An introduction to Python Programming for Research

James Hetherington

September 25, 2018

Contents

Chapter 1

Introduction to Python

1.1 Introduction

1.1.1 Why teach Python?

- In this first session, we will introduce Python.
- This course is about programming for data analysis and visualisation in research.
- It's not mainly about Python.
- But we have to use some language.

1.1.2 Why Python?

- Python is quick to program in
- Python is popular in research, and has lots of libraries for science
- Python interfaces well with faster languages
- Python is free, so you'll never have a problem getting hold of it, wherever you go.

1.1.3 Why write programs for research?

- Not just labour saving
- Scripted research can be tested and reproduced

1.1.4 Sensible Input - Reasonable Output

Programs are a rigorous way of describing data analysis for other researchers, as well as for computers.

Computational research suffers from people assuming each other's data manipulation is correct. By sharing codes, which are much more easy for a non-author to understand than spreadsheets, we can avoid the "SIRO" problem. The old saw "Garbage in Garbage out" is not the real problem for science:

- Sensible input
- Reasonable output

1.2 Many kinds of Python

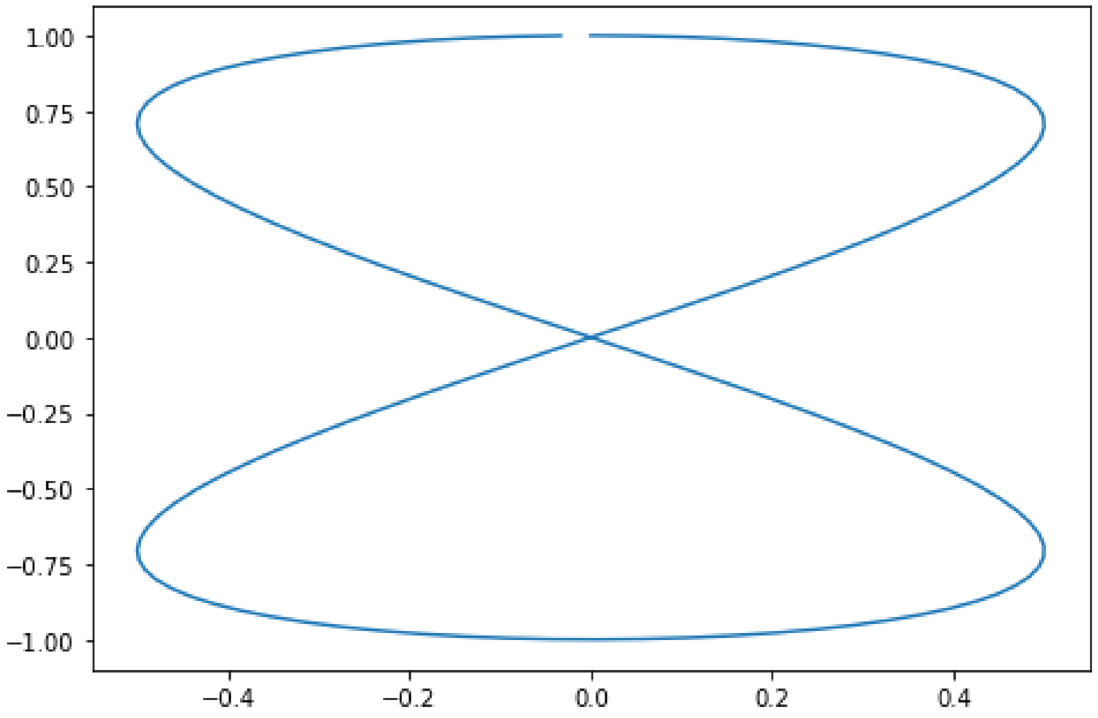
1.2.1 The Jupyter Notebook

The easiest way to get started using Python, and one of the best for research data work, is the Jupyter Notebook.

