StefinRacho-homework4

October 5, 2024

0.0.1 CS 4010: Homework 4

0.0.2 Due Friday, 10/4/2024

0.0.3 Create a New iPython (Jupyter) notebook. Name the notebook FirstAndLastName_Homework3 and save it before you start working

0.0.4 To submit, export or print your notebook as a pdf, with all outputs visible. Upload both the pdf and a copy of your notebook (.ipynb) in Canvas.

0.0.5 Histograms, Speed and Accuracy.

1) 1-D Histograms

Download the file grades.txt from Canvas.

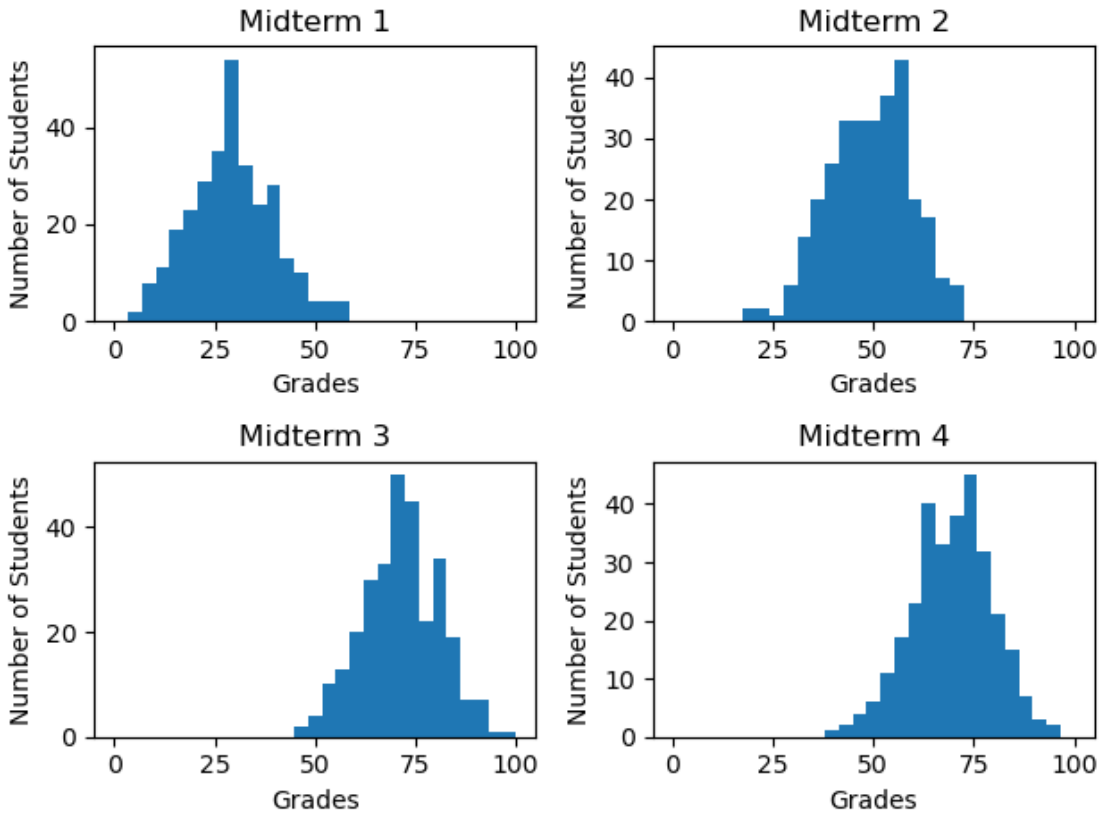
The file contains grades from a class with 300 students. Each column are the grades from their exam. Column 0 is MT1, Col1 is MT2 etc…

A) Create 4 histograms of the students' grades, 1 for each Midterm. Use the bins=linspace(0,100,nbins) functionality and adjust the nbins so the histogram isn't too "hairly" or too coarse.

