MATH 365
Python and Jupyter Notebooks

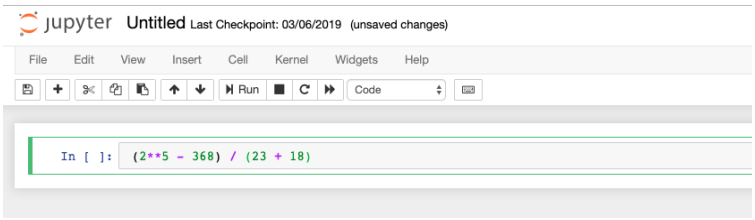
1 Introduction

1.1 Jupyter Notebooks

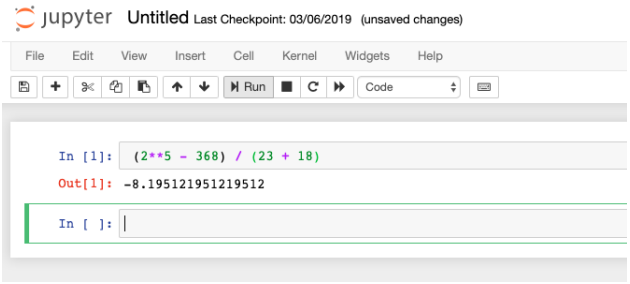
Python is a programming language and Jupyter is an interactive computational environment where you can combine code, text and graphs. A Jupyter notebook consists of a sequence of cells of different types. In a code cell you can edit and write Python code. A code cell has an input section containing your code and output section after executing the cell. You can execute or run a cell by

- clicking the Run button in the tool bar,
- selecting Cell → Run Cells in the menu bar,
- pressing Shift-Enter

For example, to compute $\frac{2^5-365}{23+18}$ you would type



and once you run it you would see



Note that the prompt numbers next to the code cells (e.g. In [1] and Out [1]) indicate which cells have been run and in which order. This is very useful, especially if you are running cells out-of-sequence. Note that not all operations have outputs, for example, print statements. This is because the print statement doesn't affect the output. If you would like to see a demonstration of someone using Jupyter notebooks I recommend watching this youtube video between 6:00 and 15:00 minutes: `https://youtu.be/CwFq3YDU6_Y`.

Jupyter notebooks are available in MB 136 and you may also choose to install it on your own computer. I suggest using the Anaconda distribution to install Python and Jupyter. Go to `https://www.anaconda.com`, choose to download the most recent version of Python, and follow the instructions.

1.2 Python Syntax

Python syntax may be different than our pseudocode in the Loops and Conditionals lab. The first significant difference is that in our pseudocode we had an *end* statement to terminate the *if*, *for* and *while* loops. Alternatively, Python uses indentation to indicate if commands sit inside a loop. The : symbol is used to start the indent suite of statements. For example, in Python we would write a while loop in the following manner

```
x = 0
while x < 4:
    x = x + 1
```

This code segment would assign value of 4 to x (justify this to yourself) and you would verify this by running

```
print(x)
```

Another significant programming difference in Python as compared to our pseudocode in the Loops and Conditionals lab is how we wrote *for loops* over consecutive integers. In Python we would do this with the range() function.

- range(n) is equivalent to to the list [0, 1, ..., n - 1]
- range(start, stop) is equivalent to the list [start, start + 1, ..., stop - 1]

Python has both lists and arrays. In this class we will mostly use arrays but we need the package NumPy to define arrays, which we will do in the next section.

If we wanted to print the first 10 integers, starting at 0 we could run

```
for i in range(10):
    print(i)
```

which is equivalent to

```
for i in range(0,10):
    print(i)
```

and equivalent to

```
list=[0,1,2,3,4,5,6,7,8,9]
for i in list:
    print(i)
```

This would produce output 0 1 2 3 4 5 6 7 8 9 since we started at 0.

We can also increment the loop and just print the even integers

```
x = range(0, 10, 2)
for n in x:
    print(n)
```

It is important to note that range() can take only integers as arguments. To iterate over more general floating point numbers we need the package NumPy.

1.3 NumPy

NumPy (numerical Python, pronounced num-pie) is a library that provides advanced mathematical operations involving statistics and linear algebra. Numpy's standard data type is an array and right now that is our main interest in using it.

NumPy is one of many modules in Python. Modules are pre-written collections of operators, usually designed for a specific purpose or task. To use a module, you must first import it, e.g.

```
import numpy
```

To invoke one of it's operations, precede the operator's name with the name of the module, followed by a period. For example, if we want to use an array of floating point numbers rather than a list of integers

```
arr = numpy.array( [ 2.1, 3.7, 4.2 ] )
```

It can be a bit tedious to write numpy repeadly so typically a pseudonym is used when we import it, e.g.

```
import numpy as np
```

so that we can write

```
arr=np.array( [ 2.1, 3.7, 4.2 ] )
```

There are numerous NumPy operations e.g.

```
arr = np.zeros( 5 )
print(arr)
```

will print `[0.  0.  0.  0.  0.]` while

```
arr = np.arange( 10, 30, 5 )
print(arr)
```

will print `[10 15 20 25]`.

