**Winter PYTHON CAMP 2020**

**Useful links**

* Open EdX site for enrolling in course: <https://openedx.seas.gwu.edu>
* Open EdX course material for Getting Data Off the Ground: <https://openedx.seas.gwu.edu/courses/course-v1:GW+EngComp1+2018/about>
* JupyterHub: <https://go.gwu.edu/jupyter>
* Markdown Syntax reference page: <https://daringfireball.net/projects/markdown/syntax>

Hello class

Leah here too if you need any help. Reminder there's a screen in the back of the room as well.

Quotes for use in lesson on playing with string methods:

AE\_quote = "Everybody is a genius. But if you judge a fish by its ability to climb a tree, it will live its whole life believing that it is stupid."

ER\_quote = "   One's philosophy is not best expressed in words; it is expressed in the choices one makes.   "

1. Go to the course site and within the lessons on the left bar, find the link to the Homework Page, for example, “Graded HW 1”.
2. Download the notebook to your computer. Remember where that is, so you can find it again!
3. Go to Jupyter Hub and upload the notebook.
4. Work on the exercises, deleting the "your code here" lines and the raiseError line. Once done, save and restart the kernel as described in the notebook.
5. Download the notebook from Jupyter Hub using File > Download as > Notebook.
6. Go back to the homework page in the course site and upload the revised notebook. Make sure you're not uploading the blank one that you previously downloaded. You may have to delete the blank one you got originally.

items = [1, 2, 3, '4', [5, 'six'], [7]]

4 in items

5 in items

7 in items

[7] in items

**Python Camp Day2 1/17/2020**

For the refresher exercise:

**Full refresher exercise:**

* 1. Go to the notepad (<https://go.gwu.edu/pythonpad)> and scroll to the section for today, where you’ll find text from a list of grant awards in the National Science Foundation BIGDATA program:  <https://go.gwu.edu/nsfdata>.  Assign it as a string to a variable called grant.
* grant = "BIGDATA: Collaborative Research: F: Nomadic Algorithms for Machine Learning in the Cloud; Award Number:1546459; Principal Investigator:Manfred Warmuth; Co-Principal Investigator:; Organization:University of California-Santa Cruz;NSF Organization:IIS Start Date:01/01/2016; Award Amount:$596,326.00; Relevance:48.0;"
* 2. Split the string on the semicolon character, so you have a list. Assign that result to a new variable called grant\_info. How many items are in the list grant\_info?
* >> grant\_info = grant.split(";")
* >> len(grant\_info)
* 3. Access the element in the list which holds the award amount and assign that to a new variable called amount. Remove the extra spaces from the front of the element. Create a variable called dollars and assign just the dollar amount to it (not the label "Award amount")
* >> amount = grant\_info[-3]
* >> dollars = amount.split(":")[-1]

Python documentation: <https://docs.python.org/3.7/library/stdtypes.html#str.strip>

* 4. FOR Iterations
* >> fruit\_bowl = ['apple', 'banana', 'orange', 'lemon']
* >> for fruit in fruit\_bowl:
* print("Eat your", fruit)
* >> print(fruit)
* >> print(type(fruit))
* >> print(fruit.capitalize(), "trees")

Exercise 1:

Say we have the following list:

all\_sizes = ['xs', 's', 'm', 'l', 'xl']

Use the string method .upper()

Write a **for** statement to:

* Create a new list with all the sizes in uppercase.
* Bonus: Test to see if the size starts with an “x” and print the True or False result.

Remember! You can’t append to a list that doesn’t exist.

* >> all\_sizes = ['xs', 's', 'm', 'l', 'xl']
* >> new\_size\_list = []
* >> for size in all\_sizes:
* new\_size\_list.append(size.upper())
* print(size.startswith('x'))
* >> print(new\_size\_list)

Exercise 2:

fullnames = [['sam','jones'],['zoe','smith'],['joe','cheek'],['tom','perez']]

firstnames = [ ]

lastnames = [ ]

Write some code that creates two lists:

* One with the first names from the list,
* Another with the last names from the list, but capitalized.

Remember! You can’t append to a list that doesn’t exist.

