Introduction to Python Workshop

Hosted by the BCB-GSO

Written by Kelby Kies

August 16, 2021

Purpose

The purpose of this workshop is not to turn you into a computer scientist or an expert Python programmer. The purpose of this workshop is to introduce you to basic syntax/commands of Python and to give you examples of how Python is used in different biological contexts.

Outline

- Getting started: 15 minutes
- Python is Awesome & here's why!: 5-10 minutes
- The Basics... (first Hour)
- Visualizations (Take a 5-10 minute break; 2nd hour)

Getting Started

Downloading the GitHub repository

1. Go to https://github.com/kelby-kies/Intro-to-Python
2. Click on the green Code button that has a little download symbol. (pic)
3. Click 'Download Zip'. This should download the entire repository (folder) to your downloads folder. I recommend moving this folder to the Desktop or somewhere easy to find.

Installation

Please install Anaconda using the 'Installing*Anaconda*Python' tutorial in this repository if you have not done so already.

Python is Awesome & here's why!

What is Python?

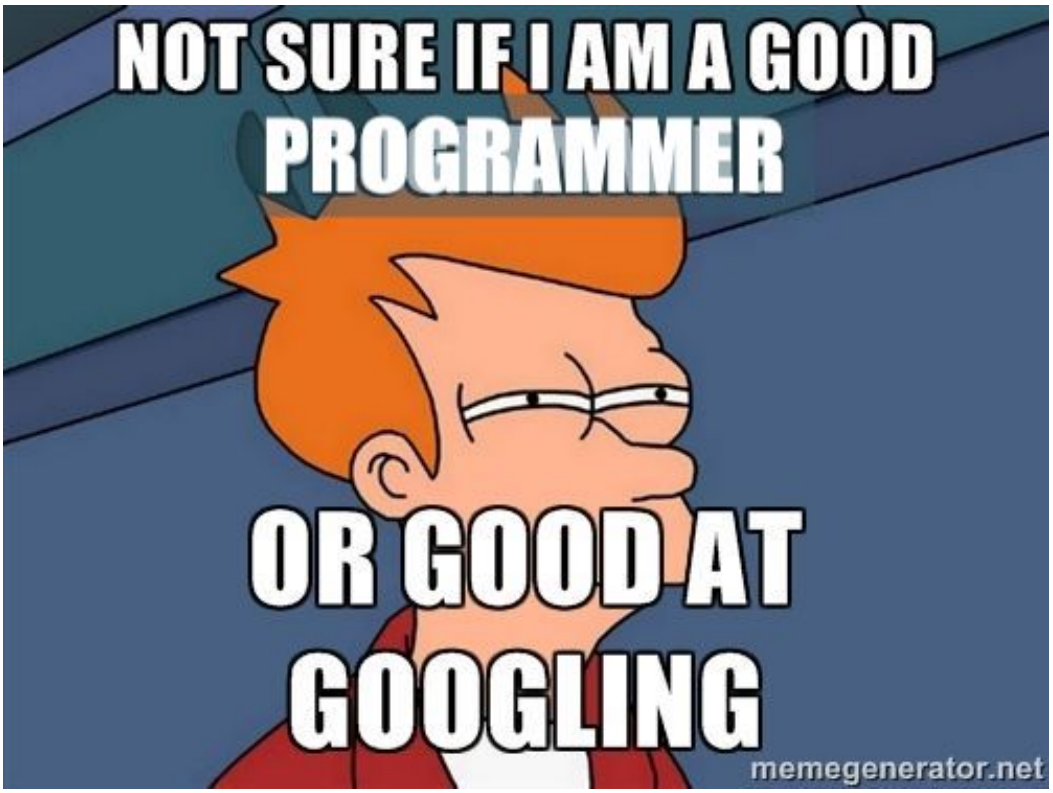
"Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python’s elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms." (1)

Pros & Cons

Pros	Cons
efficient high level data structures	speed limitations
support object oriented programming	issues with threading
simple syntax	Not native to mobile environment
versatile	high memory consumption
easy to use	
fast to develop	
Has a ton of libraries (Pandas, NumPY, BioPython etc.)	

Where to get help on Python

- http://www.python.org
- Your BCB-GSO Tutorial! Or any other tutorial out there
- https://wiki/python.org/moin/BeginnersGuide/Programmers
- Google! (This goes for anything in grad school. Google is your friend.)



