Statistics 21

Python & Other Technologies for Data Science

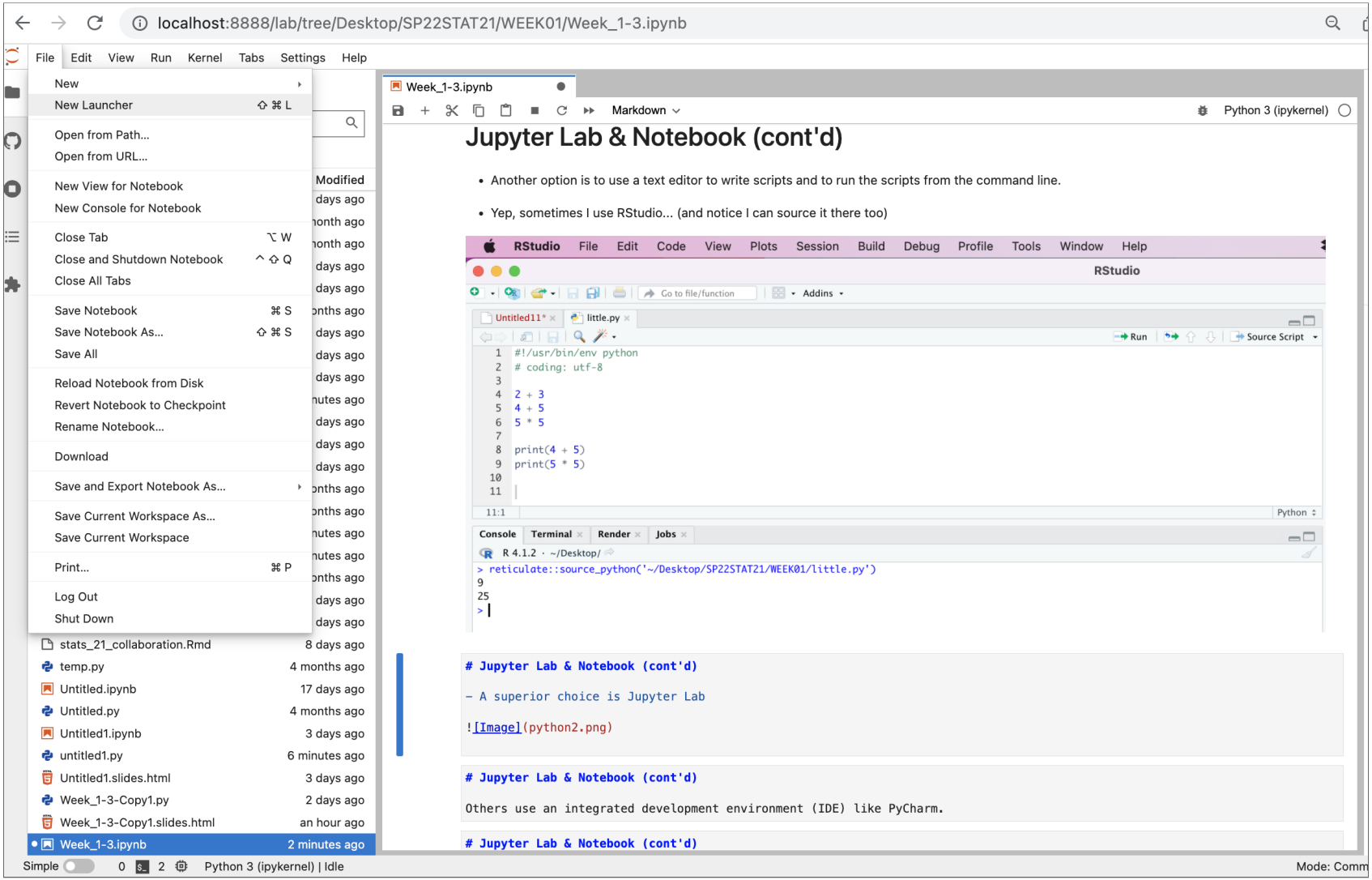
Vivian Lew, PhD - Friday, Week 1

Let’s start well

- Please try to meet someone new today
- Find three teammates, minimum team size is 4 max is 6
- Introduce yourselves, even if it is just a hello and a name
- Self assign to a team
- Take selfie/grelfie/usie (but don't upload it yet)

