Statistics 21

Introduction to Jupyter and IPython

Vivian Lew, PhD - Friday, Week 1

Let’s start well

- Please try to meet someone new today
- Find a teammate or two teammates, maximum team size is 3
- Introduce yourselves, even if it is just a hello and a name
- Take selfie/grelfie/usie
- Think of a team name

Jupyter Lab & Notebook

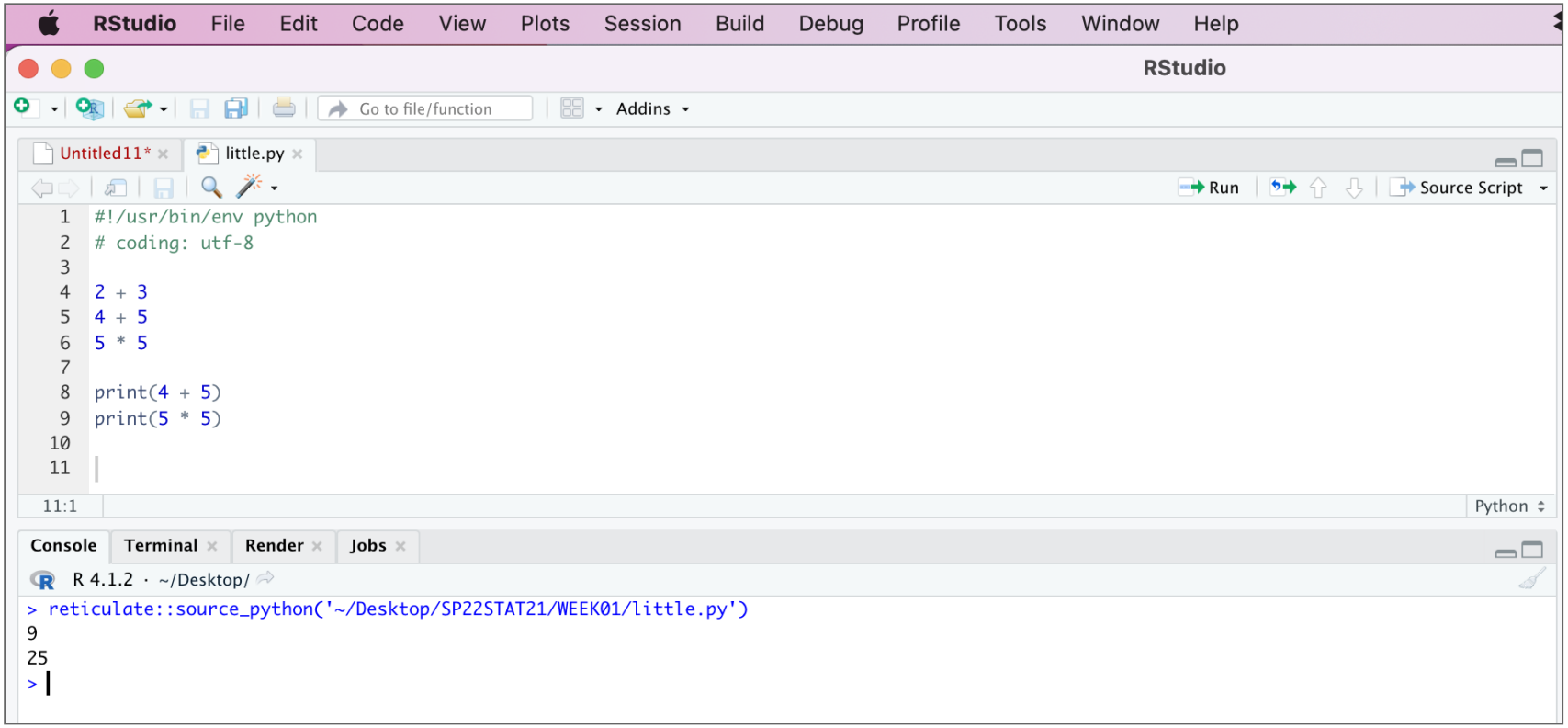
- There are many ways to run Python.
- You can run it directly in the Python interpreter in interactive mode. This is generally not recommended for doing anything other than checking a few values or expressions.

vivian — python — 80×24

```
Last login: Fri Mar 25 18:58:23 on ttys002
(base) vivian@Vivians-MacBook-Pro ~ % python
Python 3.9.7 (default, Sep 16 2021, 08:50:36)
[Clang 10.0.0 ] :: Anaconda, Inc. on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> print('hello world')
hello world
>>> x = 'Vivian'
>>> print('hello ' + x)
hello Vivian
>>>
```

