PH1110/1111 LAB 1R
E1/E2 2021

Welcome!

HOW TO ASK FOR HELP

- Google
- Talk to classmates
- ARC/MASH tutors
- Academic Honesty
- Accessibility

We are on your side, and we want you to succeed.

Any questions?

Welcome!

ABOUT THESE LABS

Skills-based - we will NOT be copying lecture content

Learning Objectives:

- Scientific Literacy
- Lab Design
- Basic Programming w/ Python

Welcome!

GOALS FOR LAB 1

- Get comfortable with Python
- Learn some data analysis skills
- Be able to talk about error
  - Uncertainty: point vs. set
  - Accuracy vs. precision

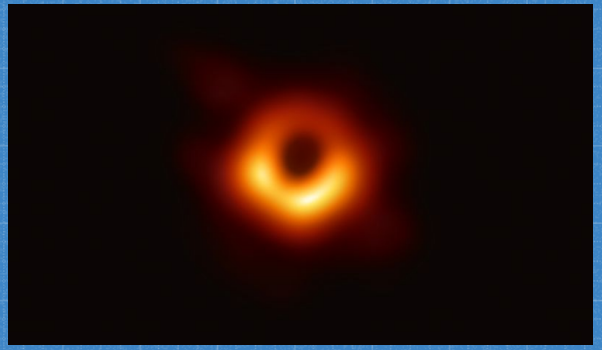
1

INTRODUCTION TO PYTHON

Let’s start with the basics

PROGRAMMING FOR PHYSICS - AN EXAMPLE

In 2019, scientists used data from the Event Horizon Telescope to capture the first ever photo of a black hole. To assemble a complete image from incomplete telescope data, the scientists had to develop advanced imaging algorithms to fill in the gaps.



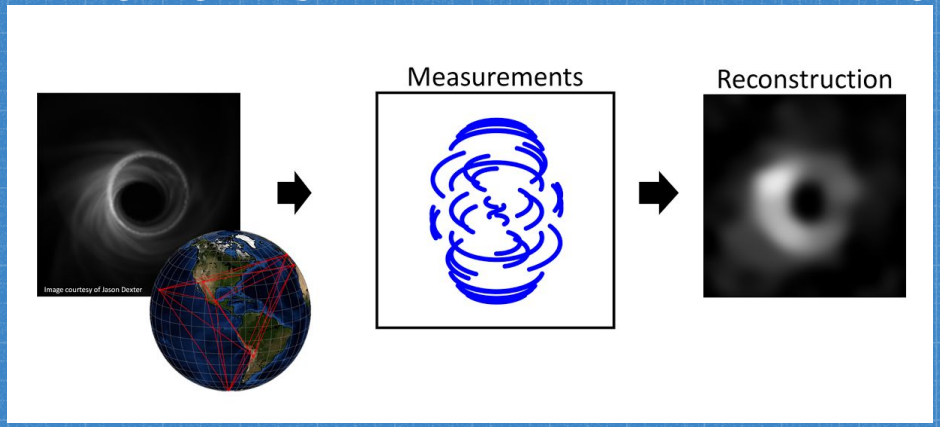


Image credit: Katie Bouman at EHT

WHY PYTHON?

Computer programming is an important skill for scientists and engineers

Python in particular is:

- Free
- Easy to learn
- Widely used and supported
- Good for data science applications

THIS IS AN INTERACTIVE TUTORIAL

Please follow along on your own computer!

There will be a few mandatory participation checkpoints.

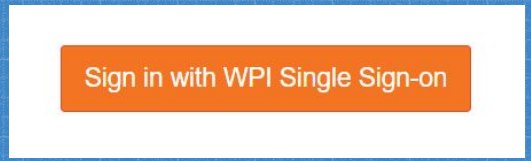
Use the chat as a discussion board. You may also send me questions directly if you'd rather ask anonymously, or speak up at any point.

NOTE: If you already have some programming experience, this will be review for you. Help out your classmates! You may also look ahead to section III in the lab guide.

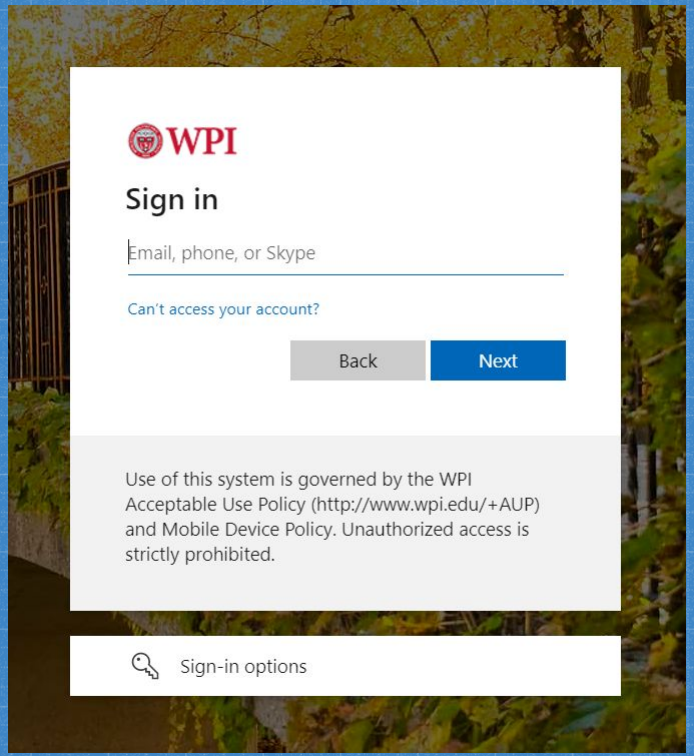
0. SETTING UP THE ENVIRONMENT

In your browser of choice, go to jupyterhub.wpi.edu

Press the button in the center of the screen:



