Rajalakshmi Engineering College

Name: ANIRUDH R
Email: 240701039@rajalakshmi.edu.in
Roll no: 240701039
Phone: 9363540767
Branch: REC
Department: I CSE FA
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 academic_alchemy():
    grades_archive = []
    while True:
        name = input().strip()
        if name.lower() == 'done':
            break
        subject1 = input().strip()
        grade1 = float(input())
        subject2 = input().strip()
        grade2 = float(input())
        if not (0 <= grade1 <= 100) or not (0 <= grade2 <= 100):
            print("Invalid grades! They must be between 0 and 100.")
            continue
        record = {
            'name': name,
            'subjects': [subject1, subject2],
            'grades': [grade1, grade2],
            'gpa': (grade1 + grade2) / 2
        }
        grades_archive.append(record)
        print(f"{record['gpa']:.2f}")
    with open("magical_grades.txt", "w") as tome:
        for record in grades_archive:
            entry = (f"{record['name']}: {record['subjects'][0]}={record['grades'][0]}, "
                    f"{record['subjects'][1]}={record['grades'][1]},
GPA={record['gpa']:.2f}\n")
            tome.write(entry)
academic_alchemy()
```

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

2. 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 record_sales():
    N = int(input())
    if N > 30:
        print("Exceeding limit!")
        return
    items_sold = list(map(int, input().split()))
    M = int(input())
    with open('sales.txt', 'w') as file:
        for sold in items_sold:
            total_earning = sold * M
            file.write(f"{total_earning}\n")
    with open('sales.txt', 'r') as file:
        for line in file:
            print(line.strip())
record_sales()
```

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 get_event_times():
    try:
        start_time = input()
        end_time = input()
        datetime.strptime(start_time, '%Y-%m-%d %H:%M:%S')
        datetime.strptime(end_time, '%Y-%m-%d %H:%M:%S')
        print(start_time)
        print(end_time)
    except ValueError:
        print("Event time is not in the format")
get_event_times()
```

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

4. 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(side1, side2, side3):
   if side1 <= 0 or side2 <= 0 or side3 <= 0:
      raise ValueError("Side lengths must be positive")
   return (side1 + side2 > side3) and (side1 + side3 > side2) and (side2 + side3 >
side1)
def main():
   try:
      side1 = int(input())
      side2 = int(input())
      side3 = int(input())
      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}")
main()
```

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