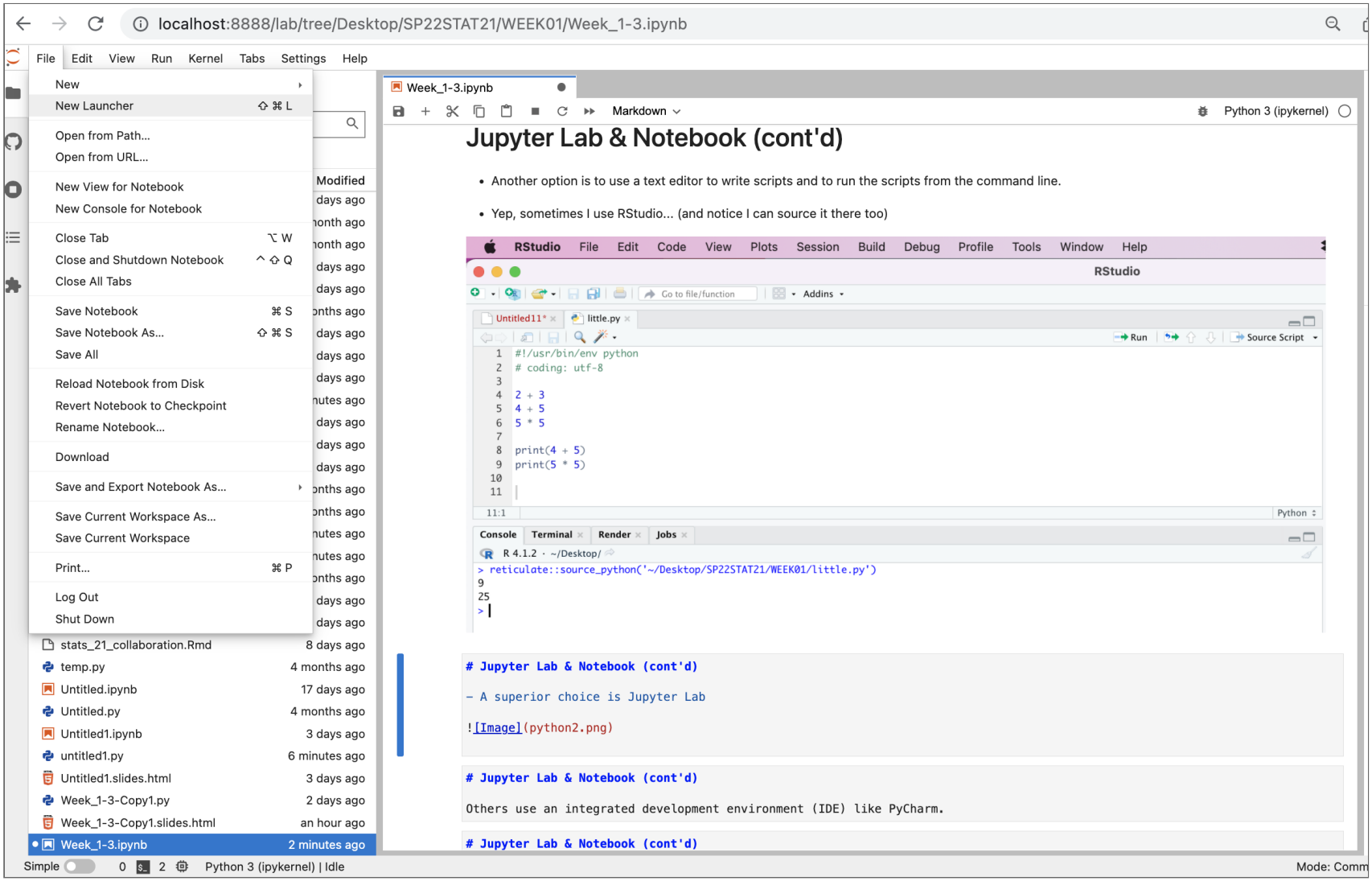
Jupyter Lab & Notebook (cont'd)

- Another option is to use any reasonable text editor to write scripts and to run the scripts from the command line.
- Yep, sometimes I use RStudio... (and notice I can source it there too)