* >> firstnames = []
* >> lastnames = []
* >> for name in fullnames:
* firstnames.append(name[0])
* lastnames.append(name[1].capitalize())
* 5. enumerate()
* >> names = ['sam', 'joe', 'nancy', 'gil']
* >> for i, name in enumerate(names):
* print(i, name)
* names[i] = name.capitalize()
* >> print(names)
* 6. IF, ELIF, and ELSE
* >> a = 3
* >> b = 5
* >> if a == b:
* print("a is equal to b")
* elif a > b:
* print("a is bigger than b")
* else:
* print("a is smaller than b")

**Exercise on if, elif, else**

1. Using if, elif and else statements write some code that does the following based on a 4-digit number you pick.

* If it is divisible by 2 and 3 you print: 'Your number is not only divisible by 2 and 3 but also by 6'.
* If it is divisible by 2 you print: 'Your number is divisible by 2'.
* If it is divisible by 3 you print: 'Your number is divisible by 3'.
* Any other option, you print: 'Your number is not divisible by 2, 3 or 6'

1. Challenge:

* Create a **for** statement containing the conditions above.
* Test your for statement on a list of several 4-digit numbers.
* >> value\_list = [9306,1231,4232,5231]
* >> for value in value\_list:
* >>    if value % 6 == 0:
* print("Your number is not only divisible by 2 and 3 but also by 6")
* elif value % 3 == 0:
* print("Your number is divisible by 3")
* elif value % 2 == 0:
* print("Your number is divisible by 2")
* else:
* print("Your number is not divisible by 2, 3, or 6")

1/17/2020 Afternoon:

from urllib.request import urlretrieve

URL = '<http://go.gwu.edu/maebulletin?accessType=DOWNLOAD>'

urlretrieve(URL, 'mae\_bulletin.txt')

mae\_bulletin\_file = open('mae\_bulletin.txt')

mae\_bulletin\_text = mae\_bulletin\_file.readlines()

courses = []

descriptions = []

**for** line **in** mae\_bulletin\_text:

    line = line.strip()            *#Remove white spaces*

**if** line == '':                 *#Skip the empty lines*

**continue**

**elif** line.startswith('MAE'):

        courses.append(line)       *#Save lines that start with MAE in list*

**else**:

        descriptions.append(line)

course\_id = []

course\_title = []

course\_credits = []

**for** course **in** courses:

    course\_info = course.split('. ')

    course\_id.append(course\_info[0])

    course\_title.append(course\_info[1])

    course\_credits.append(course\_info[2])

print(course\_id[0:5])

course\_title[:5]

print(course\_title[0:5])

course\_id.index("MAE\_3190")

print(course\_id[17])

print(course\_title[17])

print(course\_credits[17])

print(descriptions[17])

**How many courses have prerequisites?**

**for** i **in** range(4):

    print(i)

print(len(descriptions))

descriptions[:5]

course\_with\_pre = []

len(descriptions)

for i in range(len(descriptions)):

    if "Prerequisite" in descriptions[i]:

        course\_with\_pre.append(course\_id[i])

    elif "prerequisite" in descriptions[i]:

        course\_with\_pre.append(course\_id[i])

    elif "prerequisite" in descriptions[i].lower():

        course\_with\_pre.append(course\_id[i])

**Exercise:**

course\_with\_cor = []

for i in range(len(descriptions)):

    if "corequisite" in descriptions[i].lower():

        course\_with\_cor.append(course\_id[i])

**alternate option:**

for i in range(len(descriptions)):

    if "corequisite" in descriptions[i] or "Corequisite" in descriptions[i]:

        course\_with\_cor.append(course\_id[i])

print(course\_with\_cor)

fall\_and\_spring = []

fall = []

spring = []

not\_spec = []

for i in range(len(descriptions)):

    if "fall and spring" in descriptions[i].lower():

        fall\_and\_spring.append(course\_id[i])

    elif "Fall" in descriptions[i] or "fall" in descriptions[i]:

        fall.append(course\_id[i])

    elif "spring" in descriptions[i].lower():

        spring.append(course\_id[i])

    else:

        not\_spec.append(course\_id[i])

print("Fall and Spring:", fall\_and\_spring)

print(fall)

print(spring)

print(not\_spec)

Dictionaries, JSON, and APIs

location = {'building\_name': 'Gelman Library',

           'room': 301,

           'seats': 40,

           'reserved': True,

           'instructors': ['Megan', 'Laura']

           }

location.get("floor", "third") #fills in a default value of "thrid" when the key "floor" is not present.

Example JSON data

<http://api.open-notify.org/astros.json>

import requests

response = requests.get(url = "<http://api.open-notify.org/astros.json>")

data = response.json()

API means Application programming interface

World Cup API:

<https://worldcup.sfg.io/matches>

worldcup\_response = requests.get(url = "<https://worldcup.sfg.io/matches>")

worldcup\_response.status\_code

matches = worldcup\_response.json()

Exercise on World Cup API:

    Which team was the winner of each match? *Hint:*

* Create an empty list named winners.
* Access the “winners” element in each match
* Add it to the winners list.

Bonus:  How many games are high-scoring games (greater than 5 goals scored total)? What were the teams in that game? The result will be a list, where each item is a list containing the names of the two teams.

