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

Ans: The try and except blocks in programming are used for handling exceptions, which are unexpected or erroneous events that can occur during the execution of code. They allow you to handle errors and prevent your program from crashing when something goes wrong.

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

Ans: The syntax is as follows:

try:

# Code that might raise an exception

except SomeException:

# Code to handle the exception

1. What happens if an exception occurs inside a try block and there is no matching

except block?

Ans: If there’s no matching except block, the program will terminate and display an error message.

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

exception type?

Ans: Using a bare except block and specifying a specific exception type in a try-except statement have significant differences in terms of error handling and program behavior.

1. Bare except Block: A bare except block, also known as a generic except block, catches any exception that might occur within the try block. It doesn't specify a particular exception type. While it can be useful for capturing unexpected errors and preventing your program from crashing, it can also make debugging and error identification difficult because you're not explicitly handling specific exceptions
2. Specifying a Specific Exception Type: When you specify a specific exception type in an except block, you are explicitly handling only that particular type of exception. This approach provides better control and clarity in your error handling, allowing you to respond appropriately to different types of errors.
3. Can you have nested try-except blocks in Python? If yes, then give an example.

Ans: Yes, we can have nested try-except blocks in Python. Nesting try-except blocks allows you to handle different levels of exceptions in a structured manner. Each nested block can have its own set of try and except statements to handle specific exceptions at different levels of code execution.

Eg:

try:

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

try:

result = 10 / x

print("Result:", result)

except ZeroDivisionError:

print("Inner: Cannot divide by zero.")

except ValueError:

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

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

Ans: Yes, we can use multiple except blocks in a try-except statement to handle different types of exceptions.

Eg -

try:

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

result = 10 / x

print("Result:", result)

except ValueError:

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

except ZeroDivisionError:

print("Cannot divide by zero.")

except Exception as e:

print("An error occurred:", e)

1. Write the reason due to which following errors are raised:
   1. EOFError
   2. FloatingPointError
   3. IndexError
   4. MemoryError
   5. OverflowError
   6. TabError
   7. ValueError

Ans: The errors are as follows:

1. EOFError: This error occurs when the built-in input() function encounters an end-of-file condition, typically when trying to read input from the user and the user signals the end of input (e.g., by pressing Ctrl+D in Unix-like systems or Ctrl+Z in Windows). It indicates that there is no more input available to read.
2. FloatingPointError: This error occurs when a floating-point arithmetic operation results in an exceptional condition, such as division by zero or an overflow or underflow of the representable range for floating-point numbers.
3. IndexError: This error occurs when you try to access an index of a sequence (like a list or a string) that is out of range. In other words, you are trying to access an element at an index that doesn't exist.
4. MemoryError: This error occurs when the Python interpreter runs out of available memory to allocate for new objects. It indicates that your program has exhausted the available system memory.
5. OverflowError: This error occurs when a mathematical operation (such as addition or multiplication) results in a value that exceeds the maximum representable value for a numeric type.
6. TabError: This error occurs when the indentation in your Python code mix spaces and tabs inconsistently. Python uses indentation to define block structures, and mixing spaces and tabs for indentation can lead to a TabError.
7. ValueError: This error occurs when a function receives an argument of the correct data type but with an invalid value. For example, attempting to convert a string that is not a valid integer to an integer using int() will raise a ValueError.
8. Write code for the following given scenario and add try-exception block to it.
   1. Program to divide two numbers
   2. Program to convert a string to an integer
   3. Program to access an element in a list
   4. Program to handle a specific exception
   5. Program to handle any exception

Ans: The codes for the the following scenarios are as follows:

1. Program to divide two numbers -

try:

numerator = float(input("Enter the numerator: "))

denominator = float(input("Enter the denominator: "))

result = numerator / denominator

print("Result:", result)

except ZeroDivisionError:

print("Error: Cannot divide by zero.")

except ValueError:

print("Error: Please enter valid numeric inputs.")

1. Program to convert a string to an integer –

try:

input\_string = input("Enter an integer: ")

number = int(input\_string)

print("Successfully converted to integer:", number)

except ValueError:

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

1. Program to access an element in a list –

try:

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

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

value = my\_list[index]

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

except IndexError:

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

except ValueError:

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

1. Program to handle a specific exception:

try:

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

if x < 0:

raise ValueError("Number must be positive.")

print("Number entered:", x)

except ValueError as ve:

print("ValueError:", ve)

1. Program to handle any exception:

try:

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

result = 10 / x

print("Result:", result)

except Exception as e:

print("An error occurred:", e)