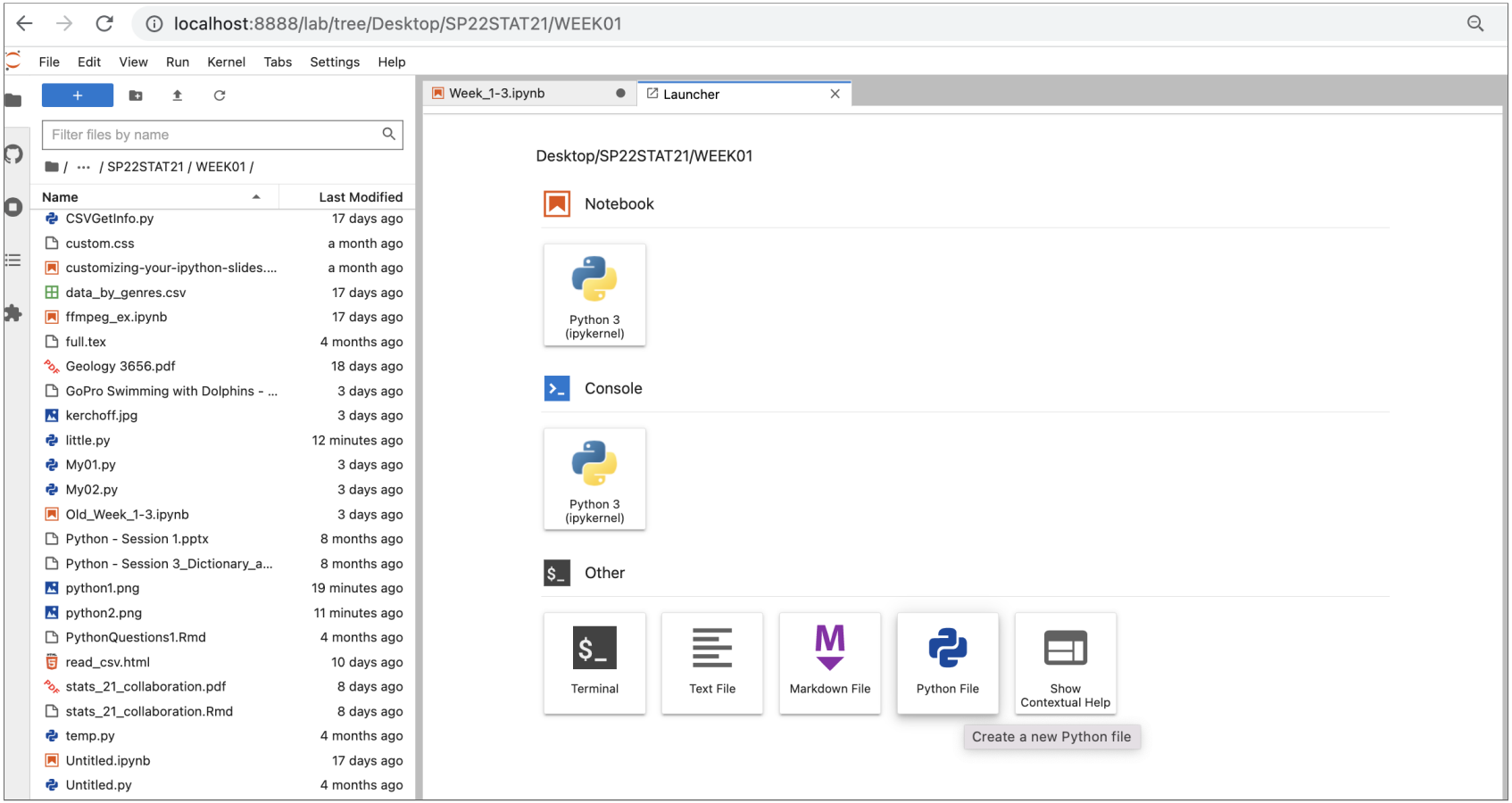
Jupyter Lab/Notebook (responses to some Qs)

