IEMS Boot Camp 2018: Introduction to Python

Instructor: Pol Boada-Collado

January 1, 2018

0. Introduction to Python

The aim of these notes is to provide a quick review of the main Python programming functionalities and features. Previous programming knowledge is assumed as the descriptions are brief, by example and informal. For more detailed introduction of programming with Python I recommend looking up to my previous document where I list recommendation of tutorial sections, readings and videos in the Appendix External Sources for Learning Python. To download and install Python: https://wiki.python.org/moin/BeginnersGuide/Download

a. Anaconda

"*With over 4.5 million users, Anaconda is the world's most popular Python data science platform. Anaconda, Inc. continues to lead open source projects like Anaconda, NumPy and SciPy that form the foundation of modern data science. Anaconda's flagship product, Anaconda Enterprise, allows organizations to secure, govern, scale and extend Anaconda to deliver actionable insights that drive businesses and industries forward.*" **Source**: https://www.anaconda.com/what-is-anaconda/

We typically use Anaconda as the environment for starting our projects. It gathers different IDE (like Spyder and Jupyter) and has already incorporated the most used libraries for data analysis and scientific computing.

b. Spyder

It is one of the most used IDE for Python. It is open source and integrates some important libraries.

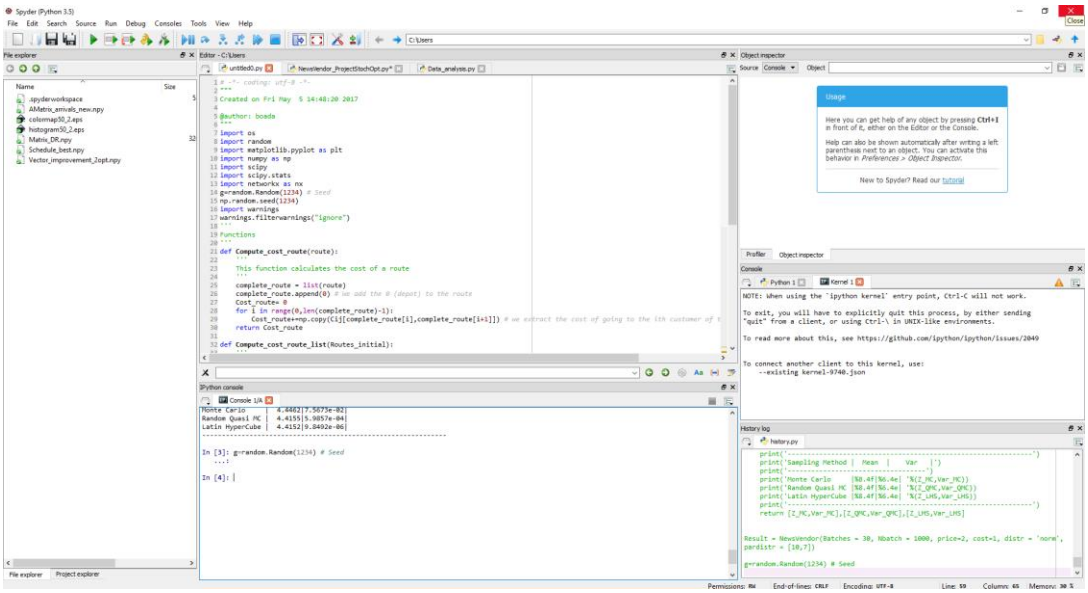

Figure 1: Example of Spyder window

It also provides a smooth transition for those who are used to working with Matlab or R (with RStudio).

Some useful tips:

- Comments in Python: "'Your Comment"', #Your Comment
- Shortcut for Comment/Uncomment: Ctrl+1
- Run selected code: F9
- Find text, replace and highlight: Ctrl+F

c. Jupyter

"The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and explanatory text." **Source**: http://jupyter.org/

It is useful for sharing our code with others as we can run the code easily and online. However, it is not so convenient for writing the code from scratch. In our courses, we will typically use it for submitting programming assignments.

