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

Try block is used to contain the code which can throw error during runtime and exception block is used to handle the exception.

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

Below is the syntax for a basic try-except block

try:

# Some Code

except:

# Executed if error in the

# try block

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

If there is no matching except block then it is passed on to the outer **try** statements. If the exception is left unhandled, then the execution stops.

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

In a bare except block all kinds of exception will be handled but if we are specifying a specific exception type then only that exception will be handled and rest will be left unhandled.

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

Yes, we can have nested try-except blocks in python. Below is an example.

try:

print("outer try block")

print(10/0)

try:

print("Inner try block")

except ZeroDivisionError:

print("Inner except block")

finally:

print("Inner finally block")

except:

print("outer except block")

finally:

print("outer finally block")

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

Yes, we can have multiple exception blocks.

try:

    f = open('missing')

    except OSError:

        print('It failed')

    except FileNotFoundError:

        print('File not found')

1. Write the reason due to which following errors are raised:
   1. EOFError- **EOFError** is raised when one of the built-in functions input() or raw\_input() hits an end-of-file condition (EOF) without reading any data.
   2. FloatingPointError- Floating-point errors can occur due to the limitations of representing real numbers using the binary floating-point format.
   3. IndexError- IndexError is a type of exception in python that is raised by the system when the index specified as subscript does not lie in the range of indices of bounds of a list.
   4. MemoryError- This error occurs when you have run out of **memory** in your RAM for your code to execute.
   5. OverflowError- An **OverflowError** exception is raised when an [arithmetic operation](https://www.pylenin.com/blogs/python-arithmetic-operators/) exceeds the limits to be represented.
   6. TabError- I**nconsistent use of tabs and spaces in indentation** error is raised when you try to indent code using both spaces and tabs.
   7. ValueError- Python ValueError is raised when a function receives an argument of the correct 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

def division(a,b):

try:

div=a/b

print(div)

except Exception as e:

print("error is: ",e)

* 1. Program to convert a string to an integer

def str\_to\_int(txt):

try:

val=int(txt)

print(type(val))

except Exception as e:

print("error is: ",e)

* 1. Program to access an element in a list

def listchk(txt,val):

try:

if val in txt:

print(val,"in list ",txt)

except Exception as e:

print("error is: ",e)

function call

l=[3,8,10,23,50]

listchk(l,8)

* 1. Program to handle a specific exception

def division(a,b):

try:

div=a/b

print(div)

except ZeroDivisionError:

print("Dividing number by zero ")

* 1. Program to handle any exception

def division(a,b):

try:

div=a/b

print(div)

except ZeroDivisionError:

print("Dividing number by zero ")

except exception as e:

print("error is ",e)