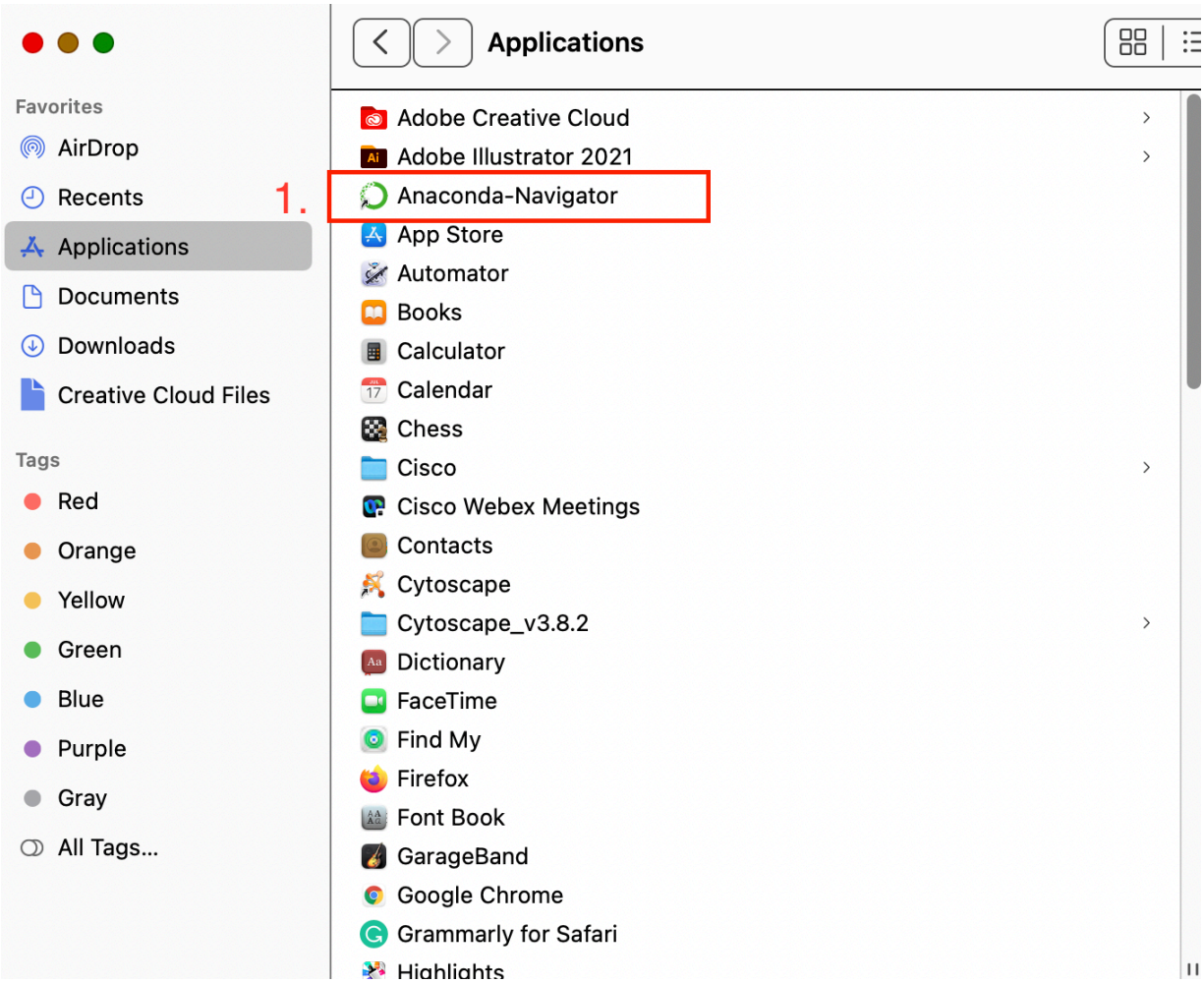
What could I use Python for?

