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

Ans- The try and except blocks in Python are used for exception handling. The try block encloses code that might raise an exception, while the except block defines the actions to take if a specific exception occurs. If an exception occurs in the try block, control is transferred to the corresponding except block that matches the exception type. This allows you to gracefully handle errors and prevent the program from terminating unexpectedly. By using try and except, you can anticipate and handle exceptions, enabling robust error handling and recovery mechanisms in your code.

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

Ans-

try:

# Code that may raise an exception

# ...

except ExceptionType:

# Code to handle the exception

# ...

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

except block?

Ans- try:

# Code that may raise an exception

# ...

x=int(input('Enter the number 1\n'))

y=int(input('Enter the number 2\n'))

result= x/y

except ZeroDivisionError:

# Code to handle the exception

# ...

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

4. What is the difference between using a bare except block and specifying a specific

exception type?

Ans- Using a specific exception type in an except block allows for targeted handling of known exceptions, enabling precise error management and appropriate actions based on the type of exception. It enhances code clarity, maintainability, and aids in debugging by providing more specific information about the nature of the error. In contrast, a bare except block catches all exceptions, including unexpected ones, which can make it harder to diagnose and handle exceptions appropriately. It is generally recommended to be specific in exception handling to promote effective error management and facilitate understanding and maintenance of the code.

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

Yes, we can have the nested try- except blocks.

try:

# Outer try block

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

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

try:

# Inner try block

result = numerator / denominator

print("Result:", result)

except ZeroDivisionError:

print("Error: Division by zero!")

except ValueError:

print("Error: Invalid input!")

except Exception as e:

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

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

Ans- Yes we can have multiple exception blocks.

# Multiple exceptions

try:

n = int(input("Please enter the numerator: "))

d = int(input("Please enter the denominator: "))

result = n / d

print("Result:", result)

except ValueError:

print("Please enter valid integers for the numerator and denominator.")

except ZeroDivisionError:

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

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

a. EOFError

b. FloatingPointError

c. IndexError

d. MemoryError

e. OverflowError

f. TabError

g. ValueError

Ans- a. EOFError- Raised when the input() function hits an end-of-file condition (EOF).

b. FloatingPointError- Raised when a floating point operation fails.

c. IndexError- Raised when a sequence subscript (index) is out of range.

d. MemoryError- Raised when an operation runs out of memory.

e. OverflowError- Raised when the result of an arithmetic operation is too large to be expressed.

f. TabError- Raised when indentation contains mixed tabs and spaces.

g. ValueError- Raised when a built-in operation or function receives an argument that has the right type but an inappropriate value.

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

a. Program to divide two numbers

Ans- try:

# Code that may raise an exception

# ...

x=int(input('Enter the number 1\n'))

y=int(input('Enter the number 2\n'))

result=int( x/y)

print(result)

except ZeroDivisionError:

# Code to handle the exception

# ...

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

b. Program to convert a string to an integer

Ans- try:

string\_num = input("Enter a number: ")

integer\_num = int(string\_num)

print("Integer value:", integer\_num)

except ValueError:

print("Error: Invalid input! Cannot convert to integer.")

c. Program to access an element in a list

Ans- try:

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

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

element = my\_list[index]

print("Element at index", index, "is", element)

except IndexError:

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

except ValueError:

print("Error: Invalid index input!")

d. Program to handle a specific exception

Ans- try:

value = int(input("Enter a value: "))

if value < 0:

raise ValueError("Error: Negative value not allowed!")

print("Value:", value)

except ValueError as e:

print(e)

e. Program to handle any exception

Ans- try:

# Code that may raise an exception

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

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

result = numerator / denominator

print("Result:", result)

string\_num = input("Enter a number: ")

integer\_num = int(string\_num)

print("Integer value:", integer\_num)

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

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

element = my\_list[index]

print("Element at index", index, "is", element)

value = int(input("Enter a value: "))

if value < 0:

raise ValueError("Error: Negative value not allowed!")

print("Value:", value)

except Exception as e:

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