

## Chapter objectives

- Learn about a small selection of “added extras” to Python which can help you build (and think about building) better code.

That just about concludes our look at the core concepts of Python. In this chapter, I’ll just be providing quick and easy reference material for the techniques on display, since by this point you’ll be comfortable with working with Python code and will really just need a brief explanation to get you started.

## 8.1 Installing Modules

One of the most useful affordances of Python as a programming language is that it has an incredible community of developers around it, producing an incredible array of tools and libraries – let’s call them modules – that can make the work of complex tasks (like using a social media API (application programming interface), or web scraping, or visualising data graphically, all of which are covered in this book) much more straightforward to code. Earlier in the book, I compared using techniques from modules as being like the difference between saying “you know that snow that’s also like rain, but also like ice and it comes down quick and hurts when it falls on you?” to simply saying “hailstones”. Modules give us easy-to-use “short-hand” code for doing complex tasks, which saves us having to build this kind of code from scratch – we’ll see some specific examples of this below. However, given that some of these modules aren’t “native” to Python (inasmuch as they don’t exist in the core Python language as you install it), we have to install them separately. You might see these called “third-party” modules. So here’s a little guide to the kinds of thing you’ll need to do to install Python modules on your system, via a demonstration of how I do these things on my Windows laptop.<sup>1</sup>

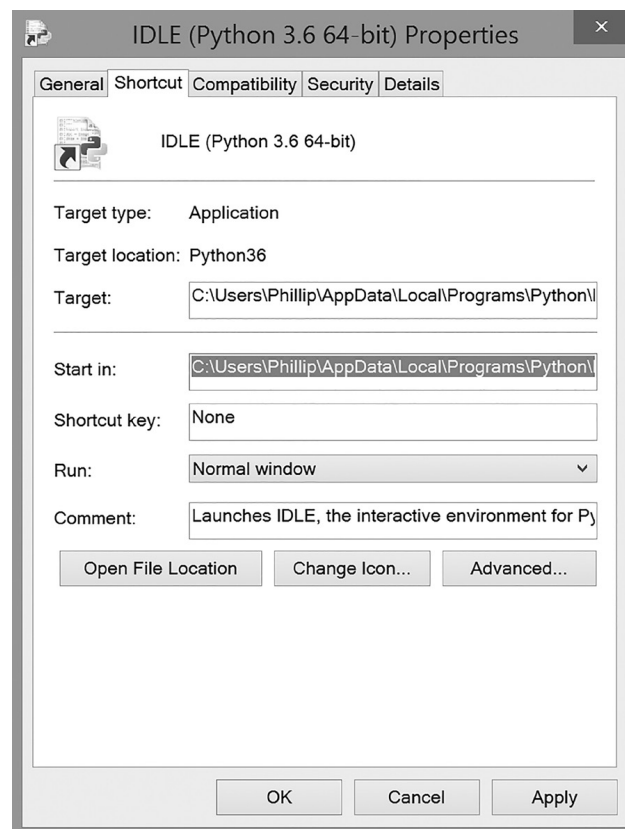
The application we need to install these Python packages is called “pip” (which stands for “Pip Installs Packages” – software developer humour “\\_(‘▽’)\_/”). Specifically, since we’re using Python 3, we need to use pip3. Handily for us, your Python installation will have already included pip3 as part of it, so we don’t need to install any extra stuff here. However,

<sup>1</sup>Like the rest of this book, this guide is written specifically from a Windows user’s perspective, though I will also comment on how Linux users might apply these same things at the end of this section for reasons that will become apparent then. Again, this is due to the unavoidable tension between having to write instructions specific enough that they can be illustrative of the actual practical work they’re intended to instruct in, whilst being general enough that they can be useful in the different contexts that readers no doubt want to apply them to. For the same reason, I should also note that as with some of the information elsewhere in this book, I have to deal with the “problem” that it’s practically impossible to write a bespoke installation routine for specific users on specific operating systems with specific directory structures for specific module installations. However, the intention is that users of any operating system will be able to see how they might go about finding out the things they need to know in order to install the modules they need – if the step-by-step instructions don’t work for you, try to make them make sense for your own specific context.

