Exercise 1: Strings, Lists and Dictionaries

Name: Panagam Mohitha

1. Read the phrase “Quick Brown fox jumped over the lazy DOG”

- **Count the number of alphabets and words in the phrase**

In [1]:
```
test_string = "Quick Brown fox jumped over the lazy DOG"
print("The original string is : " + test_string)
res = len(test_string.split())
print("The number of words in string are : " + str(res))
print("The number of alphabets in string are : ", len(test_string))
```

```
The original string is : Quick Brown fox jumped over the lazy DOG
The number of words in string are : 8
The number of alphabets in string are :  40
```

- **Count the frequency of all alphabets in the phrase**

In [2]:
```
from collections import Counter
test_str = "Quick Brown fox jumped over the lazy DOG"
test_str1=test_str.lower()
res = Counter(test_str1)
print("Count of all characters is :\n "
      + str(res))
```

```
Count of all characters is :
 Counter({' ': 7, 'o': 4, 'e': 3, 'u': 2, 'r': 2, 'd': 2, 'q': 1, 'i': 1, 'c': 1,
'k': 1, 'b': 1, 'w': 1, 'n': 1, 'f': 1, 'x': 1, 'j': 1, 'm': 1, 'p': 1, 'v': 1,
't': 1, 'h': 1, 'l': 1, 'a': 1, 'z': 1, 'y': 1, 'g': 1})
```

- **Convert the first alphabet of all words to capital**

In [3]:
```
test_str = "Quick Brown fox jumped over the lazy DOG"
# printing original string
print("The original string is : " + str(test_str))

# Using upper() + string slicing
# Initial character upper case
res = test_str[0].upper() + test_str[1:]

# printing result
print("The string after uppercasing initial character : " + str(res))
```

```
The original string is : Quick Brown fox jumped over the lazy DOG
The string after uppercasing initial character : Quick Brown fox jumped over the l
azy DOG
```

In [4]:
```
s = "Quick Brown fox jumped over the lazy DOG"
print("String before:", s)
a = s.split()
res = []
for i in a:
    x = i[0].upper()+i[1:]
    res.append(x)
res = " ".join(res)
print("String after:", res)
```

```
String before: Quick Brown fox jumped over the lazy DOG
String after: Quick Brown Fox Jumped Over The Lazy DOG
```

- **Print the phrase with every alternate word in reverse**

In [5]:
```
def main(sentence, ignores):
    for phrase in ignores:
        reversed_phrase = ' '.join([word[::-1] for word in phrase.split()])
        sentence = sentence.replace(phrase, reversed_phrase)

    return ' '.join(word[::-1] for word in sentence.split())

print(main('The quick brown fox jumps over the lazy dog', ['quick','fox','over','la
```

```
ehT quick nworb fox spmuj over eht lazy god
```

2. Read the phrase “Everybody likes to eat pizzas and ice creams”

- **Separate all words into a list**

In [6]:
```
s="Everybody likes to eat pizzas and ice creams"
list=s.split()
list
```

Out[6]:
```
['Everybody', 'likes', 'to', 'eat', 'pizzas', 'and', 'ice', 'creams']
```

- **Create a dictionary with the words and length of each word**

In [7]:
```
s="everybody likes to eat pizzas and ice creams"
l=s.split()
d={}
for word in l:
    if(word[0] not in d.keys()):
        d[word[0]]=[]
        d[word[0]].append(word)
    else:
        if(word not in d[word[0]]):
          d[word[0]].append(word)
for k,v in d.items():
        print(k,":",v, len(d))
```

```
e : ['everybody', 'eat'] 7
l : ['likes'] 7
t : ['to'] 7
p : ['pizzas'] 7
a : ['and'] 7
i : ['ice'] 7
c : ['creams'] 7
```

- **Combine the phrases mentioned above with comma as a separator**

In [8]:
```
d.values()
```

Out[8]:
```
dict_values([['everybody', 'eat'], ['likes'], ['to'], ['pizzas'], ['and'], ['ic
e'], ['creams']])
```

3. Create a program to read a phrase and highlight the word which does not contain any vowel. Run this code on the phrases mentioned above.

