Rajalakshmi Engineering College

Name: Yugesh E T
Email: 240701613@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
def record_student_grades():
    filename = "magical_grades.txt"

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

            subject1 = input().strip()
            grade1 = float(input().strip())

            subject2 = input().strip()
            grade2 = float(input().strip())

            if not (0 <= grade1 <= 100 and 0 <= grade2 <= 100):
                print("Invalid grade input. Grades must be between 0 and 100.")
                continue

            gpa = (grade1 + grade2) / 2
            print(f"{gpa:.2f}")

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

record_student_grades()
```

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

2. 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

```
# You are using Python
from datetime import datetime

def validate_time(time_str):
    try:
        datetime.strptime(time_str, '%Y-%m-%d %H:%M:%S')
        return True
    except ValueError:
        return False

start_time = input().strip()
end_time = input().strip()

if validate_time(start_time) and validate_time(end_time):
    print(start_time)
    print(end_time)
else:
    print("Event time is not in the format")
```

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

3. 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

# Custom Exceptions
class IllegalArgumentException(Exception):
   pass

class NumberFormatException(Exception):
   pass

class NoSuchElementException(Exception):
   pass

def validate_register_and_mobile(register_number, mobile_number):
    try:
        if len(register_number) != 9:
            raise IllegalArgumentException("Register Number should have exactly 9
characters.")

        if not re.match(r"^\d{2}[A-Za-z]{3}\d{4}$", register_number):
            raise IllegalArgumentException("Register Number should have the format:
2 numbers, 3 characters, and 4 numbers.")

        if not register_number.isalnum():
            raise NoSuchElementException("Register Number should only contain
digits and alphabets.")

        if len(mobile_number) != 10:
            raise IllegalArgumentException("Mobile Number should have exactly 10
characters.")

        if not mobile_number.isdigit():
            raise NumberFormatException("Mobile Number should only contain
digits.")

        print("Valid")

    except (IllegalArgumentException, NumberFormatException,
NoSuchElementException) as e:
        print(f"Invalid with exception message: {e}")

register_number = input().strip()
mobile_number = input().strip()

validate_register_and_mobile(register_number, mobile_number)
```

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

4. Problem Statement

Alice is developing a program called "Name Sorter" that helps users organize and sort names alphabetically.

The program takes names as input from the user, saves them in a file, and then displays the names in sorted order.

File Name: sorted_names.txt.

Input Format

The input consists of multiple lines, each containing a name represented as a string.

To end the input and proceed with sorting, the user can enter 'q'.

Output Format

The output displays the names in alphabetical order, each name on a new line.

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: Alice Smith
John Doe
Emma Johnson
q

Output: Alice Smith
Emma Johnson
John Doe

Answer

```
def name_sorter():
    filename = "sorted_names.txt"
    names = []

    while True:
        name = input()
        if name.lower() == 'q':
            break
        names.append(name)

    with open(filename, 'w') as file:
        for name in names:
            file.write(name + '\n')
    with open(filename, 'r') as file:
        sorted_names = sorted([line.strip() for line in file if line.strip()])

    for name in sorted_names:
        print(name)

name_sorter()
```

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