- In Jupyter Lab, just create a new file or new launcher (to see your all of your choices)



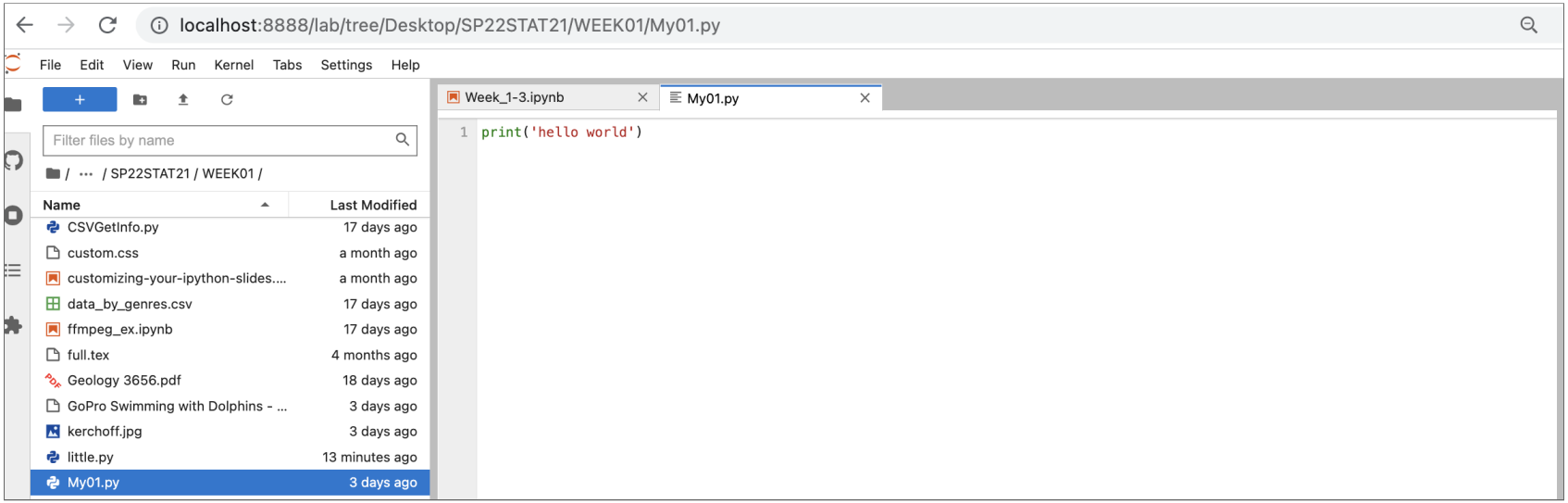
Jupyter Lab & Notebook (cont'd)

- Choose Python file (notice you can open terminal or create Python -.py- files too)



Jupyter Lab & Notebook (cont'd)

- Start typing and save it as you would any other file, the .py will be added



Jupyter Lab & Notebook (cont'd/recap)

- Jupyter Lab is an browser-based IDE centered around Notebooks.
- Jupyter Notebooks combine markdown and Python code to create a document, like an R Markdown file (R and markdown)
- It is a popular choice for data science work.
- Jupyter comes pre-installed with the Anaconda distribution.
- You can launch Jupyter Lab from the shell prompt by typing in `jupyter lab .`
- Alternatively, you can launch Jupyter Notebook alone with `jupyter notebook`

