Rajalakshmi Engineering College

Name: gladwin antony.A
Email: 241501057@rajalakshmi.edu.in
Roll no: 241501057
Phone: 8015799480
Branch: REC
Department: I AI & ML FA
Batch: 2028
Degree: B.E - AI & ML

Scan to verify results

NeoColab_REC_CS23221_Python Programming

REC_Python_Week 6_MCQ

Attempt : 1
Total Mark : 20
Marks Obtained : 18

Section 1 : MCQ

1. How do you rename a file?

Answer

os.rename(existing_name, new_name)

Status : Correct ***Marks : 1/1***

2. What is the difference between r+ and w+ modes?

Answer

in w+ the pointer is initially placed at the beginning of the file and the pointer is at the end for r+

Status : Wrong ***Marks : 0/1***

241501057

3. Which of the following is true about
fp.seek(10,1)

Answer

Move file pointer ten characters ahead from the current position

Status : Correct ***Marks : 1/1***

4. Which clause is used to clean up resources, such as closing files in Python?

Answer

finally

Status : Correct ***Marks : 1/1***

5. What is the output of the following code?

```
try:
   x = "hello" + 5
except TypeError:
   print("Type Error occurred")
finally:
   print("This will always execute")
```

Answer

Type Error occurredThis will always execute

Status : Correct ***Marks : 1/1***

6. What happens if an exception is not caught in the except clause?

Answer

The program will display a traceback error and stop execution

Status : Correct ***Marks : 1/1***

241501057

7. What is the output of the following code?

```
try:
   x = 1 / 0
except ZeroDivisionError:
   print("Caught division by zero error")
finally:
   print("Executed")
```

Answer

Caught division by zero errorExecuted

Status : Correct ***Marks : 1/1***

8. Which of the following is true about the finally block in Python?

Answer

The finally block is always executed, regardless of whether an exception occurs or not

Status : Correct ***Marks : 1/1***

9. What happens if no arguments are passed to the seek function?

Answer

error

Status : Wrong ***Marks : 0/1***

10. Fill in the code in order to get the following output:

Output:

Name of the file: ex.txt

```
fo = open(_______(1), "wb")
print("Name of the file: ",_______)(2)
```

241501057 241501057 241501057 241501057 241501057 241501057 241501057 241501057 241501057 241501057 241501057 241501057 241501057 241501057 241501057 241501057

Answer

1) "ex.txt"2) fo.name

Status : Correct ***Marks : 1/1***

11. Fill in the blanks in the following code of writing data in binary files.

```
import ____________ (1)
rec=[]
while True:
   rn=int(input("Enter"))
   nm=input("Enter")
   temp=[rn, nm]
   rec.append(temp)
   ch=input("Enter choice (y/N)")
   if ch.upper=="N":
      break
f.open("stud.dat","___________")(2)
__________.dump(rec,f)(3)
_________.close()(4)
```

Answer

(pickle,wb,pickle,f)

Status : Correct ***Marks : 1/1***

12. Match the following:

a) f.seek(5,1) i) Move file pointer five characters behind from the current position

b) f.seek(-5,1) ii) Move file pointer to the end of a file

c) f.seek(0,2) iii) Move file pointer five characters ahead from the current position

d) f.seek(0) iv) Move file pointer to the beginning of a file

Answer

a-iii, b-i, c-ii, d-iv

241501057

Status : Correct ***Marks : 1/1***

13. How do you create a user-defined exception in Python?

Answer

By creating a new class that inherits from the Exception class

Status : Correct ***Marks : 1/1***

14. Fill the code to in order to read file from the current position.

Assuming exp.txt file has following 3 lines, consider current file position is beginning of 2nd line

Meri,25

John,21

Raj,20

Ouptput:

['John,21\n','Raj,20\n']

```
f = open("exp.txt", "w+")
_________(1)
print _________(2)
```

Answer

1) f.seek(0, 1)2) f.readlines()