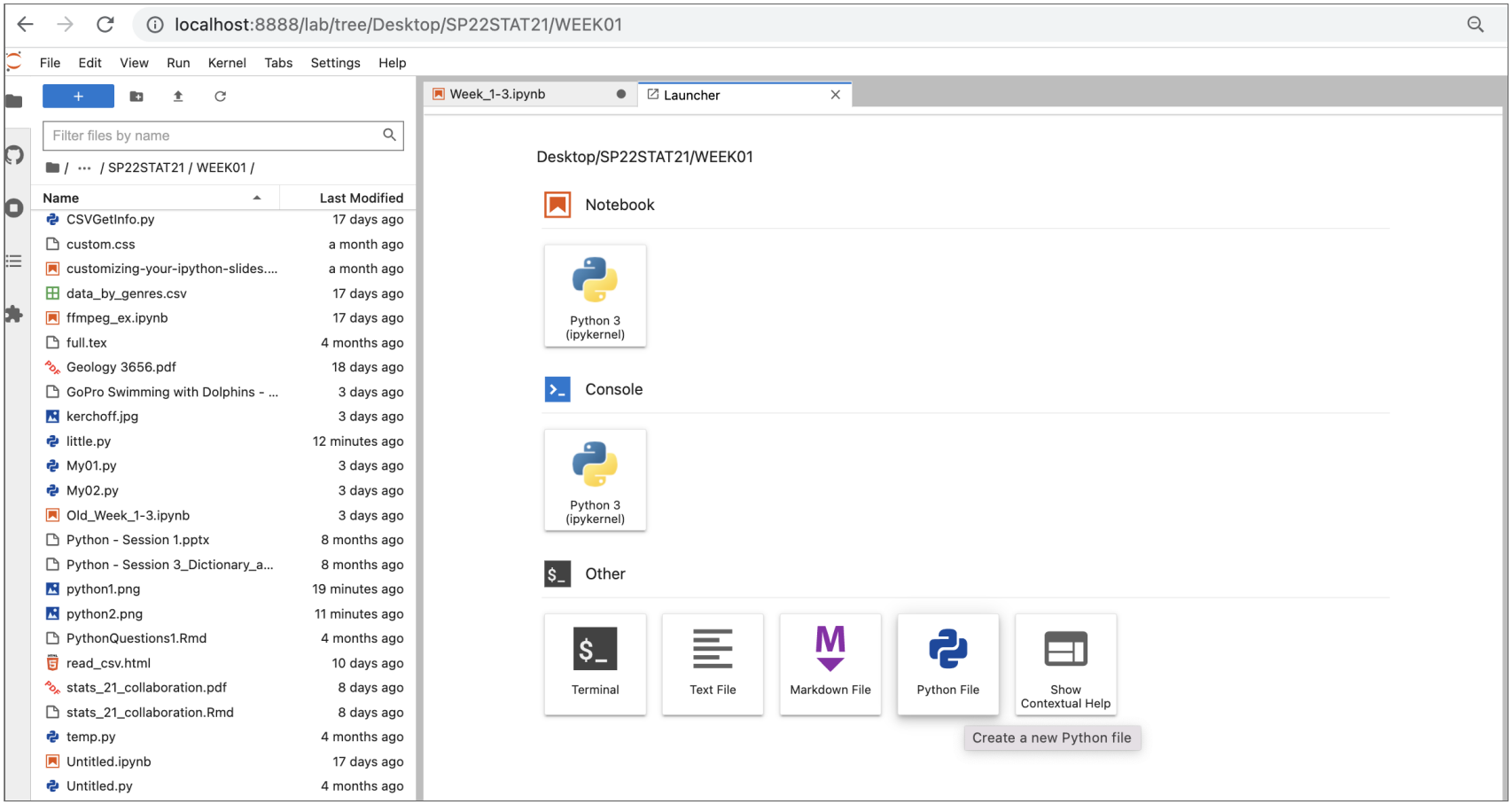
Jupyter Lab & Notebook (cont'd)

