MPP001 Research Methods - Python Introduction

Simon Martin

Apr 04, 2024

CONTENTS

This is a short introduction to python and using it on University computers (and your own computer should you wish to).

This documentation is available in website form or as a single PDF. If you are viewing the website version then the links on the left will help you navigate between chapters and the links on the right allow you to navigate within a webpage.

Notebooks vs code files

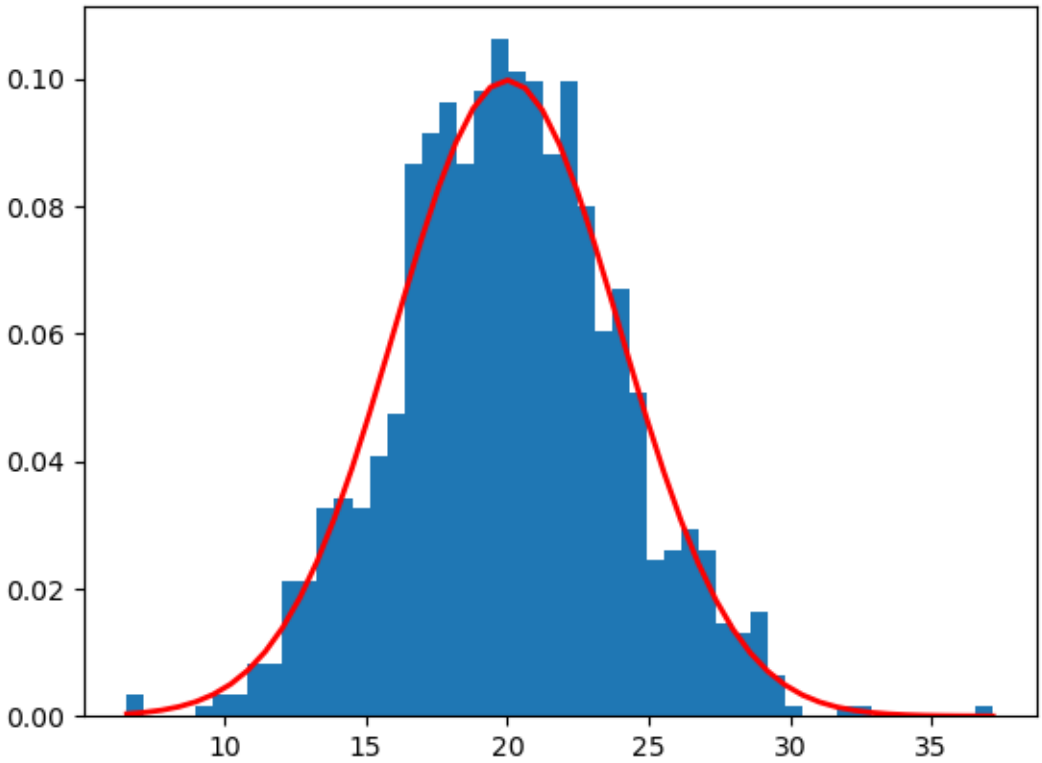
Traditionally, python coding has been done with individual files (usually with a '.py' extension). These files are edited then sent to a python interpreter to execute the code. More recently, a system based on notebooks has become very popular, particularly for science/engineering applications.

Both approaches are fine for this module. I prefer the notebook approach for small programming tasks, but still use the '.py' approach for more involved work. These notes are in part written using the notebook approach, but there will be examples of '.py' files for you to explore for the module.

