Name : SK Shivaanee

Roll no : 2310257

Section : S02 **Assignment 8 : Programs using Lists**

Aim: To learn python programming using lists by forming expressions and statements involving reading and printing the data appropriately for the given specification.

Solve the following problems using Lists in Python (CO4, K3):

Question 1: There are m students who appeared for JEE exam and n students who appeared for the NEET exam. Each student is assigned a unique ID. Consider the list JEE which has student ID and his/her JEE score in percentile. Similarly, the NEET list consists of the student ID with the NEET score of 720.

JEE=[[100,67],[101,84],[103,82],[104,95],[105,93],[106,45],[108,55]]
NEET=[[100,670],[102,710],[104,700],[105,688],[106,250],[108,478],[109,98], [110,733]]

Assume the qualifying percentile for this year is 60 for JEE and 300 for the NEET exam.

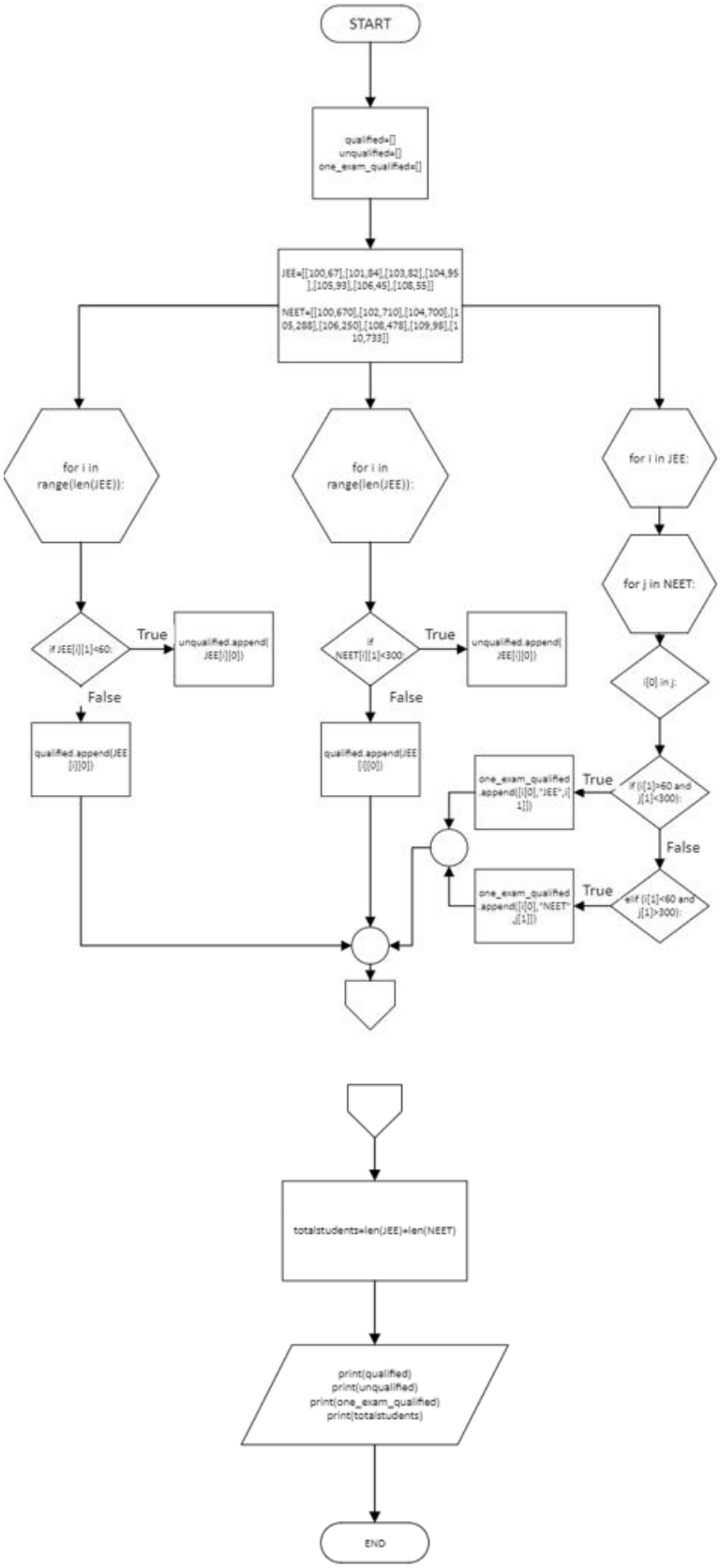
a. Create a list of students who have qualified in both exams.

b. Create a list of students who have not qualified in both exams.

c. Create a list of students who have qualified in any one exam. The list should comprise of student ID, exam type, and the score.

d. Find the total number of students who appeared for the qualifying exams this year.

