Lesson 3. An Introduction to Jupyter and Python

What is Jupyter?

- **Jupyter** is an interactive computational environment where you can combine code, text and graphs.
- Until recently, Jupyter was called **IPython Notebook**. This historical tidbit might help if you're looking for other references.
- We'll be using Jupyter with the **Python** programming langugage in this course to:
  - set up data for various models from large-scale real-world sources, and
  - solve these models and interpret their output.

Structure of a notebook document

- A notebook consists of a sequence of **cells** of different types.
- We'll use these types of cells frequently:
  - code cells
  - Markdown cells
- You can determine the type of a cell in the toolbar.
- You can run a cell by:
  - clicking the **Run** button in the tool bar,
  - selecting **Cell** → **Run Cells** in the menu bar,
  - pressing Shift-Enter

Code cells

- In a **code cell**, you can edit and write Python code.
  - We'll talk about Python shortly.
- For now, we can use a code cell as a fancy calculator.
- For example, in the code cell below, let's compute

$$\frac{2^5 - 368}{23 + 18}$$

```
In [2]: (2**5 - 368) / (23 + 18)
```

```
Out[2]: -8.195121951219512
```

- Note that a code cell has
  - an **input** section containing your code,
  - an **output** section after executing the cell.

Markdown cells

- In a **Markdown cell**, you can enter text to write notes about your code and document your workflow.
- For example, this cell is a Markdown cell.
- The **Markdown** language is a popular way to provide formatting (e.g. bold, italics, lists) to plain text. Use Google to find documentation and tutorials. Here's a pretty good cheat sheet.
- For now, here are a few basic, useful Markdown constructs:

```
You can format text as italic with *asterisks* or _underscores_.

You can format text as bold with **double asterisks** or __double underscores__.

To write an bulleted list, use *, -, or + as bullets, like this:

* One
* Two
* Three
```

- To edit a Markdown cell, double-click it. When you're done editing it, run the cell.
- Try it in the cell below:

Double-click to edit this cell. Try out Markdown here.

Manipulating cells

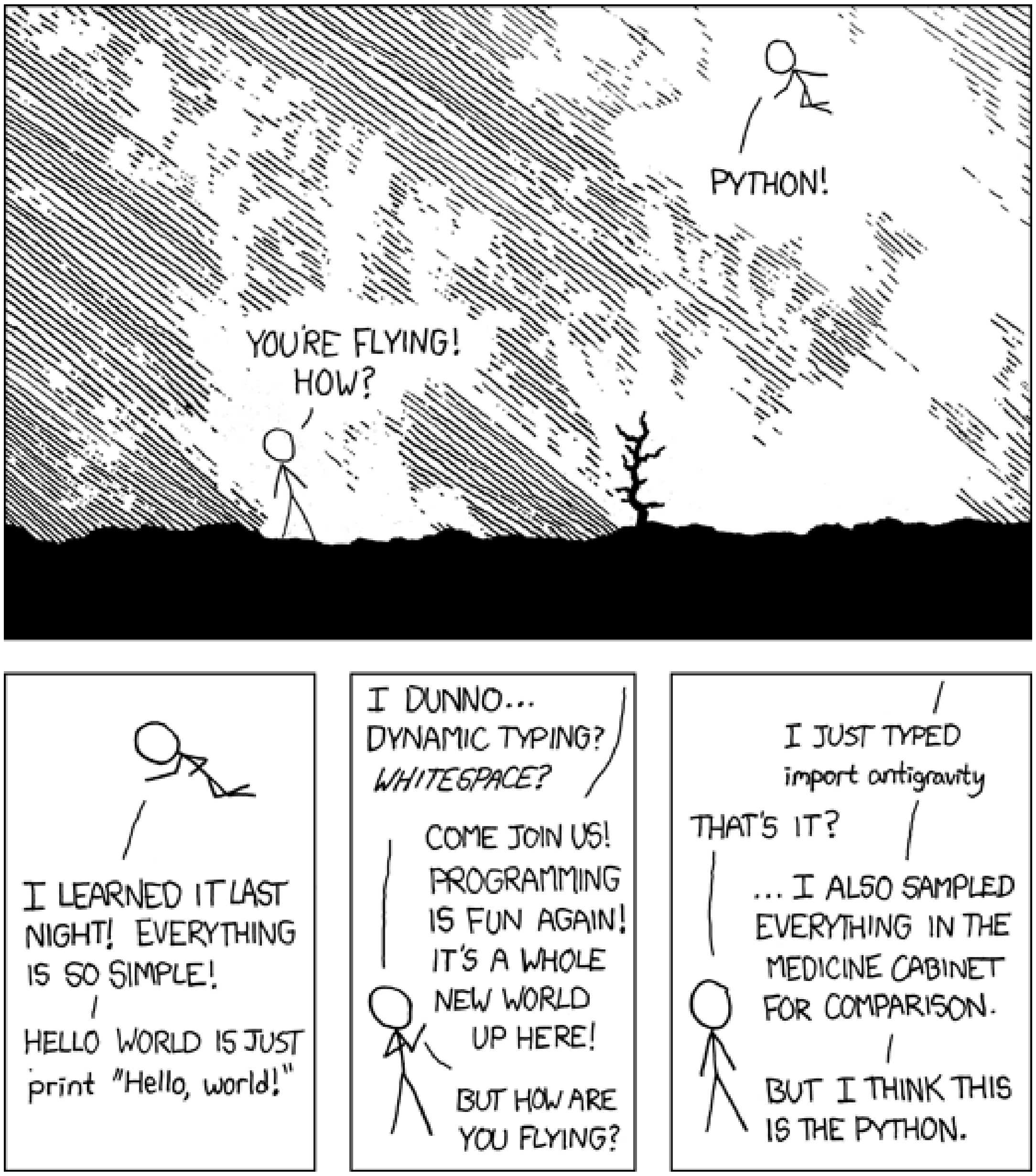
- You can insert a new cell by selecting **Insert** → **Insert Cell Above/Below** in the menu bar.
- You can copy and paste cells using the **Edit** menu.
- You can also split, merge, move, and delete cells using the **Edit** menu.