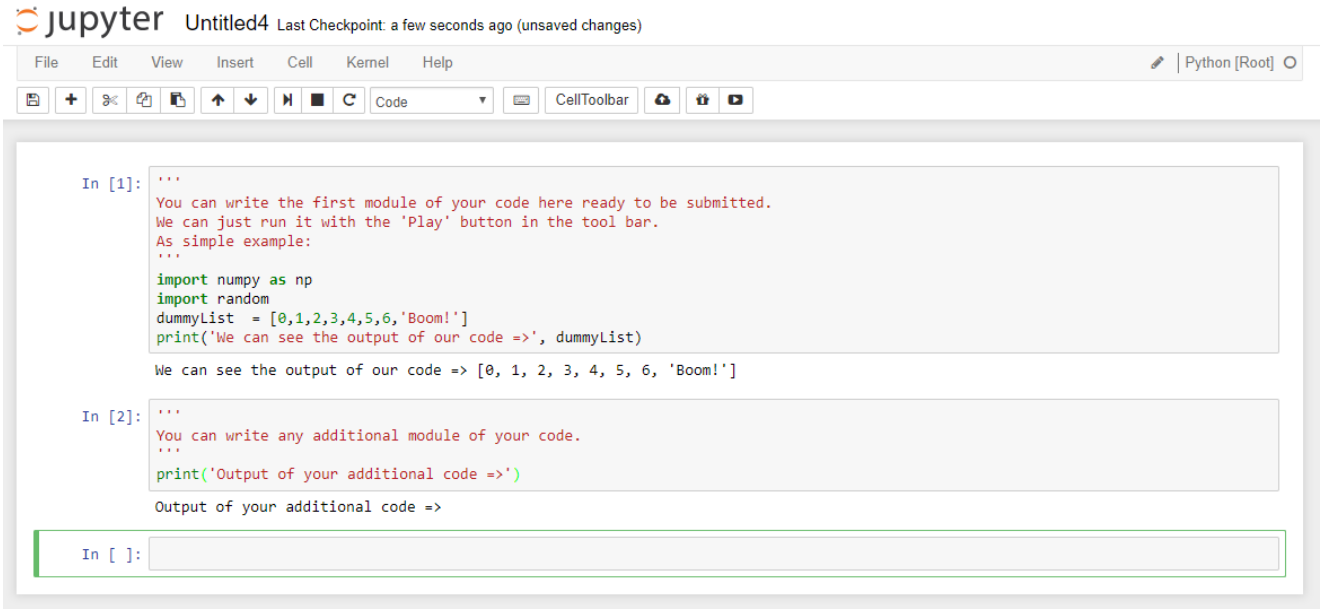

Figure 2: Example of Jupyter window

d. Python 2 vs Python 3: main differences

"What are the differences? Short version: Python 2.x is legacy, Python 3.x is the present and future of the language" **Source**: https://wiki.python.org/moin/Python2orPython3

It looks reasonable to use Python 3 if it is convenient for your projects. However, the differences are subtle and even more when we are working with libraries such like Numpy or Scipy, as we use and call the same functions in both versions. For this tutorial, we will use Python 3. Among others, these are some examples of these differences:

Print Example:

With **Python 2:**

```
1. print('Hello World')

   Hello World

2. print 'Hello World'

   Hello World
```

With **Python 3:**

```
1. print('Hello World')

   Hello World

2. print 'Hello World'

  File "<ipython-input-2-dd143c9d7342>", line
1
    print 'Hello World'
                      ^
SyntaxError: Missing parentheses in call to '
print'
```

Division example:

With **python 2**:

```
1. print(3/2)

   1

2. print(3//2)

   1

3. print(3/2.0)

   1.5
```

With **python 3**:

```
1. print(3/2)

   1.5

2. print(3//2)

   1

3. print(3/2.0)

   1.5
```