the way we're going to access this pip3 application is quite different than how we might access other applications we're more familiar with like web browsers or word processors. Whereas a web browser or word processor has an icon to click and a fancy visually oriented user interface to help us use the program, we have to access pip3 in a more "old-school" way – through a command-line interface, where we type in text instructions to tell the computer what to do. So we'll learn just what we need to know about how to use the command-line interface in Windows for the specific purpose of installing some add-on Python packages.<sup>2</sup>

## Locating the Python Directory

Before we even think about getting into the command-line interface, though, we need to first figure out where our pip3 application is located, so that we can navigate to it and execute it to



**Figure 8.1** The Properties menu

<sup>2</sup>As in the previous footnote, this is specific to Windows (and specific to my own laptop in some respects, too) – so if you're on another type of operating system, the exact command-line language I'm using might not apply. However, you should be able to see the rough structure of what I'm trying to do, and, by this point, be willing to get more comfortable with looking up extra information online to find out how you would apply those things to the specifics of your own system.

install the packages we want to install. A relatively straightforward way of doing this is to find your Python3 IDLE icon – the one that you might click to open up the Python shell – right-click on it, then select the `Properties` menu. A menu should pop up which looks something like Figure 8.1.

The `Start in:` field is what we’re interested in here (which is why it’s highlighted in Figure 8.1) – this tells us where our Python 3 application is stored. Specifically, we’re interested in where the `Python36` folder is, since within that there is a folder called `Scripts` where our `pip3` application lives. And in order to run `pip3` using a command-line interface, we need to make sure we are in the same folder as it. So, either keep the `Properties` menu open to refer to, or copy across the directory information so we can refer to it as we move on. For me, the directory information given in `Target:` is

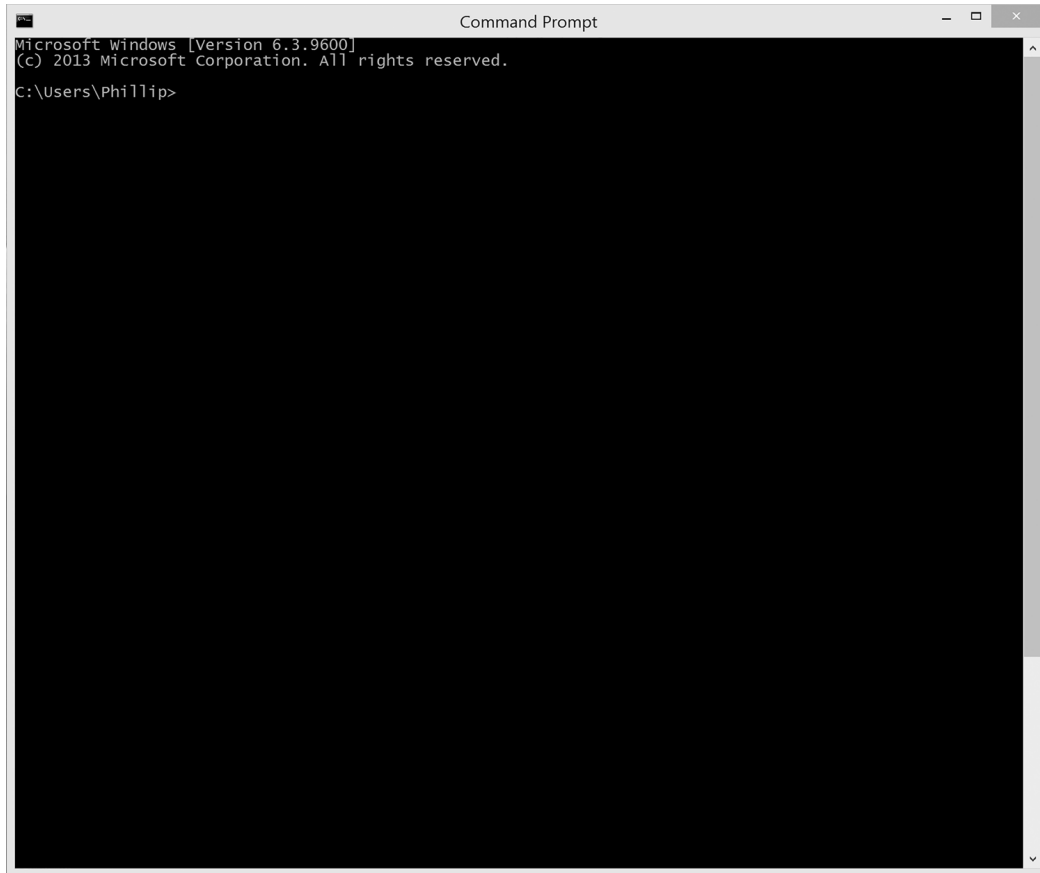
```
1 C:\Users\Phillip\AppData\Local\Programs\Python\Python36\
```