Status : Correct ***Marks : 1/1***

15. What will be the output of the following Python code?

```
f = None
for i in range (5):
 with open("data.txt", "w") as f:
  if i > 2:
```

241501057

```
   break
print(f.closed)
```

Answer

True

Status : Correct ***Marks : 1/1***

16. What is the purpose of the except clause in Python?

Answer

To handle exceptions during code execution

Status : Correct ***Marks : 1/1***

17. What is the default value of reference_point in the following code?

```
file_object.seek(offset [,reference_point])
```

Answer

0

Status : Correct ***Marks : 1/1***

18. What is the output of the following code?

```
class MyError(Exception):
   pass

try:
   raise MyError("Something went wrong")
except MyError as e:
   print(e)
```

Answer

Something went wrong

Status : Correct ***Marks : 1/1***

241501057

19. What will be the output of the following Python code?

```
# Predefined lines to simulate the file content
lines = [
   "This is 1st line",
   "This is 2nd line",
   "This is 3rd line",
   "This is 4th line",
   "This is 5th line"
]

print("Name of the file: foo.txt")

# Print the first 5 lines from the predefined list
for index in range(5):
   line = lines[index]
   print("Line No %d - %s" % (index + 1, line.strip()))
```

Answer

Displays Output

Status : Correct ***Marks : 1/1***

20. What is the correct way to raise an exception in Python?

Answer

raise Exception()

Status : Correct ***Marks : 1/1***

241501057

Rajalakshmi Engineering College

Name: gladwin antony.A
Email: 241501057@rajalakshmi.edu.in
Roll no: 241501057
Phone: 8015799480
Branch: REC
Department: I AI & ML FA
Batch: 2028
Degree: B.E - AI & ML

Scan to verify results

NeoColab_REC_CS23221_Python Programming

REC_Python_Week 6_COD

Attempt : 1
Total Mark : 50
Marks Obtained : 50

Section 1 : Coding

1. Problem Statement

In a voting system, a person must be at least 18 years old to be eligible to vote. If a user enters an age below 18, the system should raise a user-defined exception indicating that they are not eligible to vote.

Input Format

The input contains a positive integer representing age.

Output Format

If the age is less than 18, the output displays "Not eligible to vote".

Otherwise, the output displays "Eligible to vote".

241501057

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 18
Output: Eligible to vote

Answer

```
# Define a custom exception
class NotEligibleToVoteException(Exception):
   pass

# Function to check voting eligibility
def check_voting_eligibility(age):
   if age < 18:
      raise NotEligibleToVoteException("Not eligible to vote")
   else:
      return "Eligible to vote"

# Input: Age of the person
age = int(input())

# Output: Check eligibility
try:
   result = check_voting_eligibility(age)
   print(result)
except NotEligibleToVoteException as e:
   print(e)
```

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

2. Problem Statement

Sophie enjoys playing with words and wants to count the number of words in a sentence. She inputs a sentence, saves it to a file, and then reads it from the file to count the words.

Write a program to determine the number of words in the input sentence.

File Name: sentence_file.txt

241501057

Input Format

The input consists of a single line of text containing words separated by spaces.

Output Format

The output displays the count of words in the sentence.

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: Four Words In This Sentence
Output: 5

Answer

```
# Function to count words in a sentence from a file
def count_words_in_file(file_name):
   # Open the file in read mode
   with open(file_name, 'r') as file:
      # Read the sentence from the file
      sentence = file.read().strip()

   # Split the sentence by spaces to get words and filter out empty strings (in
case of multiple spaces)
   words = sentence.split()

   # Return the number of words
   return len(words)

# Write the sentence to the file (this part is assumed for testing purposes)
sentence = input()  # Read the sentence from user input

# Writing the input sentence to "sentence_file.txt"
with open('sentence_file.txt', 'w') as file:
   file.write(sentence)

# Call the function to count words and print the result
word_count = count_words_in_file('sentence_file.txt')
print(word_count)
```

241501057

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

3. Problem Statement

Tara is a content manager who needs to perform case conversions for various pieces of text and save the results in a structured manner.

