Rajalakshmi Engineering College

Name: Krishna Kumar
Email: 240701622@rajalakshmi.edu.in
Roll no:
Phone: null
Branch: REC
Department: I CSE FF
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

```
# You are using Python
total_grades = 0
count = 0

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

        subject1 = input()
        grade1 = input()
        subject2 = input()
        grade2 = input()

        # Validate grades are numeric and within 0-100
        try:
            g1 = float(grade1)
```

```
            g2 = float(grade2)
            if not (0 <= g1 <= 100) or not (0 <= g2 <= 100):
                # If grades out of range, skip saving and counting
                continue
        except ValueError:
            continue

        # Save to file
        file.write(f"{student_name} {subject1} {grade1} {subject2} {grade2}\n")

        # Sum grades for GPA calculation
        total_grades += g1 + g2
        count += 2

if count > 0:
    gpa = total_grades / count
    print(f"{gpa:.2f}")
else:
    print("0.00")
```

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

```
# You are using Python
N = int(input())
if N > 30:
    print("Exceeding limit!")
    exit()

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

# Calculate total earnings per day
total_earnings = [items * M for items in items_sold]
```

```
# Save earnings to file
with open("sales.txt", "w") as file:
    for earning in total_earnings:
        file.write(str(earning) + "\n")

# Read the file and display earnings
with open("sales.txt", "r") as file:
    earnings_from_file = file.read().split()

print(*earnings_from_file)
```

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

3. 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 !@#$%^&* 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

```
# You are using Python
name = input()
mobile = input()
username = input()
password = input()

special_chars = set("!@#$%^&*")

try:
    if not (10 <= len(password) <= 20):
        raise Exception("Should be a minimum of 10 characters and a maximum of
20 characters")

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

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

    print("Valid Password")

except Exception as e:
    print(e)
```

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

4. 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

```
# You are using Python
import re

class IllegalArgumentException(Exception):
```

```
    pass

class NoSuchElementException(Exception):
    pass

class NumberFormatException(Exception):
    pass

def validate_register_number(reg_num):
    if len(reg_num) != 9:
        raise IllegalArgumentException("Register Number should have exactly 9
characters.")
    if not reg_num.isalnum():
        raise NoSuchElementException("Register Number should have only
alphabets and digits.")
    pattern = r'^\d{2}[A-Za-z]{3}\d{4}$'
    if not re.match(pattern, reg_num):
        raise IllegalArgumentException("Register Number should have the format: 2
numbers, 3 characters, and 4 numbers.")

def validate_mobile_number(mobile_num):
    if len(mobile_num) != 10:
        raise IllegalArgumentException("Mobile Number should have exactly 10
characters.")
    if not mobile_num.isdigit():
        raise NumberFormatException("Mobile Number should only contain digits.")

try:
    reg_num = input().strip()
    mobile_num = input().strip()
    validate_register_number(reg_num)
    validate_mobile_number(mobile_num)
    print("Valid")
except (IllegalArgumentException, NoSuchElementException,
NumberFormatException) as e:
    print("Invalid with exception message:", e)
```

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