CME 211 Lecture 2: Interacting with Python

- Python is a high level language that typically runs in an *interpreter*
- An *interpreter* is a program (see `$ python3`) that executes statements from a high level language
- Examples of high level interpreted languages: Python, R, Matlab, Perl, JavaScript
- The most widely used Python interpreter is called **CPython**. It is written in C. There are others, for example **Jython** and **IronPython**. It is fairly easy (with experience) to access code written in C from **CPython**.
- Python now has a long history. Version 1.0 was released in 1994.
- This class will use Python 3. However, we will discuss important differences from Python 2 as we go along.

Getting started

To get started, let's log into `corn.stanford.edu`, start the Python 3 interpreter, and execute some Python commands. Please review the Lecture 0 notes if any of the following is unfamiliar.

1. Use SSH to login with `$ ssh [username]@corn.stanford.edu`
2. Locate the Python 3 interpreter with `$ which python3`
3. Run the Python 3 interpreter with `$ python3`
4. Execute the python statement `>>> print("Hello World!")`

Interpreter

- An *interpreter* is a program that reads and executes commands
- It is also sometimes called a REPL or read-evaluate-print-loop
- One way to interact with Python is to use the interpreter
- This is useful for interactive work, learning, and simple testing
- When you see a `$` in code blocks, it typically indicates a shell command. For example:

```
$ ls -1 *.md
0-outline.md
1-values-variables-types.md
2-strings.md
3-numbers.md
```

- When you see a `>>>` in code blocks, it typically indicates a command for the Python interpreter
- The basic Python interpreter is good for very simple computations or tests. IPython provides a lot more functionality (like tab completion and syntax highlighting), try `$ ipython3` at the command line

Python as a calculator

In the Python 3 interpreter:

```
>>> 4+7
11
>>> 55*2
110
>>> 9-1.4
7.6
>>> 5/3
```

```
1.6666666666666667
>>> 5//3
1
>>> -5//3
-2
>>> 5.0/3
1.6666666666666667
>>> 5.0//3
1.0
>>> 5 % 3
2
```

Integers and floating point

In a later lecture, we will discuss in detail the computer representation of integers and floating point numbers. For now:

- It is best to think of integers as being represented exactly over a fixed range. (This is not really true in current versions of Python, but will be true in C++)
- Floating point numbers are *approximations* of real numbers over a limited range.
- Floating point number range is not continuous. There are gaps between floating point numbers that depend on the scale. The gap between `1.0` and the next representable floating point number is smaller than the gap between `1.0e50` and the next representable floating point number.
- These things matter and bad numerical computing has resulted in a number of disasters: https://www.ima.umn.edu/~arnold/disasters/
- Division between two integers with `/` returns a floating point number
- `//` performs floor division (rounds down)
- The `%` operator is called the *modulus* operator and returns the remainder for integer division

Exiting the interpreter

- use `ctrl-d`
- use `exit()`

```
$ python3
Python 3.5.2 (default, Jun 29 2016, 13:43:58)
[GCC 4.2.1 Compatible Apple LLVM 7.3.0 (clang-703.0.31)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> exit
Use exit() or Ctrl-D (i.e. EOF) to exit
>>> exit()
```

Jupyter notebook

Jupyter Notebooks are great for teaching and interactive work. From the jupyter.org website:

> The Jupyter Notebook is a web application that allows you to create and share documents that contain live code, equations, visualizations and explanatory text. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, machine learning and much more.

In Jupyter Notebook, code is typed into code blocks:

```
print("hello from a code block!")
```

Code blocks can be re-executed with ease!

You can test Jupyter Notebook in your browser via https://try.jupyter.org/. Note that any work you do here will not be saved.

If you want to install Jupyter Notebook on your computer, I recommend Anaconda Python. Make sure to install the Python 3.5 version.

CME211 Notes:

- We do not formally support Jupyter Notebook. I show it to the class, because it is a very useful tool. You are responsible for figuring out how to install and run Jupyter Notebook. There are plenty of good tutorials online.
- The teaching staff is not responsible for helping you set up Jupyter Notebook on your computer. Python 3 on `corn.stanford.edu` is the supported computing environment.
- All Python work for CME211 must be submitted as Python scripts (`.py` files) that can be executed via the command line. We will not accept or grade any work submitted as a notebook (`.ipynb` format).
- The notes for CME211 are written in Markdown format and converted to Jupyter notebooks for lectures and screencasts. `nbconvert` is used to create PDFs.

Jupyter Notebook can render math

Here is a simple indefinite integral:

$$\int x \, dx = \frac{1}{2}x^2 + C$$

Here is an inline equation $e^{ix} = \cos x + i \sin x$ for any real number x.