She requires a program to take a user's input string, save it in a file, and then retrieve and display the string in both upper-case and lower-case versions. Help her achieve this task efficiently.

File Name: text_file.txt

Input Format

The input consists of a single line containing a string provided by the user.

Output Format

The first line displays the original string read from the file in the format: "Original String: {original_string}".

The second line displays the upper-case version of the original string in the format: "Upper-Case String: {upper_case_string}".

The third line displays the lower-case version of the original string in the format: "Lower-Case String: {lower_case_string}".

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: #SpecialSymBoLs1234

Output: Original String: #SpecialSymBoLs1234
Upper-Case String: #SPECIALSYMBOLS1234
Lower-Case String: #specialsymbols1234

Answer

```
# Function to handle the case conversion and file operations
```

241501057

```
def process_string_and_save_to_file():
    # Step 1: Read the input string from the user
    input_string = input()

    # Step 2: Save the input string into the file
    with open('text_file.txt', 'w') as file:
        file.write(input_string)

    # Step 3: Read the string from the file
    with open('text_file.txt', 'r') as file:
        string_from_file = file.read().strip()  # Read and remove any surrounding
whitespace

    # Step 4: Convert to upper and lower case
    upper_case_string = string_from_file.upper()
    lower_case_string = string_from_file.lower()

    # Step 5: Print the results in the required format
    print(f"Original String: {string_from_file}")
    print(f"Upper-Case String: {upper_case_string}")
    print(f"Lower-Case String: {lower_case_string}")

# Call the function to process the input and output
process_string_and_save_to_file()
```

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

4. Problem Statement

A retail store requires a program to calculate the total cost of purchasing a product based on its price and quantity. The program performs validation to ensure valid inputs and handles specific error conditions using exceptions:

Price Validation: If the price is zero or less, raise a ValueError with the message: "Invalid Price".Quantity Validation: If the quantity is zero or less, raise a ValueError with the message: "Invalid Quantity".Cost Threshold: If the total cost exceeds 1000, raise RuntimeError with the message: "Excessive Cost".

Input Format

241501057

The first line of input consists of a double value, representing the price of a product.

The second line consists of an integer, representing the quantity of the product.

Output Format

If the calculation is successful, print the total cost rounded to one decimal place.

If the price is zero or less prints "Invalid Price".

If the quantity is zero or less prints "Invalid Quantity".

If the total cost exceeds 1000, prints "Excessive Cost".

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 20.0
5
Output: 100.0

Answer

```
def calculate_total_cost(price, quantity):
    # Validate price
    if price <= 0:
        raise ValueError("Invalid Price")

    # Validate quantity
    if quantity <= 0:
        raise ValueError("Invalid Quantity")

    # Calculate total cost
    total_cost = price * quantity

    # Check if the cost exceeds 1000
    if total_cost > 1000:
        raise RuntimeError("Excessive Cost")

    # Return the total cost if all validations pass
```

241501057

```
    return total_cost

# Input: Read price and quantity
try:
    price = float(input())  # Price is a float
    quantity = int(input())  # Quantity is an integer

    # Call the function to calculate the total cost
    total_cost = calculate_total_cost(price, quantity)

    # Output the result
    print(f"{total_cost:.1f}")

except ValueError as ve:
    # Handle ValueError (Invalid Price or Invalid Quantity)
    print(ve)

except RuntimeError as re:
    # Handle RuntimeError (Excessive Cost)
    print(re)
```

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

5. Problem Statement

Write a program that calculates the average of a list of integers. The program prompts the user to enter the length of the list (n) and each element of the list. It performs error handling to ensure that the length of the list is a non-negative integer and that each input element is a numeric value.

Input Format

The first line of the input is an integer n, representing the length of the list as a positive integer.

The second line of the input consists of an element of the list as an integer, separated by a new line.

Output Format

If the length of the list is not a positive integer or zero, the output displays "Error:

241501057

The length of the list must be a non-negative integer."

If a non-numeric value is entered for the length of the list, the output displays "Error: You must enter a numeric value."

