# Homework 3: Arrays Computations

Welcome to the third homework! Please complete this notebook by filling in the cells provided. For all problems that you must write explanations and sentences for, please provide your answer in the designated space.

**Recommended Reading**:

- Brett M (2020) sections 3.4-3.7.
- Also see chapter 4 of McKinney (2022) as a reference for a lot more information.

**Deadline:**

This assignment is due **Friday July 11 at 10 PM in Gradescope.**

Directly sharing answers is not okay, but discussing problems with the course staff or with other students is encouraged. Refer to the policies page to learn more about how to learn cooperatively.

You should start early so that you have time to get help if you're stuck.

## Getting started

In order to complete the homework it is necessary to download a few files. Please run the code below **only once** to download data needed to complete the homework. To run the code, click in the cell below and press the play buttom (or press shift-enter).

In [1]:
```
# if you are running this notebook in colabs, please uncomment and 
# !pip install https://github.com/emeyers/YData_package/tarball/mas
```

In [1]:
```
# Please run this code once to download the files you will need to 

import YData

YData.download.download_data("ACS_2017_sample_01.csv")
YData.download.download_data("temperatures.csv")
YData.download.download_data("world_population.csv")
YData.download.download_data("old_faithful.csv")

YData.download_image("array_cumsum.png")
YData.download_image("array_diff.png")
YData.download_image("salovey.png")
```

# 0. Quote and reaction

As you know, in class we have been analyzing data on movies related to the Bechdel test. This data comes from a FiveThirtyEight article The Dollar-And_Cents Case aginst Hollywood's Exclusion of Women. To understand how the data set we have been analyzing was created, and to see how data journalists have analyzed this data, please read the original article. In the space below, please write down a quote you found of interest from the article along as well as a one paragraph description for why you thought the quote was interesting.

**Question 0.1 (5 points)** Please write down your "quote and reaction" here.

*Quote:* Looking more closely at our smaller sample and how films performed on the Bechdel test over time, we found — not unexpectedly — that a greater share of films are passing today compared to 1970, but that the level has plateaued in the past two decades. There's a surprisingly large contingent of films — about 10 percent — that have either no or only one named female character in their casts. And the number of films decisively passing the test remains below 50 percent.

Reaction: It is really amazing to find out that in a era that so many people fight for the women's right, there are still so many film that can not pass the test. It leads me to think whether this test is still good enough to reflect the equal rights level.

In [16]:

```
# This cell imports functions from packages we will use below.
# Please run it each time you load the Jupyter notebook

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from imageio.v3 import imread

%matplotlib inline
```

# 1. Creating Arrays

**Question 1.1 (3 points)** Make a numpy array called `weird_numbers` containing the following numbers (in the given order):

1. -2
2. the sine of 1.2
3. 3
4. 5 to the power of the cosine of 1.2

*Hint:* `sin()` and `cos()` are functions in the `np` module.

In [22]:
```
weird_numbers = np.array([-2,math.sin(1.2),3,5**(math.cos(1.2))])
weird_numbers
```

Out[22]:
```
array([-2.        ,  0.93203909,  3.        ,  1.79174913])
```

**Question 1.2 (2 points)** Make a numpy array called `book_title_words` containing the following three strings: "Eats", "Shoots", and "and Leaves".

In [34]:
```
book_title_words = np.array(["Eats","Shoots","and Leaves"])
book_title_words
```

Out[34]:
```
array(['Eats', 'Shoots', 'and Leaves'], dtype='<U10')
```

As we have previously discussed, strings have a method called `join`. `join` takes one argument, an array of strings. It returns a single string. Specifically, the value of `a_string.join(an_array)` is a single string that's the concatenation ("putting together") of all the strings in `an_array`, **except** `a_string` is inserted in between each string. The `.join()` method works for ndarrays of strings as well as for lists of strings. Let's explore this now.

**Question 1.3 (3 points)** Use the array `book_title_words` and the method `join` to make two strings:

1. "Eats, Shoots, and Leaves" (call this one `with_commas`)
2. "Eats Shoots and Leaves" (call this one `without_commas`)

*Hint:* If you're not sure what `join` does, first try just calling, for example, `"yale".join(book_title_words)`.

In [37]:
```
with_commas = ",".join(book_title_words)
without_commas = " ".join(book_title_words)

# These lines are provided just to print out your answers.
print('with_commas:', with_commas)
print('without_commas:', without_commas)
```

```
with_commas: Eats,Shoots,and Leaves
without_commas: Eats Shoots and Leaves
```

# 2. Basic Array Arithmetic

**Question 2.1 (3 points)** Multiply each of the following numbers by 157: 42, 4224, 42422424, and -250. For this question, **don't** use arrays.

In [41]:
```
first_product = 157*42
second_product = 157*4224
third_product = 157*42422424
fourth_product = 157*-250
```

```
print(first_product, second_product, third_product, fourth_product)
```

```
6594 663168 6660320568 -39250
```

**Question 2.2 (3 points)** Now, do the same calculation, but using an array called `numbers` and only a single multiplication ( `*` ) operator. Store the 4 results in an array named `products` .

