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

A "try" and "exception" block, often referred to as a "try-catch" block, is a fundamental construct used to handle errors and exceptions gracefully. It is primarily used to support exception handling.

Here's how a try-catch block typically works:

1. The "try" block contains the code that might generate an exception or error. If an exception occurs within the "try" block, the program immediately jumps to the corresponding "catch" block.
2. The "catch" block contains code that handles the exception. It can log the error, display a user-friendly message, or take any other action necessary to manage the error situation.

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

```
Below is a general syntax template for a basic try-except block:
try:
    # Code that may raise an exception
    # ...
except ExceptionType as e:
    # 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 catch and handle that exception, the program will terminate, and an unhandled exception error will be raised.

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

1. Bare except Block: Catches and handles any exception, but it can make code less readable, hinder debugging, and is generally discouraged in favor of specifying specific exception types.
2. Specific Exception Type: Catches and handles a particular type of exception, providing precise error handling and making code more readable and maintainable.

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.

In [7]:

```python
try:
    # Outer try block
    numerator = int(input("Enter the numerator: "))
    denominator = int(input("Enter the denominator: "))

    try:
        # Inner try block
        result = numerator / denominator
    except ZeroDivisionError:
        print("Division by zero is not allowed.")
    except ValueError:
        print("Invalid input. Please enter valid numbers.")
    else:
        print("Result of division:", result)

except Exception as e:
    print("An error occurred:", e)
```

```
Enter the numerator: 45
Enter the denominator: 0
Division by zero is not allowed.
```

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 in Python.

In [9]:
```python
try:
    # Code that may raise exceptions
    num = int(input("Enter a number: "))
    result = 10 / num

except ZeroDivisionError:
    # Handle division by zero
    print("You cannot divide by zero.")

except ValueError:
    # Handle invalid input (non-integer)
    print("Invalid input. Please enter a valid number.")

except Exception as e:
    # Handle any other exception
    print("An error occurred:", e)

else:
    # Code to execute if no exceptions occurred
    print("Result:", result)
```

```
Enter a number: r
Invalid input. Please enter a valid number.
```

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 (End of File Error):

EOFError is raised when one of the built-in functions input () or raw_input () hits an end-of-file condition (EOF) without reading any data.

b. FloatingPointError:

This exception Raised when a floating-point operation encounters an exceptional condition that cannot be represented as a valid floating-point number.

c. IndexError:

Raised when you try to access an index in a sequence (e.g., list, tuple, or string) that is outside the valid range of indices. This can happen when you attempt to access an element at an index that doesn't exist.

d. MemoryError:

Raised when an operation cannot allocate enough memory for its execution. This occurs when the program has exhausted the available memory resources.

e. OverflowError:

Raised when a mathematical operation results in a value that is too large to be represented in the current numeric data type.

f. TabError:

Raised when inconsistent use of tabs and spaces for indentation is detected in Python code.

g. ValueError:

Raised when a function or operation receives an argument of the correct data type but with an inappropriate or invalid value.

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.

In [11]:

```python
# a. Program to divide two numbers:
try:
    numerator = float(input("Enter the numerator: "))
    denominator = float(input("Enter the denominator: "))
    result = numerator / denominator
    print("Result of division:", result)
except ZeroDivisionError:
    print("You cannot divide by zero.")
except ValueError:
    print("Invalid input. Please enter valid numbers.")
except Exception as e:
    print("An error occurred:", e)
```

```
Enter the numerator: 3
Enter the denominator: 0
You cannot divide by zero.
```

In [13]:

```python
# b. Program to convert a string to an integer:
try:
    num_str = input("Enter an integer: ")
    num_int = int(num_str)
    print("Converted integer:", num_int)
except ValueError:
    print("Invalid input. Please enter a valid integer.")
except Exception as e:
    print("An error occurred:", e)
```

```
Enter an integer: 4
Converted integer: 4
```

In [15]:
```python
# c. Program to access an element in a list:
try:
    lst = [1, 2, 3]
    index = int(input("Enter an index: "))
    value = lst[index]
    print("Value at index", index, "is", value)
except IndexError:
    print("Index is out of range.")
except ValueError:
    print("Invalid index. Please enter a valid integer.")
except Exception as e:
    print("An error occurred:", e)
```

```
Enter an index: 6
Index is out of range.
```

In [18]:
```python
# d. Program to handle a specific exception:
try:
    age = int(input("Enter your age: "))
    if age < 0:
        raise ValueError("Age cannot be negative.")
    print("Your age is:", age)
except ValueError as ve:
    print("ValueError:", ve)
except Exception as e:
    print("An error occurred:", e)
```

```
Enter your age: -9
ValueError: Age cannot be negative.
```

In [19]:
```python
# e. Program to handle any exception:
try:
    result = 10 / 0  # This will raise a ZeroDivisionError
except Exception as e:
    print("An error occurred:", e)
```

```
An error occurred: division by zero
```