The version I’m using is Python 3.6 – hence why I have a folder called `Python36`. Of course, you are unlikely to have the same directory structure as I do on your own computer (even if you share the name “Phillip”). So do this for yourself, and note down your own details to be used later, though I’ll talk about my details for ease of reference from here on.

## Navigating to the Python Directory with Command Prompt

The next step is to get into the command-line interface and use it to find our way to the `Scripts` folder in our Python 3 directory, and to do that, we’ll need to open up the Windows Command Prompt program. Any of the common desktop or laptop installations of Windows will have a version of Command Prompt lurking somewhere – typically this is going to be found in the system tools or somewhere similar, and typically you’ll also have some kind of capacity to search for things on your system (in which case, running a search for “Command Prompt” will point you in the right direction). So, open Command Prompt, and you should see a screen which looks like Figure 8.2.

This is where we’re going to type our instructions to navigate to the `Script` folder, and where we’re going to tell `pip3` what we want it to do. Notice in Figure 8.2 the line of text that says `C:\Users\Phillip>` – this is the line that we will type our instructions into, and handily, it also tells us where we currently are in the directory (i.e. we are in the `Phillip` folder in `Users`, which is in the `C:` drive of my laptop). This is just another way of accessing folders on your computer; here we’re just seeing the folders in text form rather than as we might usually see them in visual form as icons, say if we went to `My Computer`, went into the `C:` drive, then went into `Users` and `Phillip` that way. If you try this on your own computer, you’ll see that you can move between folders by double-clicking them, and there may be folders within folders and files within those folders – effectively, you’re moving your way through a hierarchy of folders. We’re going to do the exact same thing, but instead of having visual icons to click, we’re going to type out instructions in Command Prompt. So Command Prompt is telling us where we currently are, and from the information in the `Properties` menu we already have, we know where we need to be. Now, how do we *get* there?



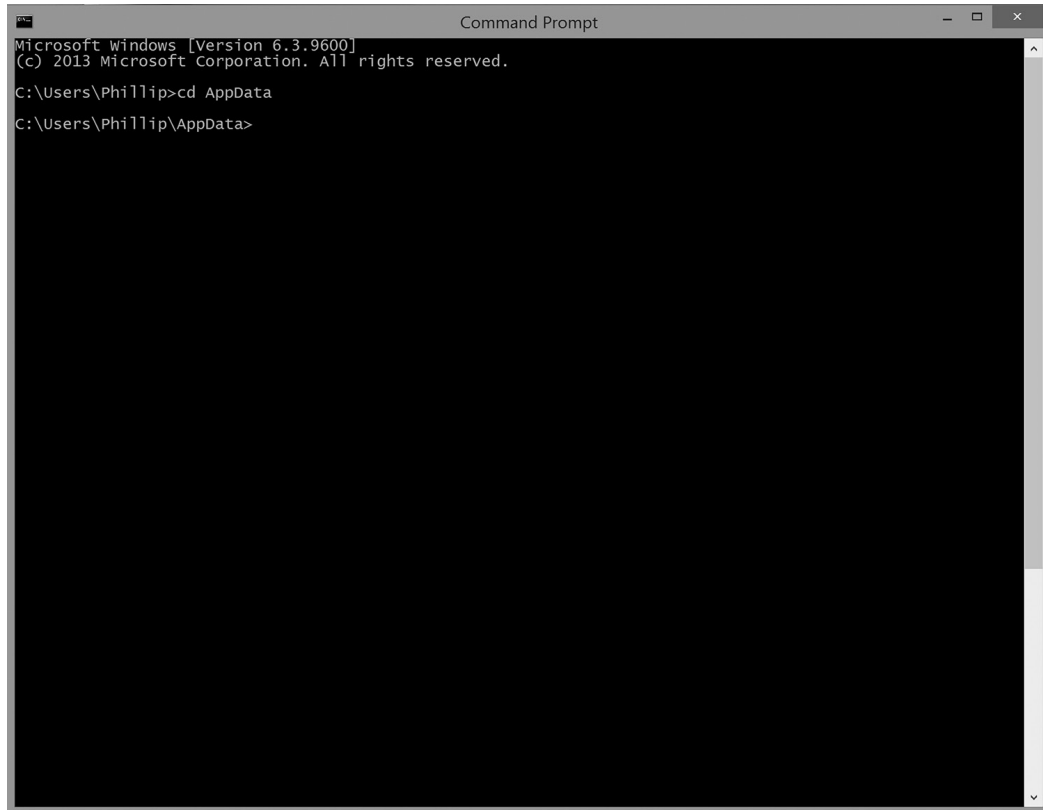
**Figure 8.2** Command Prompt: our command-line interface