Flowchart:


START
qualified=[]
unqualified=[]
one_exam_qualified=[]
JEE=[[100,67],[101,84],[103,82],[104,95],[105,93],[106,45],[108,55]]
NEET=[[100,670],[102,710],[104,700],[105,288],[106,290],[108,478],[109,98],[110,733]]
for i in range(len(JEE)):
for i in range(len(JEE)):
for i in JEE:
for j in NEET:
if JEE[i][1]<60:
True
unqualified.append(JEE[i][0])
False
qualified.append(JEE[i][0])
if NEET[i][1]<300:
True
unqualified.append(JEE[i][0])
False
qualified.append(JEE[i][0])
i[0] in j:
if (i[1]>60 and j[1]<300):
True
one_exam_qualified.append([i[0],"JEE",i[1]])
False
elif (i[1]<60 and j[1]>300):
True
one_exam_qualified.append([i[0],"NEET",i[1]])
totalstudents=len(JEE)+len(NEET)
print(qualified)
print(unqualified)
print(one_exam_qualified)
print(totalstudents)
END

Source code:

```
def qualifying_students(jee_list, neet_list, jee_threshold, neet_threshold):
    qualified_both = []
    not_qualified_both = []
    qualified_either = []

    for jee_student in jee_list:
        student_id, jee_score = jee_student
        for neet_student in neet_list:
            if student_id == neet_student[0]:
                neet_score = neet_student[1]
                if jee_score >= jee_threshold and neet_score >= neet_threshold:
                    qualified_both.append(student_id)
                else:
                    not_qualified_both.append(student_id)
                qualified_either.append((student_id, 'JEE' if jee_score >=
                    jee_threshold else 'NEET', max(jee_score, neet_score)))

    total_students = len(jee_list) + len(neet_list)

    print("a. Students qualified in both exams:", qualified_both)
    print("b. Students not qualified in both exams:", not_qualified_both)
    print("c. Students qualified in any one exam:", qualified_either)
    print("d. Total number of students who appeared for qualifying exams:",
        total_students)
```

```
# Example data
JEE = [[100, 67], [101, 84], [103, 82], [104, 95], [105, 93], [106, 45], [108,
    55]]
NEET = [[100, 670], [102, 710], [104, 700], [105, 688], [106, 250], [108, 478],
    [109, 98], [110, 733]]

# Qualifying thresholds
JEE_THRESHOLD = 60
NEET_THRESHOLD = 300

# Call the function with the provided data
qualifying_students(JEE, NEET, JEE_THRESHOLD, NEET_THRESHOLD)
```

Output:

```
a. Students qualified in both exams: [100, 104, 105]
b. Students not qualified in both exams: [106, 108]
c. Students qualified in any one exam: [(100, 'JEE', 670), (104, 'JEE', 700), (105,
    'JEE', 688), (106, 'NEET', 250), (108, 'NEET', 478)]
d. Total number of students who appeared for qualifying exams: 15
>
```

Question 2: **Consider the three different lists namely movie_title, movie_genre, and movie_rating. The movie_title contains the names of movies, and movie_genre contains the types of movies as action, comedy, horror, drama, and science/fiction. The movie_rating falls in the range 1-10 (both inclusive and float).**

a. **Create a movie database using the above 3 lists to store the information for 10 movies. Ex: [[movie_title1, movie_genre1, movie_rating1], ….[movie_title n, movie_genre n, movie_rating n]]**

b. Create a movie recommender system that suggests top k movies based on the ratings within the genre.

c. Search for the details of the given movie name. d. Create 5 different lists of movie names for 5 different genres. Ex: action=[list…], comedy=[list……], …..

