# Part 1 - Introduction to Python and Jupyter

April 4, 2018

Table of Contents

# 1 Why this topic, why now?

- Jupyter allows for living computational documents
- Explosion in use and new libraries in Python
- Streamlined sharing of knowledge
- Real-time interactive learning
- Compute Canada launched a free Jupyter instance about a year ago

# 2 Introduction to Jupyter

https://github.com/cisl/python-jupyter-intro

## 2.1 Overview of Jupyter

- Jupyter is a web-based tool for writing code, documentation, tutorials, etc.
- What you are reading was created in Jupyter
- You can access a free Compute-Canada hosted version of Jupyter at https://uvic.syzygy.ca/.

## 2.2 Jupyter Kernels (Language Support)

- Markdown
- Python
- Matlab
- R
- Fortran
- C/C++
- Julia
- Latex Expressions (MathJax)
- ~70 in total (https://github.com/jupyter/jupyter/wiki/Jupyter-kernels)

## 2.3 Getting up and running in Jupyter

- Login to Syzygy with your uvic id.
  - **Click "Keep me signed in for 8 hours" or your session will fail shortly after login**
  - `jupyter@pims.math.ca` for support
- Create (or upload) a notebook.
- Write and run your code

## 2.4 You can also

- Install Python on your computer and use it directly without Jupyter.
- Install Python and Jupyter on your computer and use Jupyter locally
- Use other languages with Jupyter...

## 2.5 Markdown

- Used to write text that can be easily formatted into HTML
- In Jupyter you can intermix markdown and code cells to create a living and executable document
- A Jupyter notebook can be exported as HTML, Latex, PDF (via Latex), code, used as slides

### 2.5.1 Markdown syntax examples

```
# Heading 1
## Heading 2

* Bullet item
    * Sub-item

**bold** *italics*
[Link](www.jupyter.org)
![Alt text](image.jpg)

a|b
---|---
c|d
```

Latex Expressions: `$e^{i\pi}=0$`, $e^{i\pi} = 0$

## 2.6 Jupyter Pros and Cons

- Pros
    - Integrated documenting, coding and executing
    - Support for programming languages and latex
    - A lot of plugins (e.g., for section numbering, Zotero)
- Cons
    - Slide functionality is problematic - mostly text-over-run

## 2.7 JupyterLab (In Beta)

Towards an integrated development environment for Jupyter

# 3 Overview of Python

- Used widely in academia and industry for data science and numerical analysis (among other uses)
- Designed to be simpler than C/Java
- There are ~2,506,000 open-source python repositories on GitHub
- There are ~134,000 packages across all versions in the Python Package Index (i.e., packages that can be installed in one command, e.g., *pip install numpy*)

## 3.1 Python 2 vs 3

- This tutorial uses Python 2 because I have a lot of code in Python 2 and only run it - but I should switch
- If you are just getting started, use Python 3.

## 3.2 Numeric/Scientific Computing in Python

- **Numpy** and **Scipy** provide the fundamentals
- There are other, more specilaiized libraries for machine learning, control systems, signal processing, networks/graphs etc.

### 3.2.1 Example Numpy modules

- Array/Matrix objects
- Sorting, searching, and counting
- Linear algebra
- Discrete Fourier Transform
- Polynomials
- Random sampling
- Logic functions
- Financial functions

### 3.2.2 Example Scipy Modules

- Integration (scipy.integrate)
- Optimization (scipy.optimize)
- Interpolation (scipy.interpolate)
- Fourier Transforms (scipy.fftpack)
- Signal Processing (scipy.signal)
- Linear Algebra (scipy.linalg)

### 3.2.3 Numpy for Matlab Users

https://docs.scipy.org/doc/numpy-dev/user/numpy-for-matlab-users.html

# 4 Python Syntax

If you cannot figure out how to do something in Python just google it. Most questions have been asked and answered

## 4.1 Variables

### 4.1.1 Define and Print Variables

```
In [52]: x = 4 # integer
         y = 3.34523534532123421 # float
```

```
In [53]: print x
         print y
         print '{:.3g}'.format(y)
```

```
4
3.34523534532
3.35
```

```
In [54]: print '{:.3g} {:.3g}'.format(x,y)
```

```
4 3.35
```

### 4.1.2 Boolean

```
In [55]: x = True
         y = False

         print x
         print y
         print x==y
```

```
True
False
False
```

### 4.1.3 Strings

```
In [56]: x = 'Hello World'
         y = 'Hello {} World {}'.format('Big',2)

         print x
         print y
         print x+y
```

```
Hello World
Hello Big World 2
Hello WorldHello Big World 2
```

### 4.1.4 Lists

```
In [57]: l = [1,2,3,4]
```

```
In [58]: print l[0] # access item at index 0
```

```
1
```

```
In [59]: print l+l
```