Sign in using your WPI email credentials

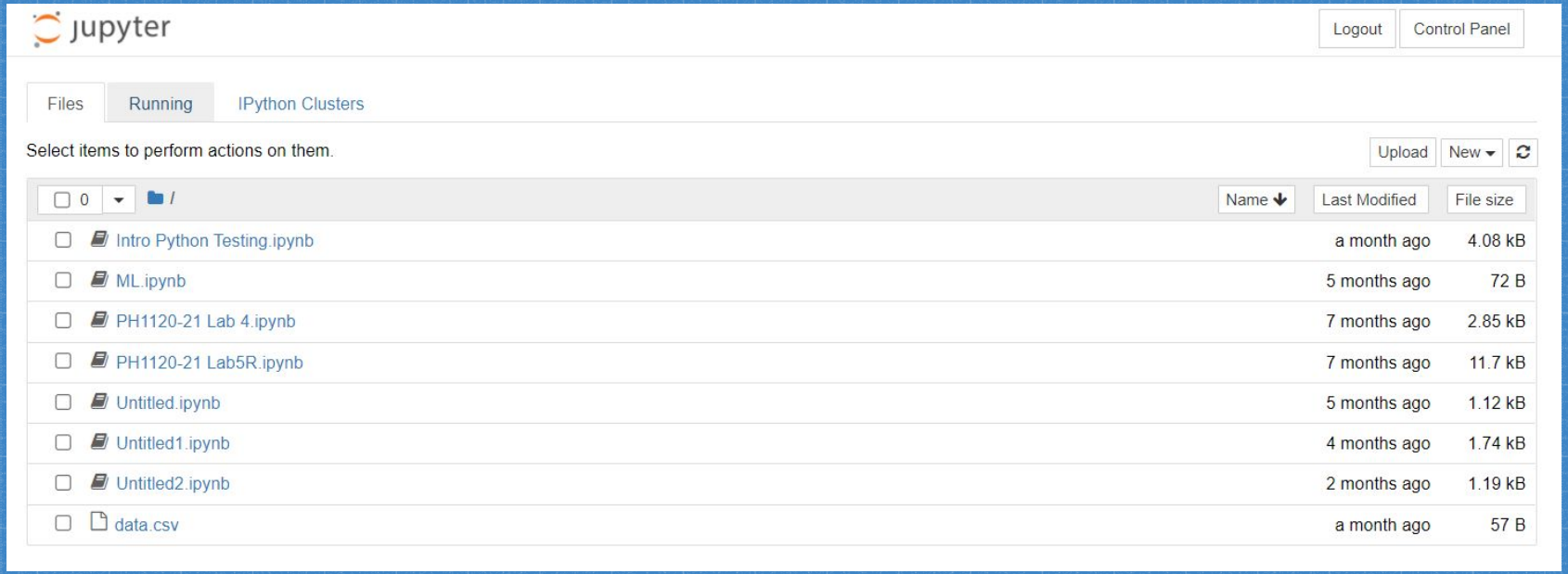


0. SETTING UP THE ENVIRONMENT

Click the “Spawn” button to start a new session. We’re not doing anything too fancy, so the profile you pick doesn’t matter.

0. SETTING UP THE ENVIRONMENT

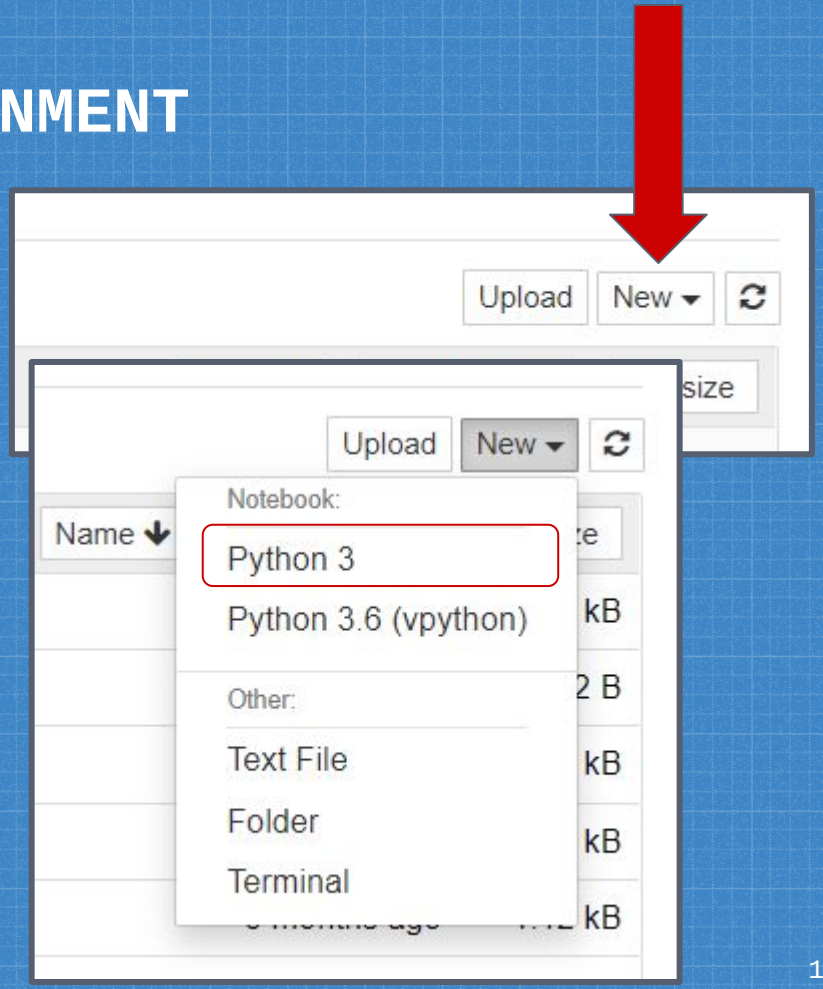
Your screen should now look something like this:



