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

- The try and except block is used to handle errors in Python. The try block contains the code that you want to run, and the except block contains the code that you want to run if an error occurs.

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

-
  ```
  try:
        #code
  except #error name :
                print("the exception occur")
  ```

3. 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, the exception will be raised to the next level of code. This means that the exception will be handled by the code that called the try block.

-
  ```
  try:
     print(10 / 0)
  except ZeroDivisionError:
     raise
  ```

4. 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 any exception, while a specific exception type will only catch that specific exception. For example, the following code will catch any exception:

-
  ```
  try:
      print(10 / 0)
  except:
  print("An error occurred")
  ```

- The output of this code will be:

  ```
  An error occurred
  ```

- The following code will only catch a ZeroDivisionError exception:

```
 try:
   print(10 / 0)
except ZeroDivisionError:
    print("You can't divide by zero!")
```

- The output of this code will be:

  ```
  You can't divide by zero!
  ```

- In general, it is better to specify a specific exception type when you are using an except block. This will make your code more readable and easier to debug.

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

```
try:
    try:
        print(10 / 0)
    except ZeroDivisionError:
        print("You can't divide by zero!")
except:
    print("An error occurred")
```

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

-
  ```
  try:
     print(10 / 0)
  except ZeroDivisionError:
     print("You can't divide by zero!")
  except ValueError:
     print("You can't convert a string to a number!")
  ```

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

a. EOFError :

- The EOFError exception is raised when the end of a file is reached unexpectedly. This can happen when you are trying to read from a file that has been closed, or when you are trying to read from a file that does not exist.

b.FloatingPointError:

- The FloatingPointError exception is raised when a floating-point operation cannot be performed. This can happen for a variety of reasons, such as when a number is too large or too small, or when a number is divided by zero. To avoid this exception, you should always check the values of your floating-point numbers before performing operations on them. You can do this using the `isfinite()` function, which returns True if a number is finite and False if it is infinite or NaN (not a number).

c. IndexError:

- The IndexError exception is raised when you try to access an element of a list, tuple, or other sequence that does not exist. For example, the following code will raise an IndexError The IndexError exception can also be raised when you try to access an element of a string that does not exist

d. MemoryError :

- The MemoryError exception is raised when a Python program attempts to allocate more memory than is available. This can happen for a variety of reasons, such as when a program tries to create a very large array or when a program tries to recurse too deeply. To avoid MemoryErrors, you should be careful about how much memory your program uses. You should also be aware of the maximum amount of memory that is available on your system.

e. OverflowError :

- The OverflowError exception is raised when a Python program attempts to perform an operation that results in a number that is too large or too small to be represented by the computer's hardware. This can happen when a program tries to divide a number by zero, or when a program tries to raise a number to a power that is too large. To avoid OverflowErrors, you should be careful about the operations that you perform on your numbers. You should also be aware of the limitations of your computer's hardware.

f. TabError:

- The TabError exception is raised when a Python program tries to use a tab character in a place where a space character is expected. This can happen when a program is trying to read or write a file, or when a program is trying to parse a string. To avoid TabErrors, you should always use spaces instead of tabs when writing Python code. You should also make sure that your code is properly indented.

g. ValueError :

- The ValueError exception is raised when a Python program attempts to use a value that is not of the correct type or that is out of range. This can happen when a program tries to convert a string to an integer, or when a program tries to access an element of an array that is out of bounds. To avoid ValueErrors, you should make sure that your values are of the correct type and that they are within the correct range. You should also make sure that you are using the correct operators and functions for the types of values that you are working with.

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

a. Program to divide two numbers :

```
try:
    a = int(input("enter the number : "))
    b = int(input("enter the second : "))

     result = a / b
    print(result)

except ZeroDivisionError:
      print("you entered zero")
```

b. Program to convert a string to an integer :

```
try:
    a = int(input("enter the number : "))
     b = int(input("enter the second : "))

    result = a / b
     print(result)

except ValueError:
        print("you entered wrong datatype ")
```

c. Program to access an element in a list :

```
try:
        x = [10,20,30,40,50]
        print(x[6])
except IndexError:
        print("you entered index out of range")
```

d. Program to handle a specific exception e. Program to handle any exception:

```
try:
    x= {"name":"alice","age":22}
    print(x["city"])

except KeyError:
      print("you entered key not found")
```