Rajalakshmi Engineering College

Name: Prithvi Yogeswaraa M
Email: 241501154@rajalakshmi.edu.in
Roll no: 2116241501154
Phone: 9345507776
Branch: REC
Department: I AIML AD
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 : 16

Section 1 : MCQ

1. 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)
```

Answer

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

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

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

Answer

raise Exception()

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

3. 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***

4. 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:
         break
print(f.closed)
```

Answer

False

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

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

Answer

To handle exceptions during code execution

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

6. 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

This will always execute

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

7. 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***

8. What is the output of the following code?

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

Answer

Executed

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

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

Answer

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

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

10. How do you rename a file?

Answer

os.rename(existing_name, new_name)

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

11. 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***

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

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

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

Answer

file position remains unchanged

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

14. 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***

15. 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***

16. 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***

17. 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***

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

Answer

finally

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

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

Compile Time Error

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

20. 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***

Rajalakshmi Engineering College

Name: Prithvi Yogeswaraa M
Email: 241501154@rajalakshmi.edu.in
Roll no: 2116241501154
Phone: 9345507776
Branch: REC
Department: I AIML AD
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

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

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

```
# You are using Python
try:
    price = float(input())
    quantity = int(input())

    if price <= 0:
        raise ValueError("Invalid Price")
    if quantity <= 0:
        raise ValueError("Invalid Quantity")

    total_cost = price * quantity

    if total_cost > 1000:
        raise RuntimeError("Excessive Cost")

    print(round(total_cost, 1))

except ValueError as ve:
    print(ve)
```

```
except RuntimeError as re:
    print(re)
```

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

2. 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

```
# You are using Python
# Read input string from the user
user_input = input()

# Save the input string to a file
with open("text_file.txt", "w") as file:
    file.write(user_input)

# Read the string back from the file
with open("text_file.txt", "r") as file:
    original_string = file.readline().rstrip("\n")

# Print the required outputs
print(f"Original String: {original_string}")
print(f"Upper-Case String: {original_string.upper()}")
print(f"Lower-Case String: {original_string.lower()}")
```

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

3. 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

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

```
# You are using Python
# Read the sentence from input
sentence = input()

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

# Read the sentence back from the file
with open("sentence_file.txt", "r") as file:
    line = file.readline().strip()

# Count the words
# If the line is empty after stripping, word count is 0
if line == "":
    word_count = 0
else:
    word_count = len(line.split())

# Print the word count
print(word_count)
```

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

4. 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".

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 18
Output: Eligible to vote

Answer

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

try:
    age = int(input())
    if age < 18:
        raise NotEligibleToVoteException("Not eligible to vote")
    else:
        print("Eligible to vote")

except NotEligibleToVoteException as e:
    print(e)
```

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: 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

```
try:
    n_input = input()
    n = int(n_input)  # Try converting directly to int

    if n <= 0:
        print("Error: The length of the list must be a non-negative integer.")
    else:
        numbers = []
```

```
        for _ in range(n):
            val = input()
            try:
                num = int(val)
                numbers.append(num)
            except ValueError:
                print("Error: You must enter a numeric value.")
                break
        else:
            avg = sum(numbers) / n
            print(f"The average is: {avg:.2f}")

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

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

Rajalakshmi Engineering College

Name: Prithvi Yogeswaraa M
Email: 241501154@rajalakshmi.edu.in
Roll no: 2116241501154
Phone: 9345507776
Branch: REC
Department: I AIML AD
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

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

```
line = input().strip()  # Read the entire line from standard input

tokens = line.split()
numbers = []

for token in tokens:
    try:
        num = float(token)
        numbers.append(num)
    except ValueError:
        print("Invalid data in the input.")
        break
else:
    avg = sum(numbers) / len(numbers)
    print(f"Average of the numbers is: {avg:.2f}")
```

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

```
MAX_CAPACITY = 30

n = int(input())
database = {}
full_error_printed = False  # To print "database is full" exception only once

for _ in range(n):
    line = input()
    try:
        if len(database) >= MAX_CAPACITY:
            if not full_error_printed:
                print("Exception caught. Error: Student database is full.")
                full_error_printed = True
            continue  # Skip adding further students

        student_id_str, student_name = line.split(maxsplit=1)
        student_id = int(student_id_str)

        if student_id in database:
            raise Exception("Student ID already exists.")

        database[student_id] = student_name
        print(f"Student with ID {student_id} added to the database.")

    except Exception as e:
        print(f"Exception caught. Error: {e}")
```

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.