0. SETTING UP THE ENVIRONMENT

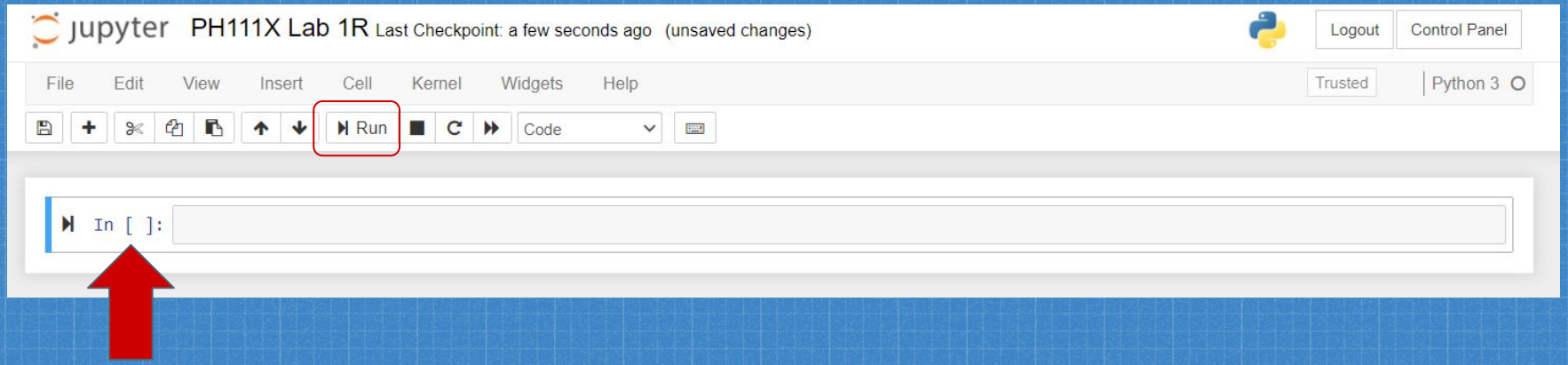
Make a new Jupyter Notebook using Python 3.

You will be submitting this notebook on Canvas instead of a lab report this week.



0. SETTING UP THE ENVIRONMENT

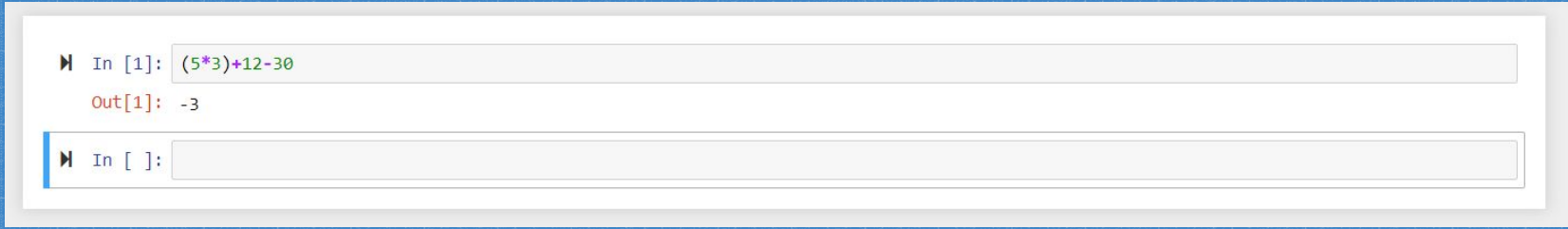
You now have a brand new notebook to play around with! It doesn’t look like much yet, but we’ll change that.



This is a cell. You can write code here, then press the Run button to execute the code in the selected cell.

1. PYTHON BASICS

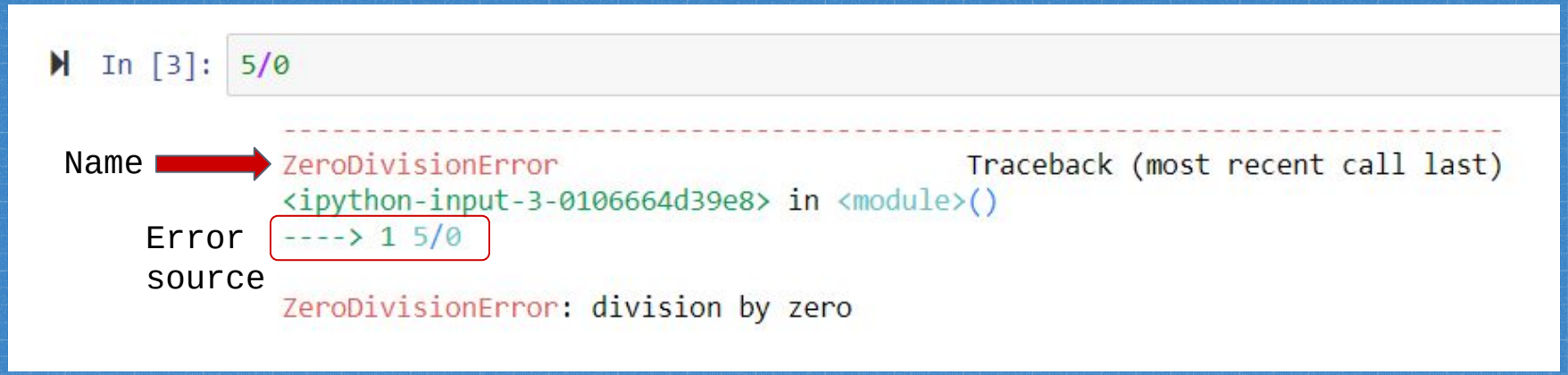
At its simplest, Python can be used as a calculator. Type in the expression you want to solve, hit Run, and you'll get an answer.



TRY IT OUT: Come up with some simple expressions for Python to solve. What happens if you put more than one in the same cell? What happens when you try to divide by zero?

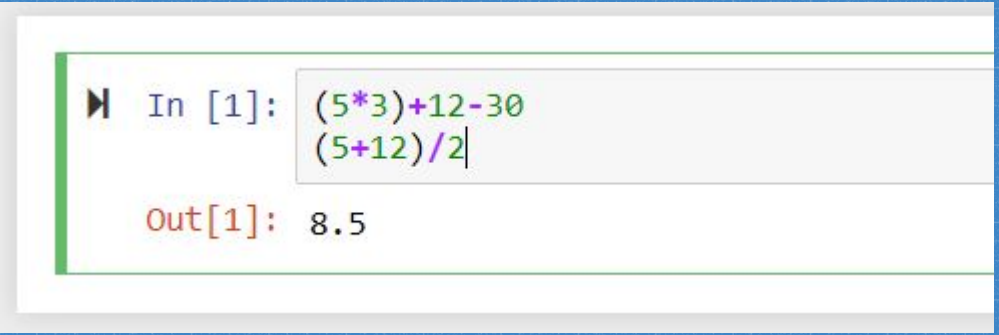
1. PYTHON BASICS

When Python encounters an error it will stop running and display a **traceback**, which shows you the name of the error and where it occurred. If you get an error that you're not sure how to fix, google its name or ask your instructor.