[4]:
```
import numpy as np
import matplotlib.pyplot as plt
```

[5]:
```
# 1a
grades = np.loadtxt("grades.txt")

nbins = 30
bins = np.linspace(0, 100, nbins)

fig, axs = plt.subplots(2, 2)

# Midterm 1 histogram
axs[0, 0].hist(grades[:, 0], bins=bins)
axs[0, 0].set_title('Midterm 1')
axs[0, 0].set_xlabel('Grades')
axs[0, 0].set_ylabel('Number of Students')

# Midterm 2 histogram
axs[0, 1].hist(grades[:, 1], bins=bins)
```

```
axs[0, 1].set_title('Midterm 2')
axs[0, 1].set_xlabel('Grades')
axs[0, 1].set_ylabel('Number of Students')

# Midterm 3 histogram
axs[1, 0].hist(grades[:, 2], bins=bins)
axs[1, 0].set_title('Midterm 3')
axs[1, 0].set_xlabel('Grades')
axs[1, 0].set_ylabel('Number of Students')

# Midterm 4 histogram
axs[1, 1].hist(grades[:, 3], bins=bins)
axs[1, 1].set_title('Midterm 4')
axs[1, 1].set_xlabel('Grades')
axs[1, 1].set_ylabel('Number of Students')

plt.tight_layout()
plt.show()
```



B) The hist function will returns 2 lists of values called a tuple, you can save this array by assigning the result to a variable like this for example

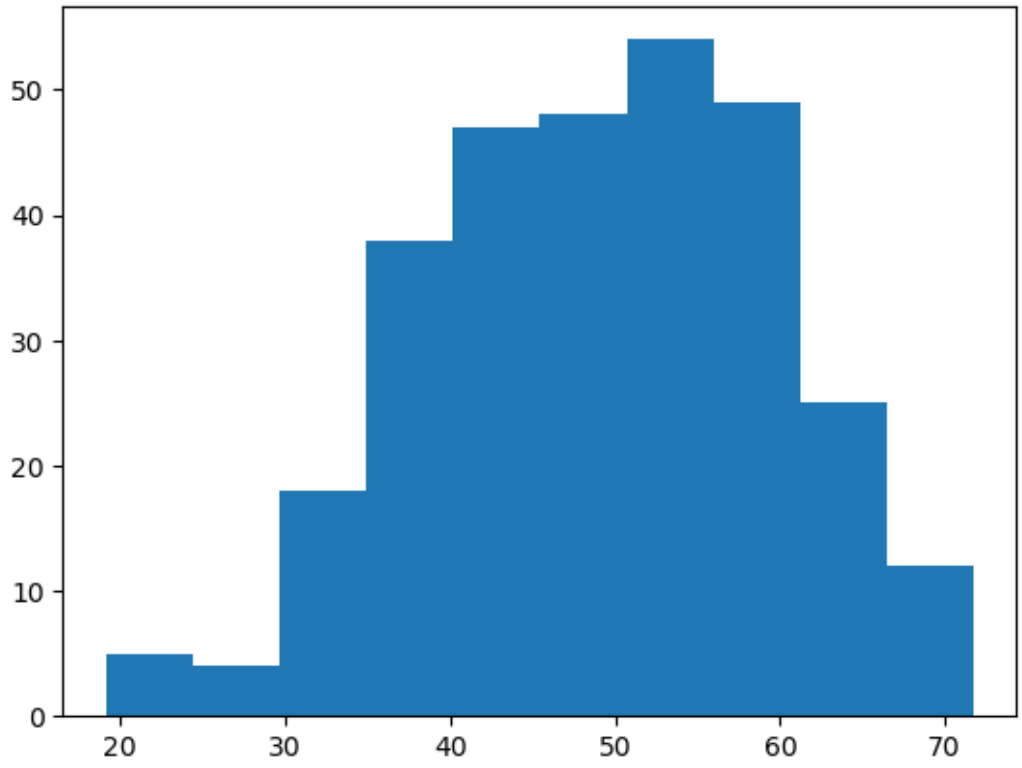
h1 = hist(grades[: , 1])

print(type(h1))

h1[0] is the array of the histogram's y-values. h1[1] are the bin edges of the histogram. So the len(h1[1]) will be greater than len(h1[0]) by 1 all the time. This is silly so let's change that to bin center instead.

