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

The try and except block in Python is used to handle exceptions. An exception is an event that occurs during the execution of a program that disrupts the normal flow of the program. Exceptions can be caused by a variety of things, such as invalid input, division by zero, or accessing a nonexistent object.

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

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 to handle that specific exception type, the exception will propagate up the call stack until it is caught by an appropriate except block, or if it reaches the top level of the program, it will result in an unhandled exception.

If there is no matching except block for the exception raised within the try block, the program will terminate and an error message will be displayed, indicating the type of the unhandled exception, along with a traceback that shows the sequence of function calls leading to the unhandled exception.

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

The difference between using a bare except block and specifying a specific exception type is that a bare except block will catch all exceptions, while specifying a specific exception type will only catch that specific exception.

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

it is possible to have nested try-except blocks in Python. This means that you can have a try-except block inside another try-except block. This allows for more fine-grained exception handling, where you can handle specific exceptions at different levels of the nested blocks.

try:

# Outer try block

outer\_var = int(input("Enter a number: "))

try:

# Inner try block

result = 10 / outer\_var

print("Result:", result)

except ZeroDivisionError:

print("Cannot divide by zero!")

except ValueError:

print("Invalid input!")

except Exception as e:

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

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

Yes, you can use multiple exception blocks in Python. This means that you can have multiple except blocks in a single try block.

For example, the following code tries to open a file and read its contents. If the file does not exist, an exception will be raised. The try block will catch the FileNotFoundError exception and print a message. The except block will also catch the IOError exception and print a message.

try:

with open("my\_file.txt", "r") as f:

contents = f.read()

except FileNotFoundError:

print("The file my\_file.txt does not exist.")

except IOError:

print("An I/O error occurred.")

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

* **EOFError** is raised when the end of a file is reached unexpectedly. This can happen if the file is empty, or if the user presses Ctrl+D to end the input.
* **FloatingPointError** is raised when a floating-point operation fails. This can happen if the operation results in a number that is too large or too small to be represented by a floating-point number.
* **IndexError** is raised when an index is used to access an element of a sequence that is out of bounds. For example, if you try to access the 10th element of a list that only has 5 elements, an IndexError will be raised.
* **MemoryError** is raised when a program runs out of memory. This can happen if the program is trying to allocate too much memory, or if the operating system is running low on memory.
* **OverflowError** is raised when the result of an arithmetic operation is too large to be represented by the data type of the operands. For example, if you try to add two very large integers, an OverflowError will be raised.
* **TabError** is raised when a tab character is used in a place where a space is expected. For example, if you try to define a variable with the name tab, an TabError will be raised.
* **ValueError** is raised when a function or operation is passed an argument of an incorrect type or value. For example, if you try to call the int() function with a string argument, a ValueError will be raised.

1. 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

**a. Program to divide two numbers**

try:

x = 10

y = 0

z = x / y

except ZeroDivisionError:

print("Division by zero")

**b. Program to convert a string to an integer**

try:

x = "hello"

y = int(x)

except ValueError:

print("The string 'hello' cannot be converted to an integer")

**c. Program to access an element in a list**

try:

my\_list = [1, 2, 3]

print(my\_list[4])

except IndexError:

print("The index 4 is out of bounds")

**d. Program to handle a specific exception**

try:

x = 10

y = 0

z = x / y

except ZeroDivisionError:

print("Division by zero")

else:

print("The division was successful")

**e. Program to handle any exception**

try:

x = 10

y = 0

z = x / y

except:

print("An exception occurred")