1. The role of the try and except block is to handle and manage exceptions that may occur during the execution of code. It allows us to catch and handle specific errors or exceptional situations that might disrupt the normal flow of the program.

2.

try:

#Code that may raise an exception

except ExceptionType:

#Code to handle the exception

3. If an exception occurs inside a try block and there is no matching except block to handle that exception, the program will terminate and an unhandled exception error will be raised.

4. The difference between using a bare except block and specifying a specific exception type is that a bare except block will catch and handle any exception that occurs, regardless of its type. Specifying a specific exception type allows you to catch and handle only that specific type of exception, providing more control over exception handling.

5. Yes, nested try-except blocks are possible in Python.

try:

# Outer try block

try:

# Inner try block

# Code that may raise an exception

except ExceptionType:

# Inner except block

# Code to handle the exception raised in the inner try block

except ExceptionType:

# Outer except block

# Code to handle the exception raised in the outer try block

6. Yes, multiple except blocks can be used in Python to handle different types of exceptions.

try:

# Code that may raise exceptions

except ExceptionType1:

# Code to handle ExceptionType1

except ExceptionType2:

# Code to handle ExceptionType2

7. a. EOFError: Raised when there is an unexpected end of input or file.

b. FloatingPointError: Raised when a floating-point operation fails to execute, such as division by zero.

c. IndexError: Raised when trying to access an index that is out of range in a sequence.

d. MemoryError: Raised when an operation runs out of memory.

e. OverflowError: Raised when the result of an arithmetic operation exceeds the maximum representable value.

f. TabError: Raised when there is an improper use of indentation with tabs and spaces.

g. ValueError: Raised when a function receives an argument of the correct type but an inappropriate value.

8. a. Program to divide two numbers

try:

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

print("Number:", num)

except ValueError:

print("Error: Invalid input, could not convert to an integer")

c. Program to access an element in a list

try:

numbers = [1, 2, 3, 4, 5]

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

value = numbers[index]

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

except IndexError:

print("Error: Index out of range")

d. Program to handle a specific exception

try:

# Code that may raise a specific exception

except SpecificException as e:

print("Error:", e)

e. Program to handle any exception

try:

# Code that may raise exceptions

except Exception as e:

print("Error:", e)