Introduction to Python

April 13, 2021

1 Introduction to Python

1.1 Lesson aims

- Provide overview of using the Python programming language for research
- Explain why you might want to use it
- Provide a starting point for further learning
- Provide basis for exercises

1.2 What is Python?

Python is a general-purpose programming language designed to be easy to learn and easy to read (in order to understand other people's code).

Python programs can be written in text files (.py script file), but for reserach we often use 'Jupyter Notebooks' that allow you to mix text and code. To run both script files and notebooks, you need to run them using the Python program.

For this course, you will use an online service that we have prepared for you.

You can also install Python on your own computer, and there are other services that make this easy such as Google Colab (https://colab.research.google.com/notebooks/intro.ipynb), among others.

1.3 What is it used for?

We will look at using Python for research, but in fact many programs are written in Python and many big institutions use it:

- Instagram and Youtube (the part that runs on the servers)
- Many big scientific projects and institutions like CERN or NASA
- Many industries have specialised use of Python, e.g. GIS (ArcGIS/QGIS), finance, or visual effects in the film industry.

1.4 What are some advantages of Python?

- Easy to learn, easy to read other people's code
- General purpose (not just for science or stats), so
- Easy to start learning but still very powerful for advanced users
- Wide range of applications and free packages for many tasks (statistics, Machine Learning, GIS, Climate…)
- Large and helpful community - lots of help, tutorials, events, etc

- Free and open source! (unlike e.g. Matlab or SPSS where you must pay or get a licence from the University)
- Rapidly growing in popularity - big community, lots of people to get help

1.5 Why learn Python (in science)?

Python is becoming increasingly popular in research in general - not only in the physical sciences (e.g. UNIGE Digital Humanities department https://www.unige.ch/lettres/humanites-numeriques/fr/accueil/). Some reasons why:

- Make your work more *reproducible*. Instead of your work being a series of manual steps, you write Python code to define all these steps. The automates the boring, repetitive work and makes it easy to go back and change or update things. Once you have made the effort to learn, you can save a lot of time
- Handle much more data, much more quickly. It allows you to process thousands of files, run millions of model iterations, and much more. This allows for new kinds of research.

Additionally, knowing Python is a valuable skill that has many different applications in different fields of research and industries.

1.6 Learning Python

Generally the best way to learn is to have a specific task or project you want to do, and learn what you need to make it happen!

Usually it is not very effective to try to learn the language 'by rote'. The syntax/grammer of Python is quite simple - the challenge is in learning how to figure out the logic needed to solve your problem, and how to use the key packages that will help you. Learn how to search effectively for what you need to know (it's impossible to remember everything!)

Working through exercises can be useful if you don't know where to start. The Software Carpentry workshop is a good one (about 4hours work) http://swcarpentry.github.io/python-novice-inflammation/

1.6.1 Useful links

- Official Python documentation page (make sure you get the Python 3.8 or newer version) https://docs.python.org/3/ , with beginner tutorial https://docs.python.org/3/tutorial/index.html
- StackOverflow https://stackoverflow.com/questions/tagged/python - community questions-and-answers website. Most often you find their results via Google - always prefer StackOverflow answers to random forums!
- Books/online courses
  - Learn Python https://www.learnpython.org/
  - Automate the boring stuff with python https://automatetheboringstuff.com/
  - Python for data science course https://www.datacamp.com/courses/intro-to-python-for-data-science
  - Software Carpetry: http://swcarpentry.github.io/python-novice-inflammation/
- Various useful links https://www.reddit.com/r/learnpython/wiki/index

1.7 Installing Python yourself

For this course, you do not need to install Python yourself.

If you would like to do so later, the recommended way is to download and install the ‘Anaconda’ Python bundle, which includes many useful tools for doing scientific research:

https://www.anaconda.com/download/.

Anaconda will install shortcuts to useful tools. I recommend using the ‘Jupyter lab’ to explore using python.

> NOTE: you can download Python by itself from python.org, but this is much less convenient for scientific work.
>
> NOTE: You should use Python versions 3.8 or newer. Sometimes you will find files or tutorials for Python 2, which is out of date.

1.8 Python files and Jupyer notebooks

1.8.1 Python files (.py)

Traditionally you write python code in a plain text file with a ‘.py’ extension.

You can run using the command line with `python yourfile.py` as well as through various editor software (PyCharm, Spyder, VSCode)

1.8.2 Jupyter notebooks (.ipynb)

We focus on using Jupyter notebooks, which must be edited and run within the Jupyter software (you can’t edit them with a normal text editor).

You can create and code ‘cells’ mixed with text cells, plot graphs which will be saved in the file (so you can see results from last time without re-running code).