Other differences:

Python 2:

```
1. print('\"Hello World\">5'),'Hello World'> 5


   "Hello World"> 5 True
```

Python 3:

```
1. print('\"Hello World\"> 5'), 'Hello World' > 5


   "Hello World"> 5
   TypeError                                 Trac
   eback (most recent call last)
   <ipython-input-6-989601d0b268> in <module>
   ()
   ----> 1 print('\"Hello World\"> 5'), 'Hell
   o World' > 5 TypeError: unorderable types:
   str() > int()
```

1. Variables, tuples, lists and general idiosyncrasies

a. General variables

In Python, we do not need to define or declare variables before using them. We can use/create variables as we need. This feature makes the coding task easy and fast but can lead to errors and bugs. See below some examples about different kind of variables commonly used:

```
a = 'Hi'
b = 5
c = 8.4
d = 123456789
e = int(8.4)
print('These are my variables!')
print('|%4s||%4s||%4s||%8s||%4s|'%('a','b','c','d','e'))
print('---------------------------------')

print( '|%4s||%4.2d||%4.2f||%4.2e||%4d|'%(a,b,c,d,e))
```

```
These are my variables!
|   a||   b||   c||       d||   e|
---------------------------------
|  Hi||  05||8.40||1.23e+08||   8|
```

Note that we also used formatting for our output. In short: The first number after the "%" indicates how many spaces we will use for printing the variable. The second number indicates the number of decimals we want to print. The letter indicates the variable type (integer:d, float:f, string:s, exponential:e, etc).

We can learn the type of a variable by using the function `type()`:

```
type(9)
```

```
int
```

b. Lists

We use lists to concatenate different variables. Variables in lists can be of different type. For example:

```
myList = [1,2.0,3.14e25,'What?',['list','inside','list!',0.005]]
print(myList)
```

```
[1, 2.0, 3.14e+25, 'What?', ['list', 'inside', 'list!', 0.005]]
```

```
print(myList[0])
```

```
1
```

```
print(myList[1])
```

```
2.0
```

```
3.  print(myList[4])
```

```
['list', 'inside', 'list!', 0.005]
```

```
4.  print(myList[4][3])
```

```
0.005
```

```
5.  print(len(myList))
```

```
5
```

Managing lists:

- We access the elements in a list with "`[]`"
- To access the elements of a list inside a list we use: `myList[Index1][Index2]`
- `len(myList)` returns the length of the list
- In Python, we always start from 0. So, the first element of a list is `myList[0]`
- If we want to return the last element we can use `myList[-1]`
- We'll get an error if we exceed the number of elements

Basic Operations:

- Append:

```
1.  List1 = [1,2.0,3.14e25,'What?',9.99]
2.  List2 = [0,0.001]
3.  List1.append(List2)
4.  print(List1)
```

```
[1, 2.0, 3.14e+25, 'What?', 9.99, [0, 0.001]]
```

- Modify:

```
1.  myList = [1,2.0,3.14e25,'What?',9.99]
2.  myList[2]=8
3.  print(myList)
```

```
[1, 2.0, 8, 'What?', 9.99]
```

- Remove:

```
1.  myList = [1,2.0,3.14e25,'What?',9.99]
2.  myList.pop(2)
3.  print(myList)
```

```
[1, 2.0, 'What?', 9.99]
```

- Insert:

```
myList = [1,2.0,3.14e25,'What?',9.99]
myList.insert(2,'wow!')
print(myList)
```

```
[1, 2.0, 'wow!', 3.14e+25, 'What?', 9.99]
```

Short comment about the Tuples:

We can think of tuples as immutable lists. Once created, we cannot modify them. They are mainly used as arguments of functions. We can define them using "()" instead of "[]".

```
myTuple = (8,9,'hi..!')
```

c. Dictionaries by example

See the following example of dictionaries creation and management in Python.

```
OptimizierOptions = {'Info': 'I Write here the options of the optimizer',
            'MaxIter': 1E6,
            'Gap': 1E-8,
            'PrimaryMethod': 'BFGS',
            'SecondaryMethod': 'SR1',
            'OutputLevel': 1}
OptimizierOptions['OutputLevel'] = 0 # Modifying values
```

d. Deleting and copying

We can remove/delete a variable using: `del myVariable1, myVariable2, myVariableN`

We can reset all our workspace with `reset` .