1. PYTHON BASICS

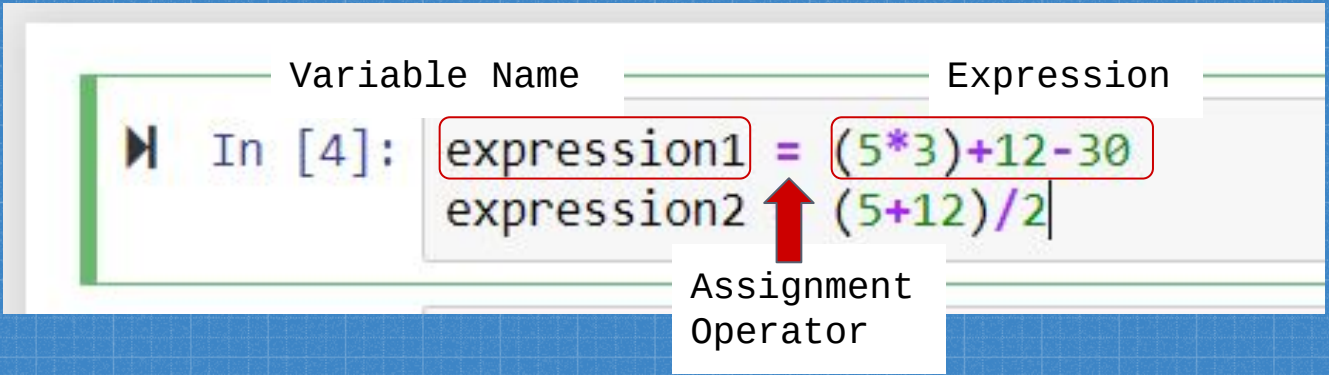
Note that if you try to solve multiple equations this way at once, Python will only give you the answer, or **return value**, for the last line.



So, how do we get Python to do more complicated stuff?

1. PYTHON BASICS - VARIABLES

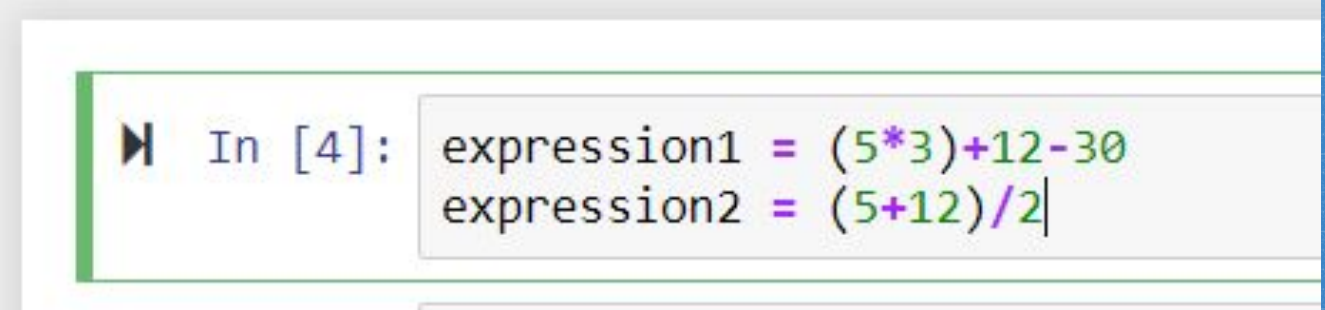
Variables can be thought of as labels or containers for pieces of data. When you assign something to a variable, you are storing it for later use.



In Python, '=' does NOT mean equals! It is only used to assign values to variables.

1. PYTHON BASICS - VARIABLES

The rules we need to follow for a language to properly read our code are called are called its **syntax**. Python syntax is simple, but may not be intuitive at first.



TRY IT OUT: Go back and assign your expressions to variables, then run the cell again. What happens this time? Try out different names for your variables. Which names are and aren't allowed? Why do you think that is?

1. PYTHON BASICS - COMMENTS

It’s good practice to give your variables descriptive names and, if needed, add **comments** to explain what’s going on in your code. Anything to the right of a ‘#’ symbol will be treated as a comment, not active code.

```
# This is a comment! Python will ignore everything on this line

# These are some sample calculations
expression1 = (5*3)+12-30
expression2 = (5+12)/2
my_sum = expression1 + expression2
```

1. PYTHON BASICS - COMMENTS

```
# This is a comment! Python will ignore everything on this line

# These are some sample calculations
expression1 = (5*3)+12-30
expression2 = (5+12)/2
my_sum = expression1 + expression2
```

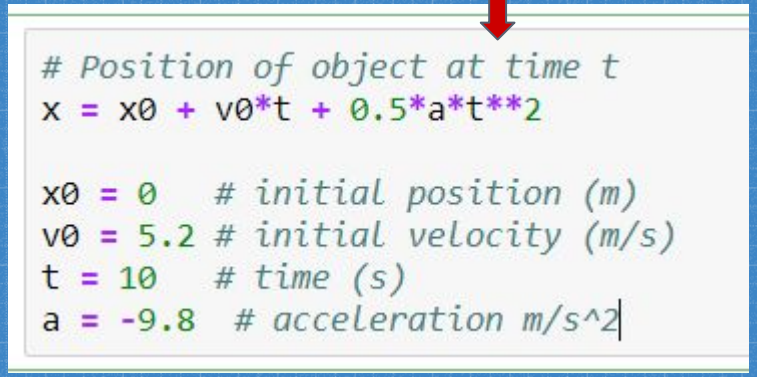
TRY IT OUT: Add a comment at the top of your Jupyter notebook with your name and today's date. Use comments to describe what different variables represent or leave notes for your future self.

