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

Answer:

The try block is used to test a block of code whereas an exception block is used to handle the error.

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

Answer:

try:  
 # code that may cause exception  
except:  
 # code to run when exception occurs

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

except block?

Answer:

If there is no matching expect block then exception is not handled and the rest of the program doesn’t work.

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

Answer:

A bare except block catches all exceptions regardless of their type. It is considered a bad practice because it can hide critical errors.

An except block is used to handle the error.

try:

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

result = 10 / x

print("Result:", result)

except ValueError:

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

Output:

Enter a number: 100

Result: 0.1

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

Answer:

Yes, we can have nested try-except blocks in python.

try:

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

try:

inner\_value = int(input("Enter another number: "))

result = outer\_value / inner\_value

print("Result:", result)

except ZeroDivisionError:

print("Cannot divide by zero.")

except ValueError:

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

Output:

Enter a number: 200

Enter another number: 100

Result: 2.0

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

Answer:

Yes, we can have multiple exception blocks .

try:

f = open('missing')

except OSError:

print('It failed')

except FileNotFoundError:

print('File not found')

7. Write the reason due to which following errors are raised:

a. EOFError

“end of file” error occurs when a compiler detects unfinished statement or block of code.

b. FloatingPointError

Floating point error occurs when the binary representation of the number is not exact.

c. IndexError

Index error occurs when the index specified is invalid. that is when the index goes beyond the bounds.

d. MemoryError

Memory error occurs when the interpreter is out of memory to allocate space for the python program.

e. OverflowError

The overflow error occurs mainly during arithmetic operations when the values go beyond its limit.

f. TabError

Tab error occurs when the interpreter detects internal error.

g. Value Error

Value error is raised when we assign wrong values to the object.

8. Write code for the following given scenario and add try-exception block to it.

a. Program to divide two numbers

Answer:

def divide\_numbers():

try:

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

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

result = dividend / divisor

print("The result of the division is:", result)

except ZeroDivisionError:

print("Error: Division by zero is not allowed.")

except ValueError:

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

divide\_numbers()

Output:

Enter the dividend: 20

Enter the divisor: 2

The result of the division is: 10.0

c. Program to access an element in a list

def access\_list\_element(lst, index):

try:

element = lst[index]

print("The element at index", index, "is:", element)

except IndexError:

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

except TypeError:

print("Error: Invalid input. Please provide a list and an integer index.")

my\_list = [10, 20, 30, 40, 50]

index\_input = input("Enter an index: ")

try:

index = int(index\_input)

access\_list\_element(my\_list, index)

except ValueError:

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

Output:

Enter an index: 3 The element at index 3 is: 40

d. Program to handle a specific exception

try:

num1 = int(input("Enter the first number: "))

num2 = int(input("Enter the second number: "))

result = num1 / num2

print("Result:", result)

except ZeroDivisionError:

print("Error: Division by zero is not allowed.")

except ValueError:

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

except Exception as e:

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

Output:

Enter the first number: 20

Enter the second number: 10

Result: 2.0

e. Program to handle any exception

try:

numerator = 10

denominator = 0

result = numerator / denominator

print("Result:", result)

except Exception as e:

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

finally:

print("This will always be executed, regardless of exceptions.")

Output:

An error occurred: division by zero This will always be executed, regardless of exceptions.