Saving your notebook

- Jupyter autosaves your notebook every few minutes.
- To manually save, click the icon, or select **File** → **Save and Checkpoint**.
- To close the notebook, select **File** → **Close and Halt**.
  - You should always close the notebook this way!
  - Just closing the tab/window will leave the notebook running in the background.
  - You can get a list of running notebooks in the **Running** tab of the Jupyter dashboard (the main Jupyter screen).

Moving on...

- We'll go over some other features of Jupyter later.
- The official documentation is here.
- There are many resources out there on using Jupyter — Google is your friend!


PYTHON!
YOU'RE FLYING! HOW?
I LEARNED IT LAST NIGHT! EVERYTHING IS SO SIMPLE!
HELLO WORLD IS JUST print "Hello, world!"
I DUNNO... DYNAMIC TYPING? WHITESPACE?
COME JOIN US! PROGRAMMING IS FUN AGAIN! IT'S A WHOLE NEW WORLD UP HERE!
BUT HOW ARE YOU FLYING?
I JUST TYPED import antigravity
THAT'S IT?
... I ALSO SAMPLED EVERYTHING IN THE MEDICINE CABINET FOR COMPARISON.
BUT I THINK THIS IS THE PYTHON.

https://xkcd.com/353/

What is Python — and why?

- **Python** is a free, open-source, general-purpose programming language.
- Python is popular and used everywhere — a few examples:
    - YouTube
    - Industrial Light and Magic
    - AstraZenica
- Python is "beautiful": its syntax was designed with an emphasis on readability.
- Python has awesome scientific computing tools: SciPy.org.
- "It's good for you": having exposure to multiple programming languages will be very useful to you as a {data scientist, economist, operations researcher, quantitative analyst, statistician, etc}.

A survival course in Python

- In this lesson, we will learn some basic Python concepts that will be useful in this course.
- We will cover other concepts throughout the semester as needed.
- There is a wealth of information on Python on the web!
- Here is the documentation for Python 3.5, which is the version we will use in this class.