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

```
In [60]: print len(l) # get the length of the list
```

```
4
```

```
In [61]: print l[::-1] # get the list in reverse order
```

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

```
In [62]: print max(l), min(l), sum(l)
```

```
4 1 10
```

```
In [63]: y = [4,2,3,1]
         y.sort()
         print y
```

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

```
In [64]: print range(4) #generate a list
```

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

```
In [65]: x = [[1,2],[2,3]] #multi-dimensional list
         print x[0][1]
```

```
2
```

### 4.1.5 Dictionaries

```
In [66]: d = {'a':1,'b':2}
```

```
In [67]: print d
```

```
{'a': 1, 'b': 2}
```

```
In [68]: print d['a']
```

```
1
```

```
In [69]: print d.keys()
```

```
['a', 'b']
```

```
In [70]: print d.values()
```

```
[1, 2]
```

Keys can be ints, floats, strings. Values can be these or lists, other dictionaries, etc.

## 4.2 Math Operators

```
In [71]: x = 2
```

```
In [72]: print x+x,x-x,x/x,x*x
```

```
4 0 1 4
```

```
In [73]: print x**2 # exponent
```

```
4
```

### 4.2.1 Working with integer and float variables

```
In [74]: x=1 #integer
         y=2.0 #float

         z = float(x)
         w = int(y)
```

```
In [75]: print x,y,w,z
```

```
1 2.0 2 1.0
```

```
In [76]: print '1+2.0 =',x+y
         print '1/2 =',x/w
         print '1.0/2.0 =',z/y
```

```
1+2.0 = 3.0
1/2 = 0
1.0/2.0 = 0.5
```

## 4.3 Variable Values and References

Number and string values are copied

```
In [77]: x = 4.0
         y = x
         x = 5
```

```
In [78]: print x,y
```

```
5 4.0
```

Lists, dictionaries and other structures are equated by an address in memory (by reference) so a change in value in one variable will also change the value of another variable.

```
In [1]: x = range(2)
        y = x
        x[0] = -1
```

```
In [2]: print x,y
```

```
[-1, 1] [-1, 1]
```

## 4.4 Loops

### 4.4.1 Standard *for* Loop

```
In [81]: for i in range (5):
             print i**i # indent lines within loops by a tab
```

```
1
1
4
27
256
```

### 4.4.2 *for* Loop using *enumerate*

```
In [82]: for i,v in enumerate(range(5)):
             print i,v*2 # prints the index and the value at that index
```

```
0 0
1 2
2 4
3 6
4 8
```

### 4.4.3 *for* Loop using Dictionaries

```
In [83]: d = {'a':1,'b':2}

         print 'by keys:'
         for key in d.keys(): # or just for key in d
             print key,d[key]

             print 'by iteritems:'
         for key,value in d.iteritems():
             print key,value
```

```
by keys:
a 1
by iteritems:
b 2
by iteritems:
a 1
b 2
```

### 4.4.4 List comprehension

```
In [84]: result = [i+i for i in range(5)]
         print result
```

```
[0, 2, 4, 6, 8]
```

## 4.5 Conditionals

```
In [85]: flag = True
         x = 1
         y = 0
         if flag == True:
             print 'Hello World'# indent lines within ifs by a tab
             if x == 1 and y == 1: # a nested IF with an AND
```

```
                print 'AND'
            if x == 1 or y == 1: # a nested IF with an OR
                print 'OR'
```

```
Hello World
OR
```

## 4.6 Functions

### 4.6.1 Standard Functions

```
In [86]: def a_function(x):
             y = x+x # indent lines within functions by a tab
             print y

         a_function(5)
         a_function('Hello World')
```

```
10
Hello WorldHello World
```

### 4.6.2 Functions that return values

```
In [87]: def b_function(x):
             return x+x

         print b_function(5)
```

```
10
```

### 4.6.3 Shorthand Functions (*lambda*)

```
In [88]: a_function2 = lambda x: x+x

         print a_function2(5)
         print a_function2('Hello, World')
```

```
10
Hello, WorldHello, World
```

## 4.7 Imports

Typically library imports are made at the top of your notebook

```
In [89]: # Importing

         # all functions and classes in numpy are accessed using np.%name%
         import numpy as np

         x = np.array([1,2,3])
         y = x # sets the reference

         # importing the function copy in the library copy
         from copy import copy

         z = copy(x)
         x[0] = -1

         print x
         print y
         print z
```

```
[-1  2  3]
[-1  2  3]
[1 2 3]
```

```
In [6]: # Usefl imports
        import numpy as np # for array/matrix manipulation
        import scipy as sp # numerical routines
        import matplotlib.pyplot as plt # for creating graphs
        import seaborn as sns # for creating nice graphs quickly
        import pandas as pd # for reading/writing tabular data
```