There are two commands that are useful here, the first of which we use to change directory (i.e. to move between folders). To do this, we have to type:

```
1 cd insert_name_of_folder
```

The command `cd` just means “change directory” – this is the instruction you’re giving to your Command Prompt, followed by the name of the directory you want to change to. In the present case, given the information we have about where my copy of Python is located on my laptop (Figure 8.1), I know that since I am already in `C:\Users\Phillip\` (where the backslashes denote a boundary between folders) the next folder I want to move to is called `AppData`, Figure 8.3 shows what I have to type to do that:

All I’ve done here is type `cd AppData`, and I’ve jumped to the `AppData` folder – great! Note, however, that there is a space separating the instruction (`cd`) from the argument we want that instruction to apply to (`AppData`). Spaces are treated as breaks in a command-line interface, which makes things a bit trickier to deal with if we have folders and files that have names that contain spaces – hence, if you’re ever naming a file yourself and might end up using a command-line interface to play around with it, it’s worth using underscores to



**Figure 8.3** We're moving between folders!

indicate spaces, or just not using spaces generally in your naming conventions. Also note, we *do* need to use the capital letters in AppData, otherwise Command Prompt won't recognise the folder – information and instructions are case-sensitive here, which means we need to be sensitive to cases too. Also also note, now the line I'm typing information into shows the updated directory to tell me where I currently am in the hierarchy of folders – handy to know! Great, we're moving closer to our Python3 directory! But before we go on, here are a few more things we can do with `cd`. In the example above, I'm in a folder called `Phillip` and I want to go to a folder within `Phillip` called `AppData`. But what can I do if I want to go back out of `AppData` to `Phillip` again? Here, we can use:

```
1 cd ..
```

Again we're using `cd` to change the directory, but instead of giving a folder name to take us further into the hierarchy of folders, we're using two full stop marks to take us in the other direction. So if you make a mistake and move to a directory you didn't want to be in, you can always get back out again.

However, moving folders one by one can be a bit tedious – this is much the same even when we're doing this visually with icons, where we might end up having to laboriously click icons lots of times to get where we want. However, we can use the command-line interface to take

