WiFi
Wework
P@ssw0rd

June 4 2019

Intro to Python

Please install Python 3.x from
https://anaconda.com/download

GENERAL ASSEMBLY

Agenda

What We'll Cover Today

In this class, we'll explore the following topics:

Time	Topic
15 min	Introductions
15 min	Python Overview
15 min	Software Install
40 min	Programming Basics
20 min	Example Program
15 min	Create a Learning Plan

Greg Godreau

SW Eng, Maishelf

About GA

Discussion:

ABOUT YOU

Introduce yourself:

- **Name**
- **What Languages have you coded in?**
  - None!, Excel, SQL, HTML, BASH/DOS, C, Python, etc.
- **What industries are you interested in?**
  - Finance, Technology, Medicine, Publishing, etc.
- **How will this course help you with your goals?**
- **Share something you recently read/watched/heard.**

About this course

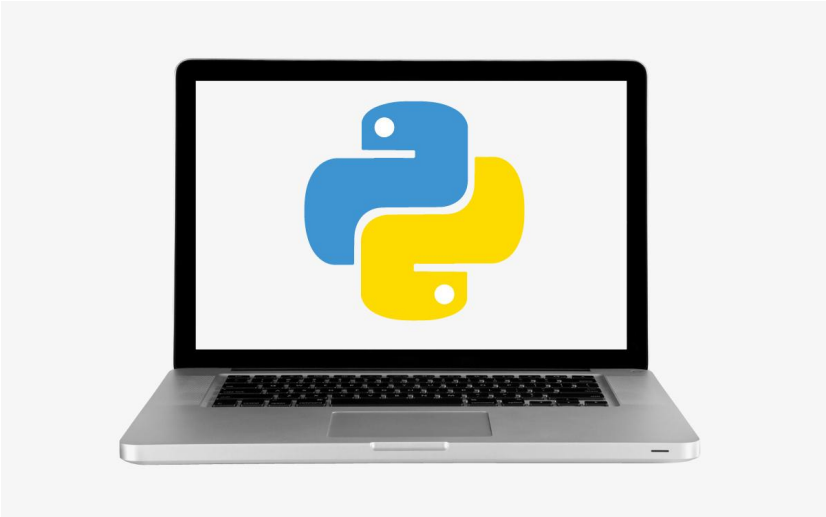
Learning Objectives

- Discuss the history of Python & how it's used in different industries
- Describe the benefits of a Python workflow when looking at data
- Demonstrate basic Python programming fundamentals to solve a real world problem
- Create a custom learning plan to build your data science skills after this workshop!

About this course

Getting the most out of this course

- Make sure you have the tools you need running smoothly
- Think / ask how Python could fit into your workflow
- The exercises are guidelines, pursue your interests in during practice
- Plan how you will continue your learning



“

Today dozens of Google engineers use Python, and we're looking for more people with skills in this language.

Peter Norvig,

Director of search quality at Google, Inc.

Intro to Python

What Is Python?

What is Python

- Created by Guido Van Rossum in 1991
- Emphasizes **productivity** and code **readability**
- **Easy** to pick up and learn
- Easier for many to contribute to **production level code**
- **Readable** code means that almost anyone can read and **understand what code is doing**

Why is Python readable

- **Interpreted language**:
  - Step by step execution for easier programming ideation
  - Write once, run anywhere
  - Performance tradeoff
- **Object-oriented (OO)**
  - Code with objects that contain data and functions to manipulate it in predefined ways
- **High-level programming**
  - Use natural language syntax where possible

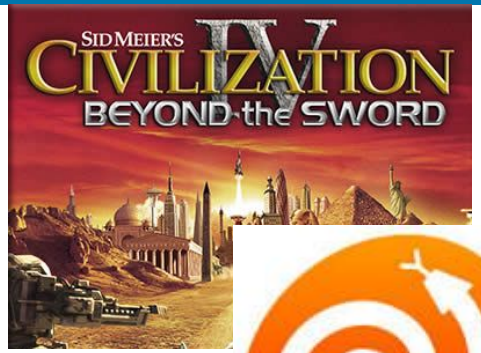
Why Python?

- **Free, Flexible,** and **Open Source**
- Rapid **prototyping** and **full-stack** commercial applications
- **Extensible** with easy to install **libraries**
- Great **documentation**
- Established and growing **community**
- **Scripts** can be run many times
  - When data changes
  - On different machines
  - At different times

Real Cases:

Who uses Python?