In [13]:
```
def highlight_word_without_vowel(phrase):
    words = phrase.split()  # split phrase into a list of words
    for word in words:
        vowels = set('aeiouAEIOU')
        if not any(char in vowels for char in word):  # check if word contains any
            print(f"\033[1;31m{word}\033[0m", end=" ")  # highlight the word in re
        else:
            print(word, end=" ")

phrase ="Everybody likes to eat pizzas and ice creams"
highlight_word_without_vowel(phrase)
phrase ="The quick brown fox jumps over the lazy dog"
highlight_word_without_vowel(phrase)
```

```
Everybody likes to eat pizzas and ice creams The quick brown fox jumps over the la
zy dog
```

4. Create the sum of the infinite series X = ½ + ¼ + …… using while loop with convergence criteria for error <0.0001

In [6]:
```
error = 1
sum = 0
n = 1

while error >= 0.0001:
    term = 1/(2**n)
    sum_prev = sum
    sum += term
    error = abs(sum - sum_prev)
    n += 1

print(f"The sum of the infinite series X = ½ + ¼ + …… is: {sum}")
```

```
The sum of the infinite series X = ½ + ¼ + …… is: 0.99993896484375
```

5. Store maximum possible decimals for e and pi and find the frequency of digits in the decimal places

In [5]:
```
import math

def find_digit_frequencies(decimal_places):
    e_decimals = str(math.e)[2:decimal_places+2]  # extract decimal places for e
    pi_decimals = str(math.pi)[2:decimal_places+2]  # extract decimal places for p

    e_digit_freq = {digit: e_decimals.count(digit) for digit in set(e_decimals)}
    pi_digit_freq = {digit: pi_decimals.count(digit) for digit in set(pi_decimals)

    print(f"Digit frequencies for e decimals (up to {decimal_places} decimal places
    print(e_digit_freq)

    print(f"\nDigit frequencies for pi decimals (up to {decimal_places} decimal pl
    print(pi_digit_freq)

decimal_places = int(input("Enter the number of decimal places to calculate: "))
find_digit_frequencies(decimal_places)
```

```
Enter the number of decimal places to calculate: 50
Digit frequencies for e decimals (up to 50 decimal places):
{'4': 2, '1': 2, '9': 1, '2': 2, '5': 2, '0': 1, '7': 1, '8': 4}

Digit frequencies for pi decimals (up to 50 decimal places):
{'4': 1, '1': 2, '3': 2, '9': 3, '2': 1, '5': 3, '6': 1, '7': 1, '8': 1}
```

Exercise 2:Numpy

1. Creating a Sampling Mean Distribution

- Create an array of 10,000 random numbers from chi-square distribution. This we would call Population Sample (Hint : Use np.random.chisquare(df=5,size=10000) to generate numbers)
- Create permutation of the population sample created (Syntax : np.random.permutation(array))
- Create a Test sample using first 1000 values of the permuted population sample
- Calculated the mean of the Test Sample and store it in a list
- Loop last 3 steps 10,000 times

In [11]:
```
import numpy as np
import matplotlib.pyplot as plt
```

In [13]:
```
pop_sample= np.random.chisquare(df=5,size=10000)
p= np.random.permutation(pop_sample)
```

In [14]:
```
mean_pop=pop_sample.mean()
mean_pop
```

Out[14]:
```
5.075839554636029
```

In [15]:
```
test_sample = np.random.choice(p, 1000, replace=True)
```

In [16]:
```
mean_p=test_sample.mean()
mean_p
```

Out[16]:
```
4.951224896269169
```

In [17]:
```
sample_props = []
for _ in range(10000):
    sample = np.random.choice(pop_sample, 5, replace=True)
    sample_props.append(sample.mean())
```