Source Code:

```
class MovieDatabase:
    def __init__(self):
        self.movie_title = []
        self.movie_genre = []
        self.movie_rating = []

    def add_movie(self, title, genre, rating):
        self.movie_title.append(title)
        self.movie_genre.append(genre)
        self.movie_rating.append(rating)

    def movie_recommender(self, genre, top_k):
        recommendations = []
        for title, g, rating in zip(self.movie_title, self.movie_genre, self
            .movie_rating):
            if g == genre:
                recommendations.append((title, rating))
        recommendations.sort(key=lambda x: x[1], reverse=True)
        return recommendations[:top_k]

    def search_movie(self, movie_name):
        for title, genre, rating in zip(self.movie_title, self.movie_genre, self
            .movie_rating):
            if title.lower() == movie_name.lower():
                return f"Movie: {title}, Genre: {genre}, Rating: {rating}"
        return "Movie not found in the database."
```

```
    def get_movies_by_genre(self):
        genres = set(self.movie_genre)
        genre_lists = {genre: [] for genre in genres}
        for title, genre in zip(self.movie_title, self.movie_genre):
            genre_lists[genre].append(title)
        return genre_lists

# Example usage
movie_db = MovieDatabase()

# Add movies to the database
movie_db.add_movie("Movie1", "Action", 8.5)
movie_db.add_movie("Movie2", "Comedy", 7.2)
movie_db.add_movie("Movie3", "Horror", 6.8)
# Add more movies as needed...

# Movie Recommender System
genre_to_recommend = "Action"
top_k_recommendations = 3
recommendations = movie_db.movie_recommender(genre_to_recommend,
    top_k_recommendations)
print(f"Top {top_k_recommendations} {genre_to_recommend} movies based on ratings:
    {recommendations}")

# Search for a movie
search_movie_name = "Movie2"
```

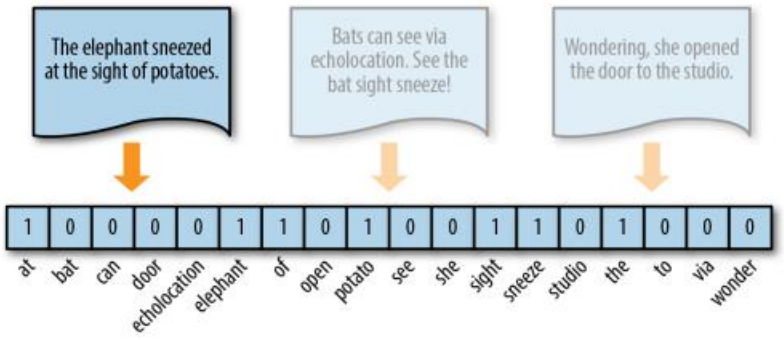
```
search_result = movie_db.search_movie(search_movie_name)
print(search_result)

# Get lists of movie names for each genre
movies_by_genre = movie_db.get_movies_by_genre()
for genre, movie_list in movies_by_genre.items():
    print(f"{genre}: {movie_list}")
```

Output:

```
Top 3 Action movies based on ratings: [('Movie1', 8.5)]
Movie: Movie2, Genre: Comedy, Rating: 7.2
Horror: ['Movie3']
Comedy: ['Movie2']
Action: ['Movie1']
```

Question 3:

Consider 5 documents where each document contains at least 2 sentences. These documents need to be vectorized into one-hot vector by constructing a bag-of-words (unique words from all the documents). Below is an example showing the one-hot-vector for the 1st document using the BoW.



a. Construct the bag-of-words (***BoW***) from the 5 documents.
b. Create one-hot-vector for each document - which is a list containing the presence of words in the document or not (1 if present, 0 otherwise)
c. Compute the similarity score between any given two documents A and B, using the below cosine-similarity formula, where *i* is the index of elements in the list.

