Getting started: Programming with Python and Jupyter Notebooks

Python is an interpreted language

- Programs do not need to be compiled before they are executed

- Allows for rapid development and exploration (important for statistics/data science)

You can run the Python interpreter directly from the command line by calling ''python'' or ''python3''

We will instead use Juypter Notebook with a Python 3 kernel

A Jupyter notebook is divided into cells

Cells may be subdivided into an input cell and an output cell

Input cells generally either contain code or text

- The Markdown language can be used to *format* text
- LaTeX can be used to input math

Cell type defaults to "Code" and can be changed to "Markdown" using the Cell->Cell Type menu or a keyboard shortcut (ESC-m in Windows)

Heading level 2

- Item 1
    - Item 2
        1. Item 3

$$\int_0^\infty e^x \, dx$$

```
2+2
def sum(a,b):
    out = a+b
    return out
```

You can use *italic* and **bold** with the * sign, and mathematical equations with $ sign, in front and at the end of the equation: $x = 1$.

- You can also type in LateX using $ sign. Inline equation: $x^2 + y^2 = \alpha^2$
- Centered equation:

$$x^2 + y^2 = \alpha^2$$

Cells also may contain "magics", which are commands to the Jupyter notebook server

- For instance, to determine which directory you are in, you can use the "%pwd" (print working directory) magic:

In [ ]:

```
%pwd
```

You can use the "%cd" magic to change your directory:

In [ ]:

```
%cd ~
```

~ is short your home directory

Hello World!

To print output in Python, use the print() function. It knows how to output many data types without providing explicit formatting (like printf in C)

In [ ]:

```
print('Hello World!')
```

In [ ]:

```
print(4773)
```

In [ ]:

```
print([1,2,3,4])
```

- Code cells may contain multiple Python statements.
- All statements in a cell will be run sequentially when the cell is run
- Cells may be run using the "play" button, via the Cell-> Run Cells button or by using a keyboard command (usually shift-Enter)

In [ ]:

```
print('Hello ')
print('World!')
```

In [ ]:

```
print('Hello!')
print(36)
```

In [ ]:

```
print('Hello!')
32+16
```

Anything that follows a # (hash) symbol is a comment:

In [ ]:

```
#It is important to use comments to document your thinking on big assignments
```

There is not really a multi-line comment in Python (like / / in C). One way to make a multiline comment is to just make a multi-line string that is not assigned to any variable. Multi-line strings are delimited by triple-ticks ('''):

In [ ]:

```
'''This
is
multi-line string'''
```

When we make functions, a multi-line string right after the function definition serves as the docstring (documentation string) for that function.

In [ ]:

```
def myFunction(x):
    '''This is the function doctring
    This is the second line'''
    return x
```

In [ ]:

```
help(myFunction)
```

In [ ]:

```
myFunction?
```

Python is a dynamically typed language

Variable types are determined when they are assigned values

In [ ]:

```
x=10
```

In [ ]:

```
print(x)
```

In [ ]:

```
type(x)
```

In [ ]:

```
x=10.0
```

In [ ]:

```
type(x)
```

In [ ]:

```
x = 'Hello World!'
print(x)
```

In [ ]:

```
type(x)
```

Python will usually do the *right thing* based on the type of the variable

In [ ]:

```
a=3
b=4
print(a+b)
```

In [ ]:

```
a='Hello '
b='World!'
```

```
print(a+b)
```

In [ ]:

```

```

Just be careful of the implications of this:

In [ ]:

```
a='3'
b='4'
print(a+b)
```

BTW, you can add color to your text by using the code `<font color=blue|red|green|pink|yellow>Text</font>`

You can also add hyperlinks to your text like this: Visit our Canvas page here

Sometimes I will use *Bootstrap alerts* to emphasize new concepts or convey new information:

All of these markdown rules can also be used when you are editing READ ME files on GitHub!