It is important to remember that **we do not copy variables with "="**! The equivalence symbol refers to the pointer of that variable. Instead, f we want to copy a variable we will use `copy()`.

```
a=[0,1,2,3,4]
b=a
b.append(9999)
print(a)  # Note that we have modified variable a by changing b!! This can lead to errors.
```

```
[0, 1, 2, 3, 4, 9999]
```

```
a=[0,1,2,3,4]
b=copy(a)
b.append(9999)
print(a)  # Now, variable a is unchanged.
```

```
[0, 1, 2, 3, 4]
```

8. Comparisons, logic operators, conditions and loops

a. Comparisons and logic operators

Comparisons and Boolean Op.
==
>,>=
<,<=
!=
`True and False`
`True or False`
`not True`

b. Conditions and loops

Python uses **indentation** for separating different loop and function statements. If the indentation is not correct, the program will return an error. See the following structures:

If:

```
1.  if Condition1 or/and Condition2:
2.      Your Code inside the "if"
```

While:

```
1.  Repetitions = 0
2.  myList = [0]
3.  while Repetitions < 10:
4.      myList.append(myList[-1] + 1)
5.      Repetitions = Repetitions + 1
6.  print(myList)
```

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

For:

"For"-loops in Python work with lists or ranges:

```
1.  myList = [1, "numberTWO!", 3.14e25, 'What?', 9.99]
2.  for element in myList:
3.      print(element)
```

```
1
numberTWO!
3.14e+25
What?
9.99
```

The following lines do the same as we have above:

```
1.  for index in range(0,len(myList)):
2.      print(myList[index])
```

Side note on range():
We typically use range to easily create lists of numbers: `range([start], stop, [step])`. If we want to print the actual numbers when we create the range, we can convert it into a list.
See the following examples:

```
1. print(range(1,8,2))

   range(1, 8, 2)

2. print(list(range(1,8,2)))

   [1, 3, 5, 7]

3. print(list(range(2,8)))

   [2, 3, 4, 5, 6, 7]

4. print(list(range(8)))

   [0, 1, 2, 3, 4, 5, 6, 7]
```

Note that the stop number is NOT included in the list!

9. Functions

We define functions with the following structure:

```
def myFunction(inputs):
        Operations
        return Result1, Result2, ..., Resultn
```

Example:

```
def mySquareRoot(X):
    '''
    This function approximates the square root of a positive number

    Input:
    -----
        X: The number we want to calculate the sqrt
    Output:
    ------
        Approx: The sqrt approximation
    '''
    if X == 0:
        Approx = 0
    elif X < 0:
        print('Hello REAL World..')
        Approx = 'Err'
    else:
        Approx = 1 #  Define the initial value
        Number_iters = 0
        while (abs(Approx**2-X)>1E-6) and (Number_iters<100):  # reversed stop cond.
            Approx = 0.5*(Approx+X/Approx) # I use newton method
            Number_iters = Number_iters + 1
    return Approx

print(mySquareRoot(88))
```

```
9.380831519670643
```