C) Write a function BinCenter that takes a list of bin edges and returns the bin centers instead. The return list should be 1 element shorter than the original.

[7]:
```
h1 = plt.hist(grades[:, 1])

# 1b, 1c
def BinCenter(bin_edges):
    return (bin_edges[:-1] + bin_edges[1:]) / 2
```



D) Use your BinCenter function to change all the h1[1] bin edge list to binCenter lists.

[9]:
```
# 1d
bin_centers = BinCenter(h1[1])

print(f"Bin edges: {h1[1]}")
print(f"Bin centers: {bin_centers}")
```

```
Bin edges: [19.16  24.416 29.672 34.928 40.184 45.44  50.696 55.952 61.208
66.464
 71.72 ]
Bin centers: [21.788 27.044 32.3   37.556 42.812 48.068 53.324 58.58  63.836
69.092]
```

E) Write a function WeightedAvg that computes the weighted average with a given list of vals and weights

$$Avg = \frac{\sum val * weight}{\sum weight}$$

[11]:
```
# 1e
def WeightedAvg(vals, weights):
    total_weight = sum(weights)

    if (total_weight == 0):
        return np.nan

    weighted_sum = sum(val * weight for val, weight in zip(vals, weights))

    return weighted_sum / total_weight
```

F) Use your BinCenter and WeightedAvg to find the mean of each histogram and see if it matches the peaks you see in the histogram.

[13]:
```
# 1f
for i in range(4):
    frequencies, bin_edges = np.histogram(grades[:, i], bins=bins)

    bin_centers = BinCenter(bin_edges)

    mean = WeightedAvg(bin_centers, frequencies)

    print(f"Midterm {i + 1} - Calculated Mean: {mean:.2f}")
```

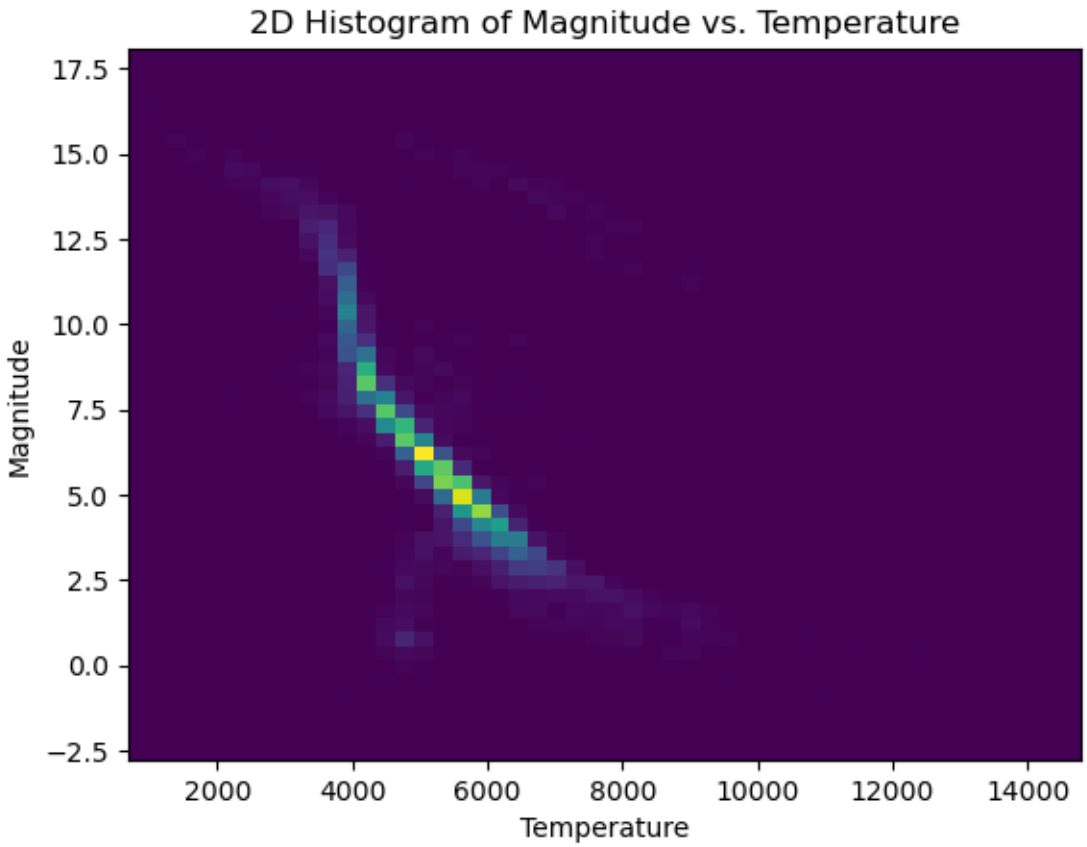
```
Midterm 1 - Calculated Mean: 29.25
Midterm 2 - Calculated Mean: 49.06
Midterm 3 - Calculated Mean: 71.45
Midterm 4 - Calculated Mean: 69.60
```

