Assignment 3

1. What is the role of try and exception block?

**Answer :** The try and except blocks in Python are used for handling exceptions or errors that may occur during the execution of a program. The purpose of using a try-except block is to handle exceptions preventing the program from terminating.

**Ex:**

try:

print(x)

except:

print("Some issue with x")

**Output:** Some issue with x

1. What is the syntax for a basic try-except block?

**Answer:**

try:

# Code block where an exception might occur

# ...

except ExceptionType:

# Code block to handle the exception

# ...

1. What happens if an exception occurs inside a try block and there is no matching except block?

**Answer:** If an exception occurs inside a **try** block and there is no matching **except** block to handle that specific exception, it will result in the program terminating with an error message.

**Ex:**

try:

    x = 10 / 0  # Division by zero, raises a ZeroDivisionError

except ValueError:

print("This block won't be executed for ZeroDivisionError")

**Output:**

ZeroDivisionError: division by zero

1. What is the difference between using a bare except block and specifying a specific exception type?

**Answer:** The difference between bare except block and specific exception type is

**Bare Except Block :**

Bare **except** block: A bare **except** block, denoted as **except:**, catches all types of exceptions. It is generally recommended to avoid using a bare **except** block without any specific exception type unless there is a specific reason to do so.

**Specific Exception Type :**

When specifying a specific exception type after the **except** keyword, such as **except ValueError:**, the **except** block will only catch exceptions of that particular type. By specifying the exception type, you can handle different types of exceptions in distinct ways.

**Ex:**

try:

x = int(input("Enter your number 1 : "))

y = int(input("Enter your number 2 : "))

result = x / y

print("Result is : ", result)

except ValueError:

print("You have entered float type number")

**Output:**

Enter your number 1 : 4

Enter your number 2 : 5.4

You have entered float type number.

1. Can you have nested try-except blocks in Python? If yes, then give an example.

**Answer :**

It is possible to have nested **try-except** blocks in Python. A nested **try-except** block is a construct where one **try** block is placed inside another **try** block, and each **try** block can have its corresponding **except** blocks.

Ex :

try:

outer\_value = 10 / 0 # Division by zero, raises a ZeroDivisionError

except ZeroDivisionError:

print("Outer exception: Division by zero!")

try:

inner\_value = int('abc') # Conversion to int, raises a ValueError

except ValueError:

print("Inner exception: Invalid conversion to int!")

print("Program continues...")

**Output :**

Outer exception: Division by zero!

Inner exception: Invalid conversion to int!

Program continues...

1. Can we use multiple exception blocks, if yes then give an example.

**Answer :** Yes, it is possible to use multiple **except** blocks in a **try-except** statement in Python. This allows you to handle different types of exceptions separately, providing specific error handling for each exception type.\

Ex:

try:

x = int(input("Enter a number: "))

result = 10 / x

print("Result:", result)

except ValueError:

print("Invalid input! Please enter a valid number.")

except ZeroDivisionError:

print("Error: Division by zero!")

except Exception as e:

print("An error occurred:", str(e))

1. Write the reason due to which following errors are raised:

a. EOFError

b. FloatingPointError

c. IndexError

d. MemoryError

e. OverflowError

f. TabError

g. ValueError.

**Answer :**

a. **EOFError**: This error is raised when the end of the file is reached unexpectedly while trying to read input from a file or standard input. It typically occurs when an input function like **input()** or **raw\_input()** is used.

b. **FloatingPointError**: This error is raised when a floating-point operation encounters an exceptional condition that cannot be handled. Invalid mathematical operation involving floating-point numbers.

c. **IndexError**: This error is raised when trying to access an index of a sequence that is out of range. It occurs when an invalid index value is used, such as accessing an element at an index that does not exist in the sequence.

d. **MemoryError**: This error is raised when the Python cannot allocate enough memory for an object or operation.

e. **OverflowError**: This error is raised when a mathematical operation exceeds the maximum representable value for a numeric type in Python.

f. **TabError**: This error is raised when there is an issue with the indentation of code using tabs and spaces inconsistently.

g. **ValueError**: This error is raised when a function or operation receives an argument of the correct type but with an invalid value.

1. Write code for the following given scenario and add try-exception block to it.

a. Program to divide two numbers

b. Program to convert a string to an integer

c. Program to access an element in a list

d. Program to handle a specific exception

e. Program to handle any exception

**Answer:**

Program to divide two numbers

try:

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

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

    result = num1 / num2

print("Result:", result)

except ZeroDivisionError:

    print("Error: Division by zero!")

except ValueError:

    print("Error: Invalid input! Please enter valid integers.")

Program to convert a string to an integer :

try:

number\_str = input("Enter a number: ")

number = int(number\_str)

print("Number:", number)

except ValueError:

print("Error: Invalid input! Please enter a valid integer.")

Program to access an element in the list:

try:

my\_list = [1, 2, 3, 4, 5]

index = int(input("Enter an index: "))

value = my\_list[index]

print("Value:", value)

except IndexError:

print("Error: Index out of range!")

except ValueError:

print("Error: Invalid index! Please enter a valid integer.")

Program to handle specific exception:

try:

num = int(input("Enter a number: "))

result = 100 / num

print("Result:", result)

except ZeroDivisionError:

print("Error: Division by zero!")

except ValueError:

print("Error: Invalid input! Please enter a valid integer.")

except Exception as e:

print("An error occurred:", str(e))

Program to handle any exception:

try:

x = int(input("Enter a number: "))

result = 10 / x

print("Result:", result)

except Exception as e:

print("An error occurred:", str(e))