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

Ans: In Python, you use **try** and **except** blocks. The try block contains the code that might raise an exception, and the **except** block contains the code that will be executed if an exception is raised.

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

Ans: try:

Print(x)

Except:

Print(“There is no X”)

Else:

Print(“no error”)

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

Ans: If an exception occurs during execution of the [try](https://docs.python.org/3/reference/compound_stmts.html#try) clause, the rest of the clause is skipped. Then, if its type matches the exception named after the [except](https://docs.python.org/3/reference/compound_stmts.html#except) keyword, the except clause is executed, and then execution continues after the try/except block.

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

Ans: he point of specifying the Exception is that ONLY that Exception will be caught, if you do not specify any Exception, then ALL Errors and Exceptions will be caught potentially masking bugs

1. 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

x = 10

y = 0

try:

print("outer try block")

try:

print("nested try block")

print(x/y)

except TypeError as te:

print("nested except block")

print(te)

except ZeroDivisionError as ze:

print("outer except block")

print(ze)

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

Ans: yes we can use multiple exception blocks

try:

n = int(input("please enter the numerator: "))

d = int(input("please enter the denominator: "))

result = n/d

print("Result",result)

except ValueError:

print("pelase enter vaild integer for numerator and denominator.")

except ZeroDivisonError:

print("Divison by zero is not allowed")

1. Write the reason due to which following errors are raised:
   1. EOFError = EOFError is short for "End-of-Line Error." This error occurs when Python has reached the end of user input without receiving any input
   2. FloatingPointError = It's a problem caused when the internal representation of floating-point numbers, which uses a fixed number of binary digits to represent a decimal number
   3. IndexError = Raised when a sequence subscript is out of range.
   4. MemoryError = Raised when an operation runs out of memory.
   5. OverflowError = Raised when the result of an arithmetic operation is too large to be expressed by the normal number format.
   6. TabError = inconsistent use of tabs and spaces in indentation" occurs when we mix tabs and spaces in the same code block
   7. ValueError = Raised when a built-in operation or function receives an argument that has the right 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:

n = int(input("please enter the numerator: "))

d = int(input("please enter the denominator: "))

result = n/d

print("Result",result)

except ValueError:

print("pelase enter vaild integer for numerator and denominator.")

* 1. Program to convert a string to an integer

a\_string = "10"

try:

an\_integer = int(a\_string)

print(an\_integer)

except ValueError:

print("its a value error")

* 1. Program to access an element in a list

try:

list1 = [1,2,3,4]

print(list1[3])

except IndexError:

print("Wrong Index")

* 1. Program to handle a specific exception

try:

print("Good morning today is good day")

except:

print("some issue")

else:

print("no issue")

* 1. Program to handle any exception

try:

numerator = 10

denominator = 0

result = numerator/denominator

print(result)

except:

print("Error: Denominator cannot be 0.")

finally:

print("This is finally block.")