10. Objects and Classes by example

We define an object in python as following:

```
class Rectangle:
    def __init__(self, Lenght, Height):
        '''''
        We write in here what we need to define the object.
        Inside the class, we use the defined self-variables as "self.NameOfVariable"
        '''
        self.length = Lenght
        self.height = Height
    def Area(self):  # Self-function: Only self-variables required
        '''''
        Note that we always have to specify "self" in our class functions
        '''
        Rec_area=self.length*self.height
        return Rec_area
    def Horizontal_Concatenation(self, Other_rectangle):
        '''
        Function that implies the current and another object.
        It creates a new rectangle by merging the current and another rect
        As it is a Horizontal concatenation we need equal heights
        Input:
        -----
        self: We always have to specify that
        Other_rectangle: The other rectangle we want to merge

        Output:
        ------
        New_rectangle: The merged rectangle '''
        if (self.height != Other_rectangle.height):
            print('Ups! I cant do that...Different heights!')
            New_rectangle = 'Err'
        else:
            New_rectangle = Rectangle(self.length+Other_rectangle.length, self.height)
        return New_rectangle

    def Vertical_Concatenation(self,Other_rectangle):
        '''
        Function that implies the current and another object.
        It creates a new rectangle by merging the current and another rect
        As it is a vertical concatenation we need equal lengths

        Input:
        -----
        self: We always have to specify that
        Other_rectangle: The other rectangle we want to merge

        Output:
        ------
        New_rectangle: The merged rectangle
        '''
        if self.length != Other_rectangle.length:
            print('Agh! I cant do that...Different lengths!')
            New_rectangle = 'Err'
        else:
            New_rectangle = Rectangle(self.length, self.height+Other_rectangle.height)
        return New_rectangle
```

Now we can play with the new class

```
a=Rectangle(2,2)
b=Rectangle(2,1)
c = a.Vertical_Concatenation(b)
print(c.Area())
```

```
6
```

11. Libraries:

We import and rename libraries to easily use them. For example:

```
import numpy as np
np.sqrt(77)
```

Among others, the most useful libraries for our purpose are the following:

- numpy
- scipy
- pandas
- matplotlib
- pyplot and plotly
- random
- os
- (networkx)
- (sklearn)
- (gurobipy)

a. numpy

Stands for Numeric Python. Many functions in that library are based on Matlab. It provides an easy way to work with vectors, arrays and matrixes. See some examples below:

Matrix and arrays:

```
import numpy as np # We will use "np" to call the library functions
A = np.array([[1, 1, 1], [2, 3, 4], [5, 5, 5]])
B = np.array([[0, 1, 0], [1, 0, 1], [0, 1, 0]])
C = A.dot(B)
```

Some basic functions:

```
myArray = np.array([0,-np.inf,9.5,np.exp(3),0.001,np.sqrt(77),
                    np.log(85),np.sin(np.pi*2/3),])
Index_max = np.argmax(myArray)
np.max(myArray) == myArray[Index_max]
```

```
True
```

More basic functions (try them to see the outputs):

```
myArray = np.array([-1,1,-1,2,-3,4,5.0,5,-5,0,1,0])
np.mean(myArray)
np.var(myArray)
np.sum(myArray)
A = np.resize(myArray,(4,3))
A_t = A.transpose()
B = np.concatenate((A,[[0,0,0]]),axis = 0) # Note here the double brackets[[]]!!
np.ones((10,5))
np.zeros((5,2))
np.eye(4)
np.sum(myArray[myArray>=0])
myArray[5] = np.nan
np.sum(myArray)
np.sum(~np.isnan(myArray))
myArray[np.isnan(myArray)] = 10
```

b. scipy

It provides many functions and routines for statistics, numerical integration and optimization. Try these statistic functions to see the outputs:

```
import scipy as sc

Matrix_norm = sc.stats.norm.rvs(loc = 3, scale = 1, size = (5, 5))
Matrix_norm_probs = sc.stats.norm.pdf(Matrix_norm, loc = 3, scale = 1)
Vector_pois = sc.stats.poisson.rvs(mu = 10, size = (15))
Vector_pois_CDF = sc.stats.poisson.cdf(Vector_pois, mu = 10)
print(Vector_pois == sc.stats.poisson.ppf(Vector_pois_CDF,  mu = 10))
```

