1 Interacting with Python and Basic Functions

In this chapter, we introduce the different ways you can interact with Python generally and some simple things you can do right from the get-go.

We start with how to install Python and issues concerned with different versions of the language. We then turn to how to invoke the interactive environment and the things you can do there. Finally, we briefly outline what it means to write a program.

1.1 Installing and Using Python

Python is a programming language. It is a way for you to talk to your computer, a way for you to get your computer to do things for you. It's basically a specialized language that is optimized for clarity for you and for converting into the internal language that your computer actually uses. You might think of it as a solution to a translation problem. You speak one language, and your computer speaks another. Python is a compromise language between the two. For you to program, you must translate your intentions into Python. For the computer to respond appropriately, it must translate Python into its own internal language.

To use Python it is not enough to simply translate your goals into Python. You must also install specialized software on your computer that will translate your Python code into what the computer can work with. We will refer to this software as a *Python installation*.

Sometimes this software is automatically part of your operating system. For example, Python is part of every MacOS and Linux installation. Sometimes this software has to be installed by you. Sometimes this software is free; some companies charge for it. My recommendation is that you at least start with a free version. If later on, when you really know what you're doing with Python, some proprietary version offers features that you feel are necessary, then that's the time a purchase may be warranted.

A related issue here is that there are different versions of Python. The language was first released in 1991, and there have been a number of different versions since then offering changes and improvement. As I write this, the current version is 3.6.2.

When you install Python, you may have a choice about versions. My recommendation is that you install something in the version 3 family. There are significant differences between versions 2 and 3, and version 3 is what we use here.

In general, you want the most recent version possible, but there are sometimes reasons not to do that. Some third-party modules are not available for all versions, and so if there is a particular module that is important to you, you may want to make sure you have a version of Python that works with it.

There are a number of free versions of Python that are available for Mac and Windows. For example, one widely used version is Anaconda.[1] Another widely used system is ActiveState.[2] Python.org[3] maintains a list of Python distributions and distribute their own as well. I've already mentioned that Python is part of any MacOS, but if you need a specific version of Python for your purposes, you can get those from MacPorts[4] or Homebrew.[5]

1.2 The Interactive Environment

There are at least four ways you can invoke Python:

(i) **interactive environment** terminal or Python window
(ii) **Idle** Python integrated development environment
(iii) **edit and run** write a program that you run in the terminal window
(iv) **Jupyter notebooks** write code that runs interactively in a web browser

Ultimately, we will want to write programs that we then run in the terminal window, but at this stage we will confine our attention to the interactive environment. This will allow us to play with Python so we can understand the basics before we move on to writing programs.

If Python is on your system – or you've properly installed it – you can start the interactive environment by simply typing `python` in the terminal window on a Mac or choosing Python from the start menu on Windows. This will produce a response like this:

```
> python
Python 3.4.6 ...
Type "help", "copyright", "credits" or
"license" for more information.
>>>
```

[1] www.continuum.io
[2] www.activestate.com
[3] wiki.python.org/moin/PythonDistributions
[4] www.macports.org
[5] brew.sh

The version of Python that you're using is displayed along with additional system information. All of this is followed by the prompt >>>. Your commands are typed at that prompt.

Before going any further, let's set out the most important commands that work here:

Quit: `quit()`. Type this at the >>> prompt to exit the interactive environment. Typing `^d` (`control-d`) has the same effect.

Help: `help()`. Typing this enters the online help system where you can get help on many aspects of using Python. While this is extremely helpful, the responses it provides may not be terribly useful at this stage. You exit the help system and return to the interactive environment with the `return` key.

Interrupt: `^c` (`control-c`). When we start playing with the system, you will occasionally get stuck in the middle of a command or while some command is running. If you do get stuck, or if some command seems to be running forever, you can often regain control and return to the >>> prompt by typing `^c`.

1.3 Basic Interactions

We will generally interact with Python by writing programs and then running them from the command line. At this point, however, let's try to understand Python a bit better in the interactive environment. In this environment, you can type legal Python statements and they are immediately evaluated. For example, you can perform mathematical calculations like multiplication, addition, and exponentiation:

```
>>> 4 * 7
28
>>> 3 + 2.9
5.9
>>> 9 ** -3
0.0013717421124828531
>>> 7 - 15
-8
```

As linguists, we will want to operate on words and sentences, and these must be entered in single or double quotes. In general terms, we will refer to these as *character strings*.

```
'This is a sentence'
"This is another one"
```

Interestingly, some of the mathematical operators above can be used with character strings as well and have different effects. With numbers + is addition,

but with strings it concatenates. With numbers * is multiplication, but with a string and a number it repeats the string:

```
>>> 'phon' + 'ology'
'phonology'
>>> 'phon' * 6
'phonphonphonphonphonphon'
```

There are some functions that operate directly on strings. The len() function returns the number of characters in a string.

```
>>> len('phonetics')
9
```

The type() function can operate on strings or numbers and returns the general "type" of the object it applies to.

```
>>> type(3)
<class 'int'>
>>> type(7.2)
<class 'float'>
>>> type('phoneme')
<class 'str'>
```

You can see that Python distinguishes integer numbers from floating point numbers (numbers with a decimal point) and from character strings.