Notebooks are great to document what you are doing and you can include formulas

$$f(x|,2) = 122e{-}(x{-})222 f(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}}\, e^{-\frac{(x-\mu)^2}{2\sigma^2}} f(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}}\, e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

2 Basic Python code examples

This is a very very brief overview of what Python code looks like

2.1 Basic syntax

- Variables, keywords, etc…
- Python is structured by indenting blocks of code with 4 spaces per level (Python editors will usually automatically insert 4 spaces when you press the ‘tab’ key)

[7]:
```
print('Hello python')
```

```
Hello python
```

2.2 Basic operations

[8]:
```
# Add
1 + 1
```

[8]:
```
2
```

[9]:
```
# Subtract
2 - 1
```

[9]:
```
1
```

[10]:
```
# Multiply
2 * 2
```

[10]:
```
4
```

[11]:
```
# Divide
2 / 2
```

[11]:
```
1.0
```

[12]:
```
# Power/exponent
2**3
```

[12]:
```
8
```

2.3 Variables and assignment

We create variables with a name and give them a value

[14]:
```
a = 1 + 1

print(a)
```

```
2
```

2.4 Logical operations and, or, not, in

[4]:
```
a = True
b = False
```

[5]:
```
# Logical operations and, or, not, in
a and a
```

[5]:
```
True
```

[6]:
```
a and b
```

[6]: False

[59]:
```
a or b
```

[59]: True

[60]:
```
a and not b
```

[60]: True

[63]:
```
1 in [1,2,3,5]
```

[63]: True

2.5 Comparison operations

[75]:
```
# Comparison operations
10 == 10
10 != 11

2 < 3
3 > 1
# NICE: python undersands chained operations in a 'natural' way (similar to how␣
↪you would read in a maths textbook)
1 < 2 < 3
```

2.6 Conditional logic - if, elif, else

Basic building blocks of programming logic using the logical and comparison operations

[34]:
```
input_value = 4

if input_value == 1:
    print('value was 1')
elif input_value == 10:
    print('lets do something else')
else:
    print('This is the default condition if none of the other conditions were␣
↪met')
```

```
This is the default condition if none of the other conditions were met
```

2.7 Loops

Loops allow you to tell the program to do repetitive things!

Python provides a lot of very useful tools for this, for now just a basic example

```
[35]: for i in range(5):
          print(i)
```

```
0
1
2
3
4
```

3 Types of variable

Variables can be numbers, text, lists, and other more complicated things

3.1 numbers

```
[7]: a = 1  # integer
     b = 2.3 # floating point (decimal) number
     c = 2.3e4 # floating point in scientific notation
```

3.2 Text: 'strings'

In programming jargon, text data is called 'string'

```
[11]: a_string = 'a string is just some text'
```

```
[12]: # We can add strings togther to make longer strings
      a_string + ' and some more text'
```

```
[12]: 'a string is just some text and some more text'
```

3.3 Lists

```
[21]: # Lists
      things = [1, 3, 4]
      things
```

```
[21]: [1, 3, 4]
```

```
[22]: # Items can be added to lists
      things.append(5)
      things
```

```
[22]: [1, 3, 4, 5]
```

! LIST INDEXING STARTS AT 0 !

- The first element of a list is the 'zeroth' element
- The index last element of the list is the length of the list -1

- Why? Because computer scientists are wierd! (Some people have different theories, but they are wrong ;)

```
[25]: # Items can be selected by 'index'
      things[0]
```

```
[25]: 1
```

```
[80]: number_of_things = len(things)
```

```
[80]: 5
```

3.4 Dictionaries or 'dicts'

A very common and useful type is a 'dictionary' mapping keys to values. Many things can be keys (strings, numbers…), while essentially anything can be stored as the value.

```
[89]: key_value_dictionary = {'key': 'values', 2: 4}
```

```
[90]: # Get the value from the dict
      key_value_dictionary['key']
```

```
[90]: 'values'
```

```
[91]: for k in key_value_dictionary:
          print(k)
```

```
key
2
```

```
[92]: for k in key_value_dictionary.values():
          print(k)
```

```
values
4
```

```
[93]: for k, v in key_value_dictionary.items():
          print(k,v)
```

```
key values
2 4
```

4 Functions

Functions are crucial to structuring code, re-using parts, avoiding copy-paste, etc.

```
[15]: def this_is_a_function(input_variable):
          # You put code inside the function
```

```
    input_variable = input_variable + 1
    return input_variable
```

[16]:
```
this_is_a_function(4)
```

[16]:
```
5
```

5 Re-using existing functionality: Modules/Packages/Libraries

You see all three names used (Modules/Packages/Libraries). Simply it is a system to allow functions and things to be bundled into a package so you can use it. For scientific work we often use a handful of important packages.

We use the 'import' keyword to load ones which have been installed.

[44]:
```
# Simple import
import datetime

# Import a name from a package
from path import Path
```

6 Python for working with data

The most used package for working with data is **Pandas**. It lets you work with tables of data (called DataFrames) in an easy way, a bit like Excel.

Together with Pandas we often use a pair of packages called Numpy and Scipy, which provide numerical and statistical functions.