In [43]:
```
numbers = np.array([42, 4224, 42422424,-250])
products = numbers*157
products
```

Out[43]:
```
array([      6594,     663168, 6660320568,     -39250])
```

**Question 2.3 (3 points)** Oops, we made a typo! Instead of 157, we wanted to multiply each number by 1577. Compute the fixed products in the cell below using array arithmetic. Notice that your job is really easy if you previously defined an array `numbers` containing the 4 numbers.

In [45]:
```
fixed_products = numbers*1577
fixed_products
```

Out[45]:
```
array([      66234,     6661248, 66900162648,     -394250])
```

**Question 2.4 (4 points)** We've loaded an array of temperatures in the next cell. Each number is the highest temperature observed on a day at a climate observation station, mostly from the US. Since they're from the US government agency NOAA, all the temperatures are in Fahrenheit. Convert them all to Celsius by first subtracting 32 from them, then multiplying the results by $\frac{5}{9}$. Make sure to **ROUND** each result to the nearest integer using the `np.round` function.

In [47]:
```
max_temperatures = np.array(pd.read_csv("temperatures.csv")["Daily M

celsius_max_temperatures = np.round((max_temperatures-32)*5/9)
celsius_max_temperatures
```

Out[47]:
```
array([-4., 31., 32., ..., 17., 23., 16.])
```

**Question 2.5 (4 points)** The cell below loads all the *lowest* temperatures from each day (in Fahrenheit). Compute the size of the daily temperature range for each day. That is, compute the difference between each daily maximum temperature and the corresponding daily minimum temperature. **Keep your answer in Fahrenheit.** Make sure **NOT** to round your answer for this question!

In [49]:
```
min_temperatures = np.array(pd.read_csv("temperatures.csv")["Daily M

temperature_ranges = max_temperatures-min_temperatures
temperature_ranges
```

Out[49]:
```
array([12, 18, 22, ..., 31, 21, 20])
```

# 3. World Population

The cell below loads a table of estimates of the world population for different years, starting in 1950 and displays the first 4 rows of the data. The estimates come from the US Census Bureau website.

In [51]:
```
# Loading the data in a pandas DataFrame. We will discuss these Data
world = pd.read_csv("world_population.csv")[['Year', 'Population']]
world.head(4)
```

Out[51]:

| | Year | Population |
|---|---|---|
| **0** | 1950 | 2557628654 |
| **1** | 1951 | 2594939877 |
| **2** | 1952 | 2636772306 |
| **3** | 1953 | 2682053389 |

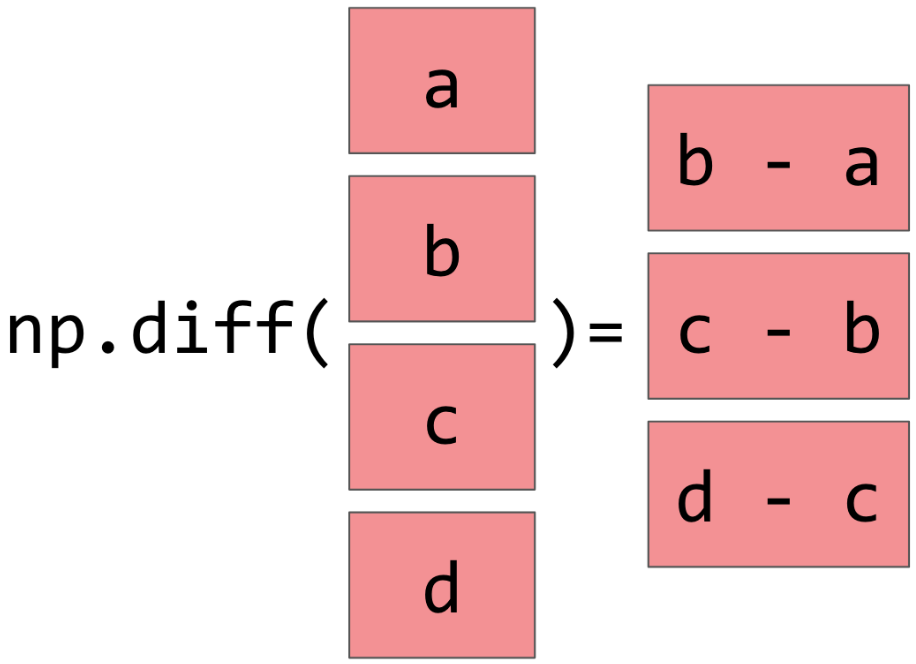
The name `population` is assigned to a numpy array of population estimates.