Note now that the quotes surrounding a character string are essential and distinguish strings from other types. If you leave them out, Python will typically give you an error:

```
>>> phoneme
  Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  NameError: name 'phoneme' is not defined
```

If you put them in with numbers, then you change the basic type of the object. Consider the difference here:

```
>>> 7 + 4
11
>>> '7' + '4'
'74'
```

Recall that the + operator can apply to strings or numbers, but has different effects with each. When it applies to numbers, it performs addition. When it applies to strings, it performs string concatenation. We see here that when numbers are put in quotes they are treated as strings rather than as numbers.

There are functions for converting back and forth between character strings, integers, and floating point numbers:

```
>>> int('7')
7
>>> float('3.6')
3.6
>>> str(17)
'17'
```

Operators and functions can be combined as well. For example:

```
>>> (7 + 4) * 2
22
>>> str(7) + str(4)
'74'
>>> int(str(7)) ** 3
343
```

The format we've used for functions is to invoke the function by putting parentheses after its name, e.g., `str()`. There is another kind of function that we will frequently need that has a different syntax. These are *methods*, functions that are very specifically associated with some particular data type. We'll have a lot more to say about these in Chapter 10, but for now we just want to alert you that they exist and show you their syntax. To invoke one of these, the method name and parentheses are suffixed to the relevant object with a period. For example, the string method `upper()` returns an uppercase version of a string. We invoke it as follows:

```
>>> 'this is not a pipe'.upper()
'THIS IS NOT A PIPE'
```

There are methods that take additional arguments as well. For example, the method `count()` returns the number of instances of its argument in the string.

```
>>> 'this is not a pipe'.count('i')
3
```

Methods are *not* the same thing as functions; the different syntax above is required. Invoking them as if they were normal functions results in errors:

```
>>> upper('this is not a pipe')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'upper' is not defined

>>> count('this is not a pipe','i')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'count' is not defined
```

1.4 Edit and Run

In general, one uses Python to write programs to be run from the command line. For example, we can write a silly little program that concatenates two strings and prints the output. We would edit a text file, say with the name `interact1.py`, to contain the following:

```
print(4+7)
print('that should be 11')
```

interact1.py

Some parts of this are familiar from what we've covered so far, but some are not. We've added the `print()` function here, which simply prints its argument. Don't worry yet about the details; at this point, we just want several lines of code for our program.

It's important that you create this file with a simple text editor, not a full-on word processor like Microsoft Word. For writing code, I use Vim on a Mac, but Emacs (or Aquamacs) is a simple and free alternative. For Windows, there are various free ports of Emacs as well, but a number of other alternatives exist. It's most useful if the editor you use does *syntax colorizing*. This highlights different terms and strings in your Python code and makes it easier to spot errors. There are all sorts of other bells and whistles that different text editors offer, but the only essential ones at this point are (i) that the program be able to edit simple text files, and (ii) that it does syntax colorizing for Python.[6]

Once you've created the program with whatever text editor you opt for, there are a number of ways to invoke it. The simplest is to

[6] I strongly encourage you *not to* spend money on a text editor, at least at this stage. There are a huge number of free options.

type `python interact1.py` in the terminal or DOS window as follows:

```
> python interact1.py
11
That should be 11
```

If you invoke the program as above, it's important that the program be in the same directory that the terminal window is in. To find out what directory you are in, you can type `pwd` at the terminal prompt. To see if this program file is in that directory, you can use the `ls` command at the terminal prompt (`dir` for Windows). If it is not, you can either move it there or you can instruct the terminal to switch to another directory with the `cd` command (followed by the path to the directory you want to be in).

One can also add comments to a code file. Comments are lines of code that remind the programmer of what the code does or should do. Comments are marked with a # on their left. Everything on the same line after that # has no effect on the program. For example, we might tweak our program above like this:

```
print(4+7) #this does some math

#the following prints a string
print('that should be 11')
```

interact2.py

Notice that comments can occur on their own line or on the right side of an actual line of code.

1.5 Summary

In this chapter we have discussed how to install Python and the different versions that exist. We've also talked about how to run Python code and exemplified some basic commands in the interactive environment. We also briefly showed how to write and run a program.

1.6 Exercises

1.1 Write commands that print out your first name, the number of characters in that name, your last name, the number of characters in that name, and then concatenate and print the two names (with a space).

1.2 Write a line of code that will calculate how many times the string 'at' occurs in this sentence.

1.3 What's the difference between these?

```
'This is Mike'.upper().lower()
'This is Mike'.lower().upper()
```

1.4 Why does `upper('This is a cat')` not work?

1.5 What does `help(help)` do in the interactive environment?

1.6 We used the mathematical operator `**` on page 3 above without explanation. Play around with it and say what it does.

1.7 In math, `6 + 2` and `2 + 6` *mean* the same thing. We've seen that + and * can be used with strings too. What happens if arguments are reversed when strings are involved? Do those expressions *mean* the same thing?

1.8 Use the `help()` function to find out about the `round()` command. Explain and demonstrate how to use it with more than one argument to produce different results.

1.9 For each of the following, explain whether it's true or false and why:

(a) `'hat' == "hat"`
(b) `hat == 'hat'`
(c) `1/3 == .33`
(d) `'three' > 'two'`
(e) `2 + 2 = 4`

1.10 **Web**: Snoop around on the web and figure out what Jupyter notebooks are and how they work. Create a Jupyter notebook that answers one of the questions above.