In the notebook, you can easily mix code with discussion and commentary, and mix code with the results of that code; including graphs and other data visualisations.

```
In [1]: ### Make plot
        %matplotlib inline
        import math

        import numpy as np
        import matplotlib.pyplot as plt

        theta = np.arange(0, 4 * math.pi, 0.1)
        eight = plt.figure()
        axes = eight.add_axes([0, 0, 1, 1])
        axes.plot(0.5 * np.sin(theta), np.cos(theta / 2))
```

```
Out[1]: [<matplotlib.lines.Line2D at 0x2ab0f9f11748>]
```



We're going to be mainly working in the Jupyter notebook in this course. To get hold of a copy of the notebook, follow the setup instructions shown on the course website, or use the installation in Desktop@UCL (available in the teaching cluster rooms or anywhere).

Jupyter notebooks consist of discussion cells, referred to as "markdown cells", and "code cells", which contain Python. This document has been created using Jupyter notebook, and this very cell is a **Markdown Cell**.

```
In [2]: print("This cell is a code cell")
```

```
This cell is a code cell
```

Code cell inputs are numbered, and show the output below.
Markdown cells contain text which uses a simple format to achive pretty layout, for example, to obtain:
bold, *italic*

- Bullet

> Quote

We write:

```
**bold**, *italic*

* Bullet

> Quote
```

See the Markdown documentation at This Hyperlink

1.2.2 Typing code in the notebook

When working with the notebook, you can either be in a cell, typing its contents, or outside cells, moving around the notebook.

- When in a cell, press escape to leave it. When moving around outside cells, press return to enter.
- Outside a cell:
- Use arrow keys to move around.
- Press `b` to add a new cell below the cursor.
- Press `m` to turn a cell from code mode to markdown mode.
- Press `shift+enter` to calculate the code in the block.
- Press `h` to see a list of useful keys in the notebook.
- Inside a cell:
- Press `tab` to suggest completions of variables. (Try it!)

Supplementary material: Learn more about Jupyter notebooks.

1.2.3 Python at the command line

Data science experts tend to use a "command line environment" to work. You'll be able to learn this at our "Software Carpentry" workshops, which cover other skills for computationally based research.

```
In [3]: %%bash
        # Above line tells Python to execute this cell as *shell code*
        # not Python, as if we were in a command line
        # This is called a 'cell magic'

        python -c "print(2 * 4)"
```

```
8
```

1.2.4 Python scripts

Once you get good at programming, you'll want to be able to write your own full programs in Python, which work just like any other program on your computer. Here are some examples:

```
In [4]: %%bash
        echo "print(2 * 4)" > eight.py
        python eight.py
```

```
8
```

We can make the script directly executable (on Linux or Mac) by inserting a hashbang and setting the permissions to execute.

```
In [5]: %%writefile fourteen.py
        #! /usr/bin/env python
        print(2 * 7)
```

```
Writing fourteen.py
```

```
In [6]: %%bash
        chmod u+x fourteen.py
        ./fourteen.py
```

```
14
```

1.2.5 Python Libraries

We can write our own python libraries, called modules which we can import into the notebook and invoke:

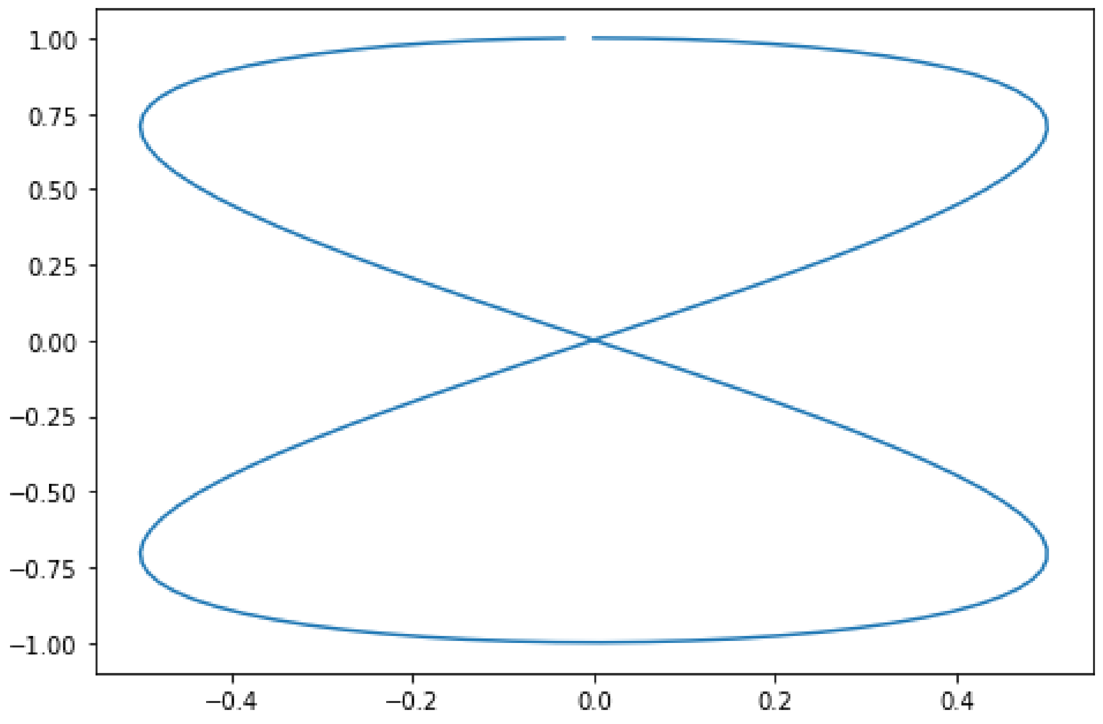
```
In [7]: %%writefile draw_eight.py
        # Above line tells the notebook to treat the rest of this
        # cell as content for a file on disk.
        import math

        import numpy as np
        import matplotlib.pyplot as plt

        def make_figure():
            theta = np.arange(0, 4 * math.pi, 0.1)
            eight = plt.figure()
            axes = eight.add_axes([0, 0, 1, 1])
            axes.plot(0.5 * np.sin(theta), np.cos(theta / 2))
            return eight
```

```
Writing draw_eight.py
```

In a real example, we could edit the file on disk using a program such as Atom or VS code.

```
In [8]: import draw_eight # Load the library file we just wrote to disk
```

```
In [9]: image = draw_eight.make_figure()
```



There is a huge variety of available packages to do pretty much anything. For instance, try `import antigravity`.

The `%%` at the beginning of a cell is called *magics*. There's a large list of them available and you can create your own.

1.3 An example Python data analysis notebook

1.3.1 Why write software to manage your data and plots?

We can use programs for our entire research pipeline. Not just big scientific simulation codes, but also the small scripts which we use to tidy up data and produce plots. This should be code, so that the whole research pipeline is recorded for reproducibility. Data manipulation in spreadsheets is much harder to share or check.

You can see another similar demonstration on the software carpentry site. We'll try to give links to other sources of Python training along the way. Part of our approach is that we assume you know how to use the internet! If you find something confusing out there, please bring it along to the next session. In this course, we'll always try to draw your attention to other sources of information about what we're learning. Paying attention to as many of these as you need to, is just as important as these core notes.

1.3.2 Importing Libraries

Research programming is all about using libraries: tools other people have provided programs that do many cool things. By combining them we can feel really powerful but doing minimum work ourselves. The python syntax to import someone else's library is "import".

```
In [1]: import geopy # A python library for investigating geographic information.
        # https://pypi.python.org/pypi/geopy
```

Now, if you try to follow along on this example in an IPython notebook, you'll probably find that you just got an error message.

You'll need to wait until we've covered installation of additional python libraries later in the course, then come back to this and try again. For now, just follow along and try get the feel for how programming for data-focused research works.

```
In [2]: geocoder = geopy.geocoders.Yandex(lang="en_US")
        geocoder.geocode('Cambridge', exactly_one=False)
```

```
Out[2]: [Location(Cambridge, Cambridgeshire County, United Kingdom, (52.208145, 0.133023, 0.0)),
         Location(Cambridge, Ontario, Canada, (43.370599, -80.318989, 0.0)),
         Location(Cambridge, Middlesex County, Massachusetts, United States of America, (42.385899, -71
         Location(Cambridge, Washington County, State of Idaho, United States of America, (44.572039, -
         Location(Cambridge, Isanti County, Minnesota, United States of America, (45.572595, -93.223783
         Location(Cambridge, East London, Republic of South Africa, (-32.978267, 27.884695, 0.0)),
         Location(Cambridge, Waikato, New Zealand, (-37.889307, 175.465014, 0.0)),
         Location(Cambridge, Cambridgeshire County, United Kingdom, (52.210303, 0.176447, 0.0)),
         Location(Cambridge Bay, Nunavut, Canada, (69.11673, -105.067782, 0.0)),
         Location(Cambridge, Saint James, Jamaica, (18.291699, -77.895883, 0.0))]
```

The results come out as a **list** inside a list: `[Name, [Latitude, Longitude]]`. Programs represent data in a variety of different containers like this.

