Rajalakshmi Engineering College

Name: n.madhu narayanan
Email: 240701295@rajalakshmi.edu.in
Roll no: 240701295
Phone: 8870065218
Branch: REC
Department: I CSE AH
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

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):
    # Check for positive side lengths
    if a <= 0 or b <= 0 or c <= 0:
        raise ValueError("Side lengths must be positive")

    # Check for triangle inequality
    if (a + b > c) and (a + c > b) and (b + c > a):
        return True
    else:
        return False

# Read input
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}")
```

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_event_time(start, end):
    try:
        # Try to parse the datetime strings
        start_time = datetime.strptime(start, "%Y-%m-%d %H:%M:%S")
        end_time = datetime.strptime(end, "%Y-%m-%d %H:%M:%S")
        # If successful, print the valid start and end times
        print(start, end)
    except ValueError:
        # If parsing fails, the format is incorrect or the date is invalid
        print("Event time is not in the format")

# Read input from user
start_input = input()
end_input = input()

# Validate and print output
validate_event_time(start_input, end_input)
```

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

3. 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 calculate_gpa():
    with open("magical_grades.txt", "w") as file:
        while True:
            student_name = input()
            if student_name.lower() == 'done':
                break
            subject1 = input()
            grade1 = float(input())
            if not (0 <= grade1 <= 100):
                print("Grade must be between 0 and 100.")
                continue
            subject2 = input()
            grade2 = float(input())
            if not (0 <= grade2 <= 100):
                print("Grade must be between 0 and 100.")
                continue
            gpa = (grade1 + grade2) / 2
            file.write(f"{gpa:.2f}\n")
            print(f"{gpa:.2f}")
```

calculate_gpa()

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

# Define custom exceptions
class IllegalArgumentException(Exception):
    pass

class NumberFormatException(Exception):
    pass

class NoSuchElementException(Exception):
    pass

def validate_register_number(reg_no):
    if len(reg_no) != 9:
        raise IllegalArgumentException("Register Number should have exactly 9 characters.")

    if not re.match(r'^[0-9]{2}[A-Za-z]{3}[0-9]{4}$', reg_no):
        if not re.match(r'^[A-Za-z0-9]+$', reg_no):
            raise NoSuchElementException("Register Number should only contain alphabets and digits.")
        raise IllegalArgumentException("Register Number should have the format: 2 numbers, 3 characters, and 4 numbers.")

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

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

def main():
    try:
        reg_no = input().strip()
        mobile_no = input().strip()

        validate_register_number(reg_no)
```

```
        validate_mobile_number(mobile_no)

        print("Valid")

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

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

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