If a non-numeric value is entered for a list element, the output displays "Error: You must enter a numeric value."

If the inputs are valid, the program calculates and prints the average of the provided list of integers with two decimal places: "The average is: [average]".

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: -2
1
2
Output: Error: The length of the list must be a non-negative integer.

Answer

```
def calculate_average():
    try:
        # Step 1: Input for length of the list (n)
        n = input()
        # Validate that n is a positive integer
        if not n.isdigit() or int(n) <= 0:
            print("Error: The length of the list must be a non-negative integer.")
            return
        n = int(n)

        # Step 2: Input for the list elements
        elements = []
        for i in range(n):
            element = input()
            # Validate that each element is a numeric value
            if not element.isdigit():
```

241501057

```
                print("Error: You must enter a numeric value.")
                return
            elements.append(int(element))

        # Step 3: Calculate the average if the inputs are valid
        average = sum(elements) / n
        print(f"The average is: {average:.2f}")

    except ValueError:
        print("Error: You must enter a numeric value.")

# Call the function
calculate_average()
```

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

241501057

Rajalakshmi Engineering College

Name: gladwin antony.A
Email: 241501057@rajalakshmi.edu.in
Roll no: 241501057
Phone: 8015799480
Branch: REC
Department: I AI & ML FA
Batch: 2028
Degree: B.E - AI & ML

Scan to verify results

NeoColab_REC_CS23221_Python Programming

REC_Python_Week 6_PAH

Attempt : 1
Total Mark : 30
Marks Obtained : 28.5

Section 1 : Coding

1. Problem Statement

Reeta is playing with numbers. Reeta wants to have a file containing a list of numbers, and she needs to find the average of those numbers. Write a program to read the numbers from the file, calculate the average, and display it.

File Name: user_input.txt

Input Format

The input file will contain a single line of space-separated numbers (as a string).

These numbers may be integers or decimals.

Output Format

241501057

If all inputs are valid numbers, the output should print: "Average of the numbers is: X.XX" (where X.XX is the computed average rounded to two decimal places)

If the input contains invalid data, print: "Invalid data in the input."

Refer to the sample output for format specifications.

Sample Test Case

Input: 1 2 3 4 5

Output: Average of the numbers is: 3.00

Answer

```
def calculate_average_from_input():
    """
    Reads a line of space-separated numbers from input, calculates their average,
    and prints the result. Handles invalid data by printing an error message.
    """
    try:
        # Read the single line of space-separated numbers from standard input.
        # In a typical competitive programming setup, this simulates reading from a
file.
        input_line = input()

        # Split the input string by spaces to get a list of number strings.
        number_strings = input_line.split()

        numbers = []
        # Iterate through each string in the list and attempt to convert it to a float.
        for num_str in number_strings:
            try:
                # Convert the string to a float. This will raise a ValueError if it's not a
valid number.
                numbers.append(float(num_str))
            except ValueError:
                # If a ValueError occurs, it means the string is not a valid number.
                # Print the error message and exit the function.
                print("Invalid data in the input.")
                return
```