In [18]:
```
sample_props = np.array(sample_props)
sample_props.mean()
```

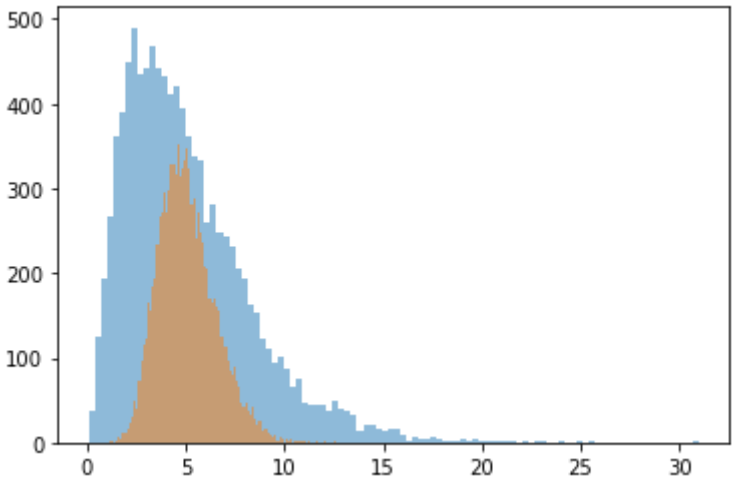
Out[18]:
```
5.060233251397004
```

2. Visualizing Sampling Mean Distribution and Population Distribution

- **Type: import matplotlib.pyplot as plt**
- **In next line, type %pylab inline**
- **Plotting Population Distribution: plt.hist(bins=100, Array of chisq distribution)**
- **Plotting Sample Mean Distribution: plt.hist(bins=100, Array of Sampling Means)**

In [19]:
```
 import matplotlib.pyplot as plt
%matplotlib inline
```

In [20]:
```
plt.hist(pop_sample, alpha=.5, bins=100);
plt.hist(sample_props, alpha=.5, bins=100);
```



Exercise 3: Introduction to Titanic Dataset

1. Read the Titanic Dataset and write a code to extract values of third last and second last column. (Assume you do not know the name of the variables)

In [21]:
```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
import scipy.stats as stats
from scipy.stats import chisquare,chi2_contingency
from scipy.stats import ttest_ind
from scipy.stats import f_oneway
from sklearn.preprocessing import LabelEncoder
import warnings
warnings.filterwarnings("ignore")
import copy
```

In [22]:
```
titanic = pd.read_csv('C:\\Users\\Mohitha Panagam\\Downloads\\Refresher_Exercises\
```

In [23]:
```
data_last_n = titanic.iloc[:, -3:-1]          # Select last columns
print(data_last_n)
```

```
        Fare Cabin
0     7.2500   NaN
1    71.2833   C85
2     7.9250   NaN
3    53.1000  C123
4     8.0500   NaN
..       ...   ...
886  13.0000   NaN
887  30.0000   B42
888  23.4500   NaN
889  30.0000  C148
890   7.7500   NaN

[891 rows x 2 columns]
```

2. Lst = [‘Embarked’,’ Sex’,’ Pclass ‘] Print the value counts of these variables from the DataFrame (Hint: Take care of the whitespaces in variable name !)

In [24]:
```
titanic[["Embarked", "Sex", "Pclass"]].value_counts()
```

Out[24]:
```
Embarked  Sex     Pclass
S         male    3         265
                  2          97
          female  3          88
          male    1          79
          female  2          67
                  1          48
C         female  1          43
          male    3          43
                  1          42
Q         male    3          39
          female  3          33
C         female  3          23
          male    2          10
          female  2           7
Q         female  2           2
          male    2           1
          female  1           1
          male    1           1
dtype: int64
```

3. Create a code to find which variables of Titanic Dataset are categorical:

- **Categorical variables are generally character variables**
- **If a Numeric variable (dtype = ‘int64’ or ‘float64’), then it can be categorical if number of categories of the variable are < 5**

In [25]:
```
titanic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

4. In a manner similar to fill rate, calculate the missing rate of all variables

In [26]:
```
titanic.isnull().sum() * 100 / len(titanic)
```

Out[26]:
```
PassengerId     0.000000
Survived        0.000000
Pclass          0.000000
Name            0.000000
Sex             0.000000
Age            19.865320
SibSp           0.000000
Parch           0.000000
Ticket          0.000000
Fare            0.000000
Cabin          77.104377
Embarked        0.224467
dtype: float64
```