In [53]:
```
population = np.array(world["Population"])
population
```

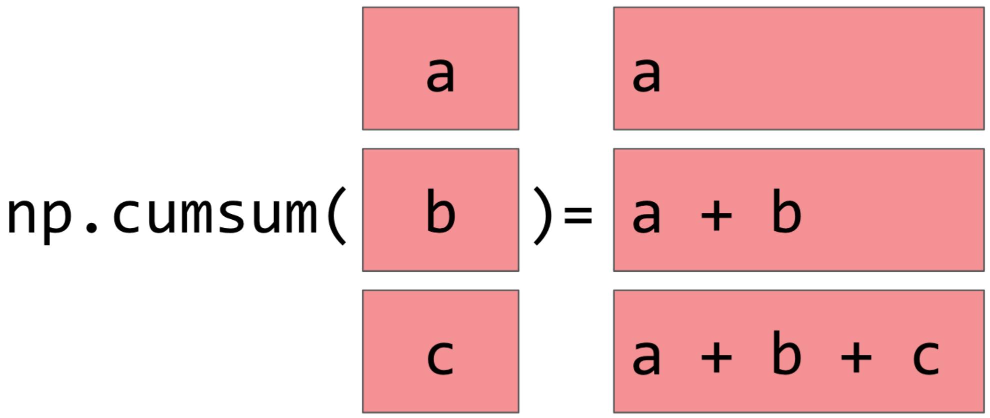
Out[53]:
```
array([2557628654, 2594939877, 2636772306, 2682053389, 2730228104,
       2782098943, 2835299673, 2891349717, 2948137248, 3000716593,
       3043001508, 3083966929, 3140093217, 3209827882, 3281201306,
       3350425793, 3420677923, 3490333715, 3562313822, 3637159050,
       3712697742, 3790326948, 3866568653, 3942096442, 4016608813,
       4089083233, 4160185010, 4232084578, 4304105753, 4379013942,
       4451362735, 4534410125, 4614566561, 4695736743, 4774569391,
       4856462699, 4940571232, 5027200492, 5114557167, 5201440110,
       5288955934, 5371585922, 5456136278, 5538268316, 5618682132,
       5699202985, 5779440593, 5857972543, 5935213248, 6012074922,
       6088571383, 6165219247, 6242016348, 6318590956, 6395699509,
       6473044732, 6551263534, 6629913759, 6709049780, 6788214394,
       6866332358, 6944055583, 7022349283, 7101027895, 7178722893,
       7256490011])
```

In this question, you will apply some built-in Numpy functions to this array.

Note: if the images below do not appear, please close and reopen your Jupyter notebook.




The difference function `np.diff` subtracts each element in an array by the element that preceeds it. As a result, the length of the array `np.diff` returns will always be one less than the length of the input array.




The cumulative sum function `np.cumsum` outputs an array of partial sums. For example, the third element in the output array corresponds to the sum of the first, second, and third elements.

**Question 3.1 (3 points)** Very often in data science, we are interested in understanding how values change with time. Use `np.diff` and `np.max` (or just `max`) to calculate the largest annual change in population between any two consecutive years.

In [55]:
```
largest_population_change = np.max(np.diff(population))
largest_population_change
```

Out [55]: 87515824

**Question 3.2 (8 points)** In class we discussed the *standard devation* as a statistic that can measure how much values vary. As you will recall, the formula for the standard deviation is:

$$s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

.

As you also recall, we can calculate the standard deviation in Python using either the `statistics` module's standard deviation function `statistics.stdev()` or using the numpy function `np.std()`. Let's now see if we can calcualte the standard deviation of the population size from year to use, *without using either of these functions*, and instead just using numpy operations. To do this, please complete the following steps:

1. Calculate the mean (average) population size and store the result in the name `mean_pop`. Do this using only numpy operations as well; i.e., **without** using `statistics.mean()` or `np.mean()`.

2. Create an array of values that has the yearly deviations, which are each year's population size minus the overall mean population size calculated across all years (i.e., the value you calculated in step 1). Store this result in the name `yearly_deviations`.

3. Next calculate the sum of the squared deviations. To do this, first square all the yearly deviation values (to make them position) and then sum them together. Store the result in the name `sum_squared_deviations`.

4. Divide this `sum_squared_deviations` by the total number of years in the data minus 1 (i.e., by $n-1$). Save this result in the name `yearly_variance`.

5. Finally, take the square root of the yearly variance to calculate the standard deviation and print out this value to "show your work".

In [65]:
```
# Calculate the mean population size
mean_pop = sum(population)/len(population)


# Calculate the each year's deviation from the mean value
yearly_deviations = population-mean_pop


# Calculate the sum of the squared deviations
```

```
sum_squared_deviations = sum((yearly_deviations)**2)


# Calculate the variance
yearly_variance = (sum_squared_deviations/(len(population)-1))**0.5


# Print out the standard deviation
print(yearly_variance)
```

```
1447259446.347368
```

**Question 3.3 (5 points)** Now compare the value you calculated for the standard deviation in Question 3.2, with using the `np.std()` function and to the `statistics.stdev()` function. In particular, print out the standard deviation's calculated using both these functions in the cell below.

In the answer section below, report if all these values the same (if they are different, you do not need to explain why, but if you're interested you can try using Google to see if you can come up with a reason).

Note: In order to use the `statistics.stdev()` function the data must be in a list, rather than in a numpy array. You can convert a numpy array to a list using the `.tolist()` method on the numpy array.

In [77]:
```
import statistics
# firstly use np.std
print(np.std(population))
print(np.std(population,ddof=1))


#then use statistivs.stdev
pop_list = population.tolist()
print(statistics.stdev(pop_list))
```

```
1436253511.3637478
1447259446.3473682
1447259446.3473682
```

**ANSWER**: They are not the same. This is because they use different degree of freedom. By adding ddof=1 to np.std(), it use the same df with statistics.stdev and then we get the same result.

# 4. Old Faithful

Old Faithful is a geyser in Yellowstone that erupts every 44 to 125 minutes (according to Wikipedia). People are often told that the geyser erupts every hour, but in fact the waiting time between eruptions is more variable. Let's take a look.