Markdown basics

https://commonmark.org/help/tutorial/

Use # symbols to indicate headings.

```
# When rendered, this would be a top level heading.
```

Make bulleted lists by using hyphens

```
- item 1
- item 2
```

heading level 1 starts with

heading level 2 starts with

heading level 3 starts with

HEADING LEVEL 4 STARTS WITH

heading level 5 starts with

heading level 6 starts wit h ######

####### ####### if you try to use 7 # symbols, it becomes normal text

Markdown basics (cont'd)

- If you want to include code, you can indicate to markdown not to render by using the back accent ` at the beginning and end: `print('Hello, World')`
- or use three tildes ~ (or 3 backticks) to indicate a code chunk.

```
for i in range(10):
    power = 2 ** i
    print(power)
```

- Use asterisks * for emphasis. Put an asterisk around a word or phrase to *italicize* it. Use two asterisks to make it **bold** .

Markdown basics (cont'd)

- Include math by using dollar signs. One dollar sign for in-line math symbols. Two dollar signs for stand-alone math equations.
- For example, you can talk about π typed `$\pi$` using in-line math.
- You write a simple equation like

```
$$E = mc^2$$
```

- Result:

$$E = mc^2$$

Markdown basics (cont'd)

Or write something more complex (checkout Detexify) like:

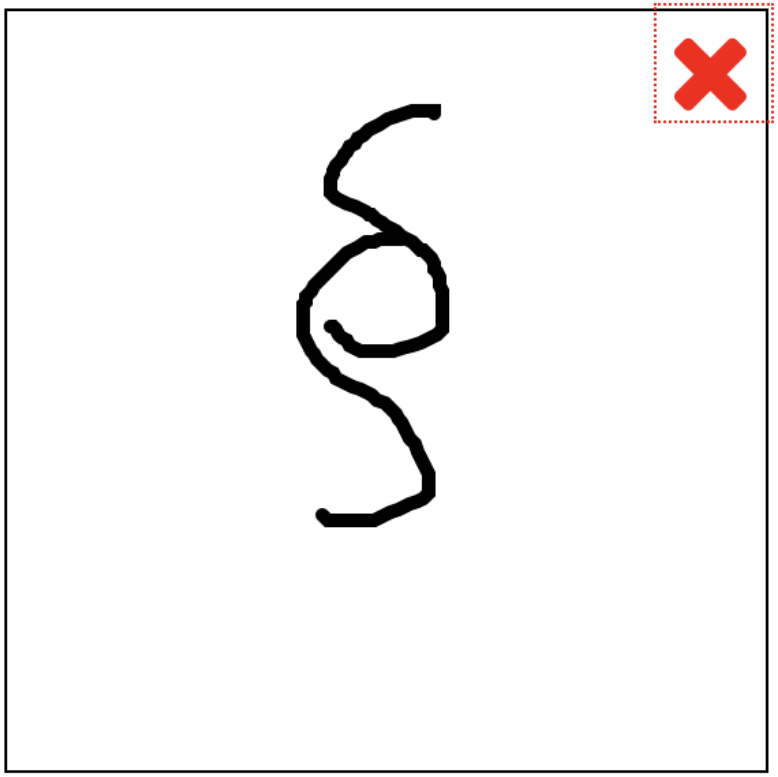
```
$$\phi(x) = \frac{1}{\sigma \sqrt{2 \pi}} \exp{
    \left( \frac{-1}{2} \left ( \frac{x - \mu}{\sigma} \right )^2 \right)} $$
```

Result:

$$\phi(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(\frac{-1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right)$$

Detexify

classify | symbols



§ Score: 0.07906932680172093
`\mathsection`
mathmode

§ Score: 0.07973780833328299
`\S`
textmode & mathmode

SS Score: 0.09138523556783595
`\SS`
textmode

§ Score: 0.09678981038646675
`\usepackage{ textcomp }`
`\textsection`
textmode

§ Score: 0.10030432028228681
`\textsection`
textmode

The symbol is not in the list? Show more

Want a Mac app?