2) 2-D Histograms

A) Download and load "stars.txt" in your program. Make a 2D Histogram of Magnitude of the Stars vs Temperature of the Stars. Choose a 50x50 bins size.

[16]:
```
stars = np.loadtxt("stars.txt")

temp = stars[:, 0]
magnitude = stars[:, 1]
```

```
plt.hist2d(temp, magnitude, bins=[50, 50])
plt.title("2D Histogram of Magnitude vs. Temperature")
plt.xlabel("Temperature")
plt.ylabel("Magnitude")

plt.show()
```



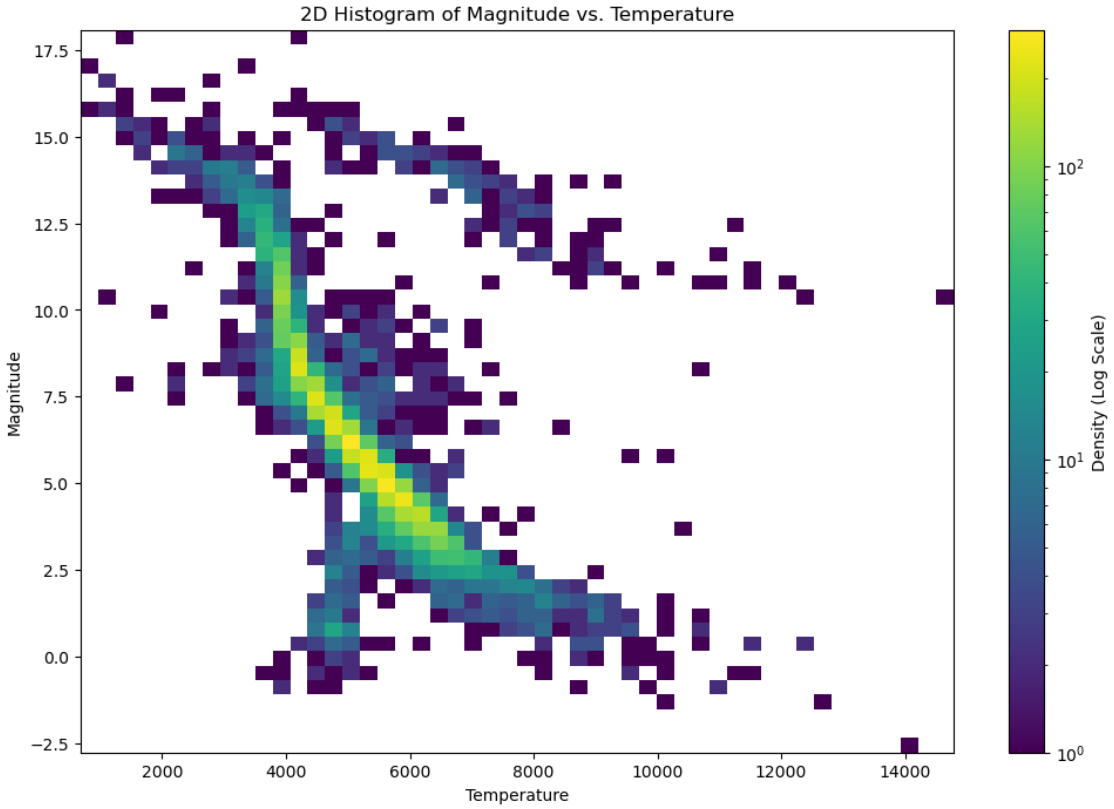
B) Change the color scale to LogNorm instead and add the colorbar to the plot and increase the figure size to (12,8)