c. os

We use this library to navigate and set the working path with our Operating System. For example:

```
import os
cwd = os.getcwd() # We get the current working directory
os.chdir('D:\\User\\Dropbox\\Bootcamp_Programming') # We set the working directory
```

d. pandas

It is a package for data structure operations. We can use it for easily uploading and working with data. See the following example:

```
import pandas
# First we read the file
df_distances = pandas.read_excel('TX_Distances.xlsx')
# We create a new data frame from that d.f. with just some columns
df_list_facilities = df_distances[['ID X', 'ZONE X',  'Zip Code X',  'Latitute X', 'Longitude X']]
# We remove the duplicates
df_list_facilities = df_list_facilities.drop_duplicates()
# We reset the index column
df_list_facilities = df_list_facilities.reset_index(drop=True)
# We rename the columns
df_list_facilities.columns = ['ID', 'ZONE', 'Zip Code', 'Latitude', 'Longitude']
# We create two new columns with the existing columns
df_distances['Distance Miles'] = df_distances['Distance Meters']*0.000621371
df_distances['Distance Hours'] = df_distances['Distance Seconds']/3600
# We create a new column with the existing columns with conditions
df_list_facilities['Status'] = ['Unavailable' if x == 'EAST' else 'Available' for x in df_list_fac
ilities['ZONE']]
# We merge the two data tables
df_distances = df_distances.merge(df_list_facilities[['ID','Status']], left_on ='ID X', right_on =
 'ID'  )
# Renaming...
df_distances.rename(columns = {'Status':'Status X'})
# We drop the column we are not interested in
df_distances = df_distances.drop('ID',1)
df_distances = df_distances.merge(df_list_facilities[['ID','Status']], left_on ='ID Y', right_on =
 'ID'  )
# We do the same as before
df_distances.rename(columns = {'Status':'Status Y'})
df_distances = df_distances.drop('ID',1)
```

e. random

It incorporates functions for random variables. Three-lined example:

```
1. import random
2. g=random.Random(1234) # Seed
3. U=g.random()   # We sample from a uniform [0, 1]
```

We fix a seed to get the same values of the random variables used: Our libraries have a huge list of uniform U[0, 1] realizations U1, U2, U3... From this list, we get the rest of r.v. realizations (you will see this in Stochastic Simulation IEMS435). Fixing the seed is fixing the starting point of this list.

f. matplotlib and pyplot

Explore these libraries to create plots, histograms, graphs etc. This is a simple example that can be used as a basic template:

```
1. import matplotlib.pyplot as plt
2. import numpy as np
3. import scipy
4. # We create our variables
5. x = np.arange(-10, 10, 0.1) # Yes! We create sequences also with numpy!
6. y = 1 + 0.8*x + sc.stats.norm.rvs(loc = 0, scale = 0.8, size = (len(x)))
7. z = np.sqrt(np.abs(y))
8. # We define the figure
9. fig1 = plt.figure(1)
10. # We plot our variables
11. plt.plot(x, y, 'r.', label = "Y sample")
12. plt.plot(x, 1+0.8*x, 'r-', label = "Y regression")
13. plt.plot(x, z, 'bx', label = "Z sample")
14. plt.plot(x, np.sqrt(np.abs(1 + 0.8*x)), 'b-', label = "Z regression")
15. # Define the plot limits
16. plt.xlim((-11, 11)) # Note that it is a tupla
17. plt.ylim((-10, 18))
18. # We plot the legend
19. plt.legend(loc = 0) #loc = 0:best, 1:upperight, 2:upperleft, 3:lowerleft, 4:lowerright, 5:right
20. # We write the labels of the axis
21. plt.xlabel('x')
22. plt.ylabel('value')
23. plt.savefig('MyFigure.jpeg') #Save the figure in our current working directory
```