1. PYTHON BASICS - VARIABLES

Variables must be created, or **declared**, before they can be used. The order matters!

For example, this code snippet would not work - trying to run it would give an error, since Python doesn't recognize any of the variables on the right side of the equation.

Double asterisks '**' are the power (exponent) operator in Python



```
# Position of object at time t
x = x0 + v0*t + 0.5*a*t**2

x0 = 0   # initial position (m)
v0 = 5.2 # initial velocity (m/s)
t = 10   # time (s)
a = -9.8  # acceleration m/s^2
```

1. PYTHON BASICS - FUNCTIONS

A **function** is a block of code that can be run when called. Many functions take one or more **arguments**, which represent data that you are giving to the function.

Python's built in **print() function** will output the value of x, or whatever else it is passed as an **argument**.

```
x0 = 0   # initial position (m)
v0 = 5.2 # initial velocity (m/s)
t = 10   # time (s)
a = -9.8  # acceleration m/s^2

# Position of object at time t
x = x0 + v0*t + 0.5*a*t**2
print(x)
```

```
-438.00000000000006
```

Is this digit relevant? Think about reasonable levels of precision.

1. PYTHON BASICS - FUNCTIONS

The print() function takes any number of arguments of different types, separated by commas, and outputs them on one line.

```
my_number = 7
temperature = 22.5
greeting = "Hello world!"

print(greeting)
print("My favorite number is:", my_number)
print("Today it's", temperature, "C outside.")
```

```
Hello world!
My favorite number is: 7
Today it's 22.5 C outside.
```

TRY IT OUT: In a new cell, create some variables like the ones above and use print statements to display their values with descriptions.

1. PYTHON BASICS - FUNCTIONS

Python has more built in functions, most of which aren't useful for our purposes. Here are some that are:

If we want more relevant functions, we'll need to write our own and/or import some more.

```
# absolute value of a number
abs(num)

# min or max value from args
min(a0, a1, a2, ...)
max(b0, b1, b2, ...)

# power operation: same as a**b
pow(a, b)

# prints argument(s) out
print(arg1, arg2, ...)

# rounds number to n digits after the decimal
round(number, n)
```

1. PYTHON BASICS - VARIABLES

Variables must be created, or **declared**, before they can be used. The order matters!

Moving the variable declarations up makes the code run, but it still doesn't output an answer...

```
x0 = 0   # initial position (m)
v0 = 5.2 # initial velocity (m/s)
t = 10   # time (s)
a = -9.8  # acceleration m/s^2

# Position of object at time t
x = x0 + v0*t + 0.5*a*t**2
```

1. PYTHON BASICS - FUNCTIONS

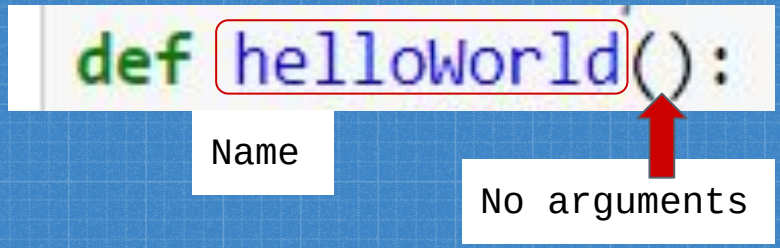
Here's the definition for a simple function.

Always start with a descriptive comment.

```
# helloWorld: prints out a greeting
def helloWorld():
    print("Hello world!")

helloWorld()
```

```
Hello world!
```

The first line defines the name of the function and how many arguments it accepts.



1. PYTHON BASICS - FUNCTIONS

Here's the definition for a simple function.

Always start with a descriptive comment.

```
# helloWorld: prints out a greeting
def helloWorld():
    print("Hello world!")

helloWorld()
```

```
Hello world!
```

Everything indented below the first line is called the **body** of the function. This is the code that will run when the function is called!

```
def helloWorld():
    print("Hello world!")
```

Indentation Matters!

1. PYTHON BASICS - FUNCTIONS

Here's the definition for a simple function.

Always start with a descriptive comment.

```
# helloWorld: prints out a greeting
def helloWorld():
    print("Hello world!")

helloWorld()
```

```
Hello world!
```

This isn't part of the function definition, but it shows how we call (or use) our new function, and its effect.

```
helloWorld()
```

```
Hello world!
```

1. PYTHON BASICS - FUNCTIONS

This function is very similar, but it takes one argument - a string.

Note that when a function is called, it replaces every instance of an argument's name with the corresponding passed in value.

```
# helloFriend: prints out a customized greeting
# -> a_name: String representing a name

def helloFriend(a_name):
    print("Hello", a_name)

helloFriend("Gompei")
helloFriend("everyone")
```

```
Hello Gompei
Hello everyone
```

1. PYTHON BASICS - FUNCTIONS

Here’s that propagation of error equation we used to use

```
import math

# u_area: propagation of uncertainty for area calculation
# -> x: width measured (cm)
# -> dx: error of width (cm)
# -> y: length measured (cm)
# -> dy: error of length (cm)
# <- uncertainty of calculated area
def u_area(x, dx, y, dy):
    area = x * y
    x_component = (dx / abs(x)) ** 2
    y_component = (dy / abs(y)) ** 2
    answer = area * math.sqrt(x_component + y_component)
    return answer

width, length = 5.3, 8.5
u_width = u_length = 0.1

area = width * length
print("Area =", area, "+/-", u_area(width, u_width, length, u_length))
```

```
Area = 45.05 +/- 1.0016985574512922
```

TRY IT OUT: In a new cell, create some variables like the ones above and use print statements to display their values with descriptions.

1. PYTHON BASICS - FUNCTIONS

This one’s a bit more complicated. Instead of printing something, it returns a value that can be saved for later!

