with Python. The introductory chapters provide a quick start of Python syntax, data types, and control structures, while the later chapters focus on data analysis and plotting, primarily with the matplotlib, Numerical Python (NumPy), and Scientific Python (SciPy) libraries.

1.2 Why Python?

There are many computer languages that can be used for scientific data analysis and visualization. These include proprietary languages such as MATLAB, Mathematica, and IDL, for which a license must be purchased, as well as open-source languages such as Perl, Fortran, and C++, which are free for use. With enough ingenuity and patience, virtually any language can be used to perform a specific data analysis and visualization task. However, Python has certain advantages over other languages (in this author's opinion). These are:

- It is open source and does not require the purchase of a license for use. Anyone can download Python and the libraries that are used in this book. You can develop your own applications and code, without fear that if you move to a different job or school that you will no longer be able to use your programs.
- Python syntax is intuitive and easy to learn. There are no cumbersome braces or semicolons cluttering up the code. Python also forces the user to indent code blocks, making for clean-looking, easy-to-read programs.

- Python is an *interpreted* language, rather than a *compiled* language. In a compiled language the entire source code text file is read and then converted into executable code (in machine language) that can be directly executed by the operating system. This executable code is often optimized for the operating system. In an interpreted language the source code text file is read line-by-line and each statement converted into machine language and executed before the next line is read, or the text file is read completely and converted into an intermediate code that is then further converted into machine language for execution by the operating system.
  Compiled languages are generally more efficient and faster than interpreted languages, since the conversion to machine language only occurs once creating a machine language code that can be executed whenever the program needs to be run. However, interpreted languages such as Python are often more flexible and changes can be easily made to the program at runtime, without going through the extra step of compilation after every minor change to the code.
- Python is widely used and supported, and its use is growing rapidly. It is neither a fad nor a flash in the pan.

1.3 Python 2 versus Python 3

Python,[1] developed by Guido van Rossum, began in 1991 with version 0.9 and has transformed via dozens of subversions since. Python 2.0 appeared in 2001, with the latest version, 2.7, appearing in 2010. Python 3.0 appeared in 2008, and is *not* backward-compatible with older versions of Python. Notable differences are the syntax for the `print` statement and string formatting, but there are others as well. Because so many scientific libraries have been developed using the Python 2 paradigm, transition and acceptance of Python 3 in the scientific community has been slower than it might have been. Versions 2.6 and 2.7 can be thought of as "overlap" versions, allowing both the older 2.0 and newer 3.0 syntaxes to be used. In this book we focus on Python 2.7, using the newer Python 3.0 syntax. Any significant differences in syntax between Python 2 and Python 3 will be pointed out as they are encountered.

[1] Python is named after the British comedy troupe Monty Python.

1.4 A Few More Things About Python

1.4.1 Python is dynamically-typed

Most variables do not need to be declared as a specific data type (real, string, integer, etc.) before assigning values to them. This feature is known as *dynamic typing*, because the type of variable is determined completely by the type of data assigned. The type of a variable can be changed at any time simply by assigning a different value of a different type to it.

1.4.2 Python is case sensitive

Python syntax, commands, and variable names are all case sensitive. Thus, three variables named `Pressure`, `pressure`, and `PRESSURE` are three completely different variables. Likewise, the `print()` function is a valid Python statement, while `Print()` would throw an error.

1.4.3 Python is object oriented

Python was built from the ground up as an object-oriented language, though it can also be readily used for both procedural programming and functional programming. Much of what we discuss in this book can be learned without dwelling on the fact that Python is object oriented. Relevant object-oriented concepts will be explained as they appear.[2]

1.5 Notation

In this book we will identify Python statements, object names, code, string literals, user-typed input, and output written by a program using a typewriter typeface; for example, `print(a, b)`. When placeholder names are used for function arguments or in assignment statements, these will be italicized: *`fig, ax`* `= plt.subplots(`*`rows, cols`*`)`. Finally, we will indicate optional arguments for functions and methods using square brackets: `contour(`*`[x, y,] z`*`)`.

1.6 Getting Python

Python is open source and can be downloaded for and installed on any of the three main operating systems—Windows, OS X, and Linux. The individual components can be downloaded separately. There are also several pre-built packages assembled by various organizations and groups that install Python

[2] Those interested in quickly and effectively learning what object-oriented programming is all about are encouraged to read the excellent book, *The Object Oriented Thought Process*, by Matt Weisfeld, published by Addison-Wesley.

and a myriad of associated libraries. Some of these pre-built packages are open source, while others charge a fee. Keep in mind that you can *always* download and use Python and most libraries free of charge.

1.6.1 Pre-built packages

The easiest way to obtain and install Python and many of the analysis and plotting libraries useful for scientists is to download a pre-built package, such as either Anaconda or Canopy. The big advantage of pre-built packages is that you do not have to install individual libraries and be concerned with whether the versions of your libraries are compatible with each other or the base Python version. Anaconda and Canopy may also be installed concurrently on the same computer without interfering with one another (although you should always be aware of which one is the default Python installation).