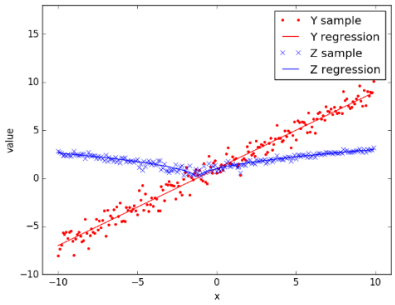

Figure 3: Example of matplotlib

g. networkx

This library is useful to work, create and analyze networks. It incorporates algorithms to obtain the MST, TSP etc.

h. Sklearn

Sklearn incorporates some basic Machine Learning functions and algorithms.

i. Gurobipy

We can use Gurobi optimizer in Python. See the following links for more details, documentation, how-to install and examples:

Registration and download of Gurobi (free license with Northwestern e-mail!):

http://www.gurobi.com/registration/download-reg

Gurobipy library:

https://pypi.python.org/pypi/gurobipy/

All you need about Gurobi and Gurobipy:

http://www.gurobi.com/documentation/7.5/quickstart_windows/py_python_interface

APPENDIX:

External Sources for Learning Python

Pol Boada-Collado (boada.pol@u.northwestern.edu)

January 1, 2018

I recommend students to watch-read the following courses (in order) to be comfortable with the session. They are all free (some of them require free registration). While watching the videos and practicing, you can use the document 'IEMS 365: Introduction to Python' as a supplementary material, in which I summarize the main concepts, structures and important piece of code that you will learn from the links and sections below. I strongly recommend playing around with the concepts learnt in A-E and doing the exercises. Programming with Python is a good asset to have in most of the fields you will be working! Do not hesitate to ask me any question about the content in any of the documents.

A. Introduction (4h):

This is a basic introduction to Python. I like the way it is introduced in this course as after each video you can actually practice with the exercises proposed. It has its own interface for writing the code!

https://www.datacamp.com/courses/intro-to-python-for-data-science

Complete the following courses (total time 4h):

1. Python Basics
2. Python Lists
3. Function and Packages
4. Numpy

B. Control flow: (WHILE, IF, FOR etc.) (4h):

Please, watch these two videos and practice the content explained:

1. For IF statements: https://www.youtube.com/watch?v=mQrci1kAwh4
2. For LOOPS (FOR and WHILE): https://www.youtube.com/watch?v=6HWK6O4-28E

*Note that they are using different syntax for the function 'print'. Do not worry, we will use the form: print("hello world").

Read and practice the content in these link (I strongly recommend to 'actually' write the code presented in the section to experiment and actually understand how it works):

3. Read the section 3.2 in this link:
https://docs.python.org/2/tutorial/introduction.html#first-steps-towards-programming

4. Read the sections: 4.1 – 4.5 of the following link:

https://docs.python.org/2/tutorial/controlflow.html

5. Do the following exercises:

https://github.com/barentsen/python-course/blob/master/lectures/04-control-flow-exercises.ipynb

Hint for the last question: There is a way to print variables and text with 'print()'. Try looking up how to do so, it is called 'print formatting' or 'Formatted Output'.

C. Writing your own functions (1.5h):

From this link:
https://www.datacamp.com/courses/python-data-science-toolbox-part-1
Complete the following course (total time 1h):

1. Writing your own functions

More on functions!: https://www.youtube.com/watch?v=qO4ZN5uZSVg

Read and practice the content in the following link (section 4.6 and 4.7.1) (0.5h):

https://docs.python.org/2/tutorial/controlflow.html

D. Matplotlib (1h):

From this link:
https://www.datacamp.com/courses/intermediate-python-for-data-science
Complete the following course (total time 1h):

1. Matplotlib

E. Pandas (30 minutes):

Here there is a nice introduction to the package Pandas. I recommend review the 10-minute introduction and try to play around a little bit with the code. (total time 20-30 minutes):
https://pandas.pydata.org/pandas-docs/stable/10min.html#