The code below is an example of python code in a Jupyter notebook - the leading notebook system for python.

```
# Code uses stats to generate an array of 1000 points belonging to a normal
# distribution with a mean of 20 and a standard deviation of 4

# Import required libraries
from scipy import stats
import numpy as np
import matplotlib.pyplot as plt

# +
# Specify the properties of the normal distribution
n_sample = 1000
mean = 20
sigma = 4

# Generate an array belonging to the normal distribution
rand_dist1 = stats.norm.rvs(loc = mean, scale = sigma, size = n_sample)

print("Mean and stddev = ", np.mean(rand_dist1), np.std(rand_dist1))
print(stats.normaltest(rand_dist1))

# Plot the histogram of the data and the distribution function
count, bins, ignored = plt.hist(rand_dist1, 50, density=True)

plt.plot(bins, 1/(sigma * np.sqrt(2 * np.pi)) * np.exp( - (bins - mean)**2 / (2 *␣
 ↪sigma**2)), linewidth=2, color='r')
plt.show()
```

```
Mean and stddev =  20.04031442558665 3.9353221804747442
NormaltestResult(statistic=4.085668166860464, pvalue=0.12966072036768458)
```


0.10
0.08
0.06
0.04
0.02
0.00
10
15
20
25
30
35

CHAPTER

ONE

INTRODUCTION

This part of the guide provides a quick overview of how to get Python running on one of the University PCs and a very short introduction to running python code.

1.1 Starting a python session on Lab PCs.

Once you are logged in to a lab PC you can start a python session by using the Start Menu – Click on the Anaconda Folder and select the Spyder app.

There is also a Python section in the Start Menu. This gives access to other versions, but it is much better to use the method above as it works reliably and you will get the same apps etc. on your own computer if you install Anaconda (see below).

1.1.1 Make sure you have the right Python version

You might be inclined to use the search box to find/start python items. The problem is that this shows up items relating to another version of python (used by SPSS) that is not set up for general use. Worse, it uses an out of date version of python that is not compatible with much of the code used in this module.

1.2 Python on you own computer

You can install Python on your own computer. I recommend the Anaconda distribution https://www.anaconda.com. It contains everything you need for this module and a lot more. Installers are available for Windows, MacOS, and Linux. If disk space is a premium for you then search on the Anaconda site for the 'miniconda' version which contains the basics and there is an easy way to add any extras that you need.

Other versions of python are available. If you do not use Anaconda do make sure you are installing a one of the Python 3 variations (I advise using version 3.8 or higher). Python 2 is still available, but has some significant differences.

MacOS (and many Linux versions) come with a version of Python however these versions often lack the science libraries needed for this module and it is generally easier to install Anaconda.

Anaconda also comes with a range of tools for editing and running python code. Spyder is probably the best (free) choice for science/engineering. You can install it separately or as part of the Anaconda installation.

Another excellent choice is pyCharm.

CHAPTER

TWO

FIRST STEPS IN PYTHON

The easiest way to get started with python is to launch the Anaconda Navigator app. You then have to choose whether to use the file based ('.py') or notebook approach. I recommend you take the time to try both.

2.1 file-based python

Follow the instructions here: Spyder

2.2 Notebook-based python

Follow the instructions here: Jupyter notebooks

2.3 Hello World!

(in Spyder) In the console window (bottom left) you should see a prompt that looks something like this:

```
In[1]:
```

This is the current line and is where what you type appears. Type in the following, but do not press the return key:

```
print('Hello world!')
```

What do you think this code does? Press the return key and find out if you were right…

The code calls the `print` function telling it you want 'Hello world!' to appear on the screen. In python anything inside quotes (single or double) are examples of strings and are just text information.

2.4 Simple maths in python

Python has a range of standard mathematical functions as sumarised in the table below. If you have used something like Excel for maths then you should be familiar with them. Try to predict the outcomes of the examples given before entering them into your python system. Do they give the results you expected?

Symbol	operation	example
+	addition	3+2
-	substraction	2-3
*	multiplication	3*2
/	division	2/3
**	exponent	3**2

There are a range of other maths operators as well, but this is enough for now.

2.4.1 Beyond simple maths

Given the similarity to Excel for the operators above, what do you think the following will produce?

```
cos(0.0)
```

Try it in the Spyder console.