**Question 4.1 (3 points)** The first line below assigns `waiting_times` to an array of 272 consecutive waiting times between eruptions, taken from a classic 1938 dataset. Assign the names `shortest`, `longest`, and `average` so that the `print` statement is correct. (*Hint:* You can round average waiting time to 3 decimal places.)

In [82]:
```
waiting_times = np.array(pd.read_csv('old_faithful.csv')['waiting']

shortest = np.min(waiting_times)
longest = np.max(waiting_times)
average = round(np.mean(waiting_times),3)

print("Old Faithful erupts every", shortest, "to", longest, "minute
```

```
Old Faithful erupts every 43 to 96 minutes and every 70.897 minutes
on average.
```

**Question 4.2 (3 points)** Assign `biggest_decrease` to the biggest decrease in waiting time between two consecutive eruptions. For example, the third eruption occurred after 74 minutes and the fourth after 62 minutes, so the decrease in waiting time was 74 - 62 = 12 minutes.

*Hint*: The function you use may report positive or negative values. You will have to determine if the biggest decrease corresponds to the highest or lowest value. Ultimately, we want to return the absolute value of the biggest decrease so the final answer is positive.

In [89]:
```
biggest_decrease = abs(min(np.diff(waiting_times)))
biggest_decrease
```

Out[89]: 45

**Question 4.3 (4 points)** If you expected Old Faithful to erupt every hour, you would expect to wait a total of `60 * k` minutes to see `k` eruptions. Set `difference_from_expected` to an array with 272 elements, where the element at index `i` is the absolute difference between the expected and actual total amount of waiting time to see the first `i+1` eruptions.

For example, since the first three waiting times are 79, 54, and 74, the total waiting time for 3 eruptions is 79 + 54 + 74 = 207. The expected waiting time for 3 eruptions is 60 * 3 = 180. Therefore, `difference_from_expected[2]` should be $|207 - 180| = 27$.

*Hint*: You'll need to compare cumulative sum to a range. The `np.cumsum()` and `np.arange()` functions might be useful.

In [100…
```
difference_from_expected = abs(np.cumsum(waiting_times)-np.arange(6
difference_from_expected
```

Out[100…

```
array([  19,   13,   27,   29,   54,   49,   77,  102,   93,  118,
112,
        136,  154,  141,  164,  156,  158,  182,  174,  193,  184,
171,
        189,  198,  212,  235,  230,  246,  264,  283,  296,  313,
319,
        339,  353,  345,  333,  353,  352,  382,  402,  400,  424,
422,
        435,  458,  462,  455,  477,  476,  491,  521,  515,  535,
529,
        552,  563,  567,  584,  605,  604,  628,  616,  638,  638,
670,
        688,  706,  711,  724,  746,  742,  761,  772,  774,  790,
790,
        808,  824,  847,  862,  884,  894,  899,  912,  940,  956,
976,
        964,  990,  990, 1020, 1010, 1028, 1031, 1043, 1067, 1082,
1073,
       1095, 1097, 1125, 1114, 1137, 1158, 1145, 1169, 1161, 1187,
1208,
       1223, 1222, 1251, 1270, 1269, 1290, 1280, 1305, 1304, 1331,
1324,
       1333, 1350, 1346, 1374, 1395, 1380, 1402, 1397, 1427, 1412,
1435,
       1431, 1460, 1446, 1468, 1459, 1485, 1478, 1497, 1518, 1518,
1540,
       1557, 1573, 1572, 1592, 1581, 1617, 1610, 1627, 1644, 1649,
1670,
       1681, 1691, 1712, 1745, 1738, 1767, 1752, 1778, 1776, 1794,
1800,
       1816, 1819, 1847, 1839, 1872, 1861, 1858, 1875, 1883, 1904,
1925,
       1938, 1928, 1953, 1967, 1962, 1979, 2002, 2025, 2016, 2034,
2058,
       2044, 2067, 2062, 2083, 2080, 2096, 2120, 2137, 2158, 2185,
2202,
       2193, 2211, 2211, 2233, 2264, 2257, 2275, 2261, 2278, 2302,
2291,
       2314, 2325, 2345, 2334, 2349, 2353, 2369, 2362, 2396, 2391,
2407,
       2397, 2419, 2413, 2428, 2446, 2465, 2483, 2501, 2511, 2530,
2540,
       2534, 2560, 2550, 2580, 2574, 2568, 2585, 2604, 2608, 2623,
2610,
       2636, 2639, 2664, 2686, 2683, 2705, 2712, 2726, 2720, 2743,
2756,
       2769, 2797, 2817, 2828, 2851, 2847, 2866, 2884, 2908, 2906,
2929,
       2912, 2912, 2927, 2948, 2934, 2964, 2950, 2964])
```

**Question 4.4 (4 points)** If instead you guess that each waiting time will be the same as the previous waiting time, how many minutes would your guess differ from the actual time, averaging over every wait time except the first one?

For example, suppose we only had 4 waiting times which were the first for values in our `waiting_times` array; i.e., 79, 54, and 74, and 62. Then the average difference between your guesses and the actual waiting times would be $\frac{|79-54|+|54-74|+|74-62|}{3} = 19.0$.