### 4.8 Helpful configurations

```
In [4]: np.set_printoptions(precision=3) # print 3 sig. digits in arrays

        # So graphs are presented inline when a cell is run:
        %matplotlib inline

        mpl.rcParams['figure.figsize'] = [10.4, 7.15] # Set figure/graph size
        mpl.rcParams['font.size'] = 20 # Set figure/graph font size
        mpl.rcParams['text.usetex'] = True # Optional (using Latex in graphs)

        sns.set_style('white') # Recommended configuration for seaborn
        sns.set_context('talk') # Recommended configuration for seaborn

        # insert a random number here to seed the (psuedo) random number generator
        np.random.seed(847) # put in your own random number
```

## 4.9 Intro to Numpy

### 4.9.1 Numpy Math

```
In [92]: print np.pi
```

```
3.14159265359
```

```
In [93]: print np.exp(1)
```

```
2.71828182846
```

```
In [94]: print np.log(np.exp(1)) #natural logarithm
```

```
1.0
```

```
In [95]: print np.sqrt(2)
```

```
1.41421356237
```

### 4.9.2 Numpy Arrays and Matrices

Like lists, but better for numerical calculations

```
In [96]: print np.array([1,2,3])
         print np.matrix([[1,2],[3,4]]) # we won't use these mutch
```

```
[1 2 3]
[[1 2]
 [3 4]]
```

```
In [97]: print np.arange(3)
```

```
[0 1 2]
```

```
In [98]: print np.ones(3), np.zeros(3)
```

```
[ 1.  1.  1.] [ 0.  0.  0.]
```

```
In [99]: print np.linspace(4,8,3)
```

```
[ 4.  6.  8.]
```

Experiment with other array functions: concatenate, +, - add, subtract, multiply, divide

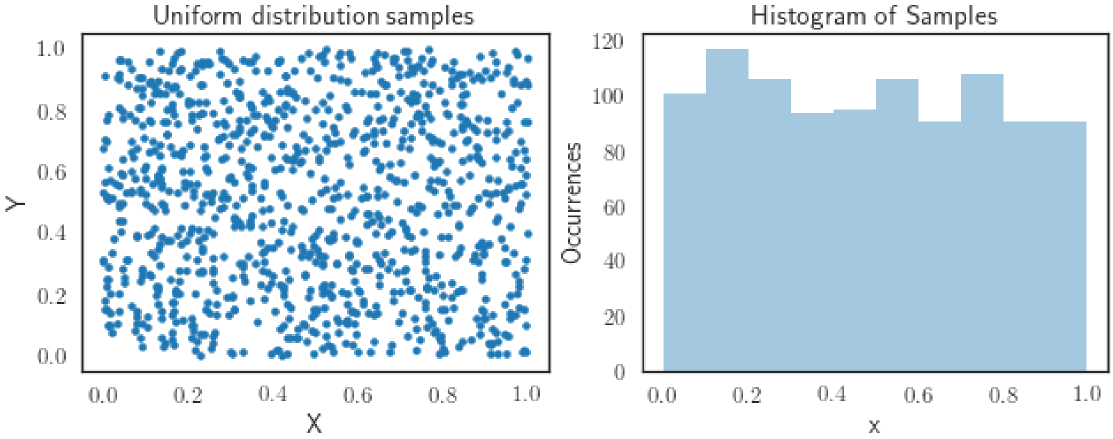
### 4.10 Random Numbers and Plotting

```
In [7]: # random numbers betweeen 0 and 1, other distributions available in np.random
        x = np.random.random(1000)
        y = np.random.random(1000)

        fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(10.4,3.5))

        axes[0].plot(x,y,'.'); # the semi-colon hides an uneeded output
        axes[0].set(title='Uniform distribution samples', xlabel='X', ylabel='Y');

        sns.distplot(x, kde=False, ax=axes[1]); # histogram
        axes[1].set(title='Histogram of Samples', xlabel='x',
                    ylabel='Occurrences');
```



### 4.11 Pandas

Read, write and work with tabular files

```
In [9]: df = pd.read_csv('../data/randomdata.csv')
        print df[0:10]
```

```
          x         y
0  0.102258  0.113201
1  0.474799  0.595240
2  0.414108  0.572583
3  0.590336  0.450980
4  0.194600  0.330423
5  0.180427  0.556806
6  0.785828  0.499291
7  0.260421  0.498740
8  0.540955  0.623766
```

```
9  0.998440  0.617484
```

```
In [102]: print type(df['x'].values)
```

```
<type 'numpy.ndarray'>
```