Rajalakshmi Engineering College

Name: Piraisoodan R
Email: 240701384@rajalakshmi.edu.in
Roll no: 240701384
Phone: 8056892546
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

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

```
names=[]
while True:
    name=input()
    if name.lower()=='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)
```

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

```
from datetime import datetime

def get_event_times():
    try:
        start_time = input()
        end_time = input()

        # Try to parse both times using the specified format
        datetime.strptime(start_time, '%Y-%m-%d %H:%M:%S')
        datetime.strptime(end_time, '%Y-%m-%d %H:%M:%S')

        print(start_time)
```

```
        print(end_time)

    except ValueError:
        print("Event time is not in the format")

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

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

3. Problem Statement

Bob, a data analyst, requires a program to automate the process of analyzing character frequency in a given text. This program should allow the user to input a string, calculate the frequency of each character within the text, save these character frequencies to a file named "char_frequency.txt," and display the results.

Input Format

The input consists of the string.

Output Format

The first line prints "Character Frequencies:".

The following lines print the character frequency in the format: "X: Y" where X is the character and Y is the count.

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: aaabbbccc

Output: Character Frequencies:
a: 3
b: 3
c: 3

Answer

```
def analyze_character_frequency():
    text = input()

    frequency = {}
    for char in text:
        frequency[char] = frequency.get(char, 0) + 1

    # Sort by character appearance order in the input (preserve order)
    from collections import OrderedDict
    frequency_ordered = OrderedDict()
    for char in text:
        if char not in frequency_ordered:
            frequency_ordered[char] = frequency[char]

    # Save to file
    with open("char_frequency.txt", "w") as file:
        file.write("Character Frequencies:\n")
        for char, count in frequency_ordered.items():
            file.write(f"{char}: {count}\n")

    # Print output
    print("Character Frequencies:")
    for char, count in frequency_ordered.items():
        print(f"{char}: {count}")

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

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

```
import re

# Define custom exceptions
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'^[0-9]{2}[A-Za-z]{3}[0-9]{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
alphabets and digits.")

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("Invalid with exception message:", str(e))

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

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