- A superior choice is Jupyter Lab, just create a new file or launcher



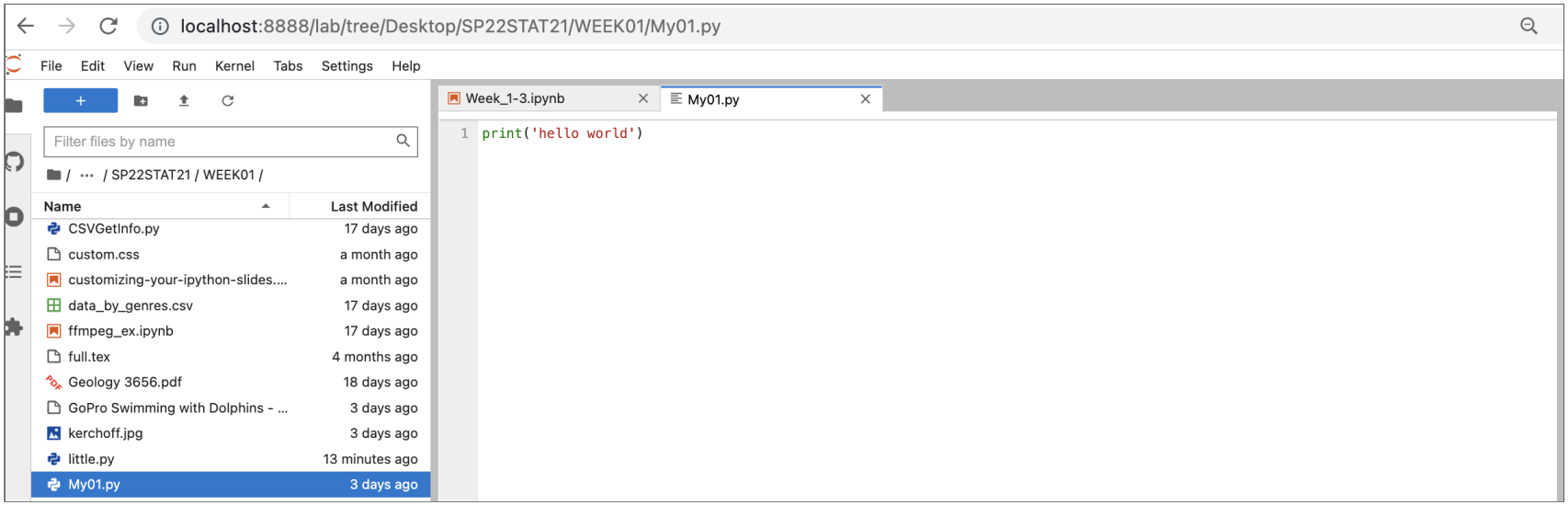
Jupyter Lab & Notebook (cont'd)

- Choose Python file



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)

Others use an integrated development environment (IDE) like PyCharm.

djtp_first_steps > polls > tests.py

```
        """
        response = self.client.get(reverse('polls:index'))
        self.assertEqual(response.status_code, 200)
        self.assertContains(response, "No polls are available.")
        self.assertQuerysetEqual(response.context['latest_question_list'], [])
        self.test
    def te
        """
        Qu
        in
        """
        cr
        re
        se
        )

    def test_index_view_with_a_future_question(self):
        """
        Questions with a pub_date in the future should not be displayed on
        the index page.
        """
        create_question(question_text="Future question.", days=30)
        response = self.client.get(reverse('polls:index'))
        self.assertContains(response, "No polls are available.",
                            status_code=200)
        self.assertQuerysetEqual(response.context['latest_question_list'], [])

    def test_index_view_with_future_question_and_past_question(self):
        """
        Even if both past and future questions exist, only past questions
        should be displayed.
        """
        create_question(question_text="Past question.", days=-30)
        create_question(question_text="Future question.", days=30)
        response = self.client.get(reverse('polls:index'))
        self.assertQuerysetEqual(
            response.context['latest_question_list'],
            ['<Question: Past question.>']
        )

    def test_index_view_with_two_past_questions(self):
        """
```

```
test_index_view_with_a_future_question(self)        QuestionViewTests
test_index_view_with_a_past_question(self)          QuestionViewTests
test_index_view_with_future_question_and_past_question   QuestionVi…
test_index_view_with_no_questions(self)             QuestionViewTests
test_index_view_with_two_past_questions(self)       QuestionViewTests
_testMethodDoc                                      TestCase
_testMethodName                                     TestCase
countTestCases(self)                                TestCase
defaultTestResult(self)                             TestCase
^↓ and ^↑ will move caret down and up in the editor >>
```

Statement seems to have no effect. Unresolved attribute reference 'test' for class 'QuestionViewTests'.

25:18 LF UTF-8 Git: master

Jupyter Lab & Notebook (cont'd)

- 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
- 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`

Stat 21 teams

-Let's take a moment talk about it

-What do you like to use to edit scripts and why? (convenience? features?)

-Listen to what your teammates have to say.

-Then summarize your team's opinion in a few sentences or even one sentence.