- writing bioinformatic tools:
  - Biopython, PyMOL, PyCogent, Galaxy, pygr and so many more!
- visualizing data
- statistically modeling data
- developing websites and software
- task automation
- data analysis

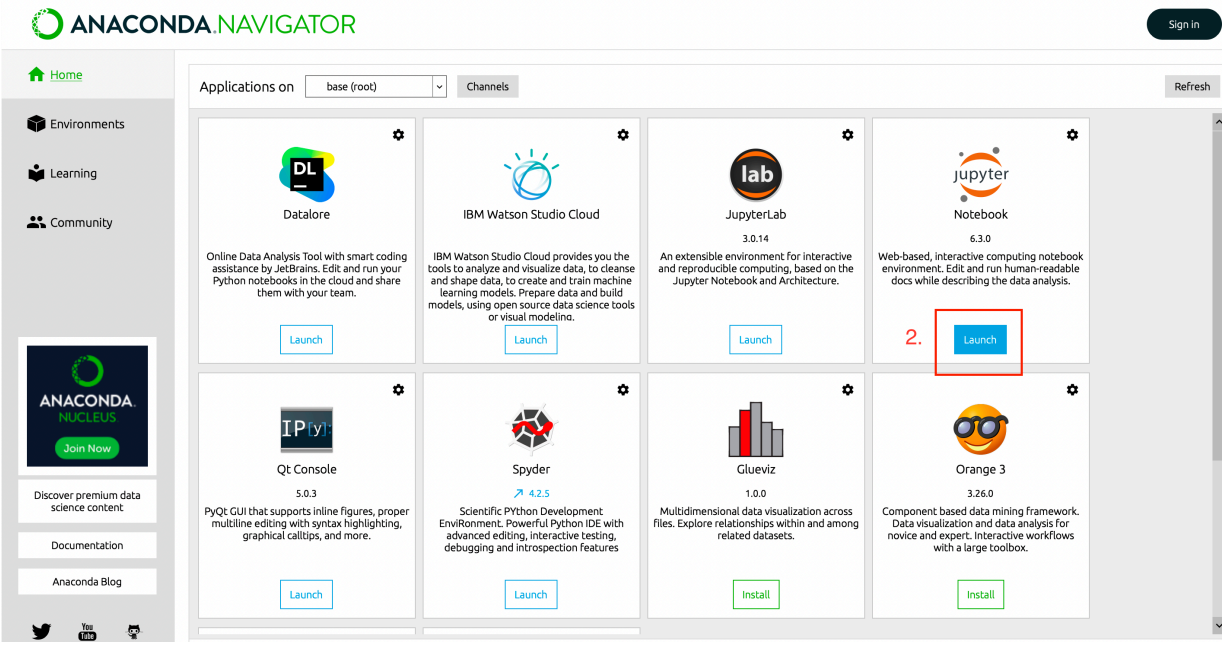
The Basics....

