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

The role of the try and except block in Python is to handle exceptions and prevent the program from crashing when an error occurs during its execution. The try block contains the code that might raise an exception, and the except block handles the exception by providing an alternative code path or displaying an error message.

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

The basic syntax for a try-except block in Python is as follows:

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?

If an exception occurs inside a try block and there is no matching except block to handle that exception, the program will terminate, and Python will display the default error message along with a traceback showing the location of the unhandled exception.

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

- Bare Except Block: Using a bare `except` block without specifying any exception type (e.g., `except:`) will catch all exceptions. It is generally not recommended to use a bare except block as it can hide bugs and make debugging difficult. It is better to catch specific exceptions to handle them properly.

- Specific Exception Type: When specifying a specific exception type (e.g., `except ValueError:`), the except block will only handle exceptions of that type. This allows you to handle different exceptions differently and provide more specific error handling.

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

Yes, you can have nested try-except blocks in Python. This allows you to handle exceptions at different levels of code execution.

Example of nested try-except blocks:

try:

# Outer try block

x = 10 / 0

except ZeroDivisionError:

print("Division by zero error.")

try:

# Inner try block

y = int("abc")

except ValueError:

print("Conversion error.")

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

Yes, you can use multiple except blocks to handle different types of exceptions separately.

Example of multiple exception blocks:

try:

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

result = 10 / x

print("Result:", result)

except ZeroDivisionError:

print("Division by zero error.")

except ValueError:

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

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

a. EOFError: Raised when the input() function hits an end-of-file condition (Ctrl+D on Unix/Linux, Ctrl+Z on Windows).

b. FloatingPointError: Raised when a floating-point operation fails, such as division by zero or an invalid operation.

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

d. MemoryError: Raised when a program runs out of available memory.

e. OverflowError: Raised when the result of an arithmetic operation exceeds the range of representable values for a numeric type.

f. TabError: Raised when inconsistent use of tabs and spaces in indentation is detected.

g. ValueError: Raised when a built-in operation or function receives an argument of the correct data type but an inappropriate value.

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

a. Program to divide two numbers

def divide\_numbers(a, b):

return a / b

try:

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

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

result = divide\_numbers(num1, num2)

print("Result:", result)

except ZeroDivisionError:

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

except ValueError:

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

b. Program to convert a string to an integer

try:

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

number = int(string\_input)

print("Number:", number)

except ValueError:

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

c. Program to access an element in a list

try:

my\_list = [1, 2, 3]

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

value = my\_list[index]

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

except IndexError:

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

except ValueError:

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

d. Program to handle a specific exception

try:

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

result = 10 / x

print("Result:", result)

except ZeroDivisionError:

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

e. Program to handle any exception.

try:

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

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

result = num1 / num2

print("Result:", result)

except Exception as e:

print("Error:", e)