241501057

241501057

```
        # Check if there are any numbers to avoid division by zero if the input is
empty.
        if not numbers:
            # Although constraints say 1 <= n <= 100, it's good practice to handle
empty case.
            # For this problem, an empty input would likely be considered invalid data.
            print("Invalid data in the input.")
            return

        # Calculate the sum of all numbers.
        total_sum = sum(numbers)
        # Calculate the count of numbers.
        count = len(numbers)

        # Calculate the average.
        average = total_sum / count

        # Print the average, rounded to two decimal places.
        print(f"Average of the numbers is: {average:.2f}")

    except Exception as e:
        # Catch any other unexpected errors during input reading or processing.
        # This is a general catch-all, though ValueError for float conversion is the
primary one.
        print(f"An unexpected error occurred: {e}")

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

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

2. Problem Statement

Peter manages a student database and needs a program to add students. For each student, Alex inputs their ID and name. The program checks for duplicate IDs and ensures the database isn't full.

If a duplicate or a full database is detected, an appropriate error message is displayed. Otherwise, the student is added, and a confirmation message is shown. The database has a maximum capacity of 30 students, and each student must have a unique ID.

Input Format

The first line contains an integer n, representing the number of students to be added to the school database.

The next n lines each contain two space-separated values, representing the student's ID (integer) and the student's name (string).

Output Format

The output will depend on the actions performed in the code.

If a student is added to the database, the output will display: "Student with ID [ID number] added to the database."

If there is an exception due to a duplicate student ID, the output will display: "Exception caught. Error: Student ID already exists."

If there is an exception due to the database being full, the output will display: "Exception caught. Error: Student database is full."

Refer to the sample outputs for the formatting specifications.

Sample Test Case

Input: 3
16 Sam
87 Sabari
43 Dani

Output: Student with ID 16 added to the database.
Student with ID 87 added to the database.
Student with ID 43 added to the database.

Answer

```
class DatabaseFullError(Exception):
    """Custom exception raised when the student database reaches its maximum
capacity."""
```

241501057

```
    pass

class DuplicateIDError(Exception):
    """Custom exception raised when an attempt is made to add a student with an
existing ID."""
    pass

# Define the maximum capacity for the student database
MAX_CAPACITY = 30

# Initialize an empty dictionary to store student data.
# Keys will be student IDs, and values will be student names.
student_database = {}

if __name__ == "__main__":
    try:
        # Read the total number of students to attempt to add from the first line of
input.
        num_students_to_add = int(input())

        # Flag to ensure the "database full" message is printed only once.
        database_full_message_printed = False

        # Loop through the specified number of students to process their data.
        for _ in range(num_students_to_add):
            # Always read the input line for the current student.
            # This ensures all input lines are consumed, even if students aren't added.
            current_input_line = input()

            # If the database is already full and the message has been printed,
            # just consume the current line and move to the next iteration.
            if database_full_message_printed:
                continue

            # Check if the database is about to exceed its capacity before processing
this student.
            # This check is placed here to determine if we should print the "database
full" message.
            if len(student_database) >= MAX_CAPACITY:
                print("Exception caught. Error: Student database is full.")
                database_full_message_printed = True # Set the flag to True after
printing
```

241501057

```
        continue # Skip adding this student, but continue loop to consume
remaining inputs.

      try:
        # Attempt to parse the student ID and name from the input line.
        # .split(maxsplit=1) handles names that might contain spaces.
        line_parts = current_input_line.split(maxsplit=1)
        student_id = int(line_parts[0])  # Convert ID to integer.
        student_name = line_parts[1]     # Get student's name.

        # Check if a student with the same ID already exists.
        if student_id in student_database:
          raise DuplicateIDError("Student ID already exists.")

        # If all checks pass, add the student to the database.
        student_database[student_id] = student_name
        print(f"Student with ID {student_id} added to the database.")

      except DuplicateIDError as e:
        # If a DuplicateIDError is caught, print the specific error message.
        # The loop continues to process subsequent students.
        print(f"Exception caught. Error: {e}")
      except ValueError:
        # Handle cases where the student ID is not a valid integer
        # or if the input line's format is unexpected.
        print("Exception caught. Error: Invalid input format or student ID.")
      except IndexError:
        # Handle cases where the input line does not contain both ID and
name.
        print("Exception caught. Error: Missing student ID or name in input.")
      except Exception as e:
        # Catch any other unexpected exceptions during individual student
processing.
        print(f"An unexpected error occurred: {e}")

