Using Python for Research Homework: Week 1

In this homework, we will use objects, functions, and randomness to find the length of documents, approximate π, and smooth out random noise.

Exercise 1

In this five-part exercise, we will count the frequency of each letter in a given string.

Exercise 1a

- Import the `string` library.
- Create a variable `alphabet` that consists of the lowercase and uppercase letters in the English alphabet using the `ascii_letters` data attribute of the `string` library.

In [7]:
```
# write your code here!
import string as str
alphabet = str.ascii_letters
```

Exercise 1b

- The lower and upper case letters of the English alphabet should stored as the string variable `alphabet`.
- Consider the sentence 'Jim quickly realized that the beautiful gowns are expensive'. Create a dictionary `count_letters` with keys consisting of each unique letter in the sentence and values consisting of the number of times each letter is used in this sentence. Count upper case and lower case letters separately in the dictionary.

In [14]:
```
sentence = 'Jim quickly realized that the beautiful gowns are expensive'

count_letters = {}

for letter in sentence: #iterates through every letter in sentence
    if letter in alphabet: #checks whether letter in in alphabet
        if letter in count_letters: #if letter is in the alphabet, it further check
            count_letters[letter] += 1 #If so, it's count value is incremented by 1
        else:
            count_letters[letter] = 1 #if the letter is not in the dictionary, its
# write your code here!
count_letters
```

Out[14]:
```
{'J': 1,
 'i': 5,
 'm': 1,
 'q': 1,
 'u': 3,
 'c': 1,
 'k': 1,
 'l': 3,
 'y': 1,
 'r': 2,
 'e': 8,
 'a': 4,
 'z': 1,
 'd': 1,
 't': 4,
 'h': 2,
 'b': 1,
 'f': 1,
 'g': 1,
 'o': 1,
 'w': 1,
 'n': 2,
 's': 2,
 'x': 1,
 'p': 1,
 'v': 1}
```

Exercise 1c

- Rewrite your code from 1b to make a function called `counter` that takes a string `input_string` and returns a dictionary of letter counts `count_letters`.
- Use your function to call `counter(sentence)`.

In [21]:
```
# write your code here!

def counter(input_string):
    count_letters = {}
    for letter in input_string:
        if letter in alphabet:
            if letter in count_letters:
                count_letters[letter] += 1
            else:
                count_letters[letter] = 1
    return count_letters
```

Exercise 1d

- Abraham Lincoln was a president during the American Civil War. His famous 1863 Gettysburg Address has been stored as `address`. Use the `counter` function from 1c to return a dictionary consisting of the count of each letter in this address and save it as `address_count`.

In [22]:
```
address = """Four score and seven years ago our fathers brought forth on this conti
conceived in Liberty, and dedicated to the proposition that all men are created equ
great civil war, testing whether that nation, or any nation so conceived and so dec
We are met on a great battle-field of that war. We have come to dedicate a portion
resting place for those who here gave their lives that that nation might live. It i
that we should do this. But, in a larger sense, we can not dedicate -- we can not c
this ground. The brave men, living and dead, who struggled here, have consecrated i
or detract. The world will little note, nor long remember what we say here, but it
```

```
It is for us the living, rather, to be dedicated here to the unfinished work which
nobly advanced. It is rather for us to be here dedicated to the great task remainir
dead we take increased devotion to that cause for which they gave the last full mea
highly resolve that these dead shall not have died in vain -- that this nation, unc
freedom -- and that government of the people, by the people, for the people, shall

# Write your code here!
```

Exercise 1f

- The frequency of each letter in the Gettysburg Address is already stored as `address_count`. Use this dictionary to find the most common letter in the Gettysburg address.

In [32]:
```
# write your code here!
address_count = counter(address)
address_count
max(address_count.values())
for k, v in address_count.items():
  if v == max(address_count.values()):
    print(k)
```

```
e
```

Exercise 2

Consider a circle inscribed in a square. The ratio of their areas (the ratio of the area of the circle to the area of the square) is $\frac{\pi}{4}$. In this six-part exercise, we will find a way to approximate this value.

Exercise 2a

- Using the `math` library, calculate and print the value of $\frac{\pi}{4}$

In [36]:
```
# write your code here
import math
math.pi/4
```

Out[36]:
```
0.7853981633974483
```

Exercise 2b

- Using `random.uniform()`, create a function `rand()` that generates a single float between -1 and 1.
- Call `rand()` once. For us to be able to check your solution, we will use `random.seed()` to fix the seed value of the random number generator.