Indentation conveys meaning in Python

Use indentation to indicate code blocks that belong together

In [ ]:

```
a=2
b=7
if a==2:
    print('Yay')
else:
    print("a!=2")
```

In [ ]:

```
for x in range(10):
    if x%2==0:
        print(x)
```

In [ ]:

```
list(range(10))
```

Ranges in Python start by default at 0 and are exclusive of the end point (I.e., if started at 0, the end point is also the length of the sequence)

In [ ]:

```
list(range(0,10,2))
```

Main Data Types

The main data types in Python are:

- numbers (int, float, complex)
- string

- list
- tuple
- dictionary

All data types are objects. That means they have methods associated with them

In [ ]:

```
x=7
```

In [ ]:

```
help(x)
```

In [ ]:

```
x.__mod__(3)
```

Methods with are designated as private, but you can still call them:

In [ ]:

```

```

Usually, you don't call the private methods because there are other ways of achieving the same thing that are easier to interpret:

In [ ]:

```
x%3
```

In [ ]:

```
y='Hello'
y.__len__()
```

In [ ]:

```
len(y)
```

Mutability

- Some data types in Python are **immutable**, i.e. they cannot be changed. These include numbers, strings, and tuples
- Lists and dictionaries are **mutable**, they can be changed

In [ ]:

```
a=(2,3)
```

In [ ]:

```
type(a)
```

In [ ]:

```
a=a+(4,)
a
```

How did a change if it is immutable?

- a did not change, a new tuple was created that added 4 to the previous tuple, and a was updated to point to the new tuple. How can we tell?

In [ ]:

```
a=(2,3)
b=a
a is b
```

In [ ]:

```
print(id(a), id(b))
```

In [ ]:

```
a+=(4,)
print(a,b)
```

In [ ]:

```
a is b
```

In [ ]:

```
print(id(a), id(b))
```

In [ ]:

```
a.append(5)
```

Lists, on the other hand, are mutable:

In [ ]:

```
a=[2,3]
b = a
```

In [ ]:

```
a is b
```

In [ ]:

```
print(id(a), id(b))
```

In [ ]:

```
a.append(4)
```

In [ ]:

```
print(a,b )
```

In [ ]:

```
print(id(a), id(b))
```

In order to leave the original list unchanged, we need to *copy* it. This will create a new list in memory:

In [ ]:

```
a=[2,3]
b=a
c = a.copy()
```

In [ ]:

```
a.append(4)
```

In [ ]:

```
print(a,b,c)
```

You can also append to lists with +:

In [ ]:

```
a+=[(5,2)]
```

In [ ]:

```
a+=['String 1', 'String 2']
```

Lists and tuples may contain any other objects, including other lists and tuples:

In [ ]:

```
a
```

In [ ]:

```

```

Note that tuples and lists are ordered collections, and we can access their members directly:

In [ ]:

```
a[4]
```

In [ ]:

```
a[3:]
```

Negative indexes start from the end of the list, with -1 denoting the last member in the list:

In [ ]:

```
a[-1]
```

In [ ]:

```
a[-4:-1]
```

Modules and Libraries

Many of the tools we will use in the class are not directly part of Python.

In [ ]:

```
sin(3.14)
```

Instead, they are libraries or modules that provide particular functionality. These include:

- **NumPy** provides arrays, linear algebra, and math functions (many similar to the core MATLAB functions)
- **Matplotlib** provides functions to generate plots similar to those in MATLAB
- **pandas** provides tools for working with data
- **SciPy** provides many tools used in scientific computing including optimization, signal processing, and statistics
- **scikit-learn** provides various classification, regression, clustering algorithms and model implementations.
- **TensorFlow** provides highly-optimized structures to run and deploy Machine Learning and Deep Learning models.

- **PyTorch** provides highly-optimized structures to run and deploy Deep Learning models (large neural network).

> If you installed the Anaconda distribution, all of these libraries are already installed in your machine!

To work with these libraries, import them:

In [ ]:

```
import numpy
```

In [ ]:

```
numpy.sin(3.14)
```

In [ ]:

```
numpy.sin(numpy.pi)
```

To reduce typing, you can relabel a library on import:

In [ ]:

```
import numpy as np
```

In [ ]:

```
np.sin(3.14)
```

When using *Matplotlib* in Jupyter notebook, I recommend using the `%matplotlib inline` magic to make your graphs appear directly in the notebook:

In [3]:

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

In [ ]:

```
%matplotlib inline
```

In [ ]:

```
squares=[]
for x in range(100):
    squares+=[x**2]

squares[:10]
```

In [ ]:

```
fig=plt.plot(squares)
plt.title('Squares');
```

Continue practicing

1. "Python Data Science Handbook" by Jake VanderPlas, available online
   - **Chapter 1 - IPython: Beyond Normal Python** provides a great complement to this Notebook.

1. "A Whirlwind Tour of Python" by Jake VanderPlas, available online

- I think **Section 10 - Errors and Exceptions** is particularly handy.

All course readings are listed and electronically available in Course Reserves.