Answer

```
# Read inputs
filename = input().strip()
content = input()
char_to_count = input()

# Write the content to the file
with open(filename, 'w') as file:
    file.write(content)

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

# Case-insensitive count: convert both to lower case
```

```
count = data.lower().count(char_to_count.lower())

# Output the result
if count > 0:
    print(f"The character '{char_to_count}' appears {count} times in the file.")
else:
    print("Character not found in the file.")
```

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

Rajalakshmi Engineering College

Name: Prithvi Yogeswaraa M
Email: 241501154@rajalakshmi.edu.in
Roll no: 2116241501154
Phone: 9345507776
Branch: REC
Department: I AIML AD
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 : 36.5

Section 1 : Coding

1. 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):
    # Check if any side length is not positive
    if a <= 0 or b <= 0 or c <= 0:
        raise ValueError("Side lengths must be positive")

    # Check triangle inequality
    if (a + b > c) and (a + c > b) and (b + c > a):
        return True
    else:
        return False

try:
    A = int(input())
    B = int(input())
    C = int(input())

    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***

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

```
class IllegalArgumentException(Exception):
    pass

class NoSuchElementException(Exception):
    pass

class NumberFormatException(Exception):
    pass

def validate_register_number(reg_num):
    if len(reg_num) != 9:
        raise IllegalArgumentException("Register Number should have exactly 9 characters.")

    # Check the format: 2 digits, 3 letters, 4 digits
    if not (reg_num[:2].isdigit() and
            reg_num[2:5].isalpha() and
            reg_num[5:].isdigit()):
        raise IllegalArgumentException("Register Number should have the format: 2 numbers, 3 characters, and 4 numbers.")

    # Check if all characters are alphanumeric
    if not reg_num.isalnum():
        raise NoSuchElementException("Register Number should only contain digits and alphabets.")

def validate_mobile_number(mobile_num):
    if len(mobile_num) != 10:
        raise IllegalArgumentException("Mobile Number should have exactly 10 characters.")

    if not mobile_num.isdigit():
        raise NumberFormatException("Mobile Number should only contain digits.")

try:
    reg_num = input().strip()
    mobile_num = input().strip()

    validate_register_number(reg_num)
```

```
    validate_mobile_number(mobile_num)

    print("Valid")

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

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

3. Problem Statement

In the enchanted realm of Academia, you, the Academic Alchemist, are bestowed with a magical quill and a parchment to weave the grades of aspiring students into a tapestry of academic brilliance.

The mission is to craft a Python program that empowers faculty members to enter student grades for any two subjects, stores these magical grades in a mystical file, and then, with a wave of your virtual wand, calculates the GPA to unveil the true essence of academic achievement.

Input Format

The input format is a string representing the student's name, any two subjects, and corresponding grades.

After entering grades, they can type 'done' when prompted for the student's name.

Output Format

The output should display the (average of grades) calculated GPA with a precision of two decimal places.

The magical grades will be saved in a mystical file named "magical_grades.txt".

Refer to the sample output for format specifications.

Sample Test Case

Input: Alice
Math
95
English
88
done

Output: 91.50

Answer

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

    with open("magical_grades.txt", "w") as file:
        while True:
            student_name = input().strip()
            if student_name.lower() == 'done':
                break

            subject1 = input().strip()
            grade1 = input().strip()
            subject2 = input().strip()
            grade2 = input().strip()

            # Save to file
            file.write(f"{student_name} {subject1} {grade1} {subject2} {grade2}\n")

            # Collect grades for GPA calculation
            grades.append(float(grade1))
            grades.append(float(grade2))

    if grades:
        average = sum(grades) / len(grades)
        print(f"{average:.2f}")
    else:
        print("0.00")  # If no grades entered

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

```
def validate_password(password):
    special_chars = set("!@#$%^&*")
```

```
    # Check digit presence first
    if not any(ch.isdigit() for ch in password):
        raise Exception("Should contain at least one digit")

    # Check special character presence
    if not any(ch in special_chars for ch in password):
        raise Exception("It should contain at least one special character")

    # Check length last
    if len(password) < 10 or len(password) > 20:
        raise Exception("Should be a minimum of 10 characters and a maximum of
20 characters")

def main():
    name = input().strip()
    mobile_number = input().strip()
    username = input().strip()
    password = input().strip()

    try:
        validate_password(password)
        print("Valid Password")
    except Exception as e:
        print(e)

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

Status : Partially correct **Marks : 6.5/10**