You probably got something like:

```
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    cos(0)
NameError: name 'cos' is not defined
```

Which is a longwinded way of the python system telling you it has no function `cos()`. One of the key features of python is that it is easily extendable. Try the following code snippet:

```
import numpy as np
np.cos(0.0)
```

The first line in the block above tells python to use the module called numpy (often pronounced num–pie) and all references to its contents will be preceded by `np.` for example `np.sin(3.141)`. Numpy is an extensive maths library that is very well optimised and we will use it a lot in MPP001–along with a lot of other libraries.

2.4.2 `as library_name`

The `as np` can be neglected and then to access numpy's functions `numpy.` will need to be added instead of 'np'. However it is very common to use `as library_name` because the full library name can be quite long and this can make the card hard to read (and longer to type). Formally the `as` sets up an alias to the original name of the library so you.

`np` is used by convention for the numpy library, you can use any text identifier you like, but remember that you and others need to be able to follow the code so keep it simple. For instance you could use `include numpy as Lb0r0R0cks` at the start of your code and then, say 100 lines later, have `Lb0r0R0cks.cos(XVal)` which would be very difficult to follow and you would have to search for what it means.

In this module you'll meet a few other widely used libraries that, like numpy, have pretty much universally accepted, standard names:

Library	Purpose	Standard short name
Scipy	A collection of scientific computing methods	`sp`
Pandas	Data manipulation and analysis	`pd`
Scikit Learn	A collection of machine learning methods	`sklearn`

There are plenty of other examples - look out for them in your reading/learning for this module.

2.4.3 Exercises

1. Investigate the operator precedence in python.
2. Experiment with the maths functions in numpy-are all the ones you expect there? Do they work the way you expect?
3. Modern python has a maths library called "math" that provides similar maths functions to numpy. Numpy provides a lot of extra functionality, but for most things there is not much to choose between them. Repeat your numpy maths experiments with the math module. Do you see any different answers?

2.5 Read the docs

Whilst discussing the standard libraries, it is worth pointing out that they have excellent online documentation. For example, for Pandas look here: https://pandas.pydata.org/pandas-docs/stable/user_guide/index.html. Smaller libraries are likely to use the ReadTheDocs web system for their documentation e.g. https://pytube.io/en/latest/index.html. This system provides a standard look for documentation that makes it easier to find you way round new libraries.

CHAPTER

THREE

PROGRAMMING (IN PYTHON)

If you are completely new to python then have a look through the syntax section on this page https://realpython.com/python-first-steps/. Syntax tells you about the rules for writing a computer language. When you begin programming you will probably spend most time getting this right. Do not worry about remembering details from that web link – you'll learn what you need through practice on this module. Hopefully you can see that python code is reasonably easy to read/understand. We are now going to look at how we can use python to "do stuff" i.e. write programs.

3.1 Algorithms

There are two absolutely key steps in writing any computer program (this applies to any programming language):

1. Understanding what it is you want to do
2. Breaking this down into individual steps

If you can do these for the problem in hand then the process of writing the code is usually straightforward.

Put together these steps allow you to develop an algorithm for the problem you are working on. In formal computer science there is more to it than this, but we need to be pragmatic and get on with solving problems.

For MPP001 step 1 should be straightforward. As you get more ambitious with programming it can be tricky to define (and understand sufficiently well) what it is you want to do. Questions you should ask yourself include:

- What is the starting point (what information do I know)?
- What result(s) should be returned/output?
- What calculations etc are necessary
- Are there any limits on the input/output?

As you gain experience with programming you will develop you own list and learn to adjust it to the situation.

Step 2 is a vital step that has to happen before you begin to write the code. With a little practice you will be able to do this step in your head for simple problems. However, for anything involving more than a handful of lines of code it is best to write out the steps and refine them.

Of course, there are other stages – testing, optimisation/refinement, etc. but for now these are enough to get going.

3.1.1 Example: how many cans of paint?

You are tasked with working out how many cans of paint are required to coat the sides and top of a tank of diameter $5m$ and height $8m$ if each can covers $2.5m^2$.