from matplotlib.colors import LogNorm from pylab import colorbar, figure

[18]:
```
from matplotlib.colors import LogNorm
from pylab import colorbar, figure
```

[19]:
```
plt.figure(figsize=(12, 8))

plt.hist2d(temp, magnitude, bins=[50, 50], norm=LogNorm())
plt.colorbar(label="Density (Log Scale)")
plt.title("2D Histogram of Magnitude vs. Temperature")
```

```
plt.xlabel("Temperature")
plt.ylabel("Magnitude")
plt.show()
```



C) hist2D also returns the histogram information to you in tuple form, but everything is in 2D now!

h2 = hist2D(stars[:,0],stars[:,1])

print(type(h2))

h2[0] is the NxM array of the values of the histogram.

h2[1] is the array of x bin edges

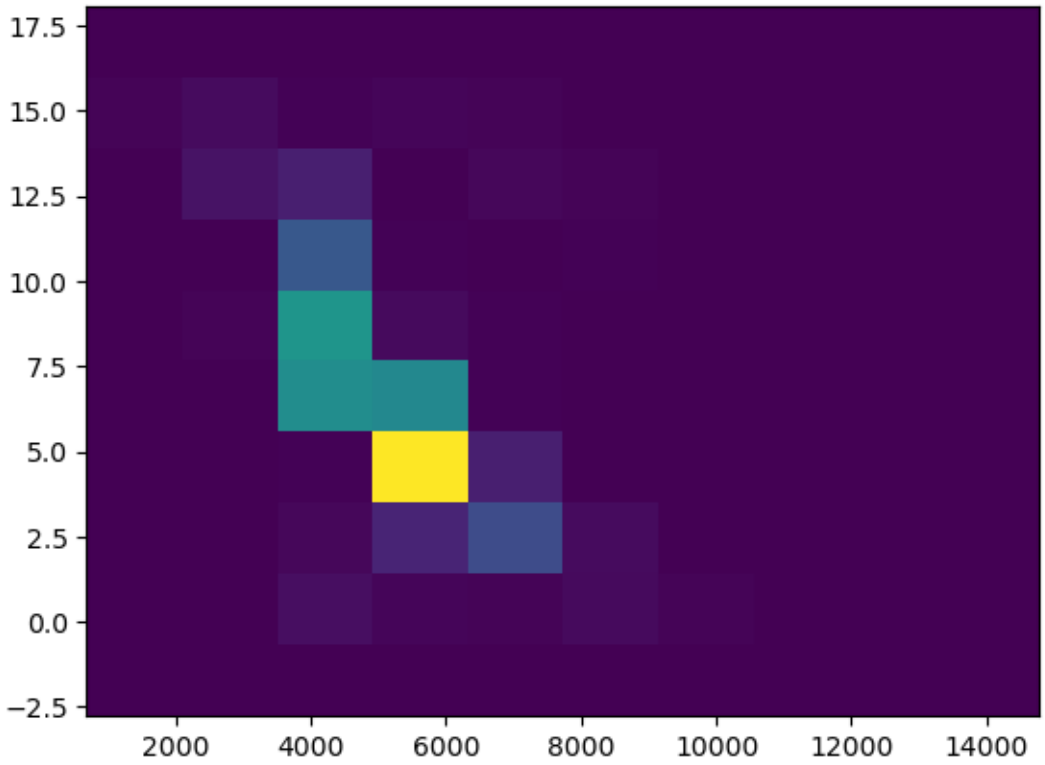
h2[2] is the array of y bin edges