In [103…
```
average_error = sum(abs(np.diff(waiting_times)))/(len(waiting_times
average_error
```

Out[103…
```
20.52029520295203
```

# 5. Selecting values through Boolean indexing

As we discussed in class, one way to select elements is through Boolean indexing (also called Boolean masking). Suppose we have a ndarray of values called `my_array`. Then if we have an ndarray of Booleans called `my_booleans` that is the same size as `my_array`, then using `my_array[my_booleans]` will return all the values in `my_array` where the values of `my_booleans` are `True`.

The code below loads the ACS data from homework 1 and creates lists for the earned income and sex variables. Let's explore how Boolean masking can be useful using this data.

In [105…
```
acs_data1 = pd.read_csv("ACS_2017_sample_01.csv")

# This code converts polars DataFrame data into Python lists so tha
earned_incomes = acs_data1["INCEARN"].to_list()
sex = acs_data1["SEX"].to_list()

acs_data1.head()
```

Out[105…

| | YEAR | SAMPLE | SERIAL | CBSERIAL | NUMPREC | HHWT | HHTYPE |
|---|---|---|---|---|---|---|---|
| 0 | 2017 | 201701 | 250395 | 2017000834943 | 2 | 79 | 2 |
| 1 | 2017 | 201701 | 1062813 | 2017000532479 | 2 | 27 | 1 |
| 2 | 2017 | 201701 | 1235164 | 2017001001296 | 1 | 48 | 6 |
| 3 | 2017 | 201701 | 142278 | 2017000769070 | 3 | 50 | 1 |
| 4 | 2017 | 201701 | 954089 | 2017000012970 | 5 | 15 | 1 |

5 rows × 61 columns

**Question 5.1 (3 point)** To start with, please convert the `earned_incomes` list into an ndarray called `earned_incomes_array`. Likewise, convert the `sex` list into an array called `sex_array`. Hint: using the `np.array()` function could be helpful here.

In [107…

```
# create numpy arrays of our data
earned_incomes_array = np.array(earned_incomes)
sex_array = np.array(sex)
```

**Question 5.2 (3 point)** Next, create an array of Booleans called `male_indicators` that has `True` values when the `sex_array` indicates a male, and `False` when the `sex_array` values indicates a female. Likewise create an array called `female_indicators` that has `True` values when the `sex_array` indicates a female, and `False` when the `sex_array` values indicates a male. Print the first 5 entries of the `female_indicators` array to show your code is working correctly.

In [121…

```
# get indicators for males and females
male_indicators = sex_array == 1
female_indicators = sex_array == 2
print(female_indicators[0:5])
                        # print first 5 values of the female indica
```

```
[False  True  True  True  True]
```

**Question 5.3 (4 point)**. Now let's create an ndarrays called `male_incomes` that has only the earned incomes from males. Likewise, create an ndarrays called `female_incomes` that has only the earned incomes from females. Print out the number of elements in both these arrays to show your code is working correctly (and you can check that you have this correct by looking at these numbers from homework 1).

In [125…

```
# get numpy arrays of males and females

# get male incomes and female incomes
male_incomes = earned_incomes_array[male_indicators]
female_incomes = earned_incomes_array[female_indicators]

# print the number of elements in the male_incomes and female_incom
print(male_incomes)
print(female_incomes)
```

```
[     0      0   7000 ... 125000      0  26000]
[    0    150 17900 ... 12000 40000     0]
```

**Question 5.4 (3 point)** Finally, print the mean earned incomes for males and for females and print these values (rounded to have no decimal places).

In [132…

```
# get mean values for male and female incomes
mean_values_male = np.mean(male_incomes)
mean_values_female = np.mean(female_incomes)
print(mean_values_male)
print(mean_values_female)
```

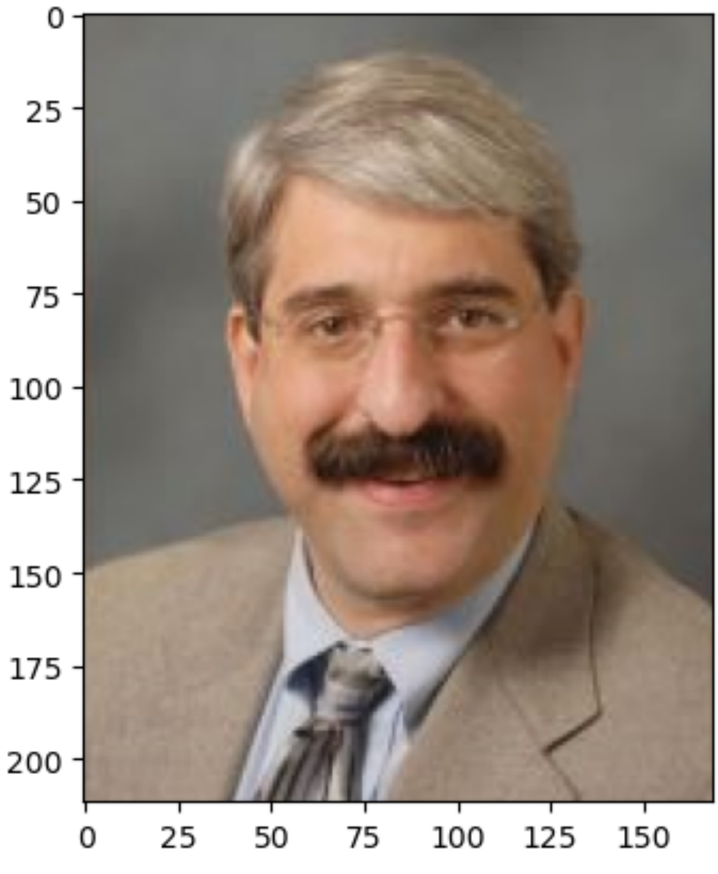
```
32916.77020408163
18759.870588235295
```

# 6. Image manipulation

As we also discussed in class, we can use Boolean indexing on higher dimensions ndarray, such as digital images. The code below loads a (slightly older) image of the former president of Yale, Peter Salovey. Let's explore Boolean masking and image manipulation on this image to make Salovey's mustache blue.