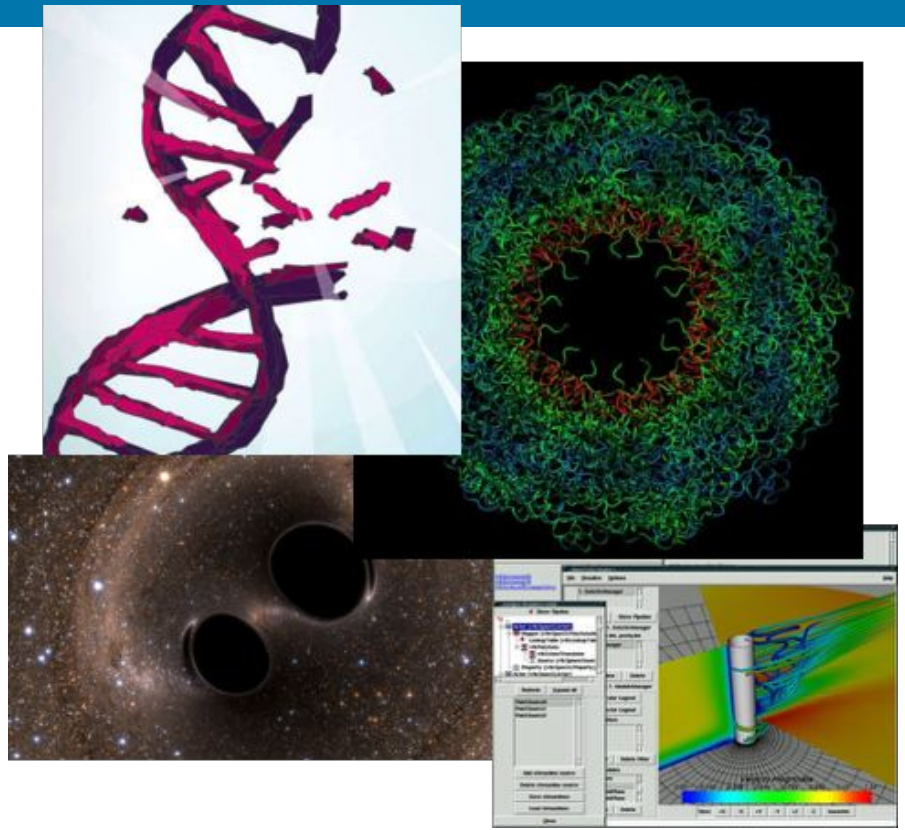
- **Industry & Academia**
  - AstroPy
  - BioPython
- **Web Development**
  - Youtube
  - DropBox
- **Game Development**
  - Civilization IV
- **Standalone Applications**
  - BitTorrent



Real Cases:

Examples

- **Industry**
  - Drug discovery
  - Financial services
  - Films and special effects
- **Academia**
  - Gravitational waves
  - Scientific visualisation
  - Biomolecule simulation
- **More**
  - Success Stories



