# David LaCharite

## Assignment 1

In [4]:
```
import numpy as np
import pandas as pd
import math
```

# Part 1

In [5]:
```
# Read Elements CSV into a pandas data frame
df_elements = pd.read_csv('elements.csv')
df_elements
```

Out[5]:

| | name | symbol | atomic_number |
|---|---|---|---|
| **0** | Hydrogen | H | 1 |
| **1** | Helium | He | 2 |
| **2** | Lithium | Li | 3 |
| **3** | Beryllium | Be | 4 |
| **4** | Boron | B | 5 |
| **5** | Carbon | C | 6 |
| **6** | Nitrogen | N | 7 |
| **7** | Oxygen | O | 8 |

In [6]:
```
# add ninth and tenth elements to dataframe
df_elements.loc[len(df_elements)]=['Fluorine','F', 9]
df_elements.loc[len(df_elements)]=['Neon','Ne', 10]
df_elements
```

Out[6]:

| | name | symbol | atomic_number |
|---|---|---|---|
| 0 | Hydrogen | H | 1 |
| 1 | Helium | He | 2 |
| 2 | Lithium | Li | 3 |
| 3 | Beryllium | Be | 4 |
| 4 | Boron | B | 5 |
| 5 | Carbon | C | 6 |
| 6 | Nitrogen | N | 7 |
| 7 | Oxygen | O | 8 |
| 8 | Fluorine | F | 9 |
| 9 | Neon | Ne | 10 |

In [7]:
```
# add a column with the atomic weights rounded to the nearest inetger
df_elements['atomic_weight'] = ['1','4','7','9','11','12','14','16','19','2(
df_elements
```

Out[7]:

| | name | symbol | atomic_number | atomic_weight |
|---|---|---|---|---|
| 0 | Hydrogen | H | 1 | 1 |
| 1 | Helium | He | 2 | 4 |
| 2 | Lithium | Li | 3 | 7 |
| 3 | Beryllium | Be | 4 | 9 |
| 4 | Boron | B | 5 | 11 |
| 5 | Carbon | C | 6 | 12 |
| 6 | Nitrogen | N | 7 | 14 |
| 7 | Oxygen | O | 8 | 16 |
| 8 | Fluorine | F | 9 | 19 |
| 9 | Neon | Ne | 10 | 20 |

# Part 2

In [8]:
```
# Make a list of strings for nine Greek letters, ‘alpha’, for example.
# Make that list such that they are not in alphabetic order
greekLetters = ['delta','alpha','phi','iota','lambda',
                'gamma','eta','tau','epsilon']
```

In [39]:
```
# Make two 9-element numpy arrays of random floating-point numbers with the
# estimated mean 10 and standard deviation 1.5
mu = 10
sigma = 1.5
NPTS = 9
random1 = [sigma * x + mu for x in np.random.randn(NPTS)]
random2 = [sigma * x + mu for x in np.random.randn(NPTS)]
```

In [40]:
```
# Make an array of nine elements ranging from zero to two times pi
range_low = 0
range_high = 2*math.pi
angle = np.random.uniform(range_low, range_high, NPTS)
```

In [41]:
```
# Make another array holding the cosine of that 'angle' array.
cosine = [math.cos(x) for x in angle]
```

In [42]:
```
# Construct a dictionary from all of the above
dict = d = {'Letter':greekLetters,
            'Random_1':random1,
            'Random_2':random2,
            'Angle' : angle,
            'Cosine' : cosine}
```

In [43]:
```
# Form a DataFrame from that dictionary and print it out
df_letters = pd.DataFrame(dict)
df_letters
```

Out[43]:

| | Letter | Random_1 | Random_2 | Angle | Cosine |
|---|---|---|---|---|---|
| **0** | delta | 7.475861 | 7.977113 | 5.396055 | 0.631639 |
| **1** | alpha | 11.068679 | 9.758309 | 0.027245 | 0.999629 |
| **2** | phi | 6.533686 | 14.392584 | 2.182581 | -0.574329 |
| **3** | iota | 10.588419 | 12.726664 | 5.694490 | 0.831666 |
| **4** | lambda | 10.289104 | 11.692746 | 0.759428 | 0.725230 |
| **5** | gamma | 11.861586 | 9.703879 | 5.781278 | 0.876667 |
| **6** | eta | 7.539113 | 10.283799 | 4.162618 | -0.522492 |
| **7** | tau | 9.686944 | 9.722367 | 6.263763 | 0.999811 |
| **8** | epsilon | 11.194167 | 8.763720 | 2.054293 | -0.464877 |

In [48]:
```
# Sort the DataFrame ascending on the Greek letters,
trimmed_df = df_letters.sort_values(by=['Letter'])
```

In [49]:
```
# drop two columns of yourchoice
trimmed_df.drop(['Random_2', 'Cosine'], axis=1, inplace=True)
```

In [50]:
```
# drop one of the rows
trimmed_df.drop(trimmed_df[trimmed_df['Letter'] == 'eta'].index, inplace=Tru
```

In [51]:
```
# and print that out
trimmed_df
```

Out[51]:

| | Letter | Random_1 | Angle |
|---|---|---|---|
| **1** | alpha | 11.068679 | 0.027245 |
| **0** | delta | 7.475861 | 5.396055 |
| **8** | epsilon | 11.194167 | 2.054293 |
| **5** | gamma | 11.861586 | 5.781278 |
| **3** | iota | 10.588419 | 5.694490 |
| **4** | lambda | 10.289104 | 0.759428 |
| **2** | phi | 6.533686 | 2.182581 |
| **7** | tau | 9.686944 | 6.263763 |

# Part 3

In [27]:
```
# Write a program in Python to create and print out the first twelve Fibona
numFibs = 12
fibsList = [0, 1]
while len(fibsList) < numFibs:
    fibsList.append(sum(fibsList[-2:]))
fibsList
```

Out[27]:
```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
```

In [28]:
```
# iterate over the last five numbers to build another list
# with the ratio of each number to its predecessor
numRatios = 5
ratioList = []
for i in range(numFibs - numRatios, numFibs):
    ratioList.append(fibsList[i] / fibsList[i-1])
ratioList
```

Out[28]:
```
[1.625,
 1.6153846153846154,
 1.619047619047619,
 1.6176470588235294,
 1.6181818181818182]
```

## What do you observe about this latter list?

The latter list is a convergent sequence with a limit of 1.618 which is ≈ the Golden Ratio.

# Part 4

In [52]:
```
# Provide a function that converts temperature in Kelvin to Rankine
def kelvin_to_rankin(K):
    return K * 1.8
```

In [53]:
```
# Make a list of five Kelvin temperatures and print out their values in Rank
kelvinTemps = [0, 223, 283, 333, 373]
rankinTemps1 = [kelvin_to_rankin(K) for K in kelvinTemps]
print(rankinTemps1)
```

```
[0.0, 401.40000000000003, 509.40000000000003, 599.4, 671.4]
```

In [54]:
```
# Repeat using a lambda function.
rankinTemps2 = list(map(lambda x: x * 1.8, kelvinTemps))
print(rankinTemps2)
```

```
[0.0, 401.40000000000003, 509.40000000000003, 599.4, 671.4]
```