Numpy provides access to elements of an array using the standard indexing operator [ ], e.g. in the last example `arr[2]` is 20 (note that counting starts at 0). It's also possible to ask for the shape of an array using numpy.shape. In the last example

```
print(np.shape(arr))
```

will output `(4,)` which means that arr is a row vector with 4 elements. The shape can also be found with the command

```
print(arr.shape)
```

Please read `https://www.pluralsight.com/guides/overview-basic-numpy-operations` for more examples. As you read these examples, for those who are more comfortable with matrix algebra than programming, think of 1D arrays as vectors and 2D arrays as matrices. This introduction is showing you how to do matrix algebra on a computer.

1.4 Plotting

Another useful Python package is matplotlib, which is a library for constructing plots. One module within this larger library is pyplot, which presents a simple and easy to use interface for constructing plots.

```
import matplotlib.pyplot as plt
```

Please read the Pyplot tutorial `https://matplotlib.org/users/pyplot_tutorial.html`.

2 Individual Lab

Part 1:
Please answer the following questions in Blackboard after reading Sections 1.1 and 1.2.

1. True or False: The pseudocode in the Loops and Conditionals Lab can be programmed in Python exactly at it is written in the Lab.
2. What will happen if we run the following statements in Python

```
sum = 0
for d in range(0, 10, 0.1):
    sum = sum + d
```

3. Which of the following commands returns a sequence 0, 1, 2, 3?

   (a) range(0, 3)
   (b) range(0, 4)
   (c) range(3)
   (d) range(4)

4. What is the output for y?

(a)

```
y = 0
for i in range(0, 10, 2):
    y = y+i
print(y)
```

(b)

```
y = 0
for i in range(10, 1, -2):
    y = y+i
print(y)
```

5. True or False: The following program will terminate:

```
balance = 10
while True:
    if balance < 9:
        break
balance = balance - 9
```

6. What is sum after the following loop terminates?

```
sum = 0
item = 0
while item < 5:
    item = item + 1
    sum = sum + item
    if sum > 4:
        break
print(sum)
```

Part 2:

Please answer the following questions in Blackboard after reading Sections 1.3 and 1.4.

1. What will be the result in Python

```
import numpy as np
integers = np.arange(1, 5)
primes = np.array([2, 3, 5, 7])
print(integers * primes)
```

2. What will be the result in Python

```
a = np.array([1.0,2.0,3.0])
b = 2.0
print(a * b)
```

3. Which Matplotlib function generates a histogram?

4. Describe the plot generated by

```
import numpy as np
import matplotlib.pyplot as plt
x = [1, 2, 3]
y = [1, 2, 1]
plt.plot(x, y, "ko")
plt.show()
```

3 Group work

Part 1:
Please work in pairs to program the pseudocode in the Loops and Conditionals Lab in Python using Jupyter notebooks. The person who identifies as the one with the least programming experience will be the scribe who types in the Jupyter notebook. The second person will be the spokesperson and should be prepared to present to the class the following reflections.

- Which part of programming did you find to be the easiest?
- What strategies did you use when you got an error message?
- How did you verify your code was correct?

Here are some suggestions to verify your code is correct:

- Start by choosing a value of n that is small enough you can check your answer by hand. Then show a result for a large value of n that would be too tedious to check by hand.
- Choose x to be a vector containing random numbers from a normal distirbution with mean mu and standard deviation sigma. Do this with the Python commands

  ```
  import numpy as np
  x = np.random.normal(mu, sigma, n)
  ```

  with n representing the number of elements in the vector. Use this strategy to check if your calculation of the mean is the same as mu. Similarly compare your calculation of the standard deviation to sigma.

Once you are finished, delete the cells with errors and save a copy of the notebook with output for each pseudocode in a pdf file. The name of the file should contain the last names of each person in the group. Upload the file into Blackboard under the Group Assignment for today.

Part 2:

1. Please write your answers to the following questions on the supplied note cards.
   Assume you import numpy as np in Python.

   (a) Identify the length of each of the following arrays and order them from longest to shortest:

```
array1=np.arange(5)
array2=np.arange(1,5)
array3=np.arange(100)
array4=np.arange(1,2,0.1)
```

(b) Consider that

```
array1=np.linspace(start = 0, stop = 1, num = 50)
```

is an array with 50 elements that starts at 0 and ends at 1. Let

```
array2=np.random.normal(0, 1, 50)
```

Describe the difference between array1 and array2.

(c) Estimate the maximum element in each of the following arrays and order them from the one with the largest element, to the one with the smallest element:

```
array1=np.random.normal(0, 2, 40)
array2= np.random.normal(2, 0, 40)
array3=np.random.normal(40, 2, 40)
```

2. Please work in pairs to create the following plot. The person who identifies as the one with the least programming experience will be the scribe who types in the Jupyter notebook. Use plt.savefig to save the plot as a .pdf file. The name of the file should contain the last names of each person in the group. Upload the file into Blackboard under the Group Assignment for today.

   (a) Plot $y = 1 + 2x$ with x linearly spaced in the interval $[2, 3]$.

   (b) Label the axes and make a title for the plot.

   (c) Plot $y = 8 - x$ on the same plot that you created in 2a.

   (d) Research documentation on how to create a legend. Create a legend that labels the two lines on your plot in 2c.