Rajalakshmi Engineering College

Name: Kevin Joe S
Email: 241501085@rajalakshmi.edu.in
Roll no: 241501085
Phone: 9894620565
Branch: REC
Department: I AIML AD
Batch: 2028
Degree: B.E - AI & ML

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 main():
    while True:
        student_name = input().strip()
        if student_name.lower() == 'done':
            break
        subjects = []
        grades = []
        for _ in range(2):
            subject = input().strip()
            grade = float(input().strip())
            if grade < 0 or grade > 100:
                print("Grade must be between 0 and 100. Please enter again.")
                return
            subjects.append(subject)
            grades.append(grade)
        gpa = sum(grades) / len(grades)
        with open("magical_grades.txt", "a") as file:
            file.write(f"{student_name}: {subjects[0]} - {grades[0]}, {subjects[1]} - {grades[1]}\n")
```

```
    print(f"{gpa:.2f}")
if __name__ == "__main__":
    main()
```

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

```
class InvalidRegisterNumber(Exception):
    pass
class InvalidMobileNumber(Exception):
    pass
def validate_register_number(reg_number):
    if len(reg_number) != 9:
        raise InvalidRegisterNumber("Register Number should have exactly 9
characters.")
    if not (reg_number[0].isdigit() and reg_number[1].isdigit()):
        raise InvalidRegisterNumber("Register Number should have the format: 2
numbers, 3 characters, and 4 numbers.")
    if not (reg_number[2:5].isalpha() and reg_number[5:9].isdigit()):
        raise InvalidRegisterNumber("Register Number should have the format: 2
numbers, 3 characters, and 4 numbers.")
def validate_mobile_number(mobile_number):
    if len(mobile_number) != 10:
        raise InvalidMobileNumber("Mobile Number should have exactly 10
characters.")
    if not mobile_number.isdigit():
        raise InvalidMobileNumber("Mobile Number should only contain digits.")
def main():
    reg_number = input().strip()
    mobile_number = input().strip()
    try:
        validate_register_number(reg_number)
        validate_mobile_number(mobile_number)
        print("Valid")
    except (InvalidRegisterNumber, InvalidMobileNumber) as e:
        print(f"Invalid with exception message: {e}")
if __name__ == "__main__":
    main()
```

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

```
# You are using Python
def is_valid_triangle(a, b, c):
    if a <= 0 or b <= 0 or c <= 0:
        raise ValueError("Side lengths must be positive")
    if a + b > c and a + c > b and b + c > a:
        return True
    else:
        return False
```

```
def main():
    try:
        side1 = int(input().strip())
        side2 = int(input().strip())
        side3 = int(input().strip())
        if is_valid_triangle(side1, side2, side3):
            print("It's a valid triangle")
        else:
            print("It's not a valid triangle")
    except ValueError as e:
        print(f"ValueError: {e}")
if __name__ == "__main__":
    main()
```

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

```
def main():
    N = int(input().strip())
    if N > 30:
        print("Exceeding limit!")
        return
    items_sold = list(map(int, input().strip().split()))
    M = int(input().strip())
    total_earnings = [items * M for items in items_sold]
    with open("sales.txt", "w") as file:
        for earnings in total_earnings:
            file.write(f"{earnings}\n")
    with open("sales.txt", "r") as file:
        for line in file:
            print(line.strip())

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

Status : Correct *Marks : 10/10*