#### **Problem Statements**

#### Event Attendance:

#### A school has two events, and the attendees are stored in two sets. Create 2 sets with names of students. Write a program to:

* Find the students who attended both events.
* Find the students who attended only one of the events.
* Find all students who attended at least one event. ( use ‘|’ operator)

#### **Solution :**

set1={"Manjesh","Rajesh","Abhishek","Raghav"}

set2={"Manjesh", "Abhishek","Ram","Ravi"}

# To Find the students who attended both events.

print("the students who attended both events are ", set1.intersection(set2))

# To Find the students who attended only one of the events.

print("the students who attended only one of the event are : ", set1.symmetric\_difference(set2))

# To Find all students who attended at least one event. ( use ‘|’ operator)

print("all students who attended at least one event are : ", set1 | set2)

#### **Out put:**

the students who attended both events are {'Manjesh', 'Abhishek'}

the students who attended only one of the event are : {'Rajesh', 'Ram', 'Raghav', 'Ravi'}

all students who attended at least one event are : {'Manjesh', 'Abhishek', 'Rajesh', 'Ram', 'Raghav', 'Ravi'}

#### 

#### Student Marks Calculation:

You are given a dictionary containing student names as keys and their marks as values.

students = {

"Rohan": 85,

"Spoorthi": 90,

"Aditi": 78,

"Tanya": 92

}

Write a program to:

* Find the student with the highest marks.
* Calculate the average marks for the class.
* Add a new student and their marks to the dictionary.

#### **Solution :**

students = {

    "Rohan": 85,

    "Spoorthi": 90,

    "Aditi": 78,

    "Tanya": 92

}

max =0

for s,m in students.items():

    if max < m :

        max = m

        student = s

print("The student having Highest marks is ",s , "and the marks is ", max)

sum = 0

n=0

for i in students.values():

    sum = sum + i

    n += 1

print("The average marks is = ", sum /n)

students["Manjesh"] = 92

print(students)

#### Output :

#### The student having Highest marks is Tanya and the marks is 92

#### The average marks is = 86.25

#### {'Rohan': 85, 'Spoorthi': 90, 'Aditi': 78, 'Tanya': 92, 'Manjesh': 92}

#### 

#### Sentence Analysis:

Given a sentence, write a program to:

* Count the number of vowels and consonants.
* Find the longest word in the sentence.
* Reverse the sentence.

#### **Solution :**

vowels="AEIOUaeiou"

space=" "

count\_v = 0

count\_c = 0

sentence="My name is Manjesh"

for i in sentence:

    if i == space:

        continue

    elif i in vowels:

        count\_v +=1

    else:

        count\_c += 1

print("Number of Vowels is = ", count\_v)

print("Number of consonants is = ", count\_c)

biggest = ""

sen = sentence.split()

for  i in sen:

    if len(biggest) < len(i):

        biggest = i

print("The largest term is : ", biggest)

str = ""

n = 0

reverse= list()

for i in range(len(sen)-1,-1,-1):

    reverse.append(sen[i])

    str = str + reverse[n] + " "

    n+=1

print("The reverse of the sentence is ",str)

Output :

Number of Vowels is = 5

Number of consonants is = 10

The largest term is : Manjesh

The reverse of the sentence is Manjesh is name My

1. You are given a list of integers , sort the list based on the frequency of the number of occurrences of the elements. Take input of your choice.

#### **Solution :**

1. You went shopping with your family at a local supermarket. Each family member picked up different items independently, creating their own shopping lists. To avoid buying duplicate items, you need to analyze these lists and calculate the final billing amount.

#### **Solution :**

list1=["shoe","bag","bottle"]

list2=["bag","paper", "car"]

list3=["Note","belt"]

set1=set(list1)

set2=set(list2)

set3=set(list3)

print("The final list will be ", list(set1 | set2 | set3))

Output:

The final list will be ['paper', 'bag', 'car', 'shoe', 'Note', 'bottle', 'belt']

1. You work at a movie theater that keeps track of daily bookings. Each booking record contains the customer's name, selected movie, and seat number in a specific format. The theater needs to analyze these booking records to manage seating and prevent duplicate bookings. Take input from the user.