To do this we will create a series of Boolean masks that isolate where Salovey's mustache is located; i.e., the final mask will be a matrix that is the size of the original image of Salovey, and will have values of `True` where his mustache is located, and `False` in regions that do not contain his mustache. We will then use this mask to turn the `True` pixel's blue.

In [134…
```
salovey_img = imread("salovey.png");
plt.imshow(salovey_img);
```




**Question 6.1 (3 point)** To start with please print out the size of the image in terms of its length, width and color depth, and also print out the total number of pixels in the image.

In [142…
```
# the length, width and color depth of the image
print(salovey_img.shape)
# the total number of pixels
print(salovey_img.shape[0]*salovey_img.shape[1])
```

```
(212, 169, 3)
35828
```

**Question 6.2 (3 point):** Let's now create a grayscale image of Salovey. We can do this by averaging (i.e., take the mean) of the image over the 3rd dimension which is the color dimension using `np.mean(img, axis)` where `img` is our image and `axis` is the axis we are averaging over.

Please take the mean over the color dimension and save the result to the name `salovey_mean`. Then use `plt.imshow(img, cmap = 'gray')` to display the grayscale image to show that your averaging worked correctly.

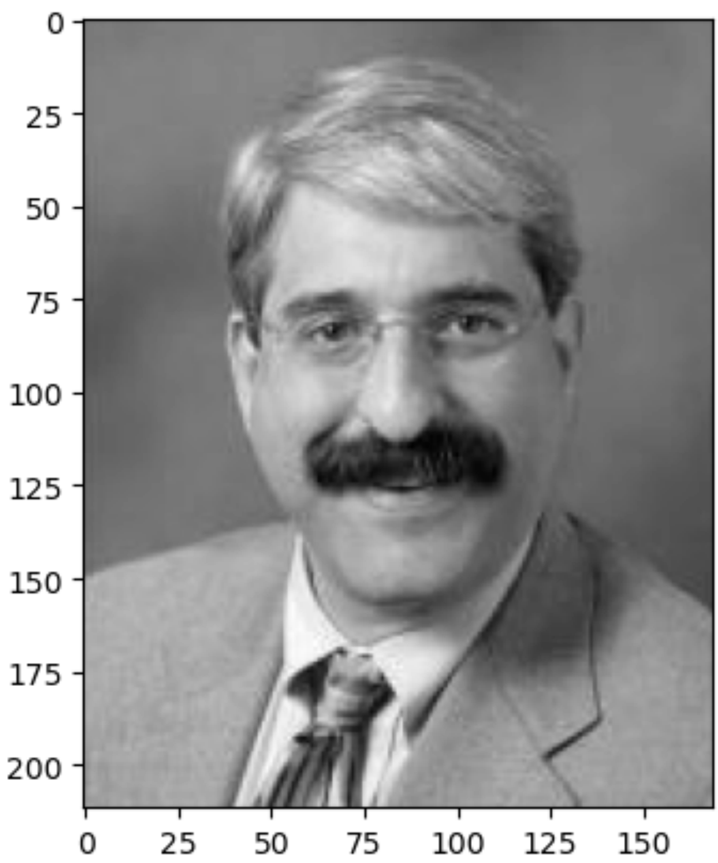
In [148…

```
# take the mean over the color dimenion of the image to create a gr
salovey_mean = np.mean(salovey_img,2)

# display the image
plt.imshow(salovey_mean,cmap='gray')
```