1.3.3 Comments

Code after a # symbol doesn't get run.

```
In [3]: print("This runs") # print "This doesn't"
        # print This doesn't either
```

```
This runs
```

1.3.4 Functions

We can wrap code up in a **function**, so that we can repeatedly get just the information we want.

```
In [4]: def geolocate(place):
            return geocoder.geocode(place, exactly_one = False)[0][1]
```

Defining **functions** which put together code to make a more complex task seem simple from the outside is the most important thing in programming. The output of the function is stated by "return"; the input comes in in brackets after the function name:

```
In [5]: geolocate('Cambridge')
```

```
Out[5]: (52.208145, 0.133023)
```

1.3.5 Variables

We can store a result in a variable:

```
In [6]: london_location = geolocate("London")
        print(london_location)
```

```
(51.507351, -0.12766)
```

1.3.6 More complex functions

The google maps API allows us to fetch a map of a place, given a latitude and longitude. The URLs look like: http://maps.googleapis.com/maps/api/staticmap?size=400x400&center=51.51,-0.1275&zoom=12 We'll probably end up working out these URLS quite a bit. So we'll make ourselves another function to build up a URL given our parameters.

```
In [7]: import requests
        def request_map_at(lat, long, satellite=True,
                           zoom=10, size=(400, 400), sensor=False):
            base = "http://maps.googleapis.com/maps/api/staticmap?"

            params = dict(
                sensor= str(sensor).lower(),
                zoom= zoom,
                size= str(size[0])+"x"+str(size[1]),
                center = str(lat)+","+str(long),
                style="feature:all|element:labels|visibility:off"
            )
            if satellite:
                params["maptype"] = "satellite"

            return requests.get(base,params=params)
```

```
In [8]: map_response = request_map_at(51.5072, -0.1275)
```

1.3.7 Checking our work

Let's see what URL we ended up with:

```
In [9]: url = map_response.url
        print(url[0:50])
        print(url[50:100])
        print(url[100:])
```

```
http://maps.googleapis.com/maps/api/staticmap?sens
or=false&zoom=10&size=400x400&center=51.5072%2C-0.
1275&style=feature%3Aall%7Celement%3Alabels%7Cvisibility%3Aoff&maptype=satellite
```

We can write **automated tests** so that if we change our code later, we can check the results are still valid.

```
In [10]: from nose.tools import assert_in

         assert_in("http://maps.googleapis.com/maps/api/staticmap?", url)
         assert_in("center=51.5072%2C-0.1275", url)
         assert_in("zoom=10", url)
         assert_in("size=400x400", url)
         assert_in("sensor=false", url)
```

Our previous function comes back with an Object representing the web request. In object oriented programming, we use the . operator to get access to a particular **property** of the object, in this case, the actual image at that URL is in the `content` property. It's a big file, so I'll just get the first few chars:

```
In [11]: map_response.content[0:20]
```

```
Out[11]: b'\x89PNG\r\n\x1a\n\x00\x00\x00\rIHDR\x00\x00\x01\x90'
```

1.3.8 Displaying results

I'll need to do this a lot, so I'll wrap up our previous function in another function, to save on typing.

```
In [12]: def map_at(*args, **kwargs):
             return request_map_at(*args, **kwargs).content
```

I can use a library that comes with Jupyter notebook to display the image. Being able to work with variables which contain images, or documents, or any other weird kind of data, just as easily as we can with numbers or letters, is one of the really powerful things about modern programming languages like Python.