Input format - "CustomerName-MovieName-SeatNumber"

Hints:

* Consider using string methods like split(), replace()
* Dictionary and set data structures may be used for tracking duplicates

#### **Solution :**

# Pre-existing dictionary of bookings

bookings = {

"MovieA": {"A1", "A2", "A3"},

"MovieB": {"B1", "B2"},

"MovieC": {"C1", "C2", "C3"}

}

def process\_bookings(bookings):

while True:

# Input format: "CustomerName-MovieName-SeatNumber"

user\_input = input("Enter booking details (or type 'exit' to stop): ").strip()

if user\_input.lower() == "exit":

break

try:

# Split the input string

customer\_name, movie\_name, seat\_number = user\_input.split("-")

except ValueError:

print("Invalid input format. Please use the format 'CustomerName-MovieName-SeatNumber'.")

continue

# Ensure movie name exists in the dictionary

if movie\_name not in bookings:

bookings[movie\_name] = set() # Initialize new movie with an empty set

# Check for duplicate bookings

if seat\_number in bookings[movie\_name]:

print(f"Error: Seat {seat\_number} for movie '{movie\_name}' is already booked.")

else:

bookings[movie\_name].add(seat\_number) # Add seat to the set

print(f"Booking successful for {customer\_name} - Movie: {movie\_name}, Seat: {seat\_number}")

# Display final bookings

print("\nFinal Bookings:")

for movie, seats in bookings.items():

print(f"Movie: {movie}, Booked Seats: {', '.join(seats)}")

# Process the pre-existing bookings

process\_bookings(bookings)

1. Write a Python program that takes two strings and checks if they are anagrams of each other. Ignore spaces and punctuation, and consider the comparison to be case-insensitive. For example, "Astronomer" and "Moon starer" should be identified as anagrams.

#### **Solution :**

def are\_anagrams(string1, string2):

    # Remove spaces and punctuation, and convert to lowercase

    clean\_string1 = ''.join(char.lower() for char in string1 if char.isalnum())

    clean\_string2 = ''.join(char.lower() for char in string2 if char.isalnum())

    # Check if sorted characters of both strings are equal

    return sorted(clean\_string1) == sorted(clean\_string2)

# Input two strings

string1 = input("Enter the first string: ")

string2 = input("Enter the second string: ")

# Check if the strings are anagrams

if are\_anagrams(string1, string2):

    print(f'"{string1}" and "{string2}" are anagrams.')

else:

print(f'"{string1}" and "{string2}" are not anagrams.')

Input:

Enter the first string: Astronomer

Enter the second string: Moon starer

Output:

"Astronomer" and "Moon starer" are anagrams.

1. Write a Python program that takes a string as input and finds the first non-repeating character using a dictionary. If a non-repeating character is found, print it; else, print an appropriate message.

#### **Solution :**

def first\_non\_repeating\_char(string):

    # Create a dictionary to store character counts

    char\_count = {}

    # Iterate through the string and count occurrences of each character

    for char in string:

        if char not in char\_count:

            char\_count[char] = 1

        else:

            char\_count[char] += 1

    # Iterate through the string again to find the first non-repeating character

    for char in string:

        if char\_count[char] == 1:

            return char

    return None  # Return None if no non-repeating character is found

# Input a string

input\_string = input("Enter a string: ")

# Find the first non-repeating character

result = first\_non\_repeating\_char(input\_string)

# Output the result

if result:

    print(f"The first non-repeating character is: {result}")

else:

    print("No non-repeating character found.")