  except ValueError:
    # This handles cases where the initial input for `num_students_to_add` is
not a valid integer.
    print("Exception caught. Error: Invalid number of students specified.")
  except Exception as e:
    # General catch-all for any other errors during the initial setup phase.
    print(f"An unexpected error occurred during setup: {e}")
```

241501057

Status : Partially correct ***Marks : 8.5/10***

3. Problem Statement

John is a data analyst who often works with text files. He needs a program that can analyze the contents of a text file and count the number of times a specific character appears in the file.

John wants a simple program that allows him to specify a file and a character to count within that file.

Input Format

The first line of input consists of the file's name to be analyzed.

The second line of the input consists of the string they want to write within the file.

The third line of the input consists of a character to count within the file.

Output Format

If the character is found, the output displays "The character 'X' appears {Y} times in the file." where X is the character and Y i the count,

If the character does not appear in the file, the output displays "Character not found."

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: test.txt
This is a test file to check the character count.
e
Output: The character 'e' appears 5 times in the file.

241501057

Answer

```
def analyze_file_character_count():
    """
    Analyzes a text file to count the occurrences of a specific character
    in a case-insensitive manner.
    """
    try:
        # Get the filename from the first line of input
        filename = input()
        # Get the string content to write into the file from the second line
        file_content = input()
        # Get the character to count from the third line
        char_to_count = input()

        # Step 1: Write the provided content to the specified file.
        # Using 'with' statement ensures the file is properly closed even if errors
occur.
        with open(filename, 'w') as file:
            file.write(file_content)

        # Step 2: Read the content back from the file.
        with open(filename, 'r') as file:
            read_content = file.read()

        # Step 3: Count the occurrences of the specified character (case-
insensitive).
        # Convert both the file content and the character to lowercase for a case-
insensitive count.
        char_count = read_content.lower().count(char_to_count.lower())

        # Step 4: Print the output based on the character count.
        if char_count > 0:
            print(f"The character '{char_to_count}' appears {char_count} times in the
file.")
        else:
            # The output for "Character not found" was slightly different in the prompt
example
            # "Character not found." vs "Character not found in the file."
            # Sticking to the exact format given in Sample 3 for "Character not found."
            print("Character not found in the file.")

    except IOError:
```

241501057

```
        # This exception handles potential issues like permission errors or disk full
when accessing the file.
        print(f"An error occurred while accessing the file '{filename}'.")
    except Exception as e:
        # Catch any other unexpected errors.
        print(f"An unexpected error occurred: {e}")

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

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

241501057 241501057 241501057 241501057 241501057 241501057 241501057 241501057 241501057 241501057 241501057 241501057 241501057 241501057 241501057 241501057

Rajalakshmi Engineering College

Name: gladwin antony.A
Email: 241501057@rajalakshmi.edu.in
Roll no: 241501057
Phone: 8015799480
Branch: REC
Department: I AI & ML FA
Batch: 2028
Degree: B.E - AI & ML

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.

241501057

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):
   # Check if the password length is between 10 and 20 characters
   if len(password) < 10 or len(password) > 20:
      return "Should be a minimum of 10 characters and a maximum of 20 
characters"

   # Check if the password contains at least one digit
   if not re.search(r'\d', password):
      return "Should contain at least one digit"

   # Check if the password contains at least one special character from !@#$
%^&*
   if not re.search(r'[!@#$%^&*]', password):
      return "It should contain at least one special character"

   # If all conditions are met
   return "Valid Password"
```

241501057

```
# Input from user
name = input()  # Name
mobile_number = input()  # Mobile number
username = input()  # Username
password = input()  # Password

# Validate the password
result = validate_password(password)

# Output the result
print(result)
```

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>".

241501057

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):
    """
    Checks whether a set of three input values can form the sides of a valid
triangle.

    Args:
        side1: The length of the first side.
        side2: The length of the second side.
        side3: The length of the third side.

    Returns:
        True if the sides form a valid triangle, False otherwise.

    Raises:
        ValueError: If any side length is not a positive value.
    """
    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
    else:
        return False

if __name__ == "__main__":
    try:
        A = int(input())
        B = int(input())
        C = int(input())
```

241501057

```
        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

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

241501057

```
2022-02-12 10:10:12
Output: 2022-01-12 06:10:00
2022-02-12 10:10:12
```

Answer

