Rajalakshmi Engineering College

Name: Tharun Sathishkumar
Email: 240701563@rajalakshmi.edu.in
Roll no: 240701563
Phone: 9363661870
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

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

```
from datetime import datetime

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

def main():
   start_time = input().strip()
   end_time = input().strip()
   if check_time_format(start_time) and check_time_format(end_time):
      print(start_time)
      print(end_time)
   else:
      print("Event time is not in the format")

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

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

2. 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

# Define custom exceptions
```

```
class InvalidRegisterNumberException(Exception):
   pass

class InvalidMobileNumberException(Exception):
   pass

class RegisterNumberFormatException(Exception):
   pass

class RegisterNumberInvalidCharacterException(Exception):
   pass

# Function to validate register number
def validate_register_number(register_number):
   # Check if the register number has exactly 9 characters
   if len(register_number) != 9:
      raise InvalidRegisterNumberException("Register Number should have
exactly 9 characters.")

   # Check if it follows the pattern: 2 digits + 3 letters + 4 digits
   if not re.match(r'^\d{2}[A-Za-z]{3}\d{4}$', register_number):
      raise RegisterNumberFormatException("Register Number should have the
format: 2 numbers, 3 characters, and 4 numbers.")

   # Check if it contains any characters other than digits and alphabets
   if not register_number.isalnum():
      raise RegisterNumberInvalidCharacterException("Register Number should
only contain digits and letters.")

# Function to validate mobile number
def validate_mobile_number(mobile_number):
   # Check if the mobile number has exactly 10 characters
   if len(mobile_number) != 10:
      raise InvalidMobileNumberException("Mobile Number should have exactly
10 characters.")

   # Check if it only contains digits
   if not mobile_number.isdigit():
      raise ValueError("Mobile Number should only contain digits.")  # Use
ValueError instead of NumberFormatException

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

      # Validate the inputs
      validate_register_number(register_number)
      validate_mobile_number(mobile_number)

      # If no exception is raised, print "Valid"
      print("Valid")

   except (InvalidRegisterNumberException, InvalidMobileNumberException,
           RegisterNumberFormatException,
RegisterNumberInvalidCharacterException,
           ValueError) as e:
      print(f"Invalid with exception message: {str(e)}")

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

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

3. 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")
    return a + b > c and a + c > b and b + c > a

def main():
    try:
        a = int(input().strip())
        b = int(input().strip())
        c = int(input().strip())
        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}")

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 main():
   N = int(input())
   if N > 30:
      print("Exceeding limit!")
      return
   items_sold = list(map(int, input().split()))
   M = int(input())
   total_earnings = []

   for i in range(N):
      earnings = items_sold[i] * M
      total_earnings.append(earnings)


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


   with open('sales.txt', 'r') as file:
      for line in file:
         print(line.strip())

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

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