Pandas also offers a lot of tools to create plots, which work with a library called **matplotlib**. It allows you to create and save graphs from your data.

Pandas lets you easily load and work with Excel files and CSV files containing tables of data.

6.1 Loading the packages

[19]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

6.2 Reading CSV and EXCEL files

[20]:
```
test_data = pd.read_csv('cooling_data_geneva.csv')
```

6.3 Showing some summaries of the data

[21]:
```
test_data.head()
```

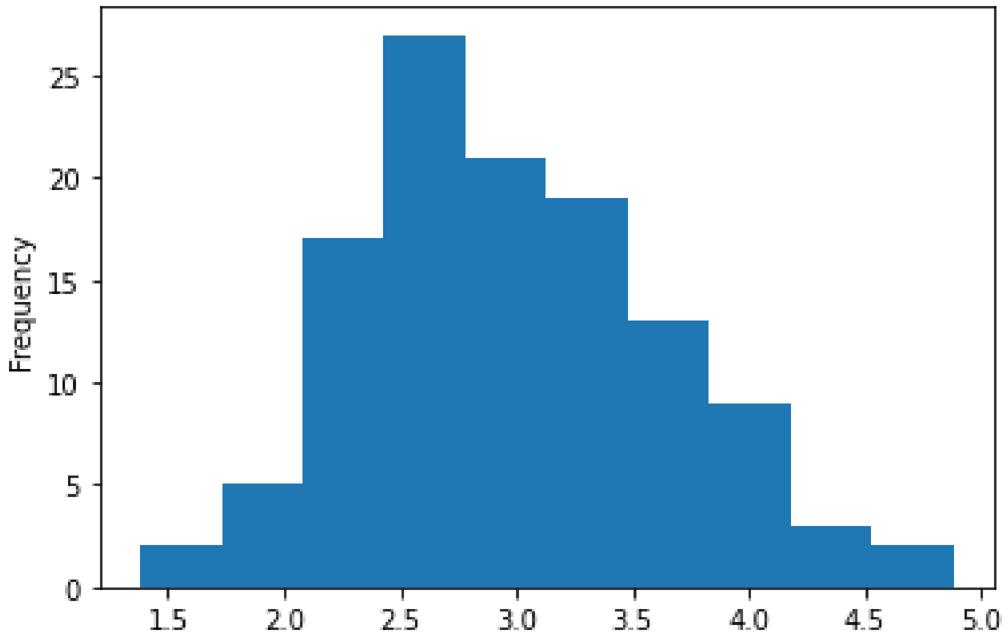
[21]:
```
   non_residential_area       cop           nh    pth_m2
0                 225.0  2.694737  2500.000000  0.113778
1                 456.0  2.545455   345.454545  0.061404
2                1024.0  3.225806  3687.500000  0.125000
3                2916.0  4.038462  2201.923077  0.048011
4                2230.0  2.790698  1976.744186  0.053812
```

[22]:
```
test_data.describe()
```

[22]:
```
       non_residential_area         cop           nh      pth_m2
count            118.000000  118.000000   118.000000  118.000000
mean            2376.480339    2.960994  1339.401646    0.104944
std             2261.322130    0.665711   951.477040    0.090505
min              105.000000    1.381818   166.666667    0.005859
25%              610.500000    2.500000   545.454545    0.053292
50%             1739.000000    2.876131  1065.277778    0.077474
75%             3537.000000    3.333333  1976.744186    0.119603
max             9654.000000    4.878049  3777.777778    0.496241
```

[23]:
```
test_data['cop'].plot.hist()
```

[23]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f49c7f23850>
```



7 Other Tools

7.1 Writing and Running code

- Jupyter Notebook: Interactive editing of text and code in your browser. Good for exploring data sets and documenting analysis
- Spyder: Working environment similar to Matlab
- PyCharm (IDE): Fully featured 'Integrated Development Environment' (IDE). Combines many tools into one, helps write, structure, and test larger and more complicated programs
- Text editors - Notebook++, Atom, Sublime Text : Enhanced text editors (like notepad) with plugins. Useful for quickly editing files, looking at raw data (e.g. CSV), text-transforms (e.g. advanced find-replace in a large file).

7.2 Software Version Control (Git)

- More advanced topic, but important! (Included more to raise awareness)
- Git is a tool for creating 'checkpoints' as you work, allows you to go back to earlier versions of files, merge changes from several people collaborating on the same code.
- More complicated to use, but CRUCIAL for larger projects and collaborations.
- Can help for individual projects, avoid saving versions of files with silly names.

Github - social programming! https://github.com/ - Github makes it easy to store and collaborate on git repositories. - It is not required in order to use Git (e.g. if your code should be private) and are many alternative services (GitLab, Bitbucket). But it is the most popular and is very often the home of open source projects. - For people serious about coding, it is a good place to keep projects which can be part of your CV/Portfolio

[ ]:
```

```