Looking ahead with Jupyter Notebooks

Jupyter notebook is a very useful tool for scientific computing. If you have a background with Matlab, the following may help you get started working with Python and NumPy.

Start by importing modules:

```
# for 2d plotting
import matplotlib.pyplot as plt
# for numerical computing
import numpy as np
```

Configure plotting for notebook:

```
# tell matplotlib to use the notebook for figures
%matplotlib inline
# tell matplotlib to use svg (they look better than png)
#%config InlineBackend.figure_format = 'svg'
```

Note: in Jupyter notebook, statements that start with `%` are known as magic commands.

A simple plot of 100 random sampled data points:

```
plt.plot(np.random.rand(100))
```

Have a look at the "NumPy for MATLAB Users" reference. Note that you will have to prefix NumPy function calls from the reference with `np.` based on the import statements above.

It may also be a good idea to skim the following to see what you can do:

- NumPy reference
- SciPy reference

Python scripts

- A more convenient way to interact with Python is to write a script
- A Python script is a text file containing Python code
- Python script file names typically end in `.py`

Let's create our first script

- Log into `corn.stanford.edu`
- Create a text file named `firstscript.py` with your favorite text editor (`$ nano firstscript.py` is a good choice)
- Insert the following Python code into `firstscript.py`:

```
print("Hello from Python.")
print("I am your first script!")
```

- Execute the command `$ python3 fisrtscript.py`

Note the use of the `$ python3` command. On many systems the command `$ python` will start the Python 2 interpreter. For this simple example, the behavior will be the same. In general, this is not the case Python versions 2 and 3 have many differences.

Why scripts? Let's write a simple Python script to compute the first `n` numbers in the Fibonacci series. As a reminder, each number in the Fibonacci series is the sum of the two previous numbers. Let `F(i)` be the `i`th number in the series. We define `F(0) = 0` and `F(1) = 1`, then `F(i) = F(i-1) + F(i-2)` for `i >= 2`. Numbers `F(0)` to `F(n)` can be computed with the following Python code:

```
n = 10

if n >= 0:
    fn2 = 0
    print(fn2,end=',')
if n >= 1:
    fn1 = 1
    print(fn1,end=',')
for i in range(2,n+1):
    fn = fn1 + fn2
    print(fn,end=',')
    fn2 = fn1
    fn1 = fn
print()
```

Note, the above code is a preview of Python syntax that we will review in this course. Now, paste this code into a file named `fib.py`. Execute the file with the command `$ python3 fib.py`. The result should like:

```
$ python3 fib.py
0,1,1,2,3,5,8,13,21,34,55,
```

To see the utility of scripts, we need to add a bit more code. Change the first line of `fib.py` to be:

```
import sys
n = int(sys.argv[1])
```

This will instruct the script to obtain the value of `n` from the command line:

```
$ python3 fib.py 0
0,

$ python3 fib.py 5
0,1,1,2,3,5,

$ python3 fib.py 21
0,1,1,2,3,5,8,13,21,34,55,89,144,233,377,610,987,1597,2584,4181,6765,10946,
```

We have increased the utility of our program by making it simple to run from the command line with different input arguments. CME211 homeworks will work like this.

Python modules

If you are familiar with MATLAB, you may come to Python and be confused by:

```
sqrt(3)
```

The Python language does not have a built in `sqrt` function. `sqrt` exists in the Python `math` module:

```
import math
math.sqrt(9)
```

About Python modules:

- A module is a collection of Python resources (functions, variables, objects, classes) that can be easily loaded into Python via `import` statements
- Modules allow for easy code reuse and organization
- Modules allow the programmer to keep various functionality in different namespaces.
- There are a large number of modules in the Python Standard Library: https://docs.python.org/3/library/index.html
- It is often useful to explore the Python documentation in the interpreter. See `>>> help(math)` and `>>> help(math.sqrt)` from the interpreter or in a notebook code block.

Printing

Jupyter Notebook will echo the output of the last (non-assignment) statement in a code block:

```
1 + 1
5 + 5
```

```
myvar = 101
# no output
```

You can use the `print()` function:

```
a = 99
print(a)
```

By default, `print()` adds a new line character at the end for printing.

```
print("hi")
print("cme211")
```

This behavior can be changed by setting the `end` keyword parameter in the print function.

```
print("hi", end=' ') # now print inserts a space instead of a newline ('\n')
print("cme211")
```

The `print()` function can print several strings at once on the same line:

```
print("apple", "bananna", "orange")
```

The default separator is a space. This can be changed by setting the `sep` keyword parameter:

```
print("apple", "bananna", "orange", sep=', ')
```

Python strings can be "formatted" with the `format` method:

```
r = 1
a = math.pi
print("the area of a circle of radius {} is {}.".format(r,a))
```

The curly braces (`{}`) get replaced by the arguments to `format()` in order.