Rajalakshmi Engineering College

Name: Parameswari P
Email: 240701378@rajalakshmi.edu.in
Roll no: 240701378
Phone: 9500133836
Branch: REC
Department: I CSE FD
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 : 37.5

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 save_grades_and_calculate_gpa():
    with open("magical_grades.txt", "w") as file:
        while True:
            name = input()
            if name.lower() == "done":
                break
            subject1 = input()
            grade1 = int(input())
            subject2 = input()
            grade2 = int(input())

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

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

```
# Run the function
save_grades_and_calculate_gpa()
```

Status : Correct **_Marks : 10/10_**

2. 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

# Main program
try:
    a = int(input())
    b = int(input())
    c = int(input())

    if is_valid_triangle(a, b, c):
        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***

3. 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

```
# You are using Python
names = []

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

# Sort the names alphabetically
names.sort()

# Write the sorted names to the file
with open("sorted_names.txt", "w") as file:
   for name in names:
      file.write(name + "\n")

# Display the sorted names
for name in names:
   print(name)
```

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
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.")
   # Check if all characters are alphanumeric
   if not reg_no.isalnum():
      raise NoSuchElementException("Register Number should contain only digits 
and alphabets.")
   # Check pattern: 2 digits, 3 letters, 4 digits
   import re
   pattern = r'^\d{2}[A-Za-z]{3}\d{4}$'
   if not re.match(pattern, reg_no):
      raise IllegalArgumentException("Register Number should have the format: 2 
numbers, 3 characters, and 4 numbers.")

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

try:
   reg_no = input().strip()
   mobile = input().strip()

   validate_register_number(reg_no)
   validate_mobile_number(mobile)
   print("Valid")

except IllegalArgumentException as e:
   print("Invalid with exception message:", e)
```

```
except NumberFormatException as e:
    print("Invalid with exception message:", e)
except NoSuchElementException as e:
    print("Invalid with exception message:", e)
```

Status : Partially correct ***Marks : 7.5/10***