Fancy calculator

- You can define a variable using the = sign.
- You can perform arithmetic operations on variables.
- You can print the value of a variable using the `print()` function.
- Don't forget to run the cell when you're done!

```
In [3]: # This is what a comment looks like in Python
        # Define dimensions of a rectangle
        length = 30
        width = 40

        # Compute area
        area = length * width

        # Print area
        print(area)
```

```
1200
```

- If you try to access a variable you haven't yet defined, Python will complain.

```
In [4]: print(volume)
```

```
        ---------------------------------------------------------------------------

        NameError                                 Traceback (most recent call last)

        <ipython-input-4-14ae0d839e78> in <module>()
    ----> 1 print(volume)


        NameError: name 'volume' is not defined
```

- Let's define the height of a box, so we can compute volume.

```
In [5]: # Define height of box
        height = 10

        # Compute volume
        volume = length * width * height

        # Print volume
        print(volume)
```

```
12000
```

- Note that the **prompt numbers** next to the code cells (e.g. `In [3]` and `Out [3]`) indicate which cells have been run and in which order.
- This is very useful, especially if you are running cells out-of-sequence.

Hello, world!

- **Strings** are lists of printable characters defined using either double quotes or single quotes.
- Just like with variables, to print a string, you can use the `print()` function.

```
In [6]: # Print "Hello, world!"
        print("Hello, world!")
```

```
Hello, world!
```

- You can use the `.format()` method to insert the value of a variable into a string.

```
In [7]: # Define a variable for your neighbor's name
        neighbor = "Nelson"

        # Print the value of the 'neighbor' variable
        print("My neighbor is {0}.".format(neighbor))
```

```
My neighbor is Nelson.
```

- The brackets and characters in them (e.g. `{0}`) are placeholders that are replaced with the arguments passed into `.format()`.
- In particular,
    - `{0}` is replaced with the first argument passed into `.format()`,
    - `{1}` is replaced with the second argument,
    - `{2}` is replaced with the third argument,
    - And so on.
- **In Python, indexing (that is, counting) starts at 0!**

Example. Define three variables, `left`, `right`, `me`, containing the names of your neighbors and yourself. Use the `print()` function to print the values of these variables in one line.

```
In [8]: # Define variables
        left = "Alice"
        right = "Carol"
        me = "Nelson"

        # Print values of variables
        print("I'm {0}, my neighbor to the left is {1}, and my neighbor to the right is
        {2}.".format(me, left, right))
```

```
I'm Nelson, my neighbor to the left is Alice, and my neighbor to the right is Carol.
```

Lists

- A **list** is a collection of items that are organized in a particular order.
- You can think of a list as an array or a vector.
- A list is written as a sequence of comma-separated items between square brackets.

```
In [9]: # Define a list containing the first 5 square numbers
        squares = [0, 1, 4, 9, 16]

        # Define a list containing the days of the week
        days_of_the_week = ["Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"]
```

- To get the first item in `days_of_the_week`, we would write

```
days_of_the_week[0]
```

- Remember, in Python, **indexing starts at 0!**

```
In [10]: # The third day of the week is...
         print(days_of_the_week[2])
```

```
Tue
```

- You can add items to the end of a list using the `.append()` method.
- You can also print lists just like any other variable.

```
In [11]: # Let's add the 6th squared number
         squares.append(25)

         # What does the list look like now?
         print("squares = {0}".format(squares))
```

```
squares = [0, 1, 4, 9, 16, 25]
```

- We can determine the length of a list using the `len()` function.

```
In [12]: # How many days of the week are there?
         len(days_of_the_week)
```

```
Out[12]: 7
```

Dictionaries

- A **dictionary** is another way to organize a collection of items.
- A dictionary maps **keys** to **values**.
    - Just like a real-world dictionary maps words to definitions.
- We can create a dictionary by starting with an empty dictionary and adding key-value pairs.
- You can also print dictionaries just like any other variable.

```
In [13]: # Create empty dictionary
         mid = {}

         # Add key-value pairs
         mid["First Name"] = "Nelson"
         mid["Last Name"] = "Uhan"
         mid["Company"] = 0

         # Print the dictionary
         print(mid)
```

```
{'Last Name': 'Uhan', 'First Name': 'Nelson', 'Company': 0}
```

- Similar to a list, we can use a key to look up the corresponding value in a dictionary as follows:

```
In [14]: # The last name of this mid is...
         print(mid["Last Name"])
```

```
Uhan
```

Loops and nesting

- We can iterate through lists using the `for` statement.

```
In [15]: # Define a list containing the months of the year
         months_of_the_year = ["Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep",
         "Nov", "Dec"]

         # Print the months of the year, one by one
         for month in months_of_the_year:
             print(month)
```

```
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Nov
Dec
```

- Python defines blocks of code using a **colon (`:`)** followed by **indentation**.
- The above code is NOT the same as

  ```
  for month in months_of_the_year:
  print(month)
  ```

- Always use the **Tab** key to indent – this will keep your indentation consistent.

