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

The try-except block in Python is used to handle exceptions (runtime errors) that may occur during the execution of code. The try block contains the code that may raise an exception, and the except block handles the exception by providing appropriate error-handling code.

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

try:

except ExceptionType:

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

except block?

If an exception occurs inside a try block and there is no matching except block to handle that specific exception type, the program will terminate abruptly, and an error message (traceback) will be displayed, indicating the type of exception and the line where the exception occurred.

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

exception type?

Using a bare except block: This catches all exceptions indiscriminately, regardless of the exception type. It is generally not recommended to use a bare except because it can make it challenging to diagnose and debug specific errors.

Specifying a specific exception type: This allows you to handle a particular type of exception and provides more targeted error handling. By specifying the exact exception type, you can handle different exceptions differently.

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

Yes, Nested try-except blocks allow you to handle exceptions at different levels of code execution.

try:

# Outer try block

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

try:

# Inner try block

result = 10 / x

print("Result:", result)

except ZeroDivisionError:

print("Error: Cannot divide by zero.")

except ValueError:

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

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

Yes you can use multiple exceptiob blocks

try:

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

result = 10 / num

print("Result:", result)

except ValueError:

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

except ZeroDivisionError:

print("Error: Cannot divide by zero.")

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

a. EOFError: Raised when there is no input from input() and an end-of-file condition is reached.

b. FloatingPointError: Raised when a floating-point operation fails due to an invalid mathematical operation, such as division by zero in float numbers.

c. IndexError: Raised when trying to access an index that is out of range in a sequence (list, tuple, etc.).

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 as a finite number in Python.

f. TabError: Raised when indentation contains inconsistent tabs and spaces.

g. ValueError: Raised when a function receives an argument of the correct data 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

try:

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

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

result = num1 / num2

print("Result:", result)

except ZeroDivisionError:

print("Error: Cannot divide by zero.")

b. Program to convert a string to an integer

try:

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

num = int(num\_str)

print("Converted number:", num)

except ValueError:

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

c. Program to access an element in a list

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

try:

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

value = my\_list[index]

print("Value at index", index, ":", value)

except IndexError:

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

d. Program to handle a specific exception

try:

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

if num < 0:

raise ValueError("Input must be a positive number.")

print("Number entered:", num)

except ValueError as ve:

print("Error:", ve)

e. Program to handle any exception

try:

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

result = 10 / x

print("Result:", result)

except Exception as e:

print("Error:", e)