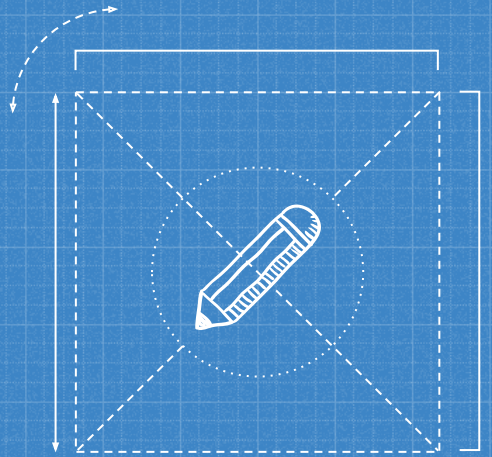
It also uses the square root function from Python’s math module.

```python
import math

# u_area: propagation of uncertainty for area calculation
# -> x: width measured (cm)
# -> dx: error of width (cm)
# -> y: length measured (cm)
# -> dy: error of length (cm)
# <- uncertainty of calculated area
def u_area(x, dx, y, dy):
    area = x * y
    x_component = (dx / abs(x)) ** 2
    y_component = (dy / abs(y)) ** 2
    answer = area * math.sqrt(x_component + y_component)
    return answer

width, length = 5.3, 8.5
u_width = u_length = 0.1

area = width * length
print("Area =", area, "+/-", u_area(width, u_width, length, u_length))
```

```
Area = 45.05 +/- 1.0016985574512922
```

GROUP WORK

FUN WITH FUNCTIONS



Introduce yourselves! Name, prospective major, fun facts?

Work together to create functions for each of these equations:

- Volume of a cylinder
- Pythagorean Theorem
- Quadratic formula*

*extra credit

Hint: The Python math module will make this much easier for you.

2

LET’S ANALYZE SOME DATA

So, what can Python actually do?

2. DATA ANALYSIS REFERENCE

Modules are collections of functions and variables that must be imported into your code before they can be used.

```
import math
import numpy
import matplotlib.pyplot as plt
import scipy.stats as stat
```

Modules with long names can be given nicknames for convenience. These four modules are the ones we will be using in this course -- they are all available through Jupyterhub, no need to download or install anything.

2. DATA ANALYSIS REFERENCE

NumPy in particular is useful because it lets us create and analyze data sets in the form of **arrays**.

Arrays are zero-indexed, and all of their elements must have the same type (string, number, etc.).

NumPy Documentation

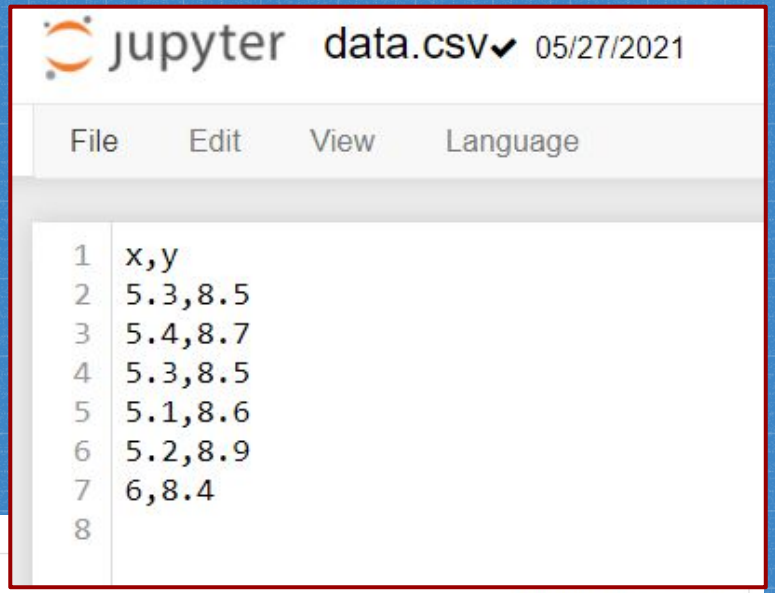
```
import numpy as np

x = np.array([1, 2, 3, 4, 5])
colors = np.array(["blue", "green", "red"])

# print the first element in colors
print(colors[0])

# print the third element of x
print(x[2])
```

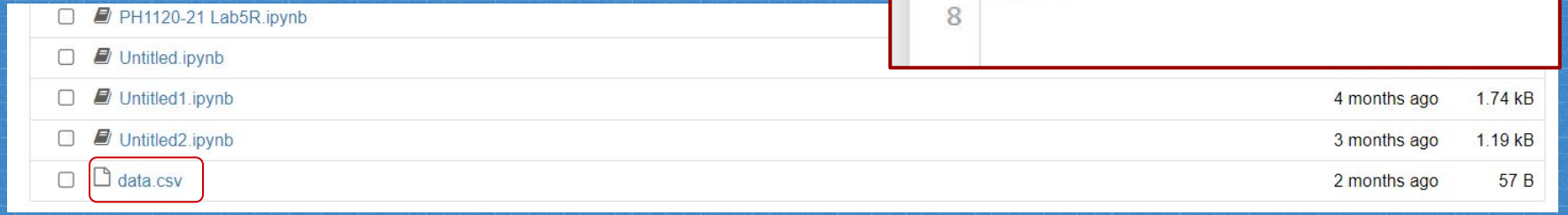
```
blue
3
```

2. DATA ANALYSIS REFERENCE

NumPy also lets you load data files as arrays for easy analysis.

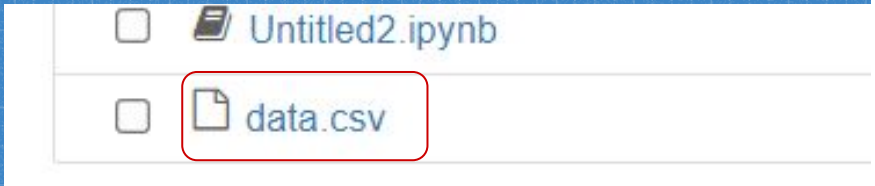
To use a file like this, you must first upload it to your Jupyter workspace.





2. DATA ANALYSIS REFERENCE

Once the file is uploaded, you can load it with the genfromtxt() function as shown here. Remember to use the exact name of the file!




```
import numpy as np

my_data = np.genfromtxt('data.csv', delimiter=',')
# delete the top row (column names)
my_data = np.delete(my_data, 0, 0)

print(my_data)
```

```
[[5.3 8.5]
 [5.4 8.7]
 [5.3 8.5]
 [5.1 8.6]
 [5.2 8.9]
 [6.  8.4]]
```

TRY IT OUT: On Canvas, find the file lab1data.csv in the lab module. Download this file, then upload it to Jupyter as shown here.