Below is a list of all pre-built packages known to the author at the time this was written. The Python landscape changes rapidly, so the following list may not be current at the time of reading. This list is not exhaustive, and the packages appearing here are not necessarily endorsed by the author.

- **Anaconda**, *http://continuum.io/downloads.html*: Anaconda is distributed free of charge by Continuum Analytics. It includes a very extensive set of libraries. It is easy to install, robust, and is available for Windows, OS X, and Linux.
- **Canopy**, *http://www.enthought.com*: Canopy is distributed by Enthought, and has both a free version with some basic libraries and a commercial version with more extensive libraries. A free academic license for the full version is also available. Canopy is available for Windows, OS X, and Linux. One drawback is that this package is large, containing far more libraries than a single user is likely to ever need.
- **WinPython**, *http://code.google.com/p/winpython*: Windows only.
- **Python(x,y)**, *http://code.google.com/p/pythonxy*: Windows only.
- **Pyzo**, *http://www.pyzo.org*: Python 3 only; Windows and Linux.

1.6.2 Installing individual libraries

As an alternative to installing one of the above distributions, one can download basic Python via *www.python.org*. Additional libraries can be downloaded individually as needed. For the examples used in this book, SciPy, NumPy, matplotlib, and Basemap may be downloaded from the following sites:

- **NumPy:** *http://www.numpy.org*
- **SciPy:** *http://www.scipy.org*
- **matplotlib:** *http://www.matplotlib.org*
- **Basemap:** *http://www.matplotlib.org/basemap*

When downloading and installing individual components, it is very important to ensure that the version of the library is compatible with the version of Python (2.6, 2.7, 3.2, etc.) being used. Also, there are certain dependencies that may need to be met. For example, installing matplotlib requires that NumPy already be installed. Careful reading of the documentation is imperative.

A website of great usefulness for Windows users is maintained by Chris Gohlke at *http://www.lfd.uci.edu/~gohlke/pythonlibs/*. This site is a treasure trove of Python extension installers tailored for Windows.

1.7 Alternate Python Implementations

Because Python is open source, there is no monopoly on interpreters for Python code. Besides the standard implementation, CPython[3] (available from Python.org), several other interpreters and implementation are available:

- **Jython** (*http://www.jython.org*) is a version of Python using a Java-based interpreter and running on the Java Virtual Machine.
- **PyPy** (*http://pypy.org*) is an implementation optimized for speed and efficiency of computational resources.
- **IronPython** (*http://ironpython.codeplex.com*) is a version that is optimized for working within the .NET framework.
- **StacklessPython** (*http://www.stackless.com*) is based on CPython but is built for thread-based programming.

The version of Python discussed in this book is strictly the standard CPython implementation.

1.8 Running Python

If you are using Enthought Canopy, you can skip this section, as you can edit and run your programs directly in Canopy. If you installed Python yourself, you may want to read on.

1.8.1 Path settings

A standard Python installation creates a directory structure where the python interpreter and libraries are stored. On Windows, this is usually `\python27`.[4] For Linux and OS X installations, the directory is usually `/usr/bin/python`,

[3]The name CPython comes from the fact that the Python interpreter itself is written in the C programming language.

[4]The "27" refers to the version number, in this case Python 2.7.

/usr/local/bin/python, or /usr/local/python. Other Python installations may have a different directory name and structure. It is important to have the PATH environment variable set to include the path to the desired Python installation.

1.8.2 Testing the Python installation

Once Python is installed, it can be tested by one of several means.

Testing from the command prompt

One method of testing a standard Python installation is to open a terminal window, and type `python` at the prompt. This should bring up an interactive shell. If nothing happens, or an error appears, it is likely that the PATH variable is not set to include the proper directory (see previous section). If Python is installed properly and the correct path is set, text similar to `Python 2.7.5 (default, May 15 2013, 22:44:16) [MSC v.1500 64 bit (AMD64)] on win32` will appear.

Testing using IDLE

A standard Python installation comes with a built-in interactive development environment named IDLE. In Windows, this is located in the directory `\python27\lib\idlelib`.[5] It can be opened by either clicking on the file idle.bat or by typing `idle` at the command prompt.[6] If successful, a new window will open and display the Python version and other text similar to that displayed when testing from the command prompt.

1.8.3 Testing libraries

Most of the examples in this book use the SciPy, NumPy, and matplotlib libraries. To ensure that these libraries are properly installed, either open the interactive Python interpreter from the command line, or open IDLE, and then type `import numpy` and hit the <Enter> key.[7] If nothing happens other than a new prompt appearing, it means that NumPy is installed and

---

[5] Running IDLE from Enthought Canopy is more difficult because the directory structure of Canopy is nonstandard and unlike a normal Python installation. There is no need to use IDLE in Canopy, but should you want to you will have to find the path for the `idle.exe` file. This is accomplished by typing `sys.prefix` from the Canopy command prompt, which will return the path to the Python executable files `idle.exe`, `python.exe`, and `pythonw.exe`.