The bin edges have 1 more elements than the number of bins as usual, but we can use BinCenter to fix that. Use BinCenter to save the arrays of the Temperature Bin Center and Magnitude Bin Center.

[21]:
```
h2 = plt.hist2d(stars[:, 0], stars[:, 1])

temp_bin_centers = BinCenter(h2[1])
```

```
magnitude_bin_centers = BinCenter(h2[2])

print(f"Temp bin centers: {temp_bin_centers}")
print(f"Magnitude bin centers: {magnitude_bin_centers}")
```

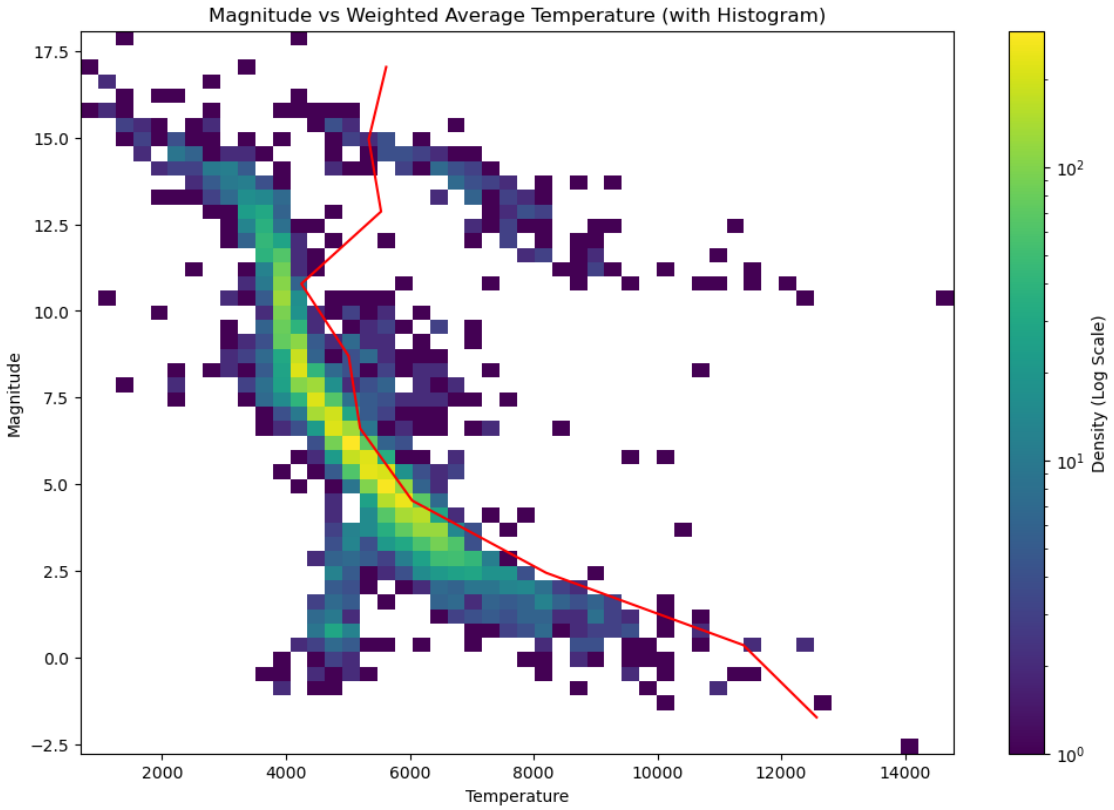
```
Temp bin centers: [ 1388.29143464  2798.58413311  4208.87683158  5619.16953005
  7029.46222852  8439.75492699  9850.04762546 11260.34032393
 12670.6330224  14080.92572087]
Magnitude bin centers: [-1.7275  0.3575  2.4425  4.5275  6.6125  8.6975 10.7825
12.8675 14.9525
 17.0375]
```