```
from datetime import datetime

if __name__ == "__main__":
    # Define the expected time format
    TIME_FORMAT = '%Y-%m-%d %H:%M:%S'

    try:
        # Read the start time from the user
        start_time_str = input()
        # Read the end time from the user
        end_time_str = input()

        # Attempt to parse the start time string into a datetime object
        # This will raise a ValueError if the format is incorrect
        parsed_start_time = datetime.strptime(start_time_str, TIME_FORMAT)

        # Attempt to parse the end time string into a datetime object
        # This will also raise a ValueError if the format is incorrect
        parsed_end_time = datetime.strptime(end_time_str, TIME_FORMAT)

        # If both parsing attempts are successful, print the original strings
        print(start_time_str)
        print(end_time_str)

    except ValueError:
        # If a ValueError occurs during parsing (due to incorrect format),
        # print the specified error message
        print("Event time is not in the format")
```

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:

241501057

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

```
# Define custom exception classes as specified in the problem statement
class IllegalArgumentException(Exception):
    """Custom exception for invalid arguments."""
    pass

class NumberFormatException(Exception):
    """Custom exception for invalid number format."""
```

241501057

```
    pass

class NoSuchElementException(Exception):
    """Custom exception for elements not found or invalid characters."""
    pass

def validate_student_data(reg_num, mobile_num):
    """
    Validates the student's Register Number and Mobile Number based on
specific rules.

    Args:
        reg_num (str): The student's Register Number.
        mobile_num (str): The student's Mobile Number.

    Raises:
        NumberFormatException: If the mobile number contains any character other
than a digit.
        IllegalArgumentException: If length constraints are violated for either
number,
                                  or if the register number format is incorrect.
        NoSuchElementException: If the register number contains any character
other than
                                digits and alphabets.
    """

    # --- Mobile Number Validation ---
    # Rule: If the Mobile Number contains any character other than a digit, raise a
NumberFormatException.
    if not mobile_num.isdigit():
        raise NumberFormatException("Mobile Number should only contain digits.")

    # Rule: If the Mobile Number does not contain exactly 10 characters, throw an
IllegalArgumentException.
    if len(mobile_num) != 10:
        raise IllegalArgumentException("Mobile Number should have exactly 10
characters.")

    # --- Register Number Validation ---
    # Rule: If the Register Number does not contain exactly 9 characters, throw an
IllegalArgumentException.
    if len(reg_num) != 9:
```

241501057

```
        raise IllegalArgumentException("Register Number should have exactly 9
characters.")

    # Rule: If the Register Number contains any character other than digits and
alphabets,
    # throw a NoSuchElementException.
    if not reg_num.isalnum():
        raise NoSuchElementException("Register Number should only contain digits
and alphabets.")

    # Rule: If the Register Number does not contain exactly 9 characters in the
specified format
    # (2 numbers followed by 3 characters followed by 4 numbers), throw an
IllegalArgumentException.
    # We use slicing and string methods (isdigit(), isalpha()) to check the format.
    # reg_num[0:2] checks the first 2 characters for digits.
    # reg_num[2:5] checks the next 3 characters for alphabets.
    # reg_num[5:9] checks the last 4 characters for digits.
    if not (reg_num[0:2].isdigit() and
            reg_num[2:5].isalpha() and
            reg_num[5:9].isdigit()):
        raise IllegalArgumentException("Register Number should have the format: 2
numbers, 3 characters, and 4 numbers.")

    # If all checks pass, the data is considered valid
    return True

if __name__ == "__main__":
    # Read the Register Number from the first line of input
    register_number = input()
    # Read the Mobile Number from the second line of input
    mobile_number = input()

    try:
        # Attempt to validate the student data
        if validate_student_data(register_number, mobile_number):
            # If no exceptions are raised, print "Valid"
            print("Valid")
    except (IllegalArgumentException, NumberFormatException,
NoSuchElementException) as e:
        # If any of the custom exceptions are caught, print the invalid message
        # along with the specific exception message.
```

241501057

```
        print(f"Invalid with exception message: {e}")
```

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

241501057