In [48]:
```
import random

random.seed(1) # Fixes the see of the random number generator.

def rand():
   return random.uniform(-1,1)
rand()
```

Out[48]:
```
-0.7312715117751976
```

Exercise 2c

- The distance between two points x and y is the square root of the sum of squared differences along each dimension of x and y. Write a function `distance(x, y)` that takes two vectors as its input and outputs the distance between them. Use your function to find the distance between $x = (0, 0)$ and $y = (1, 1)$.

In [49]:
```
def distance(x, y):

   return math.sqrt((y[1]-x[1])**2 + (y[0]-x[0])**2)

print(distance((0,0),(1,1)))
```

```
1.4142135623730951
```

Exercise 2d

- Write a function `in_circle(x, origin)` that determines whether a point in a two dimensional plane falls within a unit circle surrounding a given origin.
  - Your function should return a boolean `True` if the distance between `x` and `origin` is less than 1 and `False` otherwise.
  - Use `distance(x, y)` as defined in 2c.
- Use your function to determine whether the point (1,1) lies within the unit circle centered at (0,0).

In [55]:
```
def in_circle(x, origin = [0,0]):
  # Define your function here!
  if distance(x, origin) < 1:
    return True
  else: return False

in_circle((1, 1))
```

Out[55]:
```
False
```

Exercise 2e

- Create a list `inside` of `R=10000` booleans that determines whether or not a point falls within the unit circle centered at `(0,0)`.
  - Use the `rand` function from 2b to generate `R` randomly located points.
  - Use the function `in_circle` to test whether or not a given pint falls within the unit circle.
- Find the proportion of points that fall within the circle by summing all `True` values in the `inside` list; then divide the answer by `R` to obtain a proportion.
- Print your answer. This proportion is an estimate of the ratio of the two areas!

In [56]:
```
random.seed(1)

R = 10000
T = 0

inside = [False]*R
for i in range(R):
  inside[i] = in_circle((rand(), rand()))
```

```
  if inside[i] == True: T += 1

T/R
```

Out[56]:
```
0.779
```

Exercise 2f

- Find the difference between your estimate from part 2e and `math.pi / 4`. Note: `inside` and `R` are defined as in Exercise 2e.

In [57]:
```
# write your code here!
math.pi/4 - T/R
```

Out[57]:
```
0.006398163397448253
```

Exercise 3

A list of numbers representing measurements obtained from a system of interest can often be noisy. One way to deal with noise to smoothen the values by replacing each value with the average of the value and the values of its neighbors.

Exercise 3a

- Write a function `moving_window_average(x, n_neighbors)` that takes a list `x` and the number of neighbors `n_neighbors` on either side of a given member of the list to consider.
- For each value in `x`, `moving_window_average(x, n_neighbors)` computes the average of the value and the values of its neighbors.
- `moving_window_average` should return a list of averaged values that is the same length as the original list.
- If there are not enough neighbors (for cases near the edge), substitute the original value for a neighbor for each missing neighbor.
- Use your function to find the moving window sum of `x=[0,10,5,3,1,5]` and `n_neighbors=1`.

In [63]:
```
def moving_window_average(x, n_neighbors=5):
    n = len(x)
    width = n_neighbors*2 + 1
    x = [x[0]]*n_neighbors + x + [x[-1]]*n_neighbors
    meanvalues = []
    for i in range(n_neighbors, n+n_neighbors):
        meanvalues.append(sum(x[i-n_neighbors:i+n_neighbors+1])/(n_neighbors*2+1))
    return meanvalues

x = [0,10,5,3,1,5]
sum(moving_window_average(x, 5))
```

Out[63]:
```
19.90909090909091
```

Exercise 3b

- Compute and store `R=1000` random values from 0-1 as `x`.

- Compute the moving window average for `x` for values of `n_neighbors` ranging from 1 to 9 inclusive.
- Store `x` as well as each of these averages as consecutive lists in a list called `Y`

In [67]:
```
random.seed(1)

R = 1000
x = []
Y = []
y = []
# write your code here!
for _ in range(R):
  x.append(random.uniform(0.0, 1.0))

Y.append(x)

for i in range(1, 10):
  n = moving_window_average(x, i)
  Y.append(n)

Y[5][9]
Y[1][9]
```

Out[67]:
```
0.31932405573870365
```

Exercise 3c

- For each list in `Y`, calculate and store the range (the maximum minus the minimum) in a new list ranges.
- Print your answer. As the window width increases, does the range of each list increase or decrease? Why do you think that is?

In [68]:
```
# write your code here!
for i in range(len(Y)):
  print(max(Y[i]) - min(Y[i]))
```

```
0.9973152343362711
0.9128390185520854
0.801645771909397
0.7137391224212468
0.6230146948375028
0.5042284086774562
0.5071013753101629
0.4590090496908159
0.44659549539083265
0.4433696944090051
```

In [ ]:
```

```