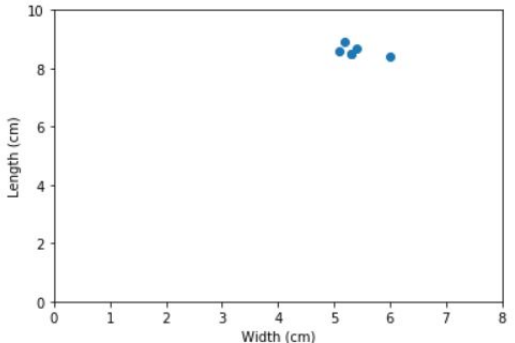
2. DATA ANALYSIS REFERENCE

Matplotlib's Pyplot provides highly flexible graphing functions we can use to visualize our data. This lets us get a good general idea of the qualitative properties of a data set, and helps us spot outliers before doing any math.

TRY IT OUT: Follow the example code to graph the data you just uploaded. Don't forget axis labels!

```
import numpy as np

my_data = np.genfromtxt('data.csv', delimiter=',')
# delete the top row (column names)
my_data = np.delete(my_data, 0, 0)

# print(my_data)
```

```
import matplotlib.pyplot as plt

# this makes it easier to plot our data
my_data = my_data.transpose()


plt.scatter(my_data[0], my_data[1])
plt.axis([0, 8, 0, 10])  # defines range of x and y axes
plt.xlabel("Width (cm)")
plt.ylabel("Length (cm)")
plt.show;
```



2. DATA ANALYSIS REFERENCE

Arrays often act as data tables, and NumPy offers many convenient functions for analyzing them. Here, we find the mean and standard deviation for each column, and save the values in new arrays.

```
import numpy as np

my_data = np.genfromtxt('data.csv', delimiter=',')
# delete the top row (column names)
my_data = np.delete(my_data, 0, 0)

# average and standard deviation of each column
# axis=0: column, axis=1: row
mean = np.mean(my_data, axis=0)
stdv = np.std(my_data, axis=0)

print("x =", mean[0], "+/-", stdv[0])
print("y =", mean[1], "+/-", stdv[1])
```

```
x = 5.383333333333333 +/- 0.2910708199428831
y = 8.6 +/- 0.16329931618554516
```

TRY IT OUT: Find the mean and standard deviation of your data set and print the values as shown here. How many digits to the left of the decimal should you keep? Think about it, then check your answer.

3

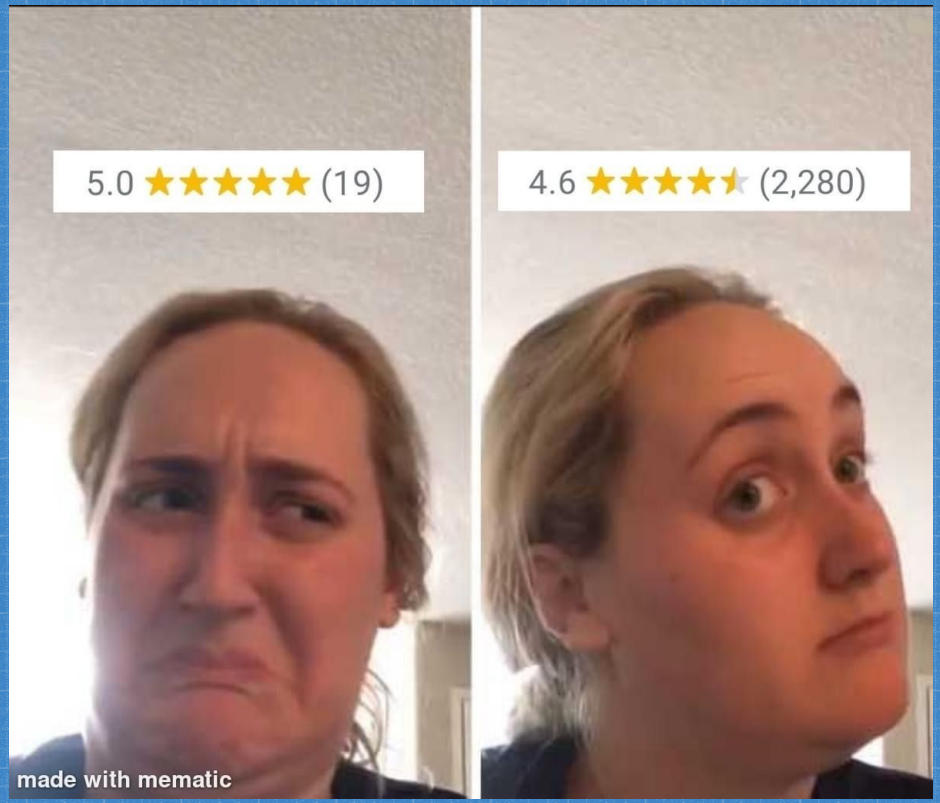
UNCERTAINTY AND MORE?

How scientists talk about error

3. THINKING ABOUT DATA

Which would you trust more, a single review or the average of five reviews?

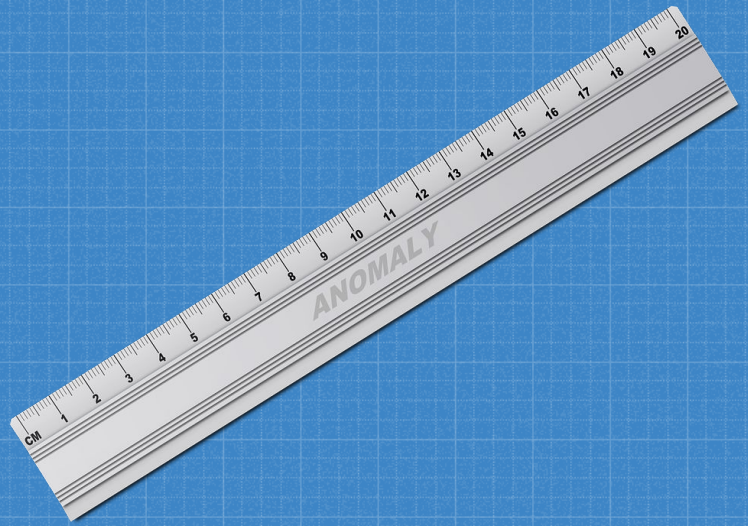
You probably already have an intuitive understanding of how sample size affects the way you look at and trust information.



