Rajalakshmi Engineering College

Name: Monish kumar v
Email: 240701333@rajalakshmi.edu.in
Roll no: 240701333
Phone: 9123501619
Branch: REC
Department: I CSE FD
Batch: 2028
Degree: B.E - CSE

Scan to verify results

NeoColab_REC_CS23221_Python Programming

REC_Python_Week 6_CY

Attempt : 1
Total Mark : 40
Marks Obtained : 40

Section 1 : Coding

1. Problem Statement

Bob, a data analyst, requires a program to automate the process of analyzing character frequency in a given text. This program should allow the user to input a string, calculate the frequency of each character within the text, save these character frequencies to a file named "char_frequency.txt," and display the results.

Input Format

The input consists of the string.

Output Format

The first line prints "Character Frequencies:".

The following lines print the character frequency in the format: "X: Y" where X is the character and Y is the count.

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: aaabbbccc
Output: Character Frequencies:
a: 3
b: 3
c: 3

Answer

```
from collections import Counter

def analyze_character_frequency(input_string):
    freq = Counter(input_string)
    output = ["Character Frequencies:"]
    for char, count in freq.items():
        output.append(f"{char}: {count}")
    with open("char_frequency.txt", "w") as file:
        file.write("\n".join(output))
    print("\n".join(output))
input_string = input()
analyze_character_frequency(input_string)
```

Status : Correct ***Marks : 10/10***

2. Problem Statement

Write a program to read the Register Number and Mobile Number of a student. Create user-defined exception and handle the following:

If the Register Number does not contain exactly 9 characters in the specified format(2 numbers followed by 3 characters followed by 4 numbers) or if the Mobile Number does not contain exactly 10 characters, throw an IllegalArgumentException. If the Mobile Number contains any character other than a digit, raise a NumberFormatException.If the Register

Number contains any character other than digits and alphabets, throw a NoSuchElementException.If they are valid, print the message ‘valid’ or else print an Invalid message.

Input Format

The first line of the input consists of a string representing the Register number.

The second line of the input consists of a string representing the Mobile number.

Output Format

The output should display any one of the following messages:

If both numbers are valid, print "Valid".

If an exception is raised, print "Invalid with exception message: ", followed by the specific exception message.

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: 19ABC1001
9949596920

Output: Valid

Answer

```
import re
class InvalidRegisterNumberException(Exception):
    pass
class InvalidMobileNumberException(Exception):
    pass
def validate_register_number(register_number):
    if len(register_number) != 9:
        raise InvalidRegisterNumberException("Register Number should have exactly 9 characters.")
    if not re.match(r'^\d{2}[A-Za-z]{3}\d{4}$', register_number):
        raise InvalidRegisterNumberException("Register Number should have the format: 2 numbers, 3 characters, and 4 numbers.")
    if not register_number.isalnum():
```

```
        raise InvalidRegisterNumberException("Register Number contains invalid
characters.")
def validate_mobile_number(mobile_number):
    if len(mobile_number) != 10:
        raise InvalidMobileNumberException("Mobile Number should have exactly
10 characters.")
    if not mobile_number.isdigit():
        raise InvalidMobileNumberException("Mobile Number should only contain
digits.")
def main():
    register_number = input().strip()
    mobile_number = input().strip()

    try:
        validate_register_number(register_number)
        validate_mobile_number(mobile_number)
        print("Valid")
    except (InvalidRegisterNumberException, InvalidMobileNumberException) as
e:
        print(f"Invalid with exception message: {e}")

if __name__ == "__main__":
    main()
```

Status : Correct **Marks : 10/10**

3. Problem Statement

Write a program to obtain the start time and end time for the stage event show. If the user enters a different format other than specified, an exception occurs and the program is interrupted. To avoid that, handle the exception and prompt the user to enter the right format as specified.

Start time and end time should be in the format 'YYYY-MM-DD HH:MM:SS'If the input is in the above format, print the start time and end time.If the input does not follow the above format, print "Event time is not in the format "

Input Format

The first line of input consists of the start time of the event.

The second line of the input consists of the end time of the event.

Output Format

If the input is in the given format, print the start time and end time.

If the input does not follow the given format, print "Event time is not in the format".

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 2022-01-12 06:10:00
2022-02-12 10:10:12

Output: 2022-01-12 06:10:00
2022-02-12 10:10:12

Answer

```
from datetime import datetime

def validate_event_time(event_time):
    try:
        # Attempt to parse the event time using the specified format
        parsed_time = datetime.strptime(event_time, "%Y-%m-%d %H:%M:%S")
        return parsed_time
    except ValueError:
        # If parsing fails, return None
        return None

def main():
    # Prompt the user for the start and end times
    start_time = input("")
    end_time = input("")

    # Validate the start and end times
    valid_start_time = validate_event_time(start_time)
    valid_end_time = validate_event_time(end_time)

    if valid_start_time and valid_end_time:
```

```
        # If both times are valid, print them
        print(f"{valid_start_time.strftime('%Y-%m-%d %H:%M:%S')}")
        print(f"{valid_end_time.strftime('%Y-%m-%d %H:%M:%S')}")
    else:
        # If either time is invalid, print an error message
        print("Event time is not in the format")

if __name__ == "__main__":
    main()
```

Status : Correct **Marks : 10/10**

4. Problem Statement

In the enchanted realm of Academia, you, the Academic Alchemist, are bestowed with a magical quill and a parchment to weave the grades of aspiring students into a tapestry of academic brilliance.

The mission is to craft a Python program that empowers faculty members to enter student grades for any two subjects, stores these magical grades in a mystical file, and then, with a wave of your virtual wand, calculates the GPA to unveil the true essence of academic achievement.

Input Format

The input format is a string representing the student's name, any two subjects, and corresponding grades.

After entering grades, they can type 'done' when prompted for the student's name.

Output Format

The output should display the (average of grades) calculated GPA with a precision of two decimal places.

The magical grades will be saved in a mystical file named "magical_grades.txt".

Refer to the sample output for format specifications.

Sample Test Case

Input: Alice
Math
95
English
88
done

Output: 91.50

Answer

```
def calculate_gpa(grades):
    return sum(grades) / len(grades)

def main():
    with open("magical_grades.txt", "w") as file:
        while True:
            # Prompt for student's name
            name = input()
            if name.lower() == 'done':
                break
            subjects = []
            grades = []
            for _ in range(2):
                subject = input()
                grade = float(input())
                if 0 <= grade <= 100:
                    subjects.append(subject)
                    grades.append(grade)
                else:
                    print("Grade must be between 0 and 100. Please re-enter.")
                    break
            else:
                gpa = calculate_gpa(grades)
                file.write(f"{name} {subjects[0]} {grades[0]} {subjects[1]} {grades[1]}
GPA: {gpa:.2f}\n")
                print(f"  {gpa:.2f}")

if __name__ == "__main__":
    main()
```

Status : Correct *Marks : 10/10*