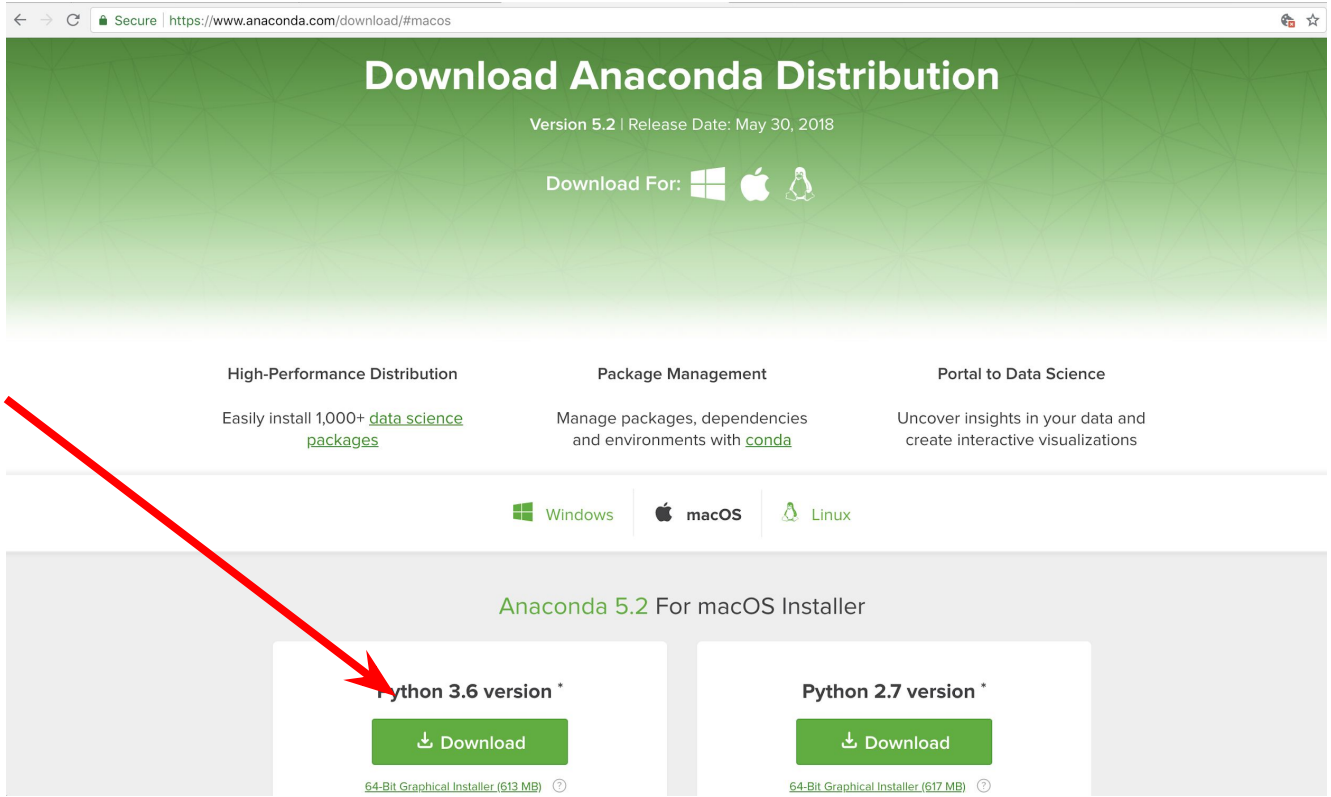
How will you use python?