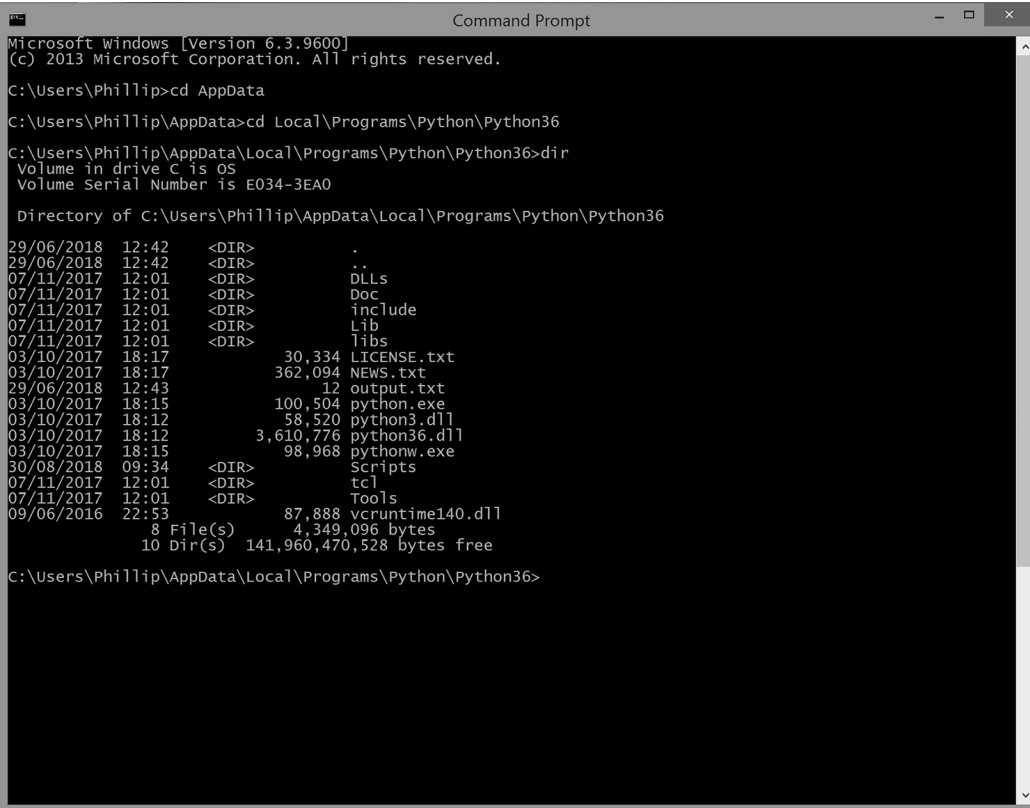
ourselves to exactly where we want to be in just one line of instruction. To do this, we can use backslashes to list the series of moves we want to make and perform them all at once, as follows:<sup>3</sup>

```
cd Local\Programs\Python\Python36\
```

This neatly leads me on to the next Command Prompt instruction we might find useful here. We're now in the `Python36` folder (which is where my Python 3 installation is located), but how do I know where to go next? Wouldn't it be *really* handy if I could see a list of everything contained in the folder I am currently in? Well, here's how to do that:

```
1 dir
```

Easy as that – `dir` is just short for directory, and we're just asking to see the contents of the current directory here. Figure 8.4 shows what happens when I type `dir` into the Command Prompt at my current location.



```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\Phillip>cd AppData
C:\Users\Phillip\AppData>cd Local\Programs\Python\Python36
C:\Users\Phillip\AppData\Local\Programs\Python\Python36>dir
Volume in drive C is OS
Volume Serial Number is E034-3EA0

Directory of C:\Users\Phillip\AppData\Local\Programs\Python\Python36

29/06/2018  12:42    <DIR>          .
29/06/2018  12:42    <DIR>          ..
07/11/2017  12:01    <DIR>          DLLs
07/11/2017  12:01    <DIR>          Doc
07/11/2017  12:01    <DIR>          include
07/11/2017  12:01    <DIR>          Lib
07/11/2017  12:01    <DIR>          libs
03/10/2017  18:17             30,334 LICENSE.txt
03/10/2017  18:17             362,094 NEWS.txt
29/06/2018  12:43                12 output.txt
03/10/2017  18:15             100,504 python.exe
03/10/2017  18:12             58,520 python3.dll
03/10/2017  18:12             3,610,776 python36.dll
03/10/2017  18:15             98,968 pythonw.exe
30/08/2018  09:34    <DIR>          Scripts
07/11/2017  12:01    <DIR>          tcl
07/11/2017  12:01    <DIR>          Tools
09/06/2016  22:53             87,888 vcruntime140.dll
                8 File(s)      4,349,096 bytes
               10 Dir(s)    141,960,470,528 bytes free

C:\Users\Phillip\AppData\Local\Programs\Python\Python36>
```

**Figure 8.4** A list of directories in my Python36 folder

<sup>3</sup>For the purposes of this example, I'm working on the assumption that I'm still in the `AppData` folder, which is to say that my starting directory is `C:\Users\Phillip\AppData`. As such, any moving between folders I'm doing has to take into account where I'm starting from.