Out[148…

```
<matplotlib.image.AxesImage at 0x1687421b0>
```



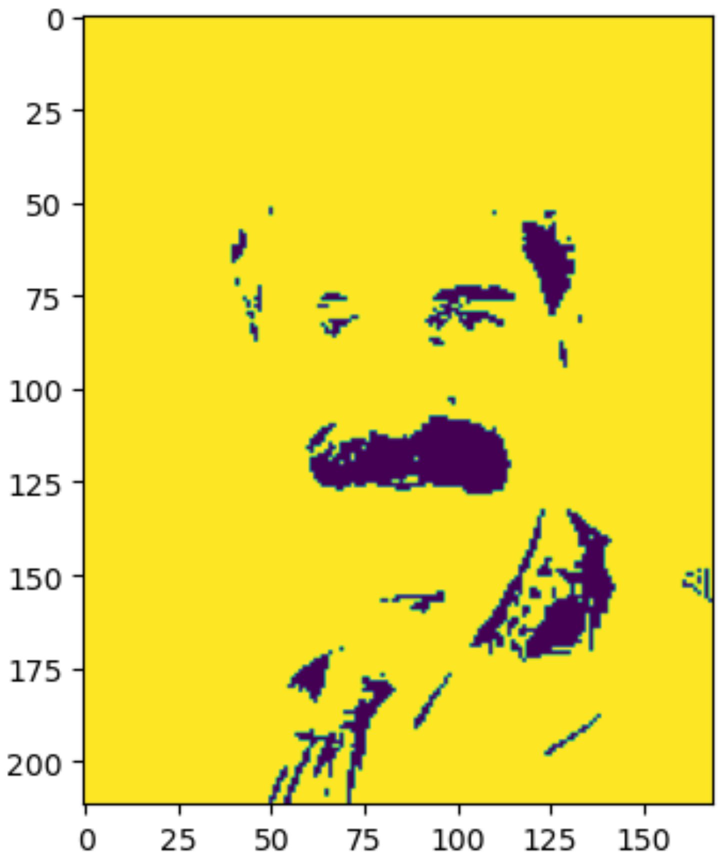

**Question 6.3 (3 point):** Let's now create Boolean array called `dark_pixel_mask` that has all the dark pixels in the grayscale image marked as `True` and the light pixels marked as `False` We will call a pixel "dark" if it has a value less than 75. I.e., you will create an ndarray the same size as the grayscale image that has only Boolean values, where values are `True` if a grayscale pixel value at the corresponding location in the grayscale image had a value less than 75, and `False` otherwise.

Once you have done this, again `plt.imshow()` to display the `dark_pixel_mask`. When displaying the `dark_pixel_mask`, use the `~` symbol to turn reverse the `True`'s and `False` which will make the dark pixels appear black (if you did this correctly, the dark pixels which should include Salovey's mustache).

In [152…
```
# create dark_pixel_mask
dark_pixel_mask = salovey_mean < 75

# display the dark_pixel_mask image
plt.imshow(~dark_pixel_mask)
```

Out[152…
```
<matplotlib.image.AxesImage at 0x1694698e0>
```




**Question 6.4 (6 point):** Let's now create another image mask (i.e., a Boolean matrix) called `mustache_region_mask` that has a rectangle around where Salovey's mustache is located.

To do this please use the folowing steps:

1. Start by creating a Boolean mask called `mustache_region_mask` as a matrix of all zeros that is the same size as the `salovey_mean` image.
2. Set the pixel values of `mustache_region_mask` to 1 in a rectangular region around where Salovey's mustache is located.
3. Convert `mustache_region_mask` into a Boolean values. Hint: using the

`.astype()` method will be useful for this.

4. Display the `mustache_region_mask`, again using the `~` symbol to turn reverse the `True`'s and `False` which will make the dark pixels appear black. The image displayed should be a black rectangle at the location of where Salovey's mustache is located in the original/grayscale image.

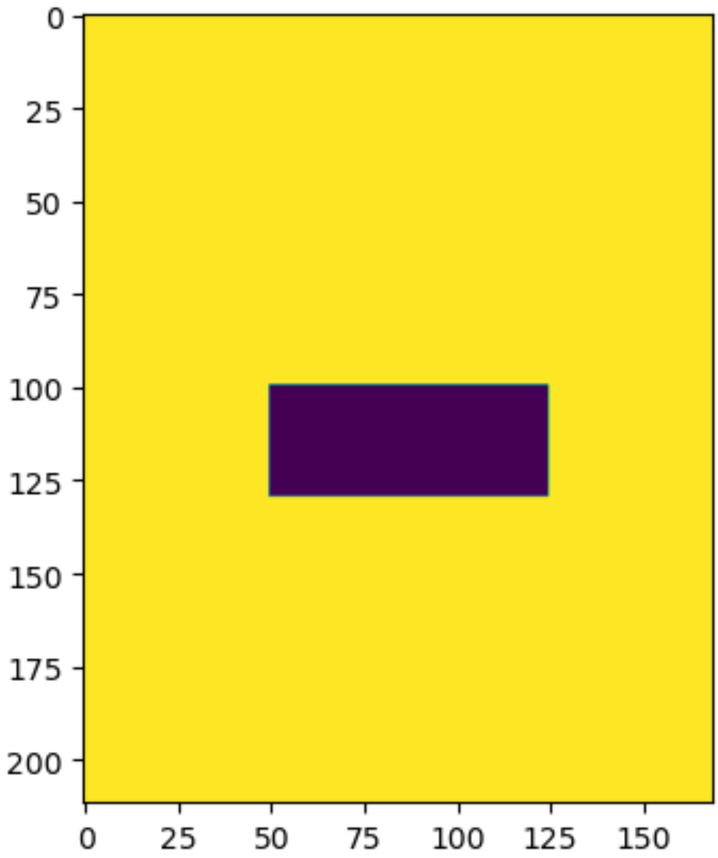
In [177…
```
# create a matrix of all 0's
mustache_rigion_mask = np.zeros((salovey_img.shape[0],salovey_img.sl

# set the region around the mustache to the value of 1
mustache_rigion_mask[100:130,50:125] = 1

# convert to Booleans
mustache_rigion_mask_bool = mustache_rigion_mask.astype(bool)

# show the mask
plt.imshow(~mustache_rigion_mask_bool)
```