Step one: Inputs: diameter, Height, coverage per can. (limits: all should be above zero)

Outputs: Number of cans (limits: should be an integer)

Equations: the surface area of a cylinder is given by:

$$\pi \times \text{diameter} \times \text{height}$$

and the area of an end is:

$$\frac{\pi \times \text{diameter}^2}{4}$$

the number of cans will be given by:

$$N = \frac{\text{total surface area}}{\text{coverage per can}}$$

Step two A first stage-wise step through of the process could be:

```
1. Store input values in sensibly named variables
2. Calculate number of cans
3. Output results
```

This could usefully be refined to:

```
1. Store input values in sensibly named variables
        * Diameter = 5, Height = 8, Coverage = 2.5
2. Calculate number of cans
        * Ncans = (Surface area)/Coverage
3. Output results
        * Ncans needs to be rounded up
```

This is about ready to turn into python code:

```
# code to calculate number of cans required to paint sides and top of cylinder
import math as math # provides math functions and constants
# define values. Names should be self explanatory
Diameter = 5.0
Height = 8.0
Coverage = 2.5
NumCans = 0. # make sure that start from zero (good practice)
# Now calculate the number of cans required
# divide surface area of cylinder side + top by coverage per can
NumCans = (math.pi*Diameter*Height + math.pi*Diameter**2/4.)/Coverage
# Output Result
print(f'You require {math.ceil(NumCans)} cans of paint') # ceil rounds to the next␣
 ↪integer
```

Notice the use of comments (anything after a # is a comment and does nothing code wise). These are a way to annotate the code so that it is easier to follow. It is difficult to have too many comments in your code especially if it contains more than a couple of active lines of code.

Now more complex problems will require more stages and you may go through three or four refinement stages before starting to convert to code. It is tempting to shortcut this, but you'll be a better programmer for taking the extra time. In fact you'll probably take less time in the end as there'll be less wrong with your code when you write it. Also, you may find that more refinement still is needed as you start the coding – don't worry about this it is bound to happen. As you get more practiced you will get used to making the judgement.

You may feel that you want to jump into coding after Stage One and a lot of the time this will work OK for simple problems. However, it is good to get into the habit of doing both steps you will benefit in the medium term–especially if you want to evidence code development for your project.

3.2 Working with files

So far, we have typed in code to a python session and observed the results. For anything involving more than two or three lines of code we need a way to save/recall code. Python code is stored in files with names that end in `.py` - this is the equivalent of `.docx` for Word files.

The left hand section of the screen in Spyder is for entering and editing code (usually python). You can enter python code over multiple lines.

To run the code, first save the file, then click the triangle at the middle/top of the screen section. The results of most of your code will appear in the console window. Any graphs/images you work with will appear in the section above the console.

3.3 The building blocks of computer code

In order to get a program with a useful level of complexity there are a few key building blocks. These are common to most computer languages (there will be syntax differences–for instance, python uses indentation of code to separate blocks whilst many others use curly brackets i.e. {}. The ones key to introductory python programming are described below.

3.3.1 Conditional statements

These check whether a condition is met and perform different code based on this check. The classic structure is the `if` statement which will execute the following code if the condition tested evaluates to TRUE. `if` statements can be extended in a number of ways: an `else` block adds code that is only run if the condition is not met. In python there is also an `elif` command that can be thought of as 'else if…'

3.3.2 Loops

These are used to repeat a block of code either a fixed number of times (usually the `for` loop) or until some condition is met (`while` loop).

In most computer languages the for-loop increments/decrements a numeric variable through a range of values. Here is a C example

```
// Loop in C language - DO NOT copy to python
for (int i = 1; i < 11; ++i)
  {
    printf("%d ", i);
  }
```

Do not try and run this code in Spyder (or any other python system) Hopefully you can see that this would print out the integers from 1 to 10.

