**Task 9: Implement Exceptions and Exceptional handling in Python.**

**Aim:**

To implement Exceptions and Exceptional handling in Python.

***Problem 9.1. You are developing a Python program that processes a list of students' grades. The program is designed to allow the user to select a grade by specifying an index number. However, you need to ensure that the program handles cases where the user inputs an index that is out of range, i.e., an index that does not exist in the list.***

**Algorithm**:

1. Start the program
2. Initializes a list of grades (e.g., [85, 90, 78, 92, 88]).
3. Prompts the user to enter the index of the grade they wish to view.
4. Attempts to display the grade at the specified index.
5. If the index is out of range, catches the IndexError and prints an error message, "Invalid index. Please enter a valid index."

**Program:**

# Initialize the list of grades

grades = [85, 90, 78, 92, 88]

# Display the grades list

print("Grades List:", grades)

# Prompt the user to enter the index of the grade they want to view

try:

index = int(input("Enter the index of the grade you want to view: "))

# Attempt to display the grade at the specified index

print(f"The grade at index {index} is: {grades[index]}")

except IndexError:

# Handle the case where the index is out of range

print("Invalid index. Please enter a valid index.")

except ValueError:

# Handle the case where the input is not an integer

print("Invalid input. Please enter a numerical index.")

**Output:**

Grades List: [85, 90, 78, 92, 88]

Enter the index of the grade you want to view: 10

Invalid index. Please enter a valid index.

Grades List: [85, 90, 78, 92, 88]

Enter the index of the grade you want to view: y

Invalid input. Please enter a numerical index.

***Problem 9.2. You are developing a Python calculator program that performs basic arithmetic operations. One of the key functionalities is to divide two numbers entered by the user. However, dividing by zero is not allowed and would cause the program to crash if not handled properly.***

**Algorithm:**

1. Define a custom exception NegativeNumberError that inherits from Exception.
2. Take two inputs from the user (num1, num2).
3. Use try-except-finally for error handling:
4. If input is not numeric → handle ValueError.
5. If denominator is 0 → handle ZeroDivisionError.
6. If any number is negative → raise NegativeNumberError.
7. Otherwise, perform the division.
8. In the finally block, print "Calculation Attempt Finished".

**Program:**

# Custom Exception

class NegativeNumberError(Exception):

pass

try:

# Take user input

num1 = int(input("Enter the numerator: "))

num2 = int(input("Enter the denominator: "))

# Check for negative numbers

if num1 < 0 or num2 < 0:

raise NegativeNumberError("Negative numbers are not allowed")

# Perform division

result = num1 / num2

print(f"Result: {num1} / {num2} = {result}")

except ValueError:

print("Error: Please enter valid numeric values.")

except ZeroDivisionError:

print("Error: Division by zero is not allowed.")

except NegativeNumberError as e:

print(f"Error: {e}")

finally:

print("Calculation Attempt Finished")

**Sample Output 1 (Valid Input):**

Enter the numerator: 20

Enter the denominator: 5

Result: 20 / 5 = 4.0

Calculation Attempt Finished

**Sample Output 2 (Denominator Zero):**

Enter the numerator: 10

Enter the denominator: 0

Error: Division by zero is not allowed.

Calculation Attempt Finished

**Sample Output 3 (Non-numeric Input):**

Enter the numerator: ten

Error: Please enter valid numeric values.

Calculation Attempt Finished

**Sample Output 4 (Negative Input):**

Enter the numerator: -15

Enter the denominator: 3

Error: Negative numbers are not allowed

Calculation Attempt Finished

***Problem 9.3: Write a Python program that prompts the user for a filename, raises a custom EmptyFileError if the file is empty, handles FileNotFoundError if the file doesn’t exist, counts and displays the number of words if the file has content, and always closes the file in a finally block displaying "File operation completed".***

**Algorithm:**

1. **Define the custom exception**

Create a class EmptyFileError that inherits from Exception.

1. **Prompt the user for input**

Ask the user to enter the filename.

1. **Check if the file exists and is empty**

Try to open the file in read mode.

If the file does not exist, handle FileNotFoundError.

If the file exists but is empty, raise EmptyFileError.

1. **Process the file content**

If the file is not empty, read its content.

Split the content into words and count them.

1. **Handle exceptions and finalize**

Catch FileNotFoundError and EmptyFileError, printing appropriate error messages.

Always close the file in the finally block and print "File operation completed".

**Program**:

import os

# 1. Define custom exception

class EmptyFileError(Exception):

pass

filename = input("Enter the filename: ")

file = None

try:

# 3. Try opening the file

file = open(filename, "r")

# 4. Check if file is empty

if os.path.getsize(filename) == 0:

raise EmptyFileError("The file is empty.")

# 5. Read and process content

content = file.read()

words = content.split()

print(f"The file has {len(words)} words.")

# 6. Handle exceptions

except FileNotFoundError:

print("Error: The specified file was not found.")

except EmptyFileError as e:

print(f"Error: {e}")

# 7. Finally block

finally:

if file:

file.close()

print("File operation completed")

**Output**:

### Sample Output1 – File Not Found

Enter the filename: sample.txt

Error: The specified file was not found.

File operation completed

### Sample Output 2 – Empty File

Enter the filename: empty.txt

Error: The file is empty.

File operation completed

### Sample Output 3 – File with Content

Enter the filename: notes.txt

The file has 12 words.

File operation completed

**Result:** Thus the program for Implement Exceptions and Exceptional handling is executed and verified successfully