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

**A: The try block lets you test a block of code for errors. The except block lets you handle the error. The else block lets you execute code when there is no error. The finally block lets you execute code, regardless of the result of the try- and except blocks.**

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

**A: The try block lets you test a block of code for errors. The except block lets you handle the error.**

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

except block?

**A: If any exception occurs, the try clause will be skipped and except clause will run. If any exception occurs, but the except clause within the code doesn't handle it, it is passed on to the outer try statements. If the exception is left unhandled, then the execution stops.**

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

exception type?

A: **The bare except is** keyboard interrupts and system exits are propagated using exceptions

**specifying a specific exception** For each try block, there can be zero or more except blocks. Multiple except blocks allow us to handle each exception differently. The argument type of each except block indicates the type of exception that can be handled by it.

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

**A: We can have nested try-except blocks in Python. In this case, if an exception is raised in the nested try block, the nested except block is used to handle it. In case the nested except is not able to handle it, the outer except blocks are used to handle the exception.**

**Example:**

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)

**Output:**

outer try block

nested try block

outer except block

division by zero

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

**A: We can use multiple exception blocks.**

**Example:**

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)

**Output:**

outer try block

nested try block

outer except block

division by zero

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

**a. 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. The reason that EOFError occurs is that Python attempts to print out your input in variable string when no data is given

**b. FloatingPointError :** Python FloatingPointError is an error that occurs when a floating-point calculation is attempted that is not supported by the available hardware or software. It occurs when trying to calculate a number that is too large or too small to represent as a floating-point number.

c**. IndexError:** This error occurs when an attempt is made to access an item in a list at an index which is out of bounds. The range of a list in Python is [0, n-1], where n is the number of elements in the list.

**d. MemoryError :** A MemoryError means that the interpreter has run out of memory to allocate to your Python program. This may be due to an issue in the setup of the Python environment or it may be a concern with the code itself loading too much data at the same time.

**e. OverflowError:** OverflowError occurs when any operations like arithmetic operations or any other variable storing any value above its limit then there occurs an overflow of values that will exceed it's specified or already defined limit. In general, in all programming languages, this overflow error means the same.

**f. TabError :** inconsistent use of tabs and spaces in indentation" occurs when we mix tabs and spaces in the same code block. To solve the error, remove the spacing and only use tabs or spaces, but don't mix the two in the same code block.

**g. ValueError:** ValueError is raised when the wrong value is assigned to an object. This can happen if the value is invalid for a given operation, or if the value does not exist. For example, if a negative integer is passed to a square root operation, a ValueError is raised.

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

**a. Program to divide two numbers**

def divide(x, y):

try:

result = x // y

print("Yeah ! Your answer is :", result)

except ZeroDivisionError:

print("Sorry ! You are dividing by zero ")

divide(3, 2)

**Output:**

Yeah ! Your answer is : 1

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

a\_string = "12"

**try**:

an\_integer = int(a\_string)

**print**(an\_integer)

**except** ValueError:

**print**("Catch Error")

**OUTPUT:**

12

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

try:

list1 = [11, 12, 14, 16]

print(list1[3])

except indexError:

print('Wrong Index')

**Output:**

16

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

try:

file\_location = "my\_file.txt"

with open(file\_location, "r") as file:

contents = file.read()

except FileNotFoundError:

print("File you are looking for is not there")

**Output:**

File you are looking for is not there

**e. Program to handle any exception**

try:

even\_numbers = [2,4,6,8]

print(even\_numbers[5])

except ZeroDivisionError:

print("Denominator cannot be 0.")

except IndexError:

print("Index Out of Bound.")

**Output:**

Index Out of Bound.