- Often we will want to write a for loop over consecutive integers. We can do this using 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]`

```
In [16]: # First 10 integers, starting at 0
         for i in range(10):
             print(i)
```

```
0
1
2
3
```

```
4
5
6
7
8
9
```

```
In [17]: # Integers between 3 and 8 inclusive
         for i in range(3, 9):
             print(i)
```

```
3
4
5
6
7
8
```

- *Technically*, `range(n)` and `range(start, stop)` aren't really lists. But it's OK to think of them as lists for now.

Example. Write code to create a list of the first 10 cubic numbers, starting with 0^3. Print the list.

Hint. Start by creating an empty list:

```
cubics = []
```

Then *append* to this list using a `for` loop.

```
In [18]: cubics = []
         for i in range(10):
             cubics.append(i**3)

         print(cubics)
```

```
[0, 1, 8, 27, 64, 125, 216, 343, 512, 729]
```

If this, then that

- The == operator performs **equality testing**:
  - If the two items on either side of == are equal, then it returns `True`.
  - Otherwise, it returns `False`.

```
In [19]: # Let's define today to be Tuesday
         today = "Tuesday"
```

```
In [20]: # Is today Thursday?
         today == "Tuesday"
```

```
Out[20]: True
```

```
In [21]: # Is today Friday?
         today == "Friday"
```

```
Out[21]: False
```

- Conditional statements are written using the same block/indentation structure as `for` statements, using the keywords `if`, `elif`, and `else`.

```
In [22]: # Today is...
         today = "Saturday"

         # What should I do?
         if today == "Friday":
             print("Go out.")
         elif today == "Saturday":
             print("Have fun.")
         else:
             print("Study.")
```

```
Have fun.
```

- Other types of comparisons:

Comparison	Meaning
`==`	equal
`!=`	not equal
`<`	less than
`>`	greater than
`<=`	less than or equal
`>=`	greater than or equal

Example. Using `if-elif-else` statements, write code to only print the first 10 cubic numbers (0^3, 1^3, 2^3, 3^3, ...) that are greater than 100. Your output should look something like this:

```
    The cube of 5 is 125.
    The cube of 6 is 216.
```

and so on.

```
In [23]: for i in range(10):
             if i ** 3 > 100:
                 print("The cube of {0} is {1}.".format(i, i**3))
```

```
The cube of 5 is 125.
The cube of 6 is 216.
The cube of 7 is 343.
The cube of 8 is 512.
The cube of 9 is 729.
```

Advanced Jupyter features that might be useful

Keyboard shortcuts

- There are keyboard shortcuts, but they can be a little tricky to use. Take a look at **Help** → **Keyboard Shortcuts**.
- If you click in the text box of a code cell or double-click in a Markdown cell, then it is outlined by a green box. This is called **Edit Mode**.
- If you click on the side of a code cell, then it is outlined by a blue box. This is called **Command Mode**.
- Here are two really useful keyboard shortcuts:
    - **Indenting multiple lines.** In Edit Mode, highlight the lines you want to indent, and then press Tab. If you want to de-indent them (i.e. indent them to the left), press Shift-Tab.
    - **Line numbers.** In Command Mode, press L to show/hide line numbers in the cell.

```
In [24]: # Try turning on and turning off line numbers in this cell.
         # Play around with indenting and de-indenting code.
         # Read the code in this cell. Make sure you understand what it does!
         student_names = ["Amy", "Bob", "Carol"]
         for name in student_names:
             print("The name of this student is {0}.".format(name))

         english_words = ["home", "navy"]
         spanish = {}
         spanish["home"] = "casa"
         spanish["navy"] = "armada"

         for word in english_words:
             print("The Spanish word for {0} is {1}.".format(word, spanish[word]))
```

```
The name of this student is Amy.
The name of this student is Bob.
The name of this student is Carol.
The Spanish word for home is casa.
The Spanish word for navy is armada.
```

Running multiple cells

- You can run all the cells in a notebook by selecting **Cell** → **Run All**.
- You can run all the cells above/below the current cell by selecting **Cell** → **Run All Above/Below**.

Clearing the output of code cells

- You can clear the output of a code cell by selecting **Cell** → **Current Output** → **Clear**.
- You can clear the output of all code cells by selecting **Cell** → **All Output** → **Clear**.