ANSWER to Bonus:

high\_goal\_matches = []

for match in matches:

    home\_team\_goals = match["home\_team"]["goals"]

    away\_team\_goals = match["away\_team"]["goals

**if** home\_team\_goals + away\_team\_goals > 4:

        game\_name = match["home\_team\_country"] + "-" + match["away\_team\_country"]

    high\_goal\_matches.append(game\_name)

print(high\_goal\_matches)

\*\*\* Morning Functions Lesson:

* > def do\_nothing():
* '' This function does nothing (yet) '''
* pass //just keep going
* > type(do\_nothing)
* > ?do\_nothing
* > help(do\_nothing)
* > type(square\_maker) # type(square\_maker()) is an error
* > square\_maker(3)
* > exponent - 2
* > def to\_power(number):
* ''' This function returns the arguement given raised to the power of exponent '''
* result = number \*\* exponent
* return result
* > to\_power(2)
* > def power\_short(number): # function should be self-contained
* xyz = 5
* value = number \*\* xyz
* return value
* > power\_short(8)
* > def greeting(name):
* ''' Returns a greeting based on input argument '''
* return "Hi, " + name
* > greeting("L")
* > def greets(name):
* ''' Returns a greeting based on input argument '''
* return "Good morning, " + name
* > greets("Me")
* > def rectangle\_area(height, width):
* ''' Returns the area based on the height and width '''
* result = height \* width
* return result
* > rectangle\_area(2,5)
* > def calc\_rect\_areas(list\_of\_rectangles):
* ''' Calculates the area of multiple rectangles by taking in a nested
* of height and width pairs and returns the calculated area for each
* pair as a list'''
* areas = []
* for rectangle in list\_of\_rectangles:
* area = rectangle[0]\*rectangle[1]
* areas.append(area)
* return areas
* > a\_few\_rectangles = [[1,2], [3,4], [4,5]]
* > calc\_rect\_areas(a\_few\_rectangles)
* > def calc\_areas\_from\_lists(list\_of\_heights, list\_of\_widths):
* #print(list\_of\_heights)
* #print(list\_of\_widths)
* areas = []
* for h in list\_of\_heights:
* for w in list\_of\_widths:
* area = h\*w
* areas.append(area)
* return areas
* > list\_of\_heights = [1, 2, 3]
* > list\_of\_widths = [2, 3, 4]
* > calc\_areas\_from\_lists(list\_of\_heights, list\_of\_widths)

\*\*\* Morning Numpy Lesson \*\*\*

* > import numpy as np
* > np.ones(6)
* > np.zeros(4)
* > np.arange(10)
* Syntax for arange()
* np.arange(start, stop, step)
* if 1 arg given np.arange(stop)
* if 2 args given np.arange(start, stop)
* if 3 args given np.arange(start, stop, step)
* Syntax for np.linspace()
* evenly spaced
* np.linspace(start, stop, num)
* default for num is 50
* start and stop are included
* np.linspace(-1,1,5)

np.linspace

* <https://docs.scipy.org/doc/numpy/reference/generated/numpy.linspace.html?highlight=numpy%20linspace#numpy.linspace>
* x\_array = np.linspace(-1, 1, 9)
* print(x\_array)
* > y\_array = x\_array \*\* 2
* > print(y\_array)
* > z\_array = np.sqrt(y\_array)
* > print(z\_array)
* > add\_array = x\_array + y\_array
* > print(add\_array)
* > mult\_array = x\_array \* z\_array
* > print(mult\_array)
* a = np.array([14, 20, 75, 90])
* b = np.array([10, 32, 18, 120])
* total\_array = a + b
* total\_array
* > less\_array = total\_array - 10
* > type(less\_array) #numpy.ndarray refers to n imensional array
* new\_array = np.array([18, 20, 3, 5, 8, 21])
* > new\_scores\_3 = []
* > for i, score in enumerate(list(scores)):
* new\_scores\_3.append(score + 1)
* > print(new\_scores\_3)
* > array\_2d = np.array([[1,2],[3,4]])
* > np.shape(array\_2d)
* > X = np.array([[1,2],[3,4]])
* > Y = np.array([[1,-1],[0,1]])
* > X+Y
* > X\*Y
* > X @ Y # dot\_product\_1
* > np.dot(X,Y) # dot\_product\_2
* > X[0,0] # accessing row 0, column 0
* > X[:, 0] # accessing all elements in column 0
* > a = np.arange(24)
* > a\_3d = np.reshape(a, (2,3,4))