[6] On Windows, the idlelib directory must be in the PATH variable, or the entire path needs to be typed at the command prompt.

[7] Python is case sensitive, so this command must be all lower case.

available. If error messages appear, it means that either NumPy is not installed or that the version of NumPy is not compatible with the version of Python installed. Likewise, typing `import scipy` and `import matplotlib` will show whether these libraries are properly installed.

Later, we will be using Basemap. To check whether this module is properly installed, use the command

```
from mpl_toolkits.basemap import Basemap
```

1.9 Executing Python Code

Python code can be executed interactively either by typing a statement directly into the Python shell or via one of the interactive development environments. Python code can also be saved to a file for later execution. The file can then be executed by running it within one of the interactive development environments such as IDLE, Canopy, IPython, etc. The file can also be executed directly from the terminal window command line.

Python files are saved using either the `.py` or `.pyw` extension. The difference between the two extensions is solely whether or not a new terminal window is opened during execution of the code.

- When a file having a `.py` extension is selected to run, the Python interpreter is opened using the executable file `python.exe`. This opens a new terminal window, and all standard input/output is then accomplished via this terminal window. Any error messages will also be displayed to the open terminal window.
- When a file having a `.pyw` extension is selected, the Python interpreter is opened using the executable file pythonw.exe. In this case a terminal window is not opened. This is appropriate if the program does not need to display or print information to the computer screen, and does not require user input.

Once a Python program file has been saved with either the `.py` or `.pyw` extension, it can be executed via several different means.

Execution within a terminal window: Stored Python programs can be executed from a terminal window prompt by typing `python` *`prog_name`* where *`prog_name`* is the name of the stored program file. As long as the directory containing the Python interpreter executable files `python.exe` or `pythonw.exe` is in the path for the system, the system will locate these files, run the interpreter, and execute your program.

Typically the user need not worry about the location of the `python.exe` or `pythonw.exe` files, as the appropriate directory is normally added to the user's path variable when Python is installed. These files normally reside in a directory named Python*XX*, where *XX* is the version number of the Python installation. However, they may be located in other directories for nonstandard installations.[8]

Execution within an interactive development environment: You can open and edit programs within an interactive development environment such as Canopy, IDLE, Scite, IPython, etc. The development environment will either have a play button or a "Run" menu heading that can be selected to run the program.

Execution from a desktop icon: It is also possible to execute stored Python code by directly clicking on the desktop icon for the program file. In this case, the program icon must be associated with either the `python.exe` or `pythonw.exe` files. Clicking on the file will then load the Python interpreter and execute the commands within the program file.

Execution using IPython Notebook: Another possible means of executing Python code is IPython Notebook, which allows text, mathematical symbols, and Python code to be combined in a single document. This makes for ease of sharing computations and results, and is also very useful for classroom instruction. IPython Notebook files have a `.ipynb` file extension. Many Python installations, including Enthought Canopy, support IPython Notebooks. More information about using IPython Notebook can be found at *http://ipython.org/notebook.html*.

1.10 Additional Resources

There is considerable online documentation of Python and of the libraries discussed in this book, if one is patient enough to track it down. Much of the information is spread across a variety of unrelated sites, and explanations are often spotty. The purpose of this book is to bring the more essential information together into a coherent tutorial and reference. However, this book certainly is not a complete reference manual for Python or its libraries, and other resources can be indispensable, including those listed below.

Books. An exhaustive bibliography is not possible here; however, a few particularly notable books relevant to new scientific users of Python are worth mentioning.

---

[8] To find the locations of `python.exe` and `pythonw.exe` in Enthought's Canopy, type `sys.prefix`. This will return a directory path string, and the executables are contained within the *scripts* subdirectory of this path.

- An outstanding overall reference book for the Python language itself is the *Python Essential Reference* by David Beazley.
- A beginning book for those working mainly in the atmospheric and oceanic sciences is *A Hands-On Introduction to Using Python in the Atmospheric and Oceanic Sciences* by Johnny Wei-Bing Lin. Scientific users outside of these fields may also find this text useful.
- *High Performance Python* by Micha Gorelick and Ian Ozsvald emphasizes techniques for using Python for numerically intensive computations.
- *Effective Computation in Physics* by Anthony Scopatz and Kathryn D. Huff discusses Python as one of several software tools for general purpose computing in the physical sciences.
- Though not a Python-specific book, the aforementioned *The Object-oriented Thought Process* by Matt Weisfeld is an excellent resource for those wanting to master object-oriented process in any programming language.

Documentation web sites. Python and most libraries each have their own documentation web sites, some of which are listed here.

- **Python:** *http://docs.python.org/2/reference*
- **NumPy:** *http://docs.scipy.org/doc/numpy/reference*
- **SciPy:** *http://docs.scipy.org/doc/scipy/reference*
- **matplotlib:** *http://matplotlib.org*
- **IPython Notebook:** *http://ipython.org/notebook.html*

Documentation web sites for other libraries can easily be found using a web search.

User forums. There are several user forums for Python and its libraries. These can be found using a web search.