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

* The role of try and exception block is to handle exceptions or errors that may occur during the execution of code. The main roles of try-except block are catching and handling exceptions, preventing program termination, providing alternative behavior and error reporting.

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

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

try:

# Code that might raise an exception

except ExceptionType:

# Code to handle the exception

1. 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 then the code will throw an error.

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

* The bare except block catches all exceptions, regardless of their type. It can be useful in certain situations where we want to handle any exception in a generic way or provide a fallback mechanism for any unexpected errors.

Specific exception type involves specifying the particular exception type that we want to handle within the except block. This way we can have a more fine-grained control over exception handling.

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

* Yes, it is possible to have nested try-except blocks in Python. This means that we can place one try-except block inside another try or except block.

For e.g.:

try:

# Outer try block

outer\_var = 10 / 0 # This will raise a ZeroDivisionError

except ZeroDivisionError:

print("Outer exception caught: Cannot divide by zero.")

try:

# Inner try block

inner\_var = int("abc") # This will raise a ValueError

except ValueError:

print("Inner exception caught: Cannot convert 'abc' to an integer.")

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

* Yes, we can use multiple except blocks to handle multiple exception types in Python. This allows you to specify different handling logic for different types of exceptions that may occur. Each except block is associated with a specific exception type, and the code within that block is executed when an exception of that type is raised.

For e.g.:

try:

# Code that may raise exceptions

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

result = 100 / num

print("Result:", result)

except ValueError:

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

except ZeroDivisionError:

print("Cannot divide by zero.")

except Exception as e:

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

1. Write the reason due to which following errors are raised:
   1. EOFError 🡪 This error is raised when an input operation reaches the end of a file (EOF) before it is expected to.
   2. FloatingPointError 🡪 This error is raised when a floating-point operation encounters an exceptional condition that cannot be handled, such as division by zero or an invalid mathematical operation.
   3. IndexError 🡪 This error is raised when attempting to access an index that is outside the range of valid indices for a list.
   4. MemoryError 🡪 This error is raised when an operation fails due to insufficient memory available for allocation.
   5. OverflowError 🡪 This error is raised when performing arithmetic operations that result in a value too large to be represented by the given data type.
   6. TabError 🡪 This error is raised when incorrect use of tabs and spaces is encountered in indentation.
   7. ValueError 🡪 This error is raised when a function receives an argument of the correct type but an inappropriate value.
2. Write code for the following given scenario and add try-exception block to it.
   1. Program to divide two numbers 🡪

try:

num1 = int(input(“Enter number 1: ”))

num2 = int(input(“Enter number 2: ”))

print(“Result: ”, num1 / num2)

expect ZeroDivisionError:

print(“Denominator (number 2) can’t be zero.”)

* 1. Program to convert a string to an integer 🡪

try:

str = “55S”

converted\_str = int(str)

print(type(converted\_str))

except ValueError:

print(“The string does not contain a number.”)

* 1. Program to access an element in a list 🡪

try:

list = [0, “iNeuron”, 4.5, True]

print(list[7])

except IndexError:

print(“Given index is not in the list.”)

* 1. Program to handle a specific exception 🡪

def check\_age(age):

if age < 0:

raise ValueError("Age cannot be negative")

elif age < 18:

raise ValueError("You must be at least 18 years old")

try:

age = int(input("Enter your age: "))

check\_age(age)

print("Access granted")

except ValueError as e:

print("Error:", str(e))

* 1. 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("An error occurred:", str(e))