Rajalakshmi Engineering College

Name: M Vaishnavi
Email: 240701576@rajalakshmi.edu.in
Roll no: 240701576
Phone: 9381045343
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

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 !@#$%^&amp;* 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

```
import re

def validate_password(password):
    errors = []


    if not (10 <= len(password) <= 20):
        errors.append("Should be a minimum of 10 characters and a maximum of
20 characters")


    if not any(char.isdigit() for char in password):
        errors.append("Should contain at least one digit")


    if not re.search(r'[!@#$%^&*]', password):
        errors.append("It should contain at least one special character")
    if errors:
        print(errors[0])
    else:
        print("Valid Password")


name = input().strip()
mobile_number = input().strip()
username = input().strip()
password = input().strip()


validate_password(password)
```

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

```
def is_valid_triangle(a, b, c):
    try:

        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:
            print("It's a valid triangle")
        else:
            print("It's not a valid triangle")

    except ValueError as e:
        print(f"ValueError: {e}")


side1 = int(input().strip())
side2 = int(input().strip())
side3 = int(input().strip())


is_valid_triangle(side1, side2, side3)
```

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

```
import re
class IllegalArgumentException(Exception):
    pass

class NumberFormatException(Exception):
    pass
```

```
class NoSuchElementException(Exception):
    pass

def validate_register_number(register_number):

    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.")

def validate_mobile_number(mobile_number):

    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.")

def main():
    try:

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

        validate_register_number(register_number)
        validate_mobile_number(mobile_number)

        print("Valid")

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

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

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

4. 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, items_sold, price):

    if n > 30:
        print("Exceeding limit!")
        return

    earnings = [items * price for items in items_sold]

    with open("sales.txt", "w") as file:
        for earning in earnings:
            file.write(f"{earning}\n")

    with open("sales.txt", "r") as file:
        print(file.read().strip())

n = int(input().strip())
items_sold = list(map(int, input().strip().split()))
price = int(input().strip())

record_sales(n, items_sold, price)
```

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