How to run Python in a Jupyter Notebook

1. Double click on the 'Anaconda Navigator' that we just downloaded

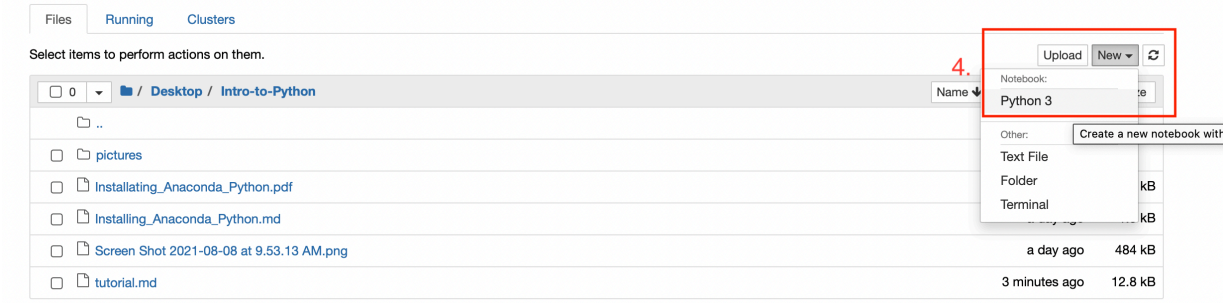


2. Launch a new Jupyter Notebook

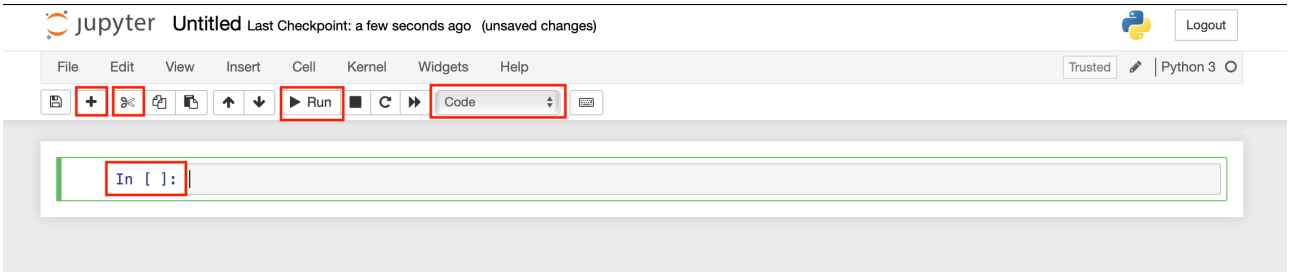


3. Navigate to the 'Intro-to-Python' directory that we just cloned from GitHub

4. Open a new Jupyter Notebook file.



Description of Jupyter Notebook



- **+** sign: Adds a new line to your notebook.
- 'scissors' symbol: If you want to delete a line then you can click on this button. This will delete the current highlighted line.
- **Code** dropdown menu: You can indicate what type of material this new line will contain using this drop down menu. Options include: Code, Markdown, Raw NBConvert, Heading. For today's purposes we will just be using the Code option.
- **Run**: To execute/run your code there is a run button at the top
- **In[]:**: The lines where we can type code are indicated by 'In[ ]:' which stands for input. We are inputting the code in this line and when we click 'Run' the notebook will take your input code and generate an output.

Let's try a quick example. Copy & Paste the line below into a 'In[]:' line.

```
# Let's write our first program!
print("Hello World")
```

Click the 'Run' button.

We can see that Hello World is printed out on a new line. Notice that the line: # Let's write our first program did not print. This is because this line is commented out using #. Comments are used for good documentation in ANY language!

Arithmetic

We can use python as a basic calculator. Let's run some basic calculations below The code lines indicated with a # are just comments. They are not actual commands being run and won't print out anything.

Add numbers with a '+' sign

```
4 + 10
2 + 2
1.5 + 3.95
```

Subtract numbers with a '-' sign

```
20 - 40
100 - 12.5
5 - 3
```

Multiply numbers with a '*'

```
2 * 3 * 10
1046 * 2345
```

Divide numbers with '/'

```
100/16
12567/3
```

The order of operations is the same as normal math in Python.

```
# First the sum within the parenthese will be exectued
# This quantity is then multiplied
# The resulting quantity is then subtracted from 100
100 - (4+3) * 2
```

Variables

A variable is a way we can store a value in memory to be used later on.

Here I am storing the value 2 in memory with the name 'x'.

```
x = 2
# When we print x the variable x, we should get 2.
print(x)
```

We can later change the value of the variable or use it in a function as well.

```
x = (x * 2)/12 + 10
print(x)
```

What is the value of x now?

It is generally good practice to make your variable name descriptive. For example, below is a list of countries. It would be hard to know what kind of values the variable holds if we called it x. Let's give it a more descriptive name like *countries*

```
countries = ['Japan', 'Mexico', 'Russia', 'Belgium', 'Venezuela']
```

Exercise #1

1. Assign `x` to a numerical value.
2. Add 100 to x.
3. Divde by 4.5.
4. Subtract 10.
5. Multiply by 2.
6. Print the new `x`

Bonus: Can you do #2-5 on the same line to get the same value of x? (Hint: Utilize Python's order of operations!)

Strings

A commonly used datatype in Python is called a *String*. A string is just a collection of characters that are enclosed in quotes.

Strings in python are either surrounded in single (') or double ("") quotes.

```
'hello'
"Hello"
```

Strings can be assigned to a variable as well.

```
blue_sky = "The sky is blue."
```

We can print a string to the standard out or 'make it literal' by using print.

```
print("Python is Awesome!")
```

Indexing a String:

Because a string is just a collection of characters we are able to *index* a string to get a specific character.

```
# Assign string to variable
school = "Iowa State University"
# print full string
print(school)
```

Great! Now let's print the W!

To print out the 'W', we need to take our variable that we assigned a string value (in this case school) and *index* it with a specific number.

To *index* means you are telling the Python interpreter to go to that position in the string and pull out that character.

```
print(school[2])
```

Yes! Awesome you printed the W, but WAIT! Did you notice that the W is the 3rd letter in Iowa, but we passed in the index 2?