$$\frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

Source Code:

```python
import re
import math

def preprocess_text(text):
    # Convert to lowercase and remove non-alphabetic characters
    cleaned_text = re.sub(r'[^a-zA-Z\s]', '', text.lower())
    return cleaned_text

def construct_bag_of_words(documents):
    words_set = set()
    for document in documents:
        cleaned_text = preprocess_text(document)
        words_set.update(cleaned_text.split())
    return list(words_set)

def create_one_hot_vector(document, bag_of_words):
    cleaned_text = preprocess_text(document)
    one_hot_vector = [1 if word in cleaned_text.split() else 0 for word in
        bag_of_words]
    return one_hot_vector

def cosine_similarity_score(vector_a, vector_b):
    dot_product = sum(a * b for a, b in zip(vector_a, vector_b))
    magnitude_a = math.sqrt(sum(a**2 for a in vector_a))
    magnitude_b = math.sqrt(sum(b**2 for b in vector_b))
    similarity = dot_product / (magnitude_a * magnitude_b) if (magnitude_a *
```

```
        magnitude_b) != 0 else 0
    return similarity

# Sample documents
documents = [
    "Document 1 contains some sample sentences.",
    "This is the second document for vectorization.",
    "The third document has unique words as well.",
    "Document number four is included in our list.",
    "This is the fifth and final document."
]

# a. Construct the bag-of-words (BoW)
bag_of_words = construct_bag_of_words(documents)

# b. Create one-hot-vector for each document
one_hot_vectors = [create_one_hot_vector(document, bag_of_words) for document in
    documents]

# c. Compute the similarity score between any two documents (A and B)
for i in range(len(documents)):
    for j in range(i + 1, len(documents)):
        similarity_score = cosine_similarity_score(one_hot_vectors[i],
            one_hot_vectors[j])
        print(f"Similarity between Document {i+1} and Document {j+1}:
            {similarity_score:.4f}")
```

Output:

```
Similarity between Document 1 and Document 2: 0.1690
Similarity between Document 1 and Document 3: 0.1581
Similarity between Document 1 and Document 4: 0.1581
Similarity between Document 1 and Document 5: 0.1690
Similarity between Document 2 and Document 3: 0.2673
Similarity between Document 2 and Document 4: 0.2673
Similarity between Document 2 and Document 5: 0.5714
Similarity between Document 3 and Document 4: 0.1250
Similarity between Document 3 and Document 5: 0.2673
Similarity between Document 4 and Document 5: 0.2673
```

Additional problems using Lists for practice:

Question 1: Write a function that accepts two positive integers a and b (a is smaller than b) and returns a list that contains all the odd numbers between a and b (including a and including b if applicable) in descending order.

PuesdoCode:

INPUT a

INPUT b

IF a>b THEN

    a,b=b,a

ENDIF

DEFINE listodd(a,b):

    l=[i FOR i in range(a,b+1) IF i%2!=0]

    RETURN l

DISPLAY listodd(a,b)

Source code:

```
def odd_numbers_descending(a, b):
    # Ensure a is smaller than b
    if a >= b:
        raise ValueError("a should be smaller than b")

    # Generate a list of odd numbers between a and b
    odd_numbers = [num for num in range(b, a - 1, -1) if num % 2 != 0]

    return odd_numbers

# Example usage
try:
    a_value = int(input("Enter the smaller positive integer (a): "))
    b_value = int(input("Enter the larger positive integer (b): "))

    result = odd_numbers_descending(a_value, b_value)

    if result:
        print(f"The odd numbers between {a_value} and {b_value} (inclusive) in
            descending order are: {result}")
    else:
        print("There are no odd numbers in the specified range.")

except ValueError:
    print("Invalid input. Please enter valid positive integers.")
```

Output:

```
Enter the smaller positive integer (a): 3
Enter the larger positive integer (b): 9
The odd numbers between 3 and 9 (inclusive) in descending order are: [9, 7, 5, 3]
```

Question 2: **Two words are anagrams if you can rearrange the letters from one to spell the other. Write a function called is_anagram(s1,s2) that takes two strings and returns True if they are anagrams: [Do not use dictionary data type or in-built methods]**

Hint: Consider the anagrams, Listen = Silent The word Listen can be translated into list of frequency of occurrences of each letter in English as follows: ['e',1,'i',1,'l',1,'n',1,'s',1,'t',1]. For anagrams, the frequency list of both the words are same. Hence the frequency list of Silent is also ['e',1,'i',1,'l',1,'n',1,'s',1,'t',1]. Other anagrams: Schoolmaster = The classroom, A gentleman = Elegant man