- Job or Industry
- Workflow Improvements
- Dream Projects

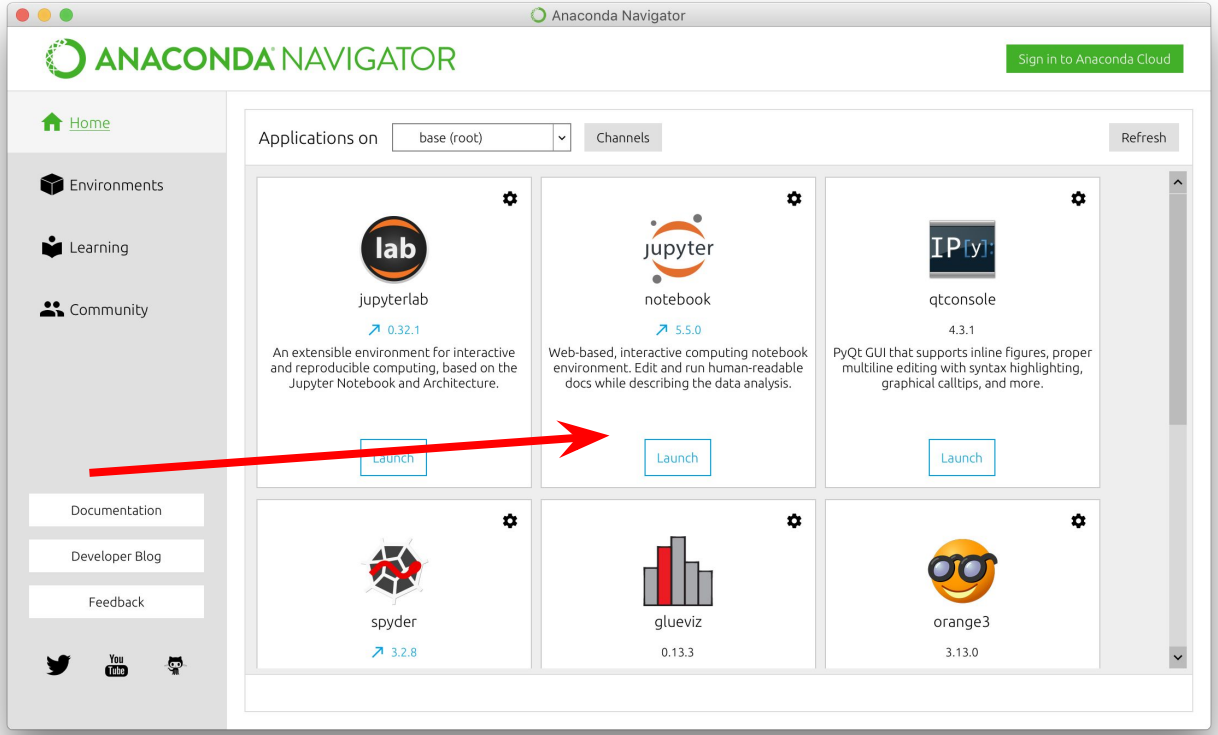
Intro to Python

Exploring Python With Anaconda

Download Anaconda https://anaconda.com/download



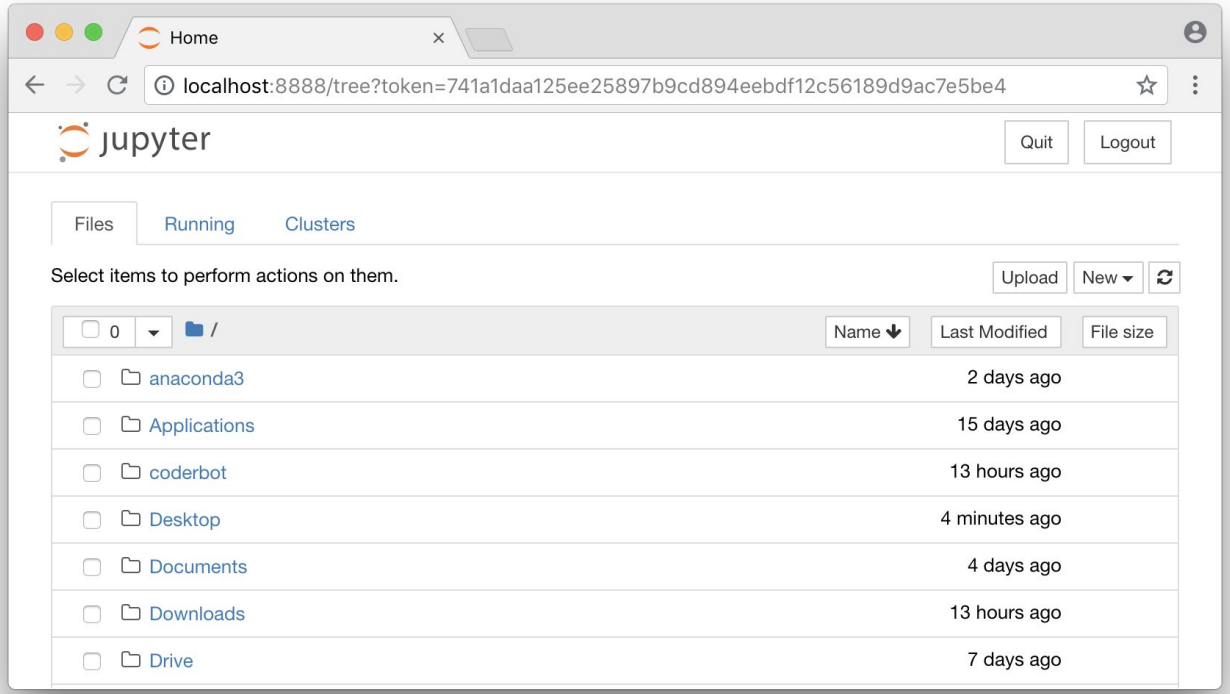
Anaconda Navigator



Terminal / Anaconda Prompt

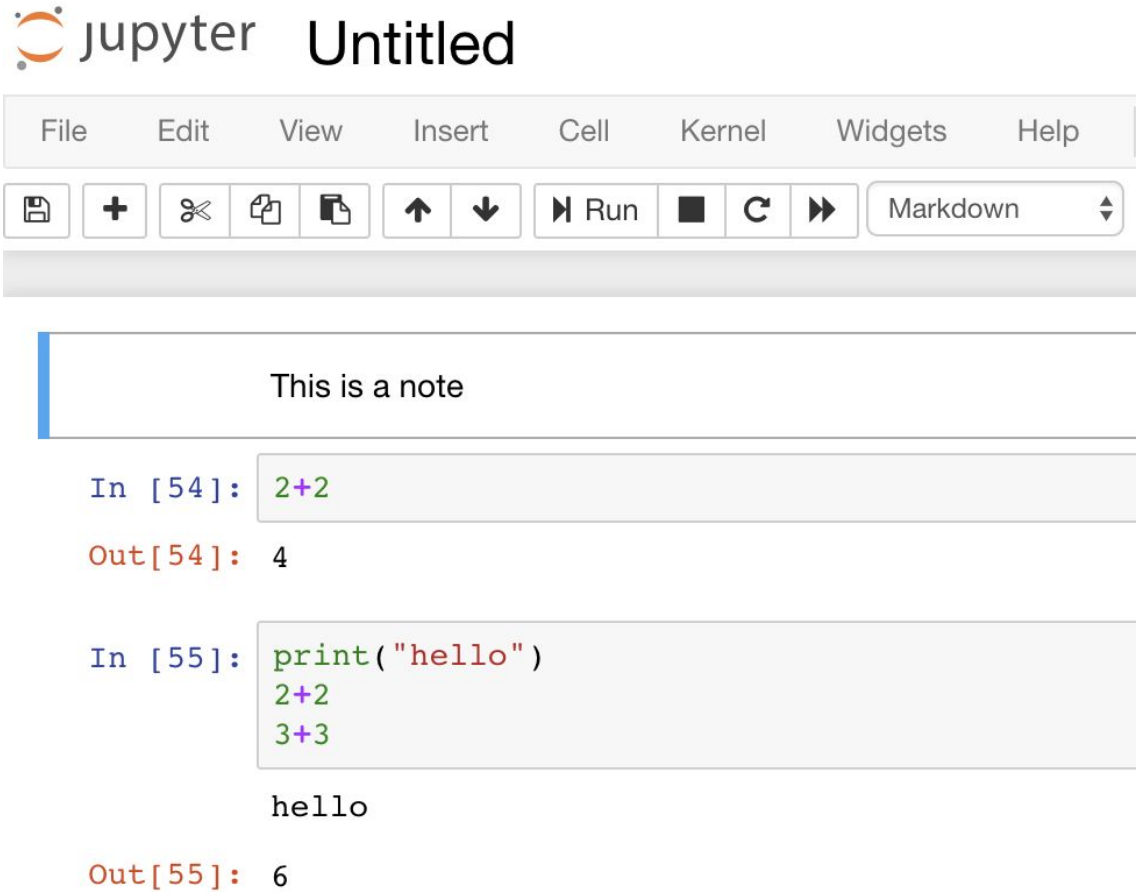
```
xterm
bash-3.2$ jupyter notebook
[I 04:00:47.123 NotebookApp] JupyterLab beta preview extension loaded from /Users/adamblomberg/anaconda3/lib/python3.6/site-packa
ges/jupyterlab
[I 04:00:47.123 NotebookApp] JupyterLab application directory is /Users/adamblomberg/anaconda3/share/jupyter/lab
[I 04:00:47.128 NotebookApp] Serving notebooks from local directory: /Users/adamblomberg
[I 04:00:47.128 NotebookApp] 0 active kernels
[I 04:00:47.128 NotebookApp] The Jupyter Notebook is running at:
[I 04:00:47.128 NotebookApp] http://localhost:8888/?token=fa133f2d6a7ab8080848634010e3ceee7393d423f60d99f5
[I 04:00:47.128 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 04:00:47.131 NotebookApp]

    Copy/paste this URL into your browser when you connect for the first time,
    to login with a token:
        http://localhost:8888/?token=fa133f2d6a7ab8080848634010e3ceee7393d423f60d99f5&token=fa133f2d6a7ab8080848634010e3ceee7393d

[I 04:00:48.196 NotebookApp] Accepting one-time-token-authenticated connection from ::1
```

New Web Browser window with personal files



Jupyter Notebook

- **Cells**:
  - **Markdown** for notes
  - **Code** for Python
- **Modes**
  - **Blue** for commands
  - **Green** for editing
- **Execution**
  - Shift + return
- **Output**
  - Print (all)
  - Return values (last)



Jupyter Notebook errors

Mistakes happen! Here's what they look like:

```
just some code
```

```
  File "<ipython-input-56-2516a36d8922>", line 1
    just some code
              ^
SyntaxError: invalid syntax
```

1. **Try to understand what went wrong**
2. **Attempt to fix the problem**
3. **Execute the cell again**

Solo Exercise:

Try out Jupyter notebook

5 minutes

Take some time to try out Jupyter notebook.