\*\*\* Afternoon MatPlotLib lesson \*\*\*

from urllib.request import urlretrieve

URL = '<http://go.gwu.edu/engcomp1data5?accessType=DOWNLOAD>'

urlretrieve(URL, 'land\_global\_temperature\_anomaly-1880-2016.csv')

import numpy as np

fname = "land\_global\_temperature\_anomaly-1880-2016.csv"

year, temp\_anomaly = np.loadtxt(fname, delimiter=",", skiprows=5, unpack=True)

from matplotlib import pyplot

%matplotlib inline

pyplot.plot(year, temp\_anomaly)

pyplot.rc('font', family="serif", size="18")

pyplot.plot(year, temp\_anomaly, color="green", linestyle="-", linewidth=2)

pyplot.plot(year, temp\_anomaly, color="green", linestyle="-", linewidth=1)

#black line

pyplot.plot(year, temp\_anomaly, color="k", linestyle="-", linewidth=1)

# color = k (black), b (blue), #2929A3 (dark blue)

#add title, figsize, axis labels

pyplot.figure(figsize=(10,5))

pyplot.title("Land global temp anomalies. \n")

pyplot.xlabel("Year")

# "$\circ$" is the degree sign

pyplot.ylabel("Land temperature anomaly [$\circ$C]")

pyplot.grid()

# ";" added to the last row of each cell to avoid unnecessary stuff appearing

\*\*\* Linear Regression \*\*\*

a\_1n, a\_0n = np.polyfit(year, temp\_anomaly, 1)

print(a\_1n, type(a\_1n))

print(a\_0n, type(a\_0n))

f\_linear = np.poly1d((a\_1n, a\_0n)) #poly-one-d

print(f\_linear)

pyplot.figure(figsize=(10,5))

pyplot.plot(year, temp\_anomaly, color="blue", linestyle="-", linewidth=1)

pyplot.plot(year, f\_linear(year), "k--", linewidth=2, label="Linear regression")

pyplot.title("Land global temp anomalies. \n")

pyplot.xlabel("Year")

pyplot.ylabel("Land temperature anomaly [$\circ$C]")

pyplot.legend(loc = "best", fontsize = 15)

pyplot.grid()

# np.where(year == 1970)

# year[0:3]

year\_1, temp\_anomaly\_1 = year[0:90], temp\_anomaly[0:90]

year\_2, temp\_anomaly\_2 = year[90:], temp\_anomaly[90:]

m1, b1 = np.polyfit(year\_1, temp\_anomaly\_1, 1)

m2, b2 = np.polyfit(year\_2, temp\_anomaly\_2, 1)

# print(m1, b1)

# print(m2, b2)

f\_linear\_1 = np.poly1d((m1, b1))

f\_linear\_2 = np.poly1d((m2, b2))

# print(f\_linear\_1)

# print(f\_linear\_2)

pyplot.figure(figsize=(10,5))

pyplot.plot(year, temp\_anomaly, color="blue", linestyle="-", linewidth=1)

pyplot.plot(year\_1, f\_linear\_1(year\_1), "g--", linewidth = 2, label = "1880-1969")

pyplot.plot(year\_2, f\_linear\_2(year\_2), "r--", linewidth = 2, label = "1970-2016")

pyplot.title("Land global temp anomalies. \n")

pyplot.xlabel("Year")

pyplot.ylabel("Land temperature anomaly [$\circ$C]")

pyplot.legend(loc = "best", fontsize = 15)

pyplot.grid()

\*\*\* Pandas lesson \*\*\*

from urllib.request import urlretrieve

URL = "<http://go.gwu.edu/engcomp2data1?accessType=DOWNLOAD>"

urlretrieve(URL, "beers.csv")

beers = pd.read\_csv("beers.csv")

type(beers)

beers["abv"]

type(beers["abv"])

beers["abv"][:10]

abv\_series = beers["abv"]

len(abv\_series)

abv\_clean = abv\_series.dropna()

abv = abv\_clean.values

type(abv)

np.max(abv\_clean)

np.min(abv\_clean)

np.mean(abv\_clean)

np.std(abv\_clean)

ibu\_series = beers["ibu"]

print(len(ibu\_series))

ibu\_clean = ibu\_series.dropna()

print(len(ibu\_clean))

from matplotlib import pyplot, rcParams

rcParams["font.family"] = "serif"

rcParams["font.size"] = 16

pyplot.figure(figsize=(10,5))

pyplot.title("ABV")

pyplot.hist(abv, bins=20,color="#114499", histtype="bar", edgecolor="white")

pyplot.xlabel("Alcohol by Volume [abv]")

pyplot.ylabel("Frequency");

beers.hist(column="abv", edgecolor="white");

style\_series = beers["style"]

type(style\_series)

style\_series[:15]

style\_counts = style\_series.value\_counts()

style\_counts[0:20].plot.barh(figsize=(10,8), color = "green", edgecolor = "gray");