This is because Python is '0-based', which means that when looking for the character we want to pull out we start counting at 0 rather than 1! This is VERY important because other languages can be different and can lead to problems!

If we want to get the last character of a string we can use `-1`

```
print(school[-1])
```

We can the length of a string using `len()`.

```
print(len(school))
```

We can subset a chunk of a string using `str[start:end]`. This will retun the subset of the given `str` return every character between `start` and `end-1`.

```
print(school[0:4])
```

Lists

We can group objects(ints, doubles, strings, etc) together into a `list` A list is defined by using [ ] and separating each object with a ,

```
list_1 = [1,2,3,4]
print(list_1)
list_2 = ['Hello', 'How', 'are', 'you', 'today?']
print(list_2)
```

You can index a list similar to how we index a string to select a specific character. (A string is just a list of characters.)

```
list_1[2]
list_1[-1]
```

We can also index a string within a list of strings. Lets say I want to get the `?` in `'Hello?'`.

```
list_2[4][-1]
```

We can add and remove objects from a list. To add a single object to a list use `append`.

```
list_1.append(15)
print(list_1)
list_2.append('Great!')
print(list_2)
```

Note: `append` will only work for adding a single object. If you try to add multiple objects then we will get an error.

One cool thing about `lists` is that they can contain mixed datatypes.

```
list_1.append('List 1 can contain Strings')
print(list_1)
# While list_2 can contain numbers
list_2.append(4)
print(list_2)
```

If we wanted to add more than 1 object, we can use `extend`.

```
list_1.extend(['I', 5, 'love', 7, 'Python'])
print(list_1)
list_2.extend([4.5, 'Happy Monday!'])
print(list_2)
```

We can even add 2 pre-defined lists together and assign to a new list name

```
new_list = list_1.extend(list_2)
print(new_list)
```

To remove objects from a list:

- Remove all items: `clear()`

  `list_1.clear()` `print(list_1)`

- Remove an item by index and get its value: `pop()`

  `print(list_2)` `#Let's remove the 3rd element in list_2`
  `print(list_2.pop(2))` `print(list_2)`

  If we don't pass in an index into `pop()` then the last element is deleted.

- Remove an item by value: `remove()`

  `#Let's remove Happy Monday from the new_list`
  `new_list.remove('Happy Monday!')`

Dictionaries

A dictionary is a collection of `key:values` pairs. One key can have multiple values. Dictionaries in Python are similar to dictionaries in real life. For a real dictionary we can look up the definition (value) of a specific word (key). THis is similar in Python; we can look up the values attached to the desired key using the command: `dictionary[key]`

We can define a dictionary by using `dict()` or by using { }. To add a new value to a dictionary: `dictionary[key] = new_value`.

```
# defining the gene_ontologies dictionary with dict()
gene_ontologies = dict()
# Let's add new values to gene_ontologies.
gene_ontologies['mesenchymal to epithelial transition'] = 'Stat1'
gene_ontologies['mesenchymal to epithelial transition'] = 'Bmp4'
gene_ontologies['mesenchymal to epithelial transition'] = 'Sall1'

gene_ontologies['angiogenesis'] = 'Nrarp'
gene_ontologies['angiogenesis'] = 'Adam15'

print(gene_ontologies)

# Let's define a new dictionary by using {}
movies_2021 = {
    "comedy":"The Hitman's Bodyguard"
    "Disney":"Cruella"
    "Thriller": "Old"
    }
print(movies_2021)
```

Dictionaries are useful because we can look up a value by using the key.

```
# Let's see the gene's that correspond to Mesenchymal to Epithelial Transit
ion.
print(gene_ontologies['mesenchymal to epithelial transition'])
```

We can also see if a sepcific key exists within a dictionary.

```
"Drama" in movies_2021
"angiogenesis" in gene_ontologies
```

Exercise #2

1. Make a list of your 3 favorite foods and assign to a variable called `food`. Print. your list using the variable name.
2. Now add your 1 of your neighbor's favorite foods to your list. For those of you that are virtual feel free to type a food in the chat or add my favorite food `'chocolate'.` Print your updated list.
3. Access the `chocolate` element and print the middle `o` character.
4. Create a dictionary with the key's food, hobby, subject and assign each key your favorite. Call this dictionary `favorites`.
5. Add a new favorite to your dictionary. Give it a unique `key` name and assign it a value.