Make sure you can:

1. Convert cells between Markdown and Code
2. Edit and execute a note cell
3. Edit and execute a code cell
4. Try to do some math
5. Make an error

Intro to Python

Python Programming Fundamentals

Programming Fundamentals Framework

Every programming language can be broken down into core components. Having a general framework for this context will help us learn specifics for Python.

- **Syntax**
  - The structure of the commands given to the computer
- **Variables**
  - How computers store information
- **Control Structures**
  - Sets the hierarchy/priorities of programming logic
- **Data Structures**
  - How computers organize data

Syntax

The **syntax** of a programming language is the set of rules that define the combinations of symbols that are considered to be correctly structured programs in that language

Just like our spoken languages have structures like paragraphs and sentences, programming languages have **code blocks** and **statements**.

Variables

Variables are symbolic names that store a specific piece of information. Variables come in different types in order to hold different kinds of information.

- For example: **r = 3**
  - Defines a variable: named "**r**"
  - This holds a numerical integer: **3**

- Other types of variables:
  - float (3.14)
  - string ("Hello")

Control Structures

A block of programming that analyses variables and chooses a direction in which to go based on given parameters is a **control structure.**

- The term **flow control** details the direction the program takes (how program logic “flows”). It determines how a computer will respond when given certain conditions and parameters. Some typical structures include:
  - **If** statements
  - **For** loops
  - **Functions**

Data Structures

A **data structure** is a particular way of storing and organizing data in a computer so that it can be used efficiently.

- Some examples in Python include:
  - Lists
  - Tuples
  - Dictionaries
  - Dataframes

Python programming

- Let’s see what a Python program looks like.
- We’ll start with the classic "*Hello World*!" program:
  - This code will print the message "Hello World!" on the screen.

```
1 | print("hello world")
```

What did we just see?

- Data
  - “hello world”
  - string
  - Denoted by quotation marks
    - Single ‘…‘
    - Double “…”
    - Triple “””…”””
- Function
  - Print
- Aside: # makes comments

```
1 | print("hello world")
```

Variables

- **Types**
  - **Bool** - 1 or 0, True or False
  - **Int** - number w/o decimal point
  - **Float** - number w/ decimal
  - **String** - text
- **Assignment**
  - = operator defines variable
- **Variable names**
  - snake_case
  - Lowercase Letters, Numbers, Underscores
- **type() function** - shows data type

```python
# variable assignments
x = 1.0
my_variable = 12.2
type(x)

y = 1
type(y)

b1 = True
type(b1)

s = "String"
type(s)
```

Variables

- **Operators -** combine data
  - **+, -, *, /** (add, subtract, multiply, divide)
  - **+=, -=, etc** (perform operation and save result)
  - ** (power)
  - **//, %** (quotient, remainder (modulus))
  - **>, <, ==, !=, <=, >=** (greater than, less than, equal, not equal, etc)
- **Functions and methods** - saved instructions to manipulate data
  - **abs(**my_int**) -** function_name(data)
  - **len(**my_string**)**
  - my_string**.lower() -** data**.**method_name()
  - my_int**.__add__(5)**
    - **"Dunder" -** double underscore (__) = important Python built in item

Partner Exercise:

Practice in Jupyter notebook

5 minutes

Now you try! Pair up with a partner to attempt the following in your notebooks. Help each other out!:

1. Create variables
    a. Bool
    b. Int
    c. Float
    d. String
2. Check their class with type()
3. Try out some operations on your variables
    a. What works? What causes errors?