D) h2[0] is a 2D array so that's harder to deal with. Write a program that loops over the y-axis (Magnitude) of h2[0] and compute the WeightedAvg for each row of x-values (Temperature). Save these values to an array. Plot Magnitude vs WeightedAvg Temperature you computed and overlay it with the Histogram.

[23]:
```
weighted_avg_temps = []
for i in range(len(h2[0])):
    row_counts = h2[0][i]
    weighted_avg = WeightedAvg(temp_bin_centers, row_counts)
    weighted_avg_temps.append(weighted_avg)
```

```
weighted_avg_temps = np.array(weighted_avg_temps)

plt.figure(figsize=(12, 8))
plt.hist2d(temp, magnitude, bins=[50, 50], norm=LogNorm())
plt.colorbar(label='Density (Log Scale)')

plt.plot(weighted_avg_temps, magnitude_bin_centers, color="red",␣
 ↪label="Weighted Avg Temp")
plt.xlabel("Temperature")
plt.ylabel("Magnitude")
plt.title("Magnitude vs Weighted Average Temperature (with Histogram)")
plt.show()
```



E) Noticed that your WeightedAvg goes through the top of the Histogram most in the bulk but does NOT near the top or bottom. Explain why briefly.

The reason why WeightedAvg goes through the top of the Histogram most in the bulk but does not near the top or bottom is because the weighted average aligns with the peak of the histogram. At the top and bottom, there are fewer data points, so the weighted average is skewed towards outliers. So, small variations have a larger impact on the average.

3) Factorials revisited

For Homework 3, you had to make two different function to calculate factorials. One used a for loop, and one used a recursive function that called itsself.

A) Which of these function was faster for calculating large factorials? (Say, 100!) You'll needed to use

import timeit

or something similar to time the two functions.

[28]:
```
import timeit
```

[29]:
```
def factorial(n):
    product = 1
    for i in range(1, n + 1):
        product *= i

    return product

def factorial_upgraded(n):
    if (n == 1):
        return 1

    return n * factorial_upgraded(n - 1)

time_factorial = timeit.timeit(lambda: factorial(100))
time_factorial_upgraded = timeit.timeit(lambda: factorial_upgraded(100))

print(f"factorial(100) = {time_factorial}")
print(f"factorial_upgraded(100) = {time_factorial_upgraded}")
```

```
factorial(100) = 7.382634600000529
factorial_upgraded(100) = 11.109075000000303
```

B) What is the largest factorial your functions can calculate as a floating point number, before they return infinity? You might need to modify your factorial function to return a float by setting the initial number (0!) to 1.0 instead of 1

[31]:
```
def largest_factorial(n):
    product = 1.0
    for i in range(1, n + 1):
        product *= i
        if product == float("inf"):
            return float("inf"), i

    return product, None

n_loop = 1
while (True):
    _, largest_num = largest_factorial(n_loop)
```

```
    if largest_num is not None:
        break
    n_loop += 1

print(f"The largest factorial the factorial function can make is {n_loop - 1}")
```

```
The largest factorial the factorial function can make is 170
```