Source code:

```
def count_frequency(s):
    # Count the frequency of each character in the string
    frequency_list = []
    for char in s:
        if char.isalpha():
            char = char.lower()
            if char not in frequency_list:
                frequency_list.append(char)
            else:
                continue
    return frequency_list
def sorting(lst):
    lst1=[]
    for i in range(97,123):
        for j in range(0,len(lst)):
            if ord(lst[j])==i:
                lst1.append(lst[j])
    return lst1

def is_anagram(s1, s2):
    # Check if the frequency lists of characters are the same
    freq_list_s1 = sorting(count_frequency(s1))
    freq_list_s2 = sorting(count_frequency(s2))

    return freq_list_s1 == freq_list_s2
word1=input("Enter word 1 ")
```

```
word2=input("Enter word 2 ")
result1 = is_anagram(word1, word2)
print(f"{word1} and {word2} are anagrams: {result1}")
```

Output:

```
Enter word 1 listen
Enter word 2 silent
listen and silent are anagrams: True
```

```
Enter word 1 dog
Enter word 2 cat
dog and cat are anagrams: False
```

Question 3: Write a function called chop that takes a list, modifies it by removing the first and last elements, and returns None.

For example:

>>>t = [1, 2, 3, 4]

>>>chop(t)

>>>t [2, 3]

Source code:

```
def chop(lst):
    if len(lst) >= 2:
        # Remove the first and last elements
        del lst[0]
        del lst[-1]

# Example usage
t = []
n=int(input("Enter the number of elements"))
for i in range(0,n):
    m=int(input(f"Enter the element number {i} "))
    t.append(m)

chop(t)
print(t)
```

Output:

```
Enter the number of elements5
Enter the element number 0 1
Enter the element number 1 2
Enter the element number 2 3
Enter the element number 3 4
Enter the element number 4 5
[2, 3, 4]
```

Question 4: Write a function sorted that takes a list of strings as a parameter and sorts the elements in a lexicographical order.

Source code:

```python
def sorted_strings(string_list):
    # Sort the list of strings in lexicographical order
    sorted_list = sorted(string_list)
    return sorted_list

n=int(input("Enter the number of elements "))
input_strings =[]
for i in range(n):
    m=input("Enter the element")
    input_strings.append(m)

result = sorted_strings(input_strings)

print("Original list:", input_strings)
print("Sorted list:", result)
```

Output:

```
Enter the number of elements 5
Enter the element dog
Enter the element bag
Enter the element ant
Enter the element egg
Enter the element cot
Original list: ['dog', 'bag', 'ant', 'egg', 'cot']
Sorted list: ['ant', 'bag', 'cot', 'dog', 'egg']
```

Question 5: The second year of engineering offers the following subjects in two departments. CSE = ["DBMS", "ADC", "SE", "DSD", "DS", "OOP"] IT = ["OS", "PQT", "DS", "SE", "OOP", "DBMS"] Write a program to create subject list for the two departments.

1. Number and names of subjects in CSE.

2. Number and names of subjects in IT.

3. Number and names of subjects common in both CSE and IT.

4. Number and names of subjects for either CSE or IT, but not in both.

5. Number and names of all subjects in unique.

Source Code:

```python
# Subjects for CSE and IT departments
CSE = ["DBMS", "ADC", "SE", "DSD", "DS", "OOP"]
IT = ["OS", "PQT", "DS", "SE", "OOP", "DBMS"]

def print_subjects(subject_list, department_name):
    print(f"\nNumber and names of subjects in {department_name}:")
    print(f"Number of subjects: {len(subject_list)}")
    print(f"Subject names: {', '.join(subject_list)}")

# 1. Number and names of subjects in CSE.
print_subjects(CSE, "CSE")

# 2. Number and names of subjects in IT.
print_subjects(IT, "IT")

# 3. Number and names of subjects common in both CSE and IT.
common_subjects = list(set(CSE) & set(IT))
print_subjects(common_subjects, "Common Subjects")

# 4. Number and names of subjects for either CSE or IT, but not in both.
exclusive_subjects = list(set(CSE) ^ set(IT))
print_subjects(exclusive_subjects, "Exclusive Subjects")

# 5. Number and names of all subjects in unique.
all_subjects = list(set(CSE + IT))
print_subjects(all_subjects, "All Subjects")
```

Output:

```
Number and names of subjects in CSE:
Number of subjects: 6
Subject names: DBMS, ADC, SE, DSD, DS, OOP

Number and names of subjects in IT:
Number of subjects: 6
Subject names: OS, PQT, DS, SE, OOP, DBMS
Number and names of subjects in Common Subjects:
Number of subjects: 4
Subject names: DBMS, DS, SE, OOP

Number and names of subjects in Exclusive Subjects:
Number of subjects: 4
Subject names: ADC, OS, PQT, DSD

Number and names of subjects in All Subjects:
Number of subjects: 8
Subject names: OS, DS, DBMS, DSD, ADC, OOP, PQT, SE
```