We can see a list of everything contained in my `Python36` folder, and of particular interest to us is the `Scripts` folder, since this is where we'll find the `pip3` application. So let's navigate to `Scripts` and use `dir` to double-check if `pip3` is in there (Figure 8.5).

```

Command Prompt

Directory of C:\Users\Phillip\AppData\Local\Programs\Python\Python36
29/06/2018 12:42 <DIR>      .
29/06/2018 12:42 <DIR>      ..
07/11/2017 12:01 <DIR>      DLLs
07/11/2017 12:01 <DIR>      Doc
07/11/2017 12:01 <DIR>      include
07/11/2017 12:01 <DIR>      Lib
07/11/2017 12:01 <DIR>      libs
03/10/2017 18:17          30,334 LICENSE.txt
03/10/2017 18:17        362,094 NEWS.txt
29/06/2018 12:43          12 output.txt
03/10/2017 18:15       100,504 python.exe
03/10/2017 18:12       58,520 python3.dll
03/10/2017 18:12     3,610,776 python36.dll
03/10/2017 18:15       98,968 pythonw.exe
30/08/2018 09:34 <DIR>      Scripts
07/11/2017 12:01 <DIR>      Tcl
07/11/2017 12:01 <DIR>      Tools
09/06/2016 22:53     87,888 vcruntime140.dll
          8 File(s)      4,349,096 bytes
        10 Dir(s)  141,960,470,528 bytes free

C:\Users\Phillip\AppData\Local\Programs\Python\Python36>cd Scripts
C:\Users\Phillip\AppData\Local\Programs\Python\Python36\Scripts>dir
Volume in drive C is OS
Volume Serial Number is E034-3EA0

Directory of C:\Users\Phillip\AppData\Local\Programs\Python\Python36\Scripts
30/08/2018 09:34 <DIR>      .
30/08/2018 09:34 <DIR>      ..
30/08/2018 09:34     102,796 chardetect.exe
27/07/2018 14:05       98,195 conv-template.exe
07/11/2017 12:01       98,197 easy_install-3.6.exe
07/11/2017 12:01       98,197 easy_install.exe
27/07/2018 14:05       98,185 f2py.exe
27/07/2018 14:05        836 f2py.py
27/07/2018 14:05       98,195 from-template.exe
01/08/2018 10:15      102,787 pip.exe
01/08/2018 10:15      102,787 pip3.6.exe
01/08/2018 10:15      102,787 pip3.exe
27/07/2018 14:05 <DIR>      __pycache__
          10 File(s)      902,962 bytes
          3 Dir(s)  141,927,297,024 bytes free

C:\Users\Phillip\AppData\Local\Programs\Python\Python36\Scripts>

```

**Figure 8.5** The Contents of Scripts

Excellent, we're in the place we need to be, with the tools we need to use – we can see that there is an executable file (i.e. an application, as denoted by the `.exe` suffix) called `pip3`. Now let's use it to install some packages!

## Using `pip3` to Install Modules

Having found our way to where `pip3` is located, the rest is really simple. All we need to do now is type the following into Command Prompt:

```
1 pip3 install name_of_python_package
```

So here, we're calling on the `pip3` application, telling it we want to install something, and then telling it the name of what we want to install. When `pip3` has all this information, it will use your web connection to identify and install the package in question from the Python Package Index (PyPI), which is a huge repository of Python packages that people have developed and

made available publicly to support lots of different types of Python programming tasks – more information on the PyPI and its contents can be found on the PyPI website (Python Software Foundation, 2019c). Of course, we’re not interested in *all* of these packages, but only in the ones that will help us do some social scientific tasks – while we’re in the right location, let’s install all the different Python packages we’re going to need throughout the rest of the book. So, what we need to do here is type out the following commands:

```
1 pip3 install bs4
2 pip3 install lxml
3 pip3 install matplotlib
4 pip3 install pandas
5 pip3 install tweepy
```