Data Structures

- **Lists**
  - A **collection** of objects
    - Mixed types are okay
  - Defined with **square brackets** [ ]
  - They can be modified
    - my_list**.append()**
    - my_list**.remove()**
  - **Slicing**
    - Access elements
    - my_list**[** start **:** end **:** step **]**

```
l = [1,2,3,4]
print(type(l))
print(l)
print(l)
print(l[1:3])
print(l[::2])

# Python starts counting from 0
print(l[0])
```

Data Structures

- **Tuples**
  - very similar to lists, but:
    - They are defined with parentheses ( ) instead of square brackets
    - They cannot be changed
      - No append() method
      - No remove() method
  - Slicing works the same way

```
point = (10, 20)
print(point, type(point))

x, y = point
print("x =", x)
print("y =", y)
```

Data Structures

- **Dictionaries**
  - Collections of **key/value pairs**
  - Defined by curly brackets { }
  - Slicing uses **keys**
  - **Order** is not preserved

Python

```python
params = {"parameter1" : 1.0, "parameter2" : 2.0,
"parameter3" : 3.0,}
print(type(params))
print(params)
```

Control Structures

- **If / elif / else**
  - Check conditions with boolean operators (i.e. <, >, ==)
  - Execute a single code block depending on result
  - If True: code runs
  - Can be:
    - if alone
    - f/else
    - if/elif(s)/else
  - Indentation controls end of **if** block
  - Data controls which code runs

Python

```python
if age_person > 18:
    return "They can drive"
else:
    return "They cannot drive"
```

Control Structures - if

Python

```python
A = 10
B = 100
if A>B:
    print("A is larger than B")
elif A==B:
    print("A is equal to B")
else:
    print("A is smaller than B")
```

Control Structures

- **for** loop
  - Repeat operations
  - Loop variable takes each value from list in turn
  - Indentation controls end of loop
  - Watch out for infinite loops!
    - Interrupt or restart kernel when this happens

Python

```python
users = ["Jeff", "Jay", "Theresa"]

for user in users:
    print("Hello %s" % user)
```

Control Structures

- **Functions**
  - Groups of instructions repeat / create abstractions of common tasks.
  - Divide our code into useful blocks
  - Provide order, making the code more readable and reusable
  - def name**(input1, input2, ...)** :
  - First line is “Document String” describing how function works
  - Definition saves instructions only - no execution!

Python

```python
def square(x):
    """
    Return the square of x.
    """
    return x ** 2
```


Control Structures

- **Functions**
  - Run functions with parentheses after the name
  - Needs to be defined before it can be executed!
  - The return value is saved in var2
  - Other functions can be run on the result!

Python

```python
var1 = 7

var2 = square(var1)

print(var2)
```

Expanding python

- **Packages**
  - Install new libraries to add functionality to Python:

    conda install <name>

    or

    pip install <name>
  - Use packages by importing them into your Python scripts

```
import math
x = math.cos(2 * math.pi)
print(x)
```

Real Cases: Expanding python

Common Packages

- Data manipulation: pandas, Numpy, scipy
- Machine Learning: scikit-learn, nltk
- Databases: psycopg2, sqlalchemy
- Visualizations: matplotlib, plotly, bokeh
- API calls / web scraping: requests, BeautifulSoup, Scrapy
- Web development: Django, Flask, Twisted, Scapy
- Game Development: Pygame, Pyglet
- Desktop App: pyQt, Tkinter

More

“

I have students learn Python in our undergraduate and graduate Web courses. Why? Because there's nothing else with the flexibility or as many pre-built libraries...

Prof. James A. Hendler, Univ. of Maryland

Intro to Python

Python Programming Practice!

Data Analysis Datafile

1. Start a new notebook and rename it to **speeding_example**
2. Import pandas and load the dataset

```
import pandas as pd
df = pd.read_csv(
    "https://vincentarelbundock.github.io/Rdatasets/csv/boot/amis.csv",
    usecols=range(1,5)
                 )
```

3. Google R datasets, should see listing as second hit linking to https://vincentarelbundock.github.io/Rdatasets/datasets.html
   a. We are using the amis dataset, fifth on that page

Data Analysis Overview

The data used here show measured car speeds with 3 other labels:

- **pair** - There are 14 pairs of data collected
- **warning** - For each pair, two sections of road were measured:
  - Where a warning sign was placed for part of the experiment
  - Whether there was a similar stretch of road in another part of town where no sign was erected during the experiment (control)
- **period** -There are 3 time periods in the data for each pair:
  - Before the warning sign was placed on road section 1
  - Just after the sign was placed on road section 1
  - Some time after the sign was erected (so the sign is no longer "new")

