IEMS 381: Introduction to Python and Jupyter Notebooks

TA: Akhil Singla

September 20th, 2022

1. 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 provided are brief, by example and informal. To download and install Python:
https://wiki.python.org/moin/BeginnersGuide/Download

A handy document for help with Python syntax: https://automatetheboringstuff.com/

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 organisations 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. Jupyter

"The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualisations 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. A good resource: https://www.dataquest.io/blog/jupyter-notebook-tutorial/

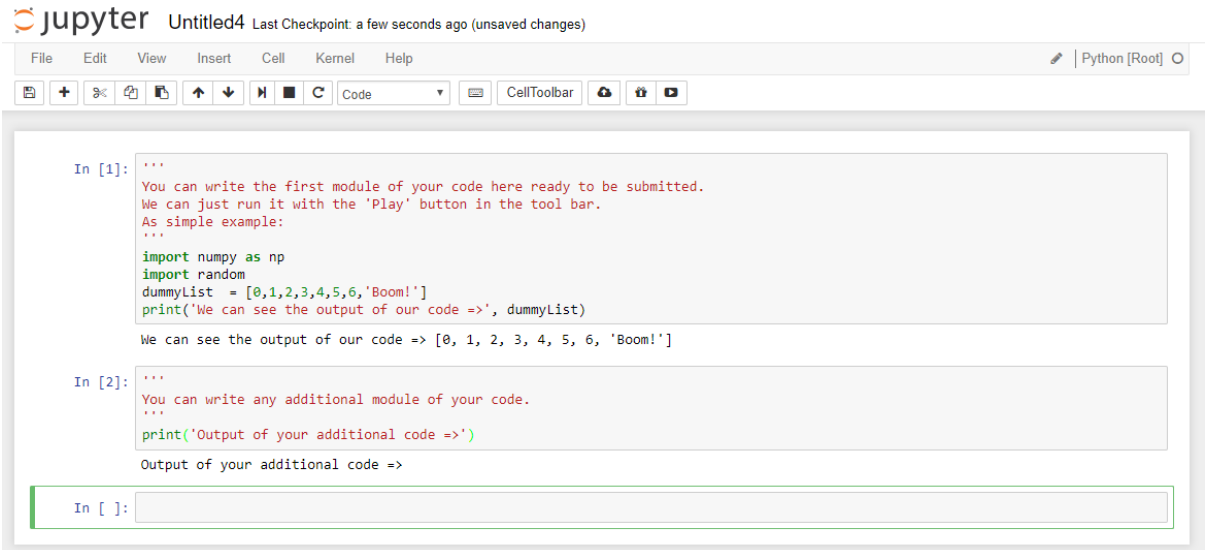

Figure 2: Example of Jupyter window

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

d. Python 3 Installation Guide

These are the instructions for using Python3 via Jupyter interface on MacOS.

Two general steps 1) Installing Anaconda and, 2) Jupyter IDE for using Python.

Step 1: Download Anaconda on your system following the guidance given here: https://docs.anaconda.com/anaconda/install/mac-os/ (for windows: https://docs.anaconda.com/anaconda/install/windows/)

Step 2: To use Jupyter Notebook as IDE:

type: `jupyter notebook`

2. 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:

```
1.  a = 'Hi'
2.  b = 5
3.  c = 8.4
4.  d = 123456789
5.  e = int(8.4)
6.  print('These are my variables!')
7.  print('|%4s||%4s||%4s||%8s||%4s|'%('a','b','c','d','e'))
8.  print('---------------------------------')
9.
10. 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).

Tips for naming any variable:

The following are the python's predefined keywords. We should avoid using them.

Python has 30 keywords:

and	elif	if	print
as	else	import	raise
assert	except	in	return
break	exec	is	try
class	finally	lambda	while
continue	for	not	with
def	from	or	yield
del	global	pass	

You can view the keywords by typing
`>>> help('keywords')`

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

```
1.  type(9)
```

```
int
```

b. Lists

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

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

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

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

```
1
```

```
print(myList[1])
2.0

print(myList[4])
['list', 'inside', 'list!', 0.005]

print(myList[4][3])
0.005

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]`
- If we want to see 3rd, 4th and 5th elements of list `myList[2:5]`
- We'll get an error if we exceed the number of elements

Basic Operations:

- Append:

```
List1 = [1,2.0,3.14e25,'What?',9.99]
List2 = [0,0.001]
List1.append(List2)
print(List1)
[1, 2.0, 3.14e+25, 'What?', 9.99, [0, 0.001]]
```

- Modify:

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

- Remove:

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

A dictionary is like a list, but more general. In a list, the indices have to be integers; in a dictionary they can be (almost) any type.
You can think of a dictionary as a mapping between a set of indices (which are called keys) and a set of values. Each key maps to a value. The association of a key and a value is called a key-value pair or sometimes an item.