Lucky you. The Mac app is finally stable enough. See how it works on Vimeo. Download the latest version here.

Restriction: In addition to the LaTeX command the unlicensed version will copy a reminder to purchase a license to the clipboard when you select a symbol.

You can purchase a license here:

Did this help?

Hosting Detexify costs money and if it helps you may consider helping to pay the hosting bill

Running or Rendering Cells

- You can run a cell with **Shift+Enter**. This will run the selected cell and then advance to the next cell. If you are at the last cell, it will insert another cell below it.
- If you want to run the current cell but do not want to advance to the next cell, use **Ctrl+Enter**. This will run the currently selected cell only.
- **Alt-Enter** or **Option+Enter** renders/runs the current cell and inserts a new cell below it.

IPython - interactive

Jupyter runs on IPython. As you write code in a code cell, you can use some of the features of IPython. Some basic ones:

```
In [1]: 2 + 3
```

```
Out[1]:

5
```

```
In [2]: # BUT if you have multiple operations...
        4 + 5
5 * 5
```

```
Out[2]:

25
```

IPython (cont'd)

if you want the other results to appear, you need to use print() commands to display:

```
In [3]: print(4 + 5)
        print(5 * 5)
```

```
9
25
```

notice that when you do this there is no `Out[ ]` for the cell. Why this matters --

Features of IPython: The In and Out

- IPython cells are preceeded by an `In[n]` or an `Out[n]`.
- These show the sequence you write code, but also allows you to access past entries and values

```
In [4]: In[1]
```

```
Out[4]:
'2 + 3'
```

Features of IPython: The In and Out (cont'd):

```
In [5]: Out[2]
```

```
Out[5]:

25
```

```
In [6]: ## but Out[3] doesn't exist
        Out[3]
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[6], line 2
      1 ## but Out[3] doesn't exist
----> 2 Out[3]

KeyError: 3
```

Features of IPython: The In and Out (cont'd):

In is a list (more on lists in a week or so):

```
In [7]: print(type(In))
        In
```

```
<class 'list'>
```

```
Out[7]:
```

```
['',
 '2 + 3',
 '# BUT if you have multiple operations...\n4 + 5\n5 * 5',
 'print(4 + 5)\nprint(5 * 5)',
 'In[1]',
 'Out[2]',
 "## but Out[3] doesn't exist\nOut[3]",
 'print(type(In))\nIn']
```

Features of IPython: The In and Out (cont'd):

Out is a dictionary (more on these later in the quarter)

```
In [8]: print(type(Out))
        Out
```

```
<class 'dict'>
```

```
Out[8]:
```

```
{1: 5,
 2: 25,
 4: '2 + 3',
 5: 25,
 7: ['',
  '2 + 3',
  '# BUT if you have multiple operations...\n4 + 5\n5 * 5',
  'print(4 + 5)\nprint(5 * 5)',
  'In[1]',
  'Out[2]',
  "## but Out[3] doesn't exist\nOut[3]",
  'print(type(In))\nIn',
  'print(type(Out))\nOut']}
```

The last output value

You can access the last value output using a single underscore character _

```
In [9]: 3 * 9
```

```
Out[9]:
27
```

```
In [10]: _ - 2
```

```
Out[10]:
25
```

```
In [11]: _
```

```
Out[11]:
25
```

Stat 21 teams

- Please create a team .ipynb file (if you also want to create a .py file today, go ahead) make sure your names are on it.
- Please use markdown to modify it... be as simple or as complex as your team wants to be. Try something.
- Please add your team photograph to it
- Export the .ipynb file to PDF, your photograph should be visible in it.
- Upload your team .ipynb and your PDF to Bruin Learn before 11:59pm

Executing Scripts from within Jupyter

- You can use the `%run` to execute Python scripts stored in external files.
- The `%` sets it apart these are "magic" commands

For example, I have a simple factorial script

```
In [13]: %run factorial2.py
```

```
120
```

To see all of the available "magic" commands run `%lsmagic` in a code cell.