Out[177…
```
<matplotlib.image.AxesImage at 0x169444c80>
```



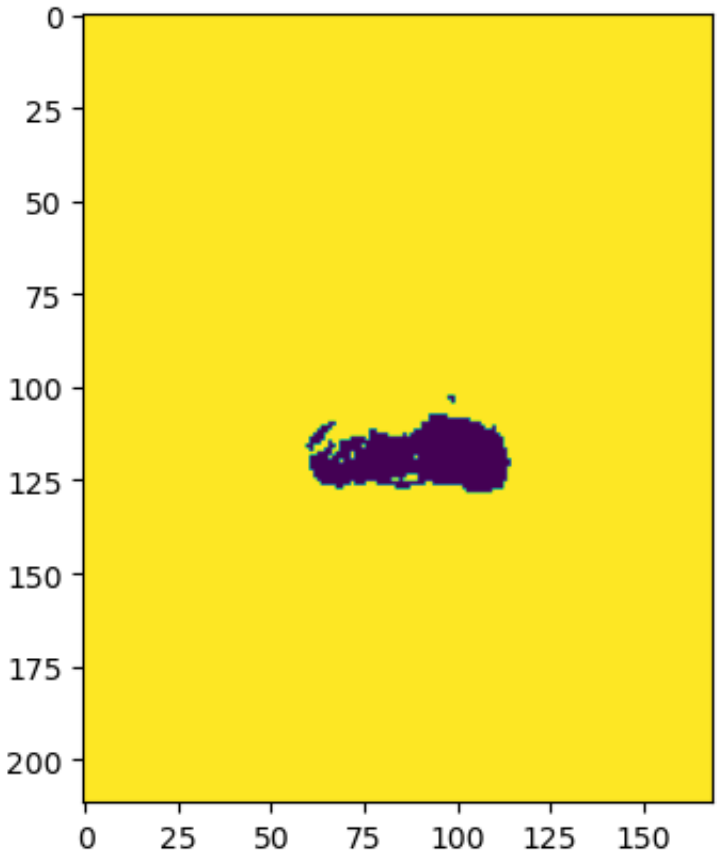

**Question 6.5 (3 point):** Now let's create a mask called `mustache_mask` that isolates just the mustache. To do this, simply multiply the `mustache_region_mask` by the `dark_pixel_mask`.

Once you have created the `mustache_mask` again using the `~` symbol to turn reverse the `True`'s and `False` and display the mask, which should just contain the mustache (it won't be perfect but should be pretty close).

In [185…
```
# create a mask of just mustache by multiplying the rectangular mas
mustache_mask = mustache_rigion_mask_bool * dark_pixel_mask

# show the mustache_mask
plt.imshow(~mustache_mask)
```

Out[185…
```
<matplotlib.image.AxesImage at 0x16c2287d0>
```




**Question 6.6 (3 point):** We are now ready to turn Salovey's mustache "Bulldog blue". To do this please complete the following steps:

1. The code below create a copy of the picture of Salovey and extracts the pixels in the red channel to a matrix called `r_image`. Please also extract the pixels in the blue and green channels to matrices called `g_image` and `b_image`.

2. "Bulldog blue" has has an RGB value of (0, 47, 108). Use the `mustache_mask` to set the `r_image`, `b_image` and `g_image` pixels values to the RGB values that will make Salovey's mustache blue. The code below shows you to do this for the `r_image` so please just do this for the `b_image` and `g_image`.

3. The code below combines the `r_image`, `g_image` and `b_image` matrices together into a single RGB image and displays the image. You do not need to change anything to this code. If your code is correct, then you

should see a blue mustache!

In [195…

```
# extract the RGB image channels
salovey_mustache = salovey_img.copy()
r_image = salovey_mustache[:, :, 0]
g_image = salovey_mustache[:, :, 1]              # extract the greei
b_image = salovey_mustache[:, :, 2]              # extract the blue


# use the mustache_mask to make the appropriate pixels in the r_imag
# b_image, and g_image matrices the appropriate values
r_image[mustache_mask] = 0      # set the red channel mustache pixe
g_image[mustache_mask] = 47                   # set the blue channe
b_image[mustache_mask] = 108                   # set the green chanı

# combine the RGB channels together into a single image and show the
# you do not need to change these lines of code
blue_stash = np.dstack((r_image, g_image, b_image))
plt.imshow(blue_stash);
plt.axis('off');
```

## 7. Reflection (3 points)

Please reflect on how the homework 3 went by going to Canvas, going to the Quizzes link, and clicking on Reflection on homework 3.

## 8. Submission

Once you're finished filling in and running all cells, you should submit your assignment as a .pdf on Gradescope. You can access Gradescope through Canvas on the left-side of the class home page. The problems in each homework assignment are numbered. **NOTE:** When submitting on Gradescope, please select the correct pages of your pdf that correspond to each problem. Failure to mark pages correctly will result in points being deducted from your homework score.

If you are running Jupyter Notebooks through an Anaconda installation on your own computer, you can produce the .pdf by completing the following steps:

1. Go to "File" at the top-left of your Jupyter Notebook
2. Under "Download as" (or "Save and Export Notebook As...") and select "HTML (.html)"
3. After the .html has downloaded, open it and then select "File" and "Print" (note you will not actually be printing)
4. From the print window, select the option to save as a .pdf

If you are running the assignment in a Google Colabs, you can use the following instructions:

1. Go to "File" at the top-left of your Jupyter Notebook and select "File" and "Print" (note you will not actually be printing)
2. From the print window, select the option to save as a .pdf
3. Be sure to look over the pdf file to make sure all your code and written work is saved in a clear way.