Input:

Enter a string: swiss

Output:

The first non-repeating character is: w

1. Password Validator

Define a simple password validator. The password must:

* Be 8 characters long
* Must contain 1 Upper case character, 1 lower case character and 1 number
* If valid return valid password if not print invalid

#### **Solution :**

def password\_validator(password):

    # Check if the password is at least 8 characters long

    if len(password) < 8:

        print("Invalid password: Must be at least 8 characters long.")

        return False

    # Initialize flags for uppercase, lowercase, and number

    has\_upper = False

    has\_lower = False

    has\_digit = False

    # Check each character in the password

    for char in password:

        if char.isupper():

            has\_upper = True

        elif char.islower():

            has\_lower = True

        elif char.isdigit():

            has\_digit = True

    # Check if all conditions are met

    if has\_upper and has\_lower and has\_digit:

        return "Valid password"

    else:

        print("Invalid password: Must contain at least 1 uppercase letter, 1 lowercase letter, and 1 number.")

        return False

# Input a password

password = input("Enter a password: ")

# Validate the password

result = password\_validator(password)

# Print the result

if result:

    print(result)

Input:

Enter a password: Pass1234

Output:

Valid password

Input:

Enter a password: pass123

Output:

Invalid password: Must be at least 8 characters long.

1. Isomorphic Strings

Given two strings s and t, determine if they are isomorphic. Two strings s and t are isomorphic if the characters in s can be replaced to get t. All occurrences of a character must be replaced with another character while preserving the order of characters. No two characters may map to the same character, but a character may map to itself.

#### **Solution :**

def are\_isomorphic(s, t):

    # If the strings have different lengths, they cannot be isomorphic

    if len(s) != len(t):

        return False

    # Dictionaries to store mappings from s to t and t to s

    mapping\_s\_to\_t = {}

    mapping\_t\_to\_s = {}

    # Iterate through characters of both strings

    for char\_s, char\_t in zip(s, t):

        # Check if there's already a mapping for char\_s in s to char\_t in t

        if char\_s in mapping\_s\_to\_t:

            if mapping\_s\_to\_t[char\_s] != char\_t:

                return False

        else:

            mapping\_s\_to\_t[char\_s] = char\_t

        # Check if there's already a mapping for char\_t in t to char\_s in s

        if char\_t in mapping\_t\_to\_s:

            if mapping\_t\_to\_s[char\_t] != char\_s:

                return False

        else:

            mapping\_t\_to\_s[char\_t] = char\_s

    return True

# Input two strings

s = input("Enter the first string: ")

t = input("Enter the second string: ")

# Check if the strings are isomorphic

if are\_isomorphic(s, t):

    print(f'"{s}" and "{t}" are isomorphic.')

else:

    print(f'"{s}" and "{t}" are not isomorphic.')

Input:

Enter the first string: egg

Enter the second string: add

Output:

"egg" and "add" are isomorphic.

1. Anonymous Feedback Aggregator

Scenario: Your company collects anonymous feedback from employees with a dictionary storing feedback themes as keys and a list of feedback messages as values.

Use the dictionary feedback = {

"Work Environment": ["Great work culture", "Need more team activities"],

"Salary": ["Fair pay, but bonuses are inconsistent"],

"Management": ["Leadership can improve", "Need more transparency"],

}

Write a program to:

* Add new feedback to the relevant theme.
* Identify the theme with the most feedback.

**Solution :**

feedback = {

    "Work Environment": ["Great work culture", "Need more team activities"],

    "Salary": ["Fair pay, but bonuses are inconsistent"],

    "Management": ["Leadership can improve", "Need more transparency"],

}

# Function to add new feedback to the relevant theme

def add\_feedback(theme, message):

    if theme in feedback:

        feedback[theme].append(message)

    else:

        feedback[theme] = [message]

# Function to identify the theme with the most feedback

def theme\_with\_most\_feedback(feedback\_dict):

    max\_theme = None

    max\_count = 0

    for theme, messages in feedback\_dict.items():

        if len(messages) > max\_count:

            max\_count = len(messages)

            max\_theme = theme

    return max\_theme, max\_count

# Example usage

# Adding new feedback

add\_feedback("Work Environment", "More flexible work hours")

add\_feedback("Salary", "Increase in base salary would be great")

# Identifying the theme with the most feedback

theme, count = theme\_with\_most\_feedback(feedback)

print(f"The theme with the most feedback is '{theme}' with {count} feedback messages.")

Output

The theme with the most feedback is 'Work Environment' with 3 feedback messages.