3. THINKING ABOUT DATA

The error of a single measurement is mostly determined by the precision of the measuring tool, plus any external factors. As such, uncertainty can often be attributed to **instrumental limitations**.

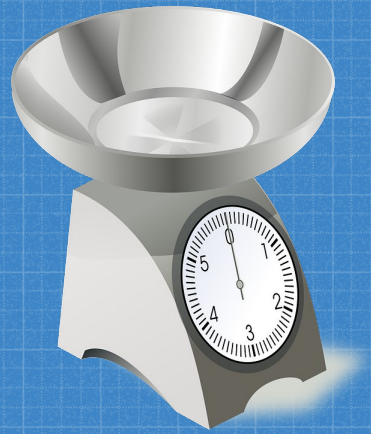
Measurements made with a ruler like this, for example, will always have an error of at least 0.1cm (the smallest unit marked); more if it's difficult to tell where the object's boundaries are.



3. THINKING ABOUT DATA

Systematic errors are caused by factors that do not change between measurement. They don’t affect the uncertainty, but they can cause a shift in the average value. A common source of these is improperly calibrated equipment.

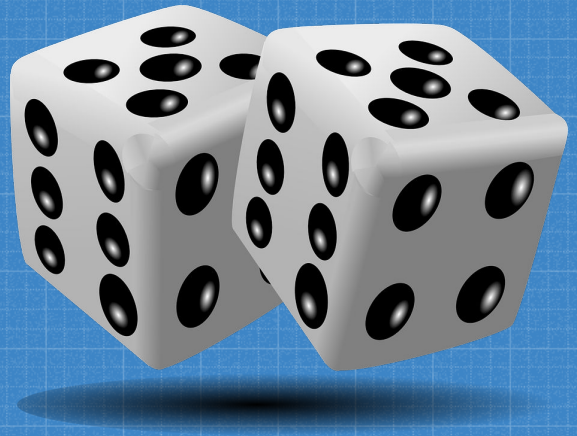
If a scale isn’t zeroed correctly before measurements are made, it can cause systematic error.



3. THINKING ABOUT DATA

Single measurements are far more likely to be affected by **random error**, or slight variations in environment and process. Combining the results from multiple trials will reduce the effect of random errors and improve precision.

Not to be confused with **human error**, which means “I did the experiment wrong” and is not a legitimate source of error!



3. THINKING ABOUT DATA

In common speech, **precision** and **accuracy** mean the same thing. In a scientific context, they have similar but distinct meanings! Both refer to qualities of a data set:

- Precision: How closely independent measurements of the same thing are clustered together around some point that may or may not be the “true” value
- Accuracy: How close the average value from an independent set of measurements is to the “true” value

3. THINKING ABOUT DATA

A. Low-precision, Low-accuracy:

The average (the X) is not close to the center

B. Low-precision, High-accuracy:

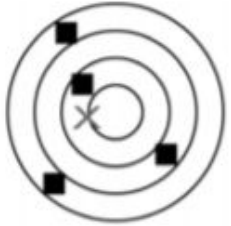
The average is close to the true value, but data points are far apart

C. High-precision, Low-accuracy:

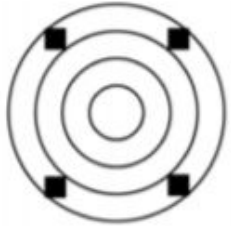
Data points are close together, but he average is not close to the true value

D. High-precision, High-accuracy

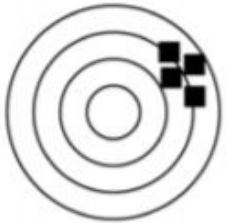
All data points close to the true value



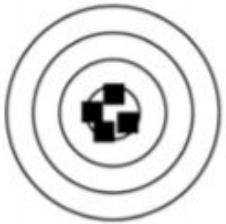
A. Low Precision, Low Accuracy



B. Low Precision, High Accuracy



C. High Precision, Low Accuracy

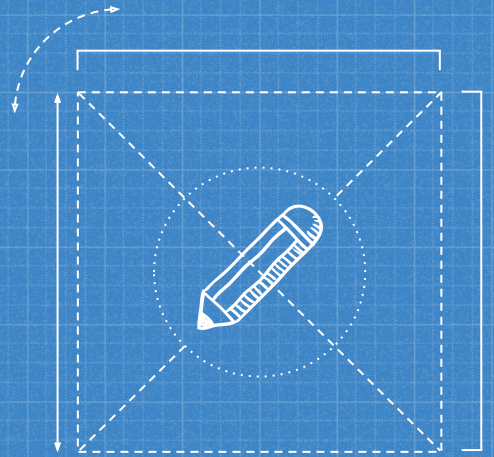


D. High Precision, High Accuracy

Image credit: UPenn Department of Physics

GROUP WORK

UHHH, UNCERTAINTY?



Your data set represents the estimated mass (in $M_{\oplus}$*) and distance (in parsecs) of a newly discovered planet, as calculated by some amateur astronomers. What would you recommend they do next to improve this measurement? Why? Discuss using error terms from this section!

Summarize your group's findings (in your own words) in a 2-3 sentence comment at the end of your notebook.

MORE PYTHON RESOURCES

- W3schools - easy, readable reference for basic Python functionality
- Numpy Docs - reference manual for Numpy
- Matplotlib Docs - pyplot tutorials
- Lab GitHub - Example code for many of the labs, feel free to use and modify
- Talk to your classmates!
- Google it!

That’s all!

ANY QUESTIONS?

Email me: anishimura@wpi.edu

Don’t forget to submit your Jupyter notebook on Canvas!

Hope you enjoyed this lab :)