Rajalakshmi Engineering College

Name: Mohamed Aaris
Email: 240701316@rajalakshmi.edu.in
Roll no: 240701316
Phone: 6383899392
Branch: REC
Department: I CSE FC
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

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 is_valid_grade(grade_str):
    try:
        grade = float(grade_str)
        return 0 <= grade <= 100
    except ValueError:
        return False

with open("magical_grades.txt", "w") as file:
    while True:
        student_name = input()
        if student_name.lower() == 'done':
            break

        subject1 = input()
        grade1 = input()
        if not is_valid_grade(grade1):
            print("Invalid grade. Must be a number between 0 and 100.")
            continue
        subject2 = input()
        grade2 = input()
        if not is_valid_grade(grade2):
            print("Invalid grade. Must be a number between 0 and 100.")
            continue

        grade1 = float(grade1)
        grade2 = float(grade2)
        gpa = (grade1 + grade2) / 2

        # Write to mystical file
        file.write(f"{student_name},{subject1}:{grade1},{subject2}:{grade2},GPA:
{gpa:.2f}\n")

        # Reveal GPA
        print(f"{gpa:.2f}")
```

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

2. Problem Statement

Alex is creating an account and needs to set up a password. The program prompts Alex to enter their name, mobile number, chosen username, and desired password. Password validation criteria include:

Length between 10 and 20 characters.At least one digit.At least one special character from !@#$%^&amp;* set. Display "Valid Password" if criteria are met; otherwise, raise an exception with an appropriate error message.

Input Format

The first line of the input consists of the name as a string.

The second line of the input consists of the mobile number as a string.

The third line of the input consists of the username as a string.

The fourth line of the input consists of the password as a string.

Output Format

If the password is valid (meets all the criteria), it will print "Valid Password"

If the password is weak (fails any one or more criteria), it will print an error message accordingly.

Refer to the sample outputs for the formatting specifications.

Sample Test Case

Input: John
9874563210
john
john1#nhoj
Output: Valid Password

Answer

```
def validate_password(password):
    special_chars = "!@#$%^&*"

    if len(password) < 10 or len(password) > 20:
        raise ValueError("Should be a minimum of 10 characters and a maximum of 
20 characters")


    if not any(char.isdigit() for char in password):
        raise ValueError("Should contain at least one digit")


    if not any(char in special_chars for char in password):
        raise ValueError("It should contain at least one special character")

    return True


name = input()
mobile = input()
username = input()
password = input()
```

```
try:
   if validate_password(password):
      print("Valid Password")
except ValueError as ve:
   print(ve)
```

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

3. Problem Statement

Implement a program that checks whether a set of three input values can form the sides of a valid triangle. The program defines a function is_valid_triangle that takes three side lengths as arguments and raises a ValueError if any side length is not a positive value. It then checks whether the sum of any two sides is greater than the third side to determine the validity of the triangle.

Input Format

The first line of input consists of an integer A, representing side1.

The second line of input consists of an integer B, representing side2.

The third line of input consists of an integer C, representing side3.

Output Format

The output prints either "It's a valid triangle" if the input side lengths form a valid triangle,

or "It's not a valid triangle" if they do not.

If there is a ValueError, it should print "ValueError: <error_message>".

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: 3
4
5
Output: It's a valid triangle

Answer

```
def is_valid_triangle(a, b, c):
    if a <= 0 or b <= 0 or c <= 0:
        raise ValueError("Side lengths must be positive")
    return (a + b > c) and (a + c > b) and (b + c > a)

try:
    A = int(input())
    B = int(input())
    C = int(input())

    if is_valid_triangle(A, B, C):
        print("It's a valid triangle")
    else:
        print("It's not a valid triangle")

except ValueError as ve:
    print(f"ValueError: {ve}")
```

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

4. Problem Statement

A shopkeeper is recording the daily sales of an item for N days, where the price of the item remains the same for all days. Write a program to calculate the total sales for each day and save them in a file named sales.txt that can store the data for a maximum of 30 days. Then, read the file and display the total earnings for each day.

Note: Total Earnings for each day = Number of Items sold in that day × Price of the item.

Input Format

The first line of input consists of an integer N, representing the number of days.

The second line of input consists of N space-separated integers representing the number of items sold each day.

The third line of input consists of an integer M, representing the price of the item that is common for all N days.

Output Format

If the number of days entered exceeds 30 (N > 30), the output prints "Exceeding limit!" and terminates.

Otherwise, the code reads the contents of the file and displays the total earnings for each day on separate lines.

Contents of the file: The total earnings for N days, with each day's earnings appearing on a separate line.

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: 4
5 10 5 0
20

Output: 100
200
100
0

Answer

```
N = int(input())


if N > 30:
    print("Exceeding limit!")
else:

    items_sold = list(map(int, input().split()))

    M = int(input())


    with open("sales.txt", "w") as file:
        for count in items_sold:
            total = count * M
            file.write(f"{total}\n")

    with open("sales.txt", "r") as file:
        for line in file:
            print(line.strip())
```

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