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

**Ans 1:** In Python, the try and except blocks are used for error handling, allowing you to gracefully handle exceptions that might occur during the execution of your code. Here's how they work:

1. **try block**: This is where you place the code that might raise an exception. Python will attempt to execute the code inside this block.
2. **except block**: If an exception occurs inside the try block, Python will jump to the except block. Here, we can specify which exceptions we want to catch and provide code to handle them. If no specific exception is caught, it will catch all exceptions.

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

**Ans 2:** The basic syntax for a try-except block in Python is as follows:

try:

# Code that might 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 3:** If an exception occurs inside a try block and there is no matching except block to handle it, the exception will propagate up the call stack until it is handled by an appropriate except block, or if it's not handled anywhere in the call stack, the program will terminate with an error message.

For example:

try:

x = 10 / 0 # This will raise a ZeroDivisionError

except ValueError:

print("This won't be executed because the exception type doesn't match")

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

**exception type?**

**Ans 4:**

**Specific Exception Type:**

* When you specify a specific exception type in the except block (except ExceptionType:), it catches only the exceptions of that type or its subclasses.
* This allows you to handle different exceptions differently based on their types.
* It is considered good practice to be specific about the exceptions you catch to prevent unintentionally catching and handling exceptions that you might not have anticipated.

**Bare Except Block:**

* A bare except block (except:) catches all exceptions, regardless of their type.
* It can catch both expected and unexpected exceptions, making it potentially risky because it might hide errors that you didn't intend to catch.
* Using a bare except block is generally discouraged in most cases because it can make debugging more difficult and lead to unexpected behavior.

For Example :

try:

x = 10 / 0 # This will raise a ZeroDivisionError

except ZeroDivisionError:

print("Handling ZeroDivisionError")

except ValueError:

print("Handling ValueError")

except:

print("This won't be executed because there is a specific exception handler")

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

**Ans 5:** Yes, you can have nested try-except blocks in Python. This means you can place one try-except block inside another try block or except block. Here's an example to demonstrate nested try-except blocks:

try:

# Outer try block

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

try:

# Inner try block

result = 10 / x

print("Result:", result)

except ZeroDivisionError:

# Inner except block

print("Error: Division by zero occurred in inner try block")

except ValueError:

# Outer except block

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

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

**Ans 6:** Yes, you can use multiple except blocks to handle different types of exceptions in Python. Here's an example:

try:

# Code that might raise exceptions

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

result = 10 / x

print("Result:", result)

except ValueError:

# Handling ValueError (e.g., when input cannot be converted to an integer)

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

except ZeroDivisionError:

# Handling ZeroDivisionError (e.g., when dividing by zero)

print("Error: Division by zero occurred.")

except Exception as e:

# Handling other exceptions

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

Using multiple except blocks allows you to handle different types of exceptions more selectively and provide specific error messages or error handling logic tailored to each type of exception.

**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 7:**

**EOFError**: This error is raised when the input() function hits an unexpected end-of-file condition (EOF) without reading any data or when it reaches the end of a file while trying to read input from it.

**FloatingPointError**: This error occurs when a floating-point operation fails, typically due to the result being too large or too small to be represented as a floating-point number.

**IndexError**: This error is raised when you try to access an index that is out of range for the given sequence (e.g., list, tuple, string).

**MemoryError**: This error occurs when an operation cannot be completed due to lack of available memory. It indicates that the program has attempted to allocate more memory than the system can provide.

**OverflowError**: This error happens when the result of an arithmetic operation is too large (positive overflow) or too small (negative overflow) to be represented within the available memory or range of the data type.

**TabError**: This error is raised when indentation using tabs is inconsistent or mixed with spaces in Python code, particularly when Python expects consistent indentation levels to define blocks of code (e.g., in loops or function definitions).

**ValueError**: This error is raised when a built-in operation or function receives an argument with the right type but an inappropriate value, or when a function receives the correct type but an invalid value for that type.

**8. 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**

Ans 8:

**# 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: Division by zero is not allowed.")

except ValueError:

print("Error: Please enter valid integers.")

**# Program to convert a string to an integer**

try:

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

num = int(num\_str)

print("Integer:", num)

except ValueError:

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

**# Program to access an element in a list**

try:

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

index = int(input("Enter the index of the element to access: "))

value = my\_list[index]

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

except IndexError:

print("Error: Index out of range. Please enter a valid index.")

except ValueError:

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

**# Program to handle a specific exception**

try:

# Code that might raise a specific exception

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

result = 10 / x

print("Result:", result)

except ZeroDivisionError:

print("Error: Division by zero occurred.")

**# Program to handle any exception**

try:

# Code that might raise any exception

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

result = 10 / x

print("Result:", result)

except Exception as e:

print("An error occurred:", e)