Rajalakshmi Engineering College

Name: Sherwin G M
Email: 240701496@rajalakshmi.edu.in
Roll no: 240701496
Phone: 7708605966
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 : 40

Section 1 : Coding

1. 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_and_mobile(register_number, mobile_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 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.")

    return "Valid"

def main():
    try:
        register_number = input().strip()
        mobile_number = input().strip()

        result = validate_register_and_mobile(register_number, mobile_number)

        print(result)

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

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

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(side1, side2, side3):
    if side1 <= 0 or side2 <= 0 or side3 <= 0:
        raise ValueError("Side lengths must be positive")
    if side1 + side2 > side3 and side1 + side3 > side2 and side2 + side3 > side1:
        return True
    return False

def main():
    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}")

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

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

```
def main():
    names = []

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

    names.sort()

    with open('sorted_names.txt', 'w') as file:
        for name in names:
            file.write(name + '\n')

    for name in names:
        print(name)

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

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

4. 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

```
import re

def validate_password(password):
    if len(password) < 10 or len(password) > 20:
        raise ValueError("Should be a minimum of 10 characters and a maximum of 20 characters")

    if not re.search(r"\d", password):
        raise ValueError("Should contain at least one digit")

    if not re.search(r"[!@#$%^&*]", password):
        raise ValueError("It should contain at least one special character")
```

```
    return "Valid Password"

def main():
    name = input()
    mobile = input()
    username = input()
    password = input()

    try:
        result = validate_password(password)
        print(result)
    except ValueError as e:
        print(e)

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

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