Functions

What is a function?

a function is a chunk of code that takes input values and performs a 'function' on it to get an output. Some functions are simple, but some can be complicated.

Structure of a function

```
def function_name(input, values)
{
output_value = input * values
print(output_value)
}
```

To call a function you must use the give name with `()`:

```
function_name(2,4)
```

Built in Python functions

Python has many built in functions. Example 1: `print(value)`. Here the variable named `value` is the input value into the `print()` and the print function will take that value and print it to the standard out.

Example 2: `abs(-14)`. Here the `-14` is the input value into the `abs()`. `abs()` will return the absolute value of whatever object is passed into it.

Importing new functions using Libraries

One can import a library with < import package_name> to load all of the functions of that package.

Popular packages used by Biologists:

- **NumPy**: "NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more." https://numpy.org/doc/stable/user/absolute_beginners.html
- **Pandas**: "Pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language." https://pandas.pydata.org/docs/getting_started/index.html#getting-started
- **Biopython**: "The goal of Biopython is to make it as easy as possible to use Python for bioinformatics by creating high-quality, reusable modules and classes. Biopython features include parsers for various Bioinformatics file formats (BLAST, Clustalw, FASTA, Genbank,...), access to online services (NCBI, Expasy,...), interfaces to common and not-so-common programs (Clustalw, DSSP, MSMS...), a standard sequence class, various clustering modules, a KD tree data structure etc. and even documentation."http://biopython.org/DIST/docs/tutorial/Tutorial.html
    - Fun Fact: Dr. Iddo Friedberg here at ISU helped create BioPython!

Pandas

Series

Dataframes

Visualizations

Scientists in all areas typically look at their results in the form of a graph. Visualizing data is one of the most utlized features in Python!

Python Packages for Visualization:

- **Matplotlib**: uses numPy; comes with a wide variety of plots like line, bar, scatter, historgram, etc.; John Hunter 2022
- **Seaborn**: dataset oriented library for making statistical representations in Python; uses Pandas data structures;
- **plotly**: interactive, open-source, high-level, declarative and browsers-abased visualization library for Python. scientifc chars, 3D graphs, statistical charts, financial charts;
- **ggplot**: grammar of graphics implemented in Python; mapping of data to asthetic attributes and geometric objects.

When to use the right visualization?

It all depends on the type of data and the question you are trying to answer!

Let's import some example data and look at some visualizations.

Reading in Data

```
import pandas as pd

df = pd.read_csv('data.csv')
```

Different types of graphs:

Dataset # 1: This data contains the details of over 11,000 athletes, with 47 disciplines, along with 743 Teams taking part in the 2021(2020) Tokyo Olympics. This dataset contains the details of the Athletes, Coaches, Teams participating as well as the Entries by gender. It contains their names, countries represented, discipline, gender of competitors, name of the coaches. Source: Tokyo Olympics 2020 Website

```
# Read in File
```

- bar graphs:

    - **bar chart** (horizontal) : used to compare metric values across diff. subgroups of data. more groups is better for a bar chart.
    - **column chart** (vertical): used to compare a single category of data between individual sub-items
        - bar graph of gold, silver bronze #'s - Medals
    - **grouped bar chart**: used to compare values in certain groups and sub-groups
        - compare the # of Male to Female atheletes for each sport : EntriesGender
    - **stacked bar chart**: compare the total sizes across the available groups and composition of different subgroups.
        - show breakdown of countries for each sport - teams.csv
- **line chart**: used to represent continuous data; trend across time.
- **histogram**: observe distribution of a single variale with a few data points.
- **scatter plot**: idetntify relations between 2 variables. can be used when the dependent(y) variable can have multiple values for the independtnt(x) variable.
- **box plot**: used to show shape of distribution and the sumamry of the data.

Exercise #3

1. Read in the 2nd dataset.
2. Make a Bar plot
3. Make a Box plot
4. Make a scatter plot

Citations:

This tutorial was based on Paul Villanueva's 2018 Python tutorial https://nbviewer.jupyter.org/github/pommevilla/p3.bootcamp.python.2018/blob/master/lessons/P3Bootcamp2018-00.ipynb & BCB546 class material https://eeob-biodata.github.io/BCB546X-python/

1. https://docs.python.org/3.6/tutorial/
2. https://www.analyticsvidhya.com/blog/2021/02/an-intuitive-guide-to-visualization-in-python/