In python the loop goes through a list of values. The must all be the same type, but beyond that there are very few limits. So this example prints out the primary colours:

```
colours = ['red','green','blue']
for colour in colours:
    print(f'{colour}')
```

```
red
green
blue
```

The C example above could be reproduced in python with something like:

```
for i in range(11):
    print(i)
```

```
0
1
2
3
4
5
6
7
8
9
10
```

range() is a function that returns a list of numbers. It defaults to starting at 0, but this can be changed (e.g. `range(1,11)` would give 1, 2, …, 10).

Top tip Loops are often used for calculating sums, products, etc. make sure the variable that you are storing the result in starts from where it is supposed to. For sums (`i = i+NextValue`) this will usually be zero. For products (`y = y*NextValue`) one is a better choice.

3.3.3 Functions

These are blocks of code that you send values to and the block returns one or more values. `print()` is an example of a built-in function. You can add your own functions to python code either by importing them from another file/script or by defining them in the script you are working on.

A minimal python example would be:

```
def HelloWorld():
        print('Hello world!')
```

Note the code after the `def` line is indented – this is how python tracks the extent of the current code block. You can see a similar use of indenting in the for loop above.

The brackets after `HelloWorld` can take the names of arguments (parameters) that are to be passed to the function. E.g.

```
def Quotient(x,y):
        print(x/y)
```

which could be called by `Quotient(2,3)`. Note the order of arguments matters.

You can give arguments default values:

```
def Sum(x,y=3):
        return(x+y)
```

Calling this with `Sum(4)` will return `7` whereas `Sum(4,2)` will return `6`. Note all arguments after the first with a default value must have default values.

3.4 Finding out more about python

An excellent resource for getting started in python is week one of this online course: introPy

There are lots of other learning resources available via the internet. LinkedIn Learning (for which you have access via the University) has lots of professionally created courses to choose from and Google and/or YouTube may have one or two things of interest as well.

3.5 Exercises

3.5.1 Coding challenges

For each of the following use the development process outlined above. You may need to ask for help and/or search for information on some of the coding needed.

1. Write a function that returns the volume of a pipe of length `l`, outer diameter `D`, and inner diameter `d`.
2. Adapt the above program so that it gives the result to three decimal places and to three significant figures.
3. Write a function that calculates the sum of the first `N` odd integers. For example: if `N = 5` then the sum is `25` (i.e. 1 + 3 + 5 + 7 + 9 =25).
4. Write a program that uses a loop to call the function you created for exercise 3 for values of N from one to ten. What do you notice about the sums?

3.5.2 Spot the difference

The `numpy` library has a `ceil()` function that has a sublte difference to the one in the `math` library. Can you spot this difference? What impact might it have on your code?

CHAPTER

FOUR

NOTEBOOKS WITH MYST MARKDOWN

Jupyter Book also lets you write text-based notebooks using MyST Markdown. See the Notebooks with MyST Markdown documentation for more detailed instructions. This page shows off a notebook written in MyST Markdown.

4.1 An example cell

With MyST Markdown, you can define code cells with a directive like so:

```
print(2 + 2)
```

```
4
```

When your book is built, the contents of any `{code-cell}` blocks will be executed with your default Jupyter kernel, and their outputs will be displayed in-line with the rest of your content.

See also:

Jupyter Book uses Jupytext to convert text-based files to notebooks, and can support many other text-based notebook files.

4.2 Create a notebook with MyST Markdown

MyST Markdown notebooks are defined by two things:

1. YAML metadata that is needed to understand if / how it should convert text files to notebooks (including information about the kernel needed). See the YAML at the top of this page for example.
2. The presence of `{code-cell}` directives, which will be executed with your book.

That's all that is needed to get started!

4.3 Quickly add YAML metadata for MyST Notebooks

If you have a markdown file and you'd like to quickly add YAML metadata to it, so that Jupyter Book will treat it as a MyST Markdown Notebook, run the following command:

```
jupyter-book myst init path/to/markdownfile.md
```