- Upload your team photograph and your opinion statement to Bruin Learn before 11:59pm

Jupyter has two modes:

command mode

- Type `Esc` to enter command mode

edit mode

- Type `Enter` / `Return` to enter edit mode

Try to avoid mixing up command mode and edit mode (frustrating and unproductive)

Keyboard Shortcuts

- We encourage you to learn and use the keyboard shortcuts if you want to gain speed and proficiency.
- Try to do as much as you can without touching your mouse or trackpad.

Adding Cells (command mode)

- Type **b** to add a new cell below your current cell.
- Type **a** to add a new cell above your current cell.

Deleting Cells (command mode)

- Type **dd** to delete a cell. That is type the letter d twice.

Keyboard Shortcuts (cont'd)

Cut, Copy, Paste (command mode)

While you have a cell selected in command mode, you can use

- **x** to cut
- **c** to copy
- **v** to paste (it will paste the cell below the selected cell)

Don't hold ctrl, just type the letter.

Navigating (command mode)

- You can use the up or down arrows to switch cells.
- If you don't want to leave your home row, you can also use **j or k** to move up and down.

Jupyter has three types of cells:

- Markdown Cells
    - used for text
- Code Cells
    - used to run code
- raw cells (used infrequently)

While in command mode, you can convert a cell to markdown by typing **m**
You can convert a cell to code by typing **y**

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

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
```

Markdown basics (cont'd)

- If you want to include code, you can indicate to markdown not to render by using the back accent ` and the beginning and end of one line, or use three tildes ~ to indicate a code chunk.
- Use asterisks *for emphasis*. *Put on 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 a more complex equation like:

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

Result:

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

Running or Rendering Cells

- You can run a code 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
```

note that when you do this, notice 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]:

```
print(Out[2])
print(Out.get(2))
```

```
25
25
```

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

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

In [6]:

```
print(type(Out))
```

```
Out
```

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

Out[6]:

```
{1: 5, 2: 25, 4: '2 + 3'}
```

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

In is a list (more on these 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]',
 'print(Out[2])\nprint(Out.get(2)) ',
 'print(type(Out))\nOut',
 'print(type(In))\nIn']
```

”math operations” with strings

multiplication repeats the string

In [8]:

```
3 * 'hello!'
```

Out[8]:

```
'hello!hello!hello!'
```

more ”math operations” with strings

addition concatenates strings

In [9]:

```
"hi" + "bye"
```

Out[9]:

```
'hibye'
```

Executing Scripts from within Jupyter

You can use the `%run` (a function special to iPython, the `%` sets it apart) to execute python scripts stored in separate files. For example, I have a simple script that simply prints hello world stored in a script

In [10]:
```
%run My01.py
```

```
hello world
```

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 [11]:
```
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 - see what happens when I don't do the right thing: (note, My02.py needs the Python library pytube to run)

```
In [12]:
%run My02.py
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~/Desktop/SP22STAT21/WEEK01/My02.py in <module>
      6 import pytube
      7
----> 8 pytube.YouTube(url).streams.get_highest_resolution().download()
      9
     10

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.):

In [13]:
```
%run -i My02.py
```

Proof

In [14]:

```
from IPython.display import Video
Video("GoPro Swimming with Dolphins - Santa Cruz CA.mp4")
```

Out[14]:

Stat 21 teams

- Let's take a moment talk about Python libraries/packages/modules informally

- Turn to your teammate(s) and find out if the teammate has experience installing libraries

- Listen to what your teammates have to say.

- As a class Let's talk about installation informally

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 [15]:
```
!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
```

The last output value

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

In [16]:
```
3 * 9
```

Out[16]:

```
27
```

In [17]:
```
_ - 2
```

Out[17]:

```
25
```

In [18]:
```
_
```

Out[18]:

```
25
```

Help

In [19]:
```
help(print)
```

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

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=Fals
e)

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

Important Notes about Python Syntax

based on A Whirlwind Tour of Python by Jake VanderPlas

Comments Are Marked by #

In [20]:
```
# 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 [21]:

```
# 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 [22]:

```
# 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 [23]:

```
a = 1 + 2 + 3 \
    + 4 + 5
print(a)
```

```
15
```

In [24]:

```
# or use parenthesis
b = (1 + 2 + 3
    + 4 + 5)
print(b)
```

```
15
```

Lines (cont'd)

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

In [25]:
```
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 [26]:

```
# 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 [27]:

```
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 [28]:

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

```
4
hello
```