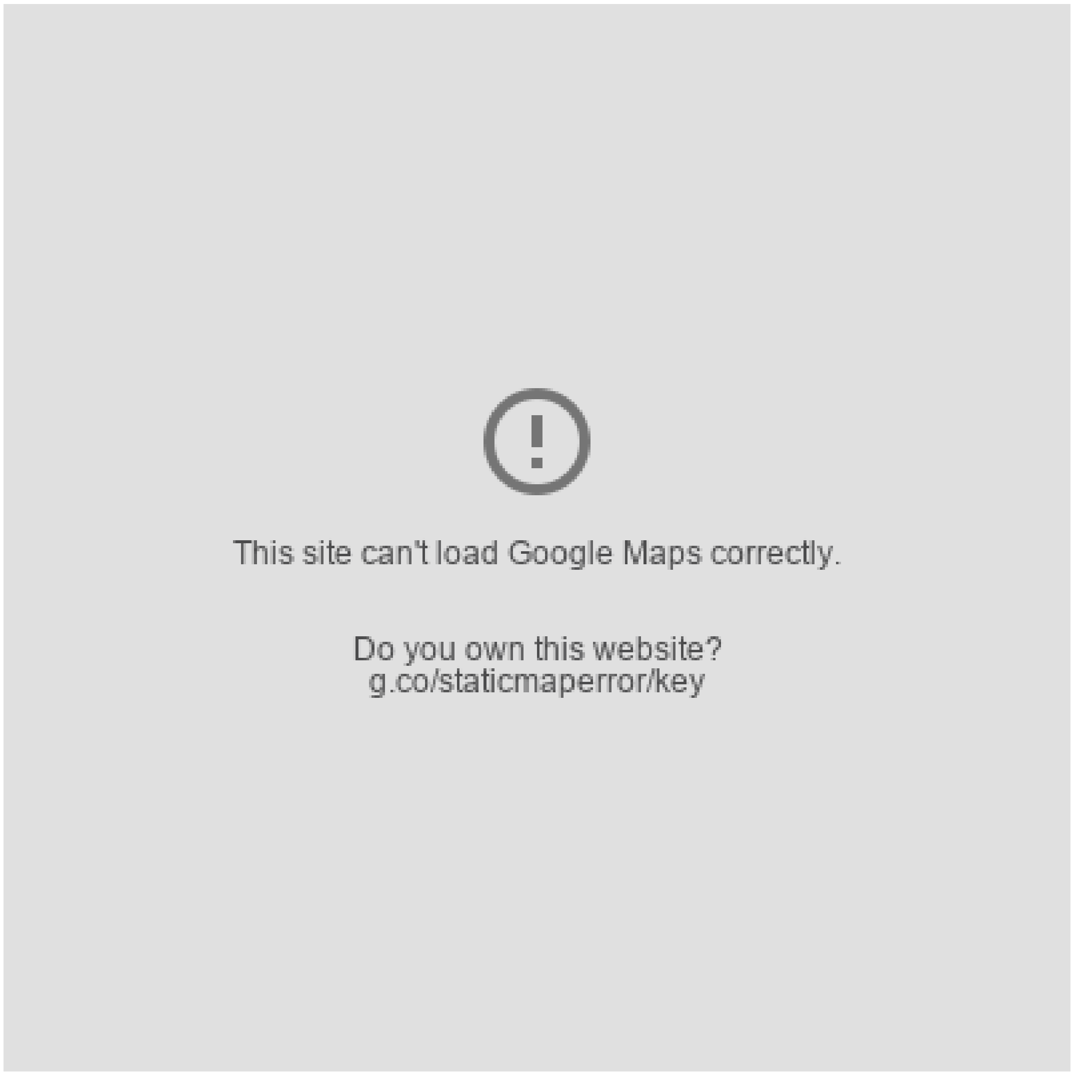
```
In [13]: import IPython
         map_png = map_at(*london_location)
```

```
In [14]: print("The type of our map result is actually a: ", type(map_png))
```

```
The type of our map result is actually a:  <class 'bytes'>
```

```
In [15]: IPython.core.display.Image(map_png)
```

```
Out[15]:
```



This site can't load Google Maps correctly.

Do you own this website?
g.co/staticmaperror/key