Accessing variables defined in the notebook:

If you want the script to have access to variables that you have defined in the notebook, so here we define a url

```
In [14]: url = 'https://www.youtube.com/watch?v=LStXdttFj_o'
         print(url)
```

```
https://www.youtube.com/watch?v=LStXdttFj_o
```

Accessing variables defined in the notebook (cont'd):

Then use `%run -i` when accessing a .py script on your drive. When I don't do the right thing: (note, My02.py needs the Python library pytube to run)

```
In [15]: %run My02.py
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
File ~/Desktop/SP23STAT21/_Week01/Lecture3/My02.py:8
      1 # Download a YouTube video
      2 # need to install pytube for this to run
      3 # suggest trying it first in a virtual environment
      4 # will ask for input of a url
      6 import pytube
----> 8 pytube.YouTube(url).streams.get_highest_resolution().download()

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

Accessing variables defined in the notebook (cont'd):

- The correct way (there is no NameError thrown (we'll learn about the different ones) and our YouTube video is downloaded.)
- Without the -i flag, the script runs in a separate process and any variables or functions defined in the script are not available to the notebook.

```
In [16]: %run -i My02.py
```

Proof

In [17]:
```
from IPython.display import Video
Video("GoPro Swimming with Dolphins - Santa Cruz CA.mp4")
```

Out[17]:

Using bash/shell/cmd commands within a jupyter notebook

You can use bash commands like `cat` which displays the contents of a file by preceeding the command with an exclamation point. The commands you can use will be different for windows or unix based (mac) machines. For example, there is no `cat` command in windows, and you must use `type`

```
In [18]: !cat My02.py # Mac OS
```

```
# Download a YouTube video
# need to install pytube for this to run
# suggest trying it first in a virtual environment
# will ask for input of a url

import pytube

pytube.YouTube(url).streams.get_highest_resolution().download()
```

Using bash/shell/cmd commands within a jupyter notebook (cont'd):

- If you are using Windows, use the `type` command instead:

```
!type My02.py # Windows
```

Important Notes about Python Syntax

based on A Whirlwind Tour of Python by Jake VanderPlas

(**https://jakevdp.github.io/WhirlwindTourOfPython/**)

Comments Are Marked by #

```
In [19]: # this is a comment and is not run
```

Lines

The end of a line terminates a statement. No need for using a semi-colon to end a statement ; although you can optionally use the semi-colon to write two statements in one line.

```
In [20]: # example
         x = 5
print(x)
```

```
5
```

Lines

If you want to have a single statement cover multiple lines, you can use a backslash \ or encase the statement in parenthesis. If you are defining a list or other data structure that already uses some sort of bracket, this is handled automatically.

In [21]:
```
# semicolon to include multiple statements in one line
y = 6; z = 7
print(y + z)
```

```
13
```

Having multiple statements in one line is generally considered bad style and should be avoided.

Lines

We use the backslash or parentheses or in certain cases brackets to continue a statement over multiple lines

In [22]:
```
a = 1 + 2 + 3 \
        + 4 + 5
print(a)
```

```
15
```

In [23]:
```
# or use parenthesis
b = (1 + 2 + 3
    + 4 + 5)
print(b)
```

```
15
```

Lines (cont'd)

But some data structures (list here) use a comma to continue over multiple lines:

```
In [24]: l = ['a', 2, 3, 'd',
              'e', 6,
     'b']
print(l)
```

```
['a', 2, 3, 'd', 'e', 6, 'b']
```

(Important) Indentation defines code blocks

- Python does not use curly braces `{}` to define code blocks.
- IPython is smart enough to automatically indent lines after you use a colon `:` which indicates that the following lines are part of a code block.
- We haven't covered conditionals yet, but I'll introduce them here briefly to show how code blocks work.

In [25]:
```
# we will learn if statements later, but here's an example
        x = 8
if(x > 5):
    print('x is greater than 5')    # the two indented lines only run
    print(x)                        # when the if statement is true
print('hello')    # this line is not indented and will run regardless of if statement
```

```
x is greater than 5
8
hello
```

In [26]:
```
x = 4
    if(x > 5):
        print('x is greater than 5')    # the two indented lines only run
        print(x)                        # when the if statement is true
print('hello')    # this line is not indented and will run regardless of if statement
```

```
hello
```

In [27]:
```
x = 4
        if(x > 5):
    print('x is greater than 5')
print(x)
print('hello')
```

```
4
hello
```

Getting Help

- Official Python Documentation **https://docs.python.org/3/**
- PEP **https://peps.python.org/** (e.g., PEP 8, PEP 20)
- Connect with others -- in here and elsewhere

and the help function

```
In [28]: help(print)
```

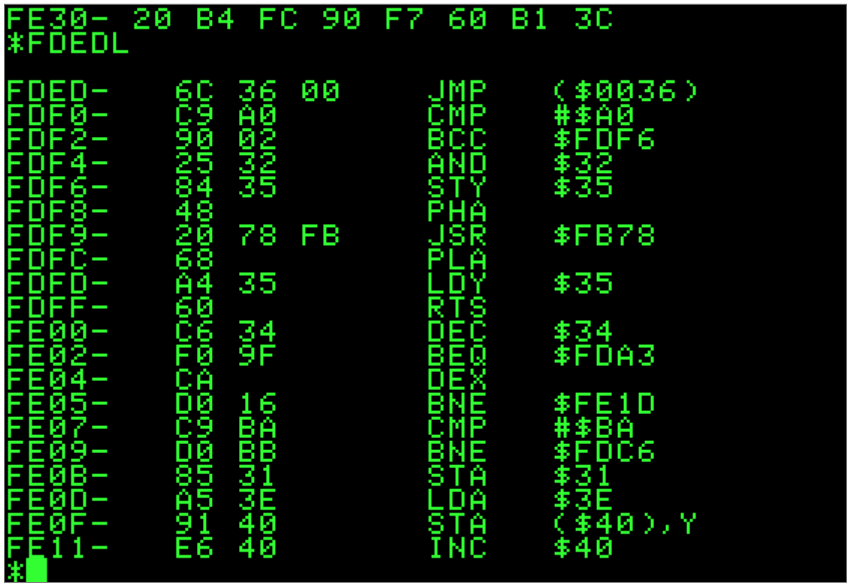
```
Help on built-in function print in module builtins:

print(*args, sep=' ', end='\n', file=None, flush=False)
    Prints the values to a stream, or to sys.stdout by default.

    sep
      string inserted between values, default a space.
    end
      string appended after the last value, default a newline.
    file
      a file-like object (stream); defaults to the current sys.stdout.
    flush
      whether to forcibly flush the stream.
```

Just some technical fundamentals if we have time

- Python is an interpreted language. When we write Python code and submit it, the Python interpreter reads the code and translates (compiles) into machine code. executing each statement as it is translated.
- Some languages compile an entire script into an executable that is run by the computing system
- This is an example of machine code:



Lexis and Semantics

- Lexis - the words and symbols used to write programs

A lexically correct example in Python (no syntax errors):

```
x = "1" * 2
```

- Syntax - the way you combine words and symbols has rules and structure

but this example breaks

- semantic rules, multiplying a string and a number together may not make sense and may not produce the desired results.

Lexical/syntax errors are easy to catch (Python will do this well). Semantic errors are a challenge - always consider "Is this what we want/need?"

Statistics 21

Have an awesome weekend!