Data Analysis

We want to study how the **average speed changes** in one section of road after the sign was erected, so we need to:

1. √Read the data
2. Loop over the rows of data
3. Select data only from the group of interest (just pair 7)
4. Compute the answer

Data Analysis Recap

1. Create lists to store data of interest
2. For loop over all the data
3. Nested if statements to filter just the:
   a. Pair
   b. Warning
   c. Period
   d. we are look for
4. Save data to appropriate list
5. Print average of list after loop

```
before = []
after = []

for row in df.values:
    if row[3] == 7:
        if row[2] == 1:
            if row[1] == 1:
                before.append(row[0])
            if row[1] == 3:
                after.append(row[0])

print("average before sign: ",
      sum(before)/len(before))
print("average after sign: ",
      sum(after)/len(after))
```

Data Analysis Script

5. Change the Notebook name to **speeding_example**
    a. Note no spaces!
    b. File > Download As > Python (.py)
    c. Click **Keep** if Browser warns file may be dangerous
6. Open new Terminal (or Anaconda Prompt) window
7. Change to Downloads folder in Terminal
    a. type: **cd Downloads** and press enter
8. Run the python script
    a. Type: **python speeding_example.py** and press enter

Note only the printed output lines appear in the terminal!

“

The goal is to turn data into information, and information into insight.

Carly Fiorina,
Former CEO, Hewlett-Packard

Intro to Python

Let’s Review

Review and Recap

In this workshop, we've covered the following topics:

- Python as a popular, flexible programming language
- Python has applications in many different areas
- Python is particularly great for data manipulation
- Python programming basics include: types, variables, functions, and more!

Finish That Sentence

What are your biggest takeaways from this lesson?

"Something that really got me thinking is..."

"The best thing I got out of this unit is..."

"I discovered..."

"I still want to learn about..."

"I was surprised to learn that..."

Ask Me Anything!

We appreciate your feedback!

Please take 60 seconds to complete our survey.

Intro to Python

Next Steps

Create a learning plan

What's next?

Solidify your learning:

- Go through the parts of Learn How to Think Like a Computer Scientist.
- Familiarize yourself with the language by going through A Beginner's Python Tutorial.

Practice Practice Practice! Problems to expand your skills are available at:

- HackerRank
- CodeWars

Create a learning plan

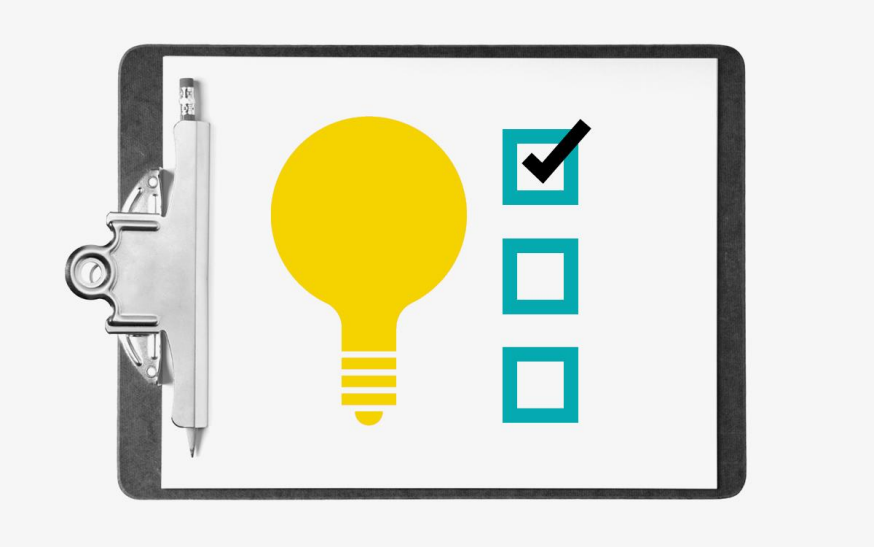
General Assembly also offers courses that teach you how to use Python!

Check out our:

- Part-time Data Science Course
- Data Science Immersive Course
- Online Part-time Data Science Course
- Part-time Python Course
- Online Part-time Python Course

A Few Good References

1. Official Python Documentation
2. PEP-8 Official Guide
3. Anaconda Tutorials
4. Jupyter Documentation
5. Example Notebooks



See you next time!

THANK YOU!

GENERAL ASSEMBLY

GA