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
```

Example1: Find the number of 'a', and 'b' in a given string.

```
mystr = raw_input('Enter a string ')
mydict= dict()
mydict['a']  = 0
mydict['b']  = 0
for i in range(len(mystr)):
    if mystr[i]=='a':
        mydict['a']  = mydict['a'] +1
    elif  mystr[i]=='b':
        mydict['b']  = mydict['b'] +1
print mydict
```

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, if 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.

c. 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:

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

While:

```
Repetitions = 0
myList = [0]
while Repetitions < 10:
    myList.append(myList[-1] + 1)
    Repetitions = Repetitions + 1
print(myList)
```

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

For:

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

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

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

The following lines do the same as we have above:

```
for index in range(0, len(myList)):
    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
```

Question:
1. Solve $2x^2 - 4x - 3 = 0$ using the quadratic formula. Also, try to code a general version for quadratic equations with real solutions.
2. Write a code to find minimum element of a list by defining a general function.

Use of Class Object: Define your own data structure for a list for which you can:

1. Add elements in the list
2. Remove elements from beginning, end, and from a particular location.
3. Print each of the element
4. Find summation of the elements
5. Find its length
6. Find an element in the list.

(You can use Python in-built functions ☺)

10. Objects and Classes by example

We define an object in python as following:

```
class Rectangle:
    def __init__(self, Length, 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 = Length
        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())
```

11. Libraries:

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

```
1. import numpy as np
2. 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)
- amplpy

Tip:

Installing package in Jupyter notebook directly
Run the following command in jupyter kernal and “simpy” is the name of the package

---
!pip install pandas
---

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 matrices. See some examples below:

Matrix and arrays:

```
1. import numpy as np       # We will use "np" to call the library functions
2. A = np.array([[1, 1, 1], [2, 3, 4], [5, 5, 5]])
3. B = np.array([[0, 1, 0], [1, 0, 1], [0, 1, 0]])
4. C = A.dot(B)             # This function returns the dot product between two arrays or the
5.                            multiplication between two matrices
```

Some basic functions:

```
1. myArray = np.array([0, -np.inf, 9.5, np.exp(3), 0.001, np.sqrt(77),
2.                     np.log(85), np.sin(np.pi*2/3),])
3. Index_max = np.argmax(myArray)          #Returns de indices of the position of the maximum
   value
4. np.max(myArray) == myArray[Index_max]    #Returns maximum value in the array
   True
```

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

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

```
np.sum(~np.isnan(myArray))
myArray[np.isnan(myArray)] = 10
```

b. 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_BC.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
_facilities['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)
```

c. 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:

```
import matplotlib.pyplot as plt
import numpy as np
# We create our variables
x = np.arange(-10, 10, 0.1) # Yes! We create sequences also with numpy!
y = 1 + 0.8*x
z = np.sqrt(np.abs(y))
# We define the figure
fig1 = plt.figure(1)
# We plot our variables
plt.plot(x, y, 'r.', label = "xy points")
plt.plot(x, 1+0.8*x, 'b', label = "xy line ")
plt.plot(x, z, 'bx', label = "xz points")
plt.plot(x, np.sqrt(np.abs(1 + 0.8*x)), 'r', label = "xz line")
# Define the plot limits
plt.xlim((-11, 11)) # Note that it is a tupla
plt.ylim((-10, 18))
# We plot the legend
plt.legend(loc = 0) #loc = 0:best, 1:upperight, 2:upperleft, 3:lowerleft, 4:lowerright, 5:righ
t
# We write the labels of the axis
plt.xlabel('x')
plt.ylabel('value')
```

```
22. plt.savefig('MyFigure.jpeg') #Save the figure in our current working directory
```

Generate the plot by yourself and match it from the Lab Session-1 jupyter notebook.

Helpful commands for Faster Implementation:

We can define the above lists, *X* and *S* as follows. >>> X=[1,2,3,4,5,6,7,8,9]
>>> S=[]
>>> for x in X:

S.append(x**2)
>>> print S

Now, using **list comprehension**, we can define both the above lists in a concise form:

>>> Xnew = [x for x in range(1,10)]
>>> Snew = [s**2 for s in Xnew]
>>> print Xnew ← X*new* is exactly same as *X*.

>>> print Snew ← *Snew* is exactly same as *S*.

Let's define T and S using list comprehension:

>>> P = [2**x for x in range(-6, 7)]
>>> print P
>>> T = [x for x in Snew if x%2 == 0]
>>> print T

Printing three tables in the range (3 to 100):

>>> [y for y in range(3, 100, 3)]