Question 6: Given two sorted arrays nums1 and nums2 of size m and n respectively, return the median of the two sorted arrays

Source Code:

```python
def findMedianSortedArrays(nums1, nums2):
    merged = sorted(nums1 + nums2)
    total_length = len(merged)
    if total_length % 2 == 0:
        # If the total length is even, calculate the average of the middle two
            elements
        mid_right = total_length // 2
        mid_left = mid_right - 1
        median = (merged[mid_left] + merged[mid_right]) / 2
    else:
        # If the total length is odd, return the middle element
        median = merged[total_length // 2]
    return median

n=int(input("Enter the number of elements in array 1 "))
lst1=[]
for i in range(n):
    m=int(input(f"Enter the element {i+1} "))
    lst1.append(m)
n1=int(input("Enter the number of elements in array 1 "))
lst2=[]
for i in range(n1):
    m1=int(input(f"Enter the element {i+1} "))
    lst2.append(m1)
result = findMedianSortedArrays(lst1, lst2)
print(f"The median of the two sorted arrays is: {result}")
```

Output:

```
Enter the number of elements in array 1 3
Enter the element 1 1
Enter the element 2 3
Enter the element 3 5
Enter the number of elements in array 1 2
Enter the element 1 2
Enter the element 2 4
The median of the two sorted arrays is: 3
```

Question 7: A list of students' names who registered for the Python course. Perform the following operations (use inbuilt functions) and print the output:

i. A new student registered for the course.

ii. Count the number of students registered for the course.

Iii. Modify a name in the list.

iv. Sort the name list.

v. Insert a new student name in a given position.

vi. Search for a student.

vii. Remove a given name from the list.

Source Code:

```
# Initial list of students' names
students = ["Alice", "Bob", "Charlie", "David", "Eva"]

# i. A new student registered for the course.
new_student = "Frank"
students.append(new_student)
print(f"i. A new student registered for the course: {students}")

# ii. Count the number of students registered for the course.
num_students = len(students)
print(f"ii. Number of students registered for the course: {num_students}")

# iii. Modify a name in the list.
old_name = "Bob"
new_name = "Bobby"
if old_name in students:
    index = students.index(old_name)
    students[index] = new_name
print(f"iii. Modified name list: {students}")

# iv. Sort the name list.
students.sort()
print(f"iv. Sorted name list: {students}")

# v. Insert a new student name in a given position.
new_student_at_position = "Grace"
```

```python
position_to_insert = 2
students.insert(position_to_insert, new_student_at_position)
print(f"v. Inserted a new student at position {position_to_insert}: {students}")

# vi. Search for a student.
search_student = "David"
if search_student in students:
    print(f"vi. {search_student} is found in the list.")
else:
    print(f"vi. {search_student} is not found in the list.")

# vii. Remove a given name from the list.
name_to_remove = "Charlie"
if name_to_remove in students:
    students.remove(name_to_remove)
    print(f"vii. Removed {name_to_remove} from the list: {students}")
else:
    print(f"vii. {name_to_remove} is not in the list.")
```

Output:

```
i. A new student registered for the course: ['Alice', 'Bob', 'Charlie', 'David', 'Eva',
    'Frank']
ii. Number of students registered for the course: 6
iii. Modified name list: ['Alice', 'Bobby', 'Charlie', 'David', 'Eva', 'Frank']
iv. Sorted name list: ['Alice', 'Bobby', 'Charlie', 'David', 'Eva', 'Frank']
v. Inserted a new student at position 2: ['Alice', 'Bobby', 'Grace', 'Charlie', 'David',
    'Eva', 'Frank']
vi. David is found in the list.
vii. Removed Charlie from the list: ['Alice', 'Bobby', 'Grace', 'David', 'Eva', 'Frank']
```

Learning outcome:

1. Reading inputs / Printing the result
2. Using appropriate datatypes for the given input
3. Variable assignment
4. Converting the formula into python expressions

Result: Thus I learned to implement a simple problems in Python and solve the same using Lists.