You should press return after each of these commands and wait until they have successfully installed before moving on to the next one.<sup>4</sup> Installing all five now will save you having to come back to the command prompt multiple times later on. You don’t really need to know what these names refer to at this point, but for future reference, `bs4` is the PyPI name for the Beautiful Soup package which we’ll use for web scraping, `lxml` is a package for processing data that comes in XML format, `matplotlib` and `pandas` are to do with managing and visualising data for analysis, and `tweepy` is a package that helps us play around with Twitter as a data source and service. Each of these packages is identifiable in PyPI by the name given in the command-line excerpt above; these names can easily be found out by visiting the web-pages for each of the packages and looking at their respective “how to install this package” sections (so, if you ever need to find out the PyPI name of a package you want to install, it’s information you can easily locate with a search engine).

And that’s that – everything you need to work your way through the rest of the book is now installed and ready to use!

## Summing Up

The whole process described above can be boiled down to a series of steps to follow, where you have to fill in the gaps that make the steps relevant to whatever system you’re working on (unless you’re using a Linux-based operating system, in which case, see the note following this summary):

---

<sup>4</sup>If these packages *don’t* successfully install, then unfortunately there’s not much I can do in the pages of a book – the reasons why an installation might fail are local to specific computers and therefore there are no real general guidelines I can give, other than try to read the error messages that happen, and use a search engine to find people who have used programming question-and-answer platforms such as Stack Exchange and Stack Overflow to get support when they’ve had the same problem. That’s how a lot of programming issues are resolved really, so it’s not bad practice to get familiar with the idea of tweaking and applying somebody else’s solution to your own specific problem. However, I will say that some of these chapters are entirely premised on you having the appropriate packages installed, to the extent that if you *don’t* have those packages installed you won’t be able to get anything out of those chapters. So do try to find a solution if you run into problems here.



1. Figure out where your Python 3 directory is - you'll need to know where the `Scripts` file is within that, since that is where you will find your `pip3` application.
2. Open up a command-line interface (such as Command Prompt) and navigate to the `Scripts` file.
3. Use the `pip3` application contained therein to install your Python libraries.

## A Final Note for Linux Users

The intention throughout this section has been to focus on the specifics of my own (Windows) system to give a general flavour of what you need to know and do to install modules for any operating system you might be using. However, it's also worth commenting briefly on how to do the same work on Linux-based operating systems, just because it's very simple. Here's how you install all these libraries on a Linux-based operating system:

1. Open the Terminal
2. Type in the following commands (do each one separately, that is, press return at the end of each line below):

```
1 sudo pip3 install bs4
2 sudo pip3 install lxml
3 sudo pip3 install matplotlib
4 sudo pip3 install pandas
5 sudo pip3 install tweepy
```

That's it! There *are* some differences in how you code in Python between Windows and Linux<sup>5</sup> which mean that the odd point in this book (which is Windows-oriented) would need tweaking to suit a Linux system, but that being said, see how much more straightforward using Linux can be in some respects, especially when it comes to having a bit of extra control over how your computer works? Worth thinking about, now that you'll be routinely getting deeper into your computer as a Python programmer!

## 8.2 Importing Modules

We have already seen how to use various objects, commands, methods and so on within Python. However, there are many different techniques that we *could* use which are not stored by default in the core Python language. There are lots of extra bits of code that Python can use as “modules” – banks of “add-on” techniques and methods and so on which we can bring into the main Python language as and when we require them. Some of these we'll need to

---

<sup>5</sup>For instance, I got horrendously stuck trying to figure out why a graph wouldn't save as an image on Windows, only to later realise that the code I wrote on my little (Linux) Raspberry Pi microcomputer needed an ever-so-slight tweaking to make it fit its new Windows environment. It still infuriates me how long it took me to figure that out. So when you come to the chapter on data visualisation with Matplotlib later in the book, be forewarned that you might be able to detect the faintest whiff of “absolutely done with this”.