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

   These blocks are used for exception handling. This blocks manage and handle exceptions which arises during program execution.

   Try block allows to write code that may potentially can raise an exception. When exception raise within the try block, the execution of this block halted and control is transferred to corresponding except block.

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

```
try:
    #Code that might raise an exception
except Exception Type:
    # Code to handle the exception
```

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

   Then the exception will not be caught and cannot be handle gracefully. And when that exception occurs, python's default exception handling mechanism takes over and this print the traceback to the console,This shows the sequence of function calls which leads to unhandled exception.

   If it happens in already produced application, then that application crashes.

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

   Bare except block catches all the exception that may occur in corresponding try block. Thus, prevent the program from crashing. However, it makes debugging and resolving issues very difficult.

   Whereas specifying exception type will raise that specific exception and thus make debugging and resolving issues-task easy.

   It is always recommended to use specific exception type while programing.

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

   Yes, we have nested try-except blocks.

```
try:
    a=int(input("Enter the numerator:"))
    try:
        b=int(input("Enter the denominator:"))
        print(f" The result is:{a/b}")
    except ValueError:
        print("Entered denominator value not valid")
    except ZeroDivisionError:
        print("You cannot input Zero as dinominator")
except ValueError:
    print("Please enter valid numerator value")
```

```
output➔ Enter the numerator:12.1
Please enter valid numerator value
```

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

   Yes, we can use multiple except blocks.

```
try:
```

```
        num1 = int(input("Enter a dividend: "))
        num2 = int(input("Enter a divisor: "))
        result = num1 / num2
    except ValueError:
        print("Please enter valid integer values.")
    except ZeroDivisionError:
        print("Cannot divide by zero.")
    else:
        print("The division result is:", result)
```

Output➔

```
 Enter a dividend: 14
Enter a divisor: 5
The division result is: 2.8
```

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

   a. EOFError: This error is raised when the input() function or a similar function reaches the end of the file (EOF) before obtaining the expected input. It occurs when attempting to read from an input source, such as a file or standard input, and the end of the input is encountered unexpectedly.

   b. FloatingPointError: This error is raised when a floating-point calculation or operation fails to produce a valid result. It can occur due to various reasons, such as dividing a number by zero, taking the square root of a negative number, or performing an invalid mathematical operation on floating-point numbers.

   c. IndexError: This error is raised when you try to access an index of a sequence (e.g., a list, tuple, or string) that is outside the valid range. It occurs when you attempt to access an element using an index that is either negative or greater than or equal to the length of the sequence.

   d. MemoryError : This error is raised when an operation fails due to insufficient memory. It occurs when a program or process tries to allocate more memory than is available in the system.

   e. OverflowError: This error is raised when the result of an arithmetic operation exceeds the maximum representable value for a numeric type. It typically occurs when performing calculations that lead to a value that is too large to be stored within the range of the data type.

   f. TabError: This error is raised when there are inconsistencies or incorrect usage of tabs and spaces for indentation in Python code. It typically occurs when mixing tabs and spaces for indentation or when the indentation level is not consistent within a block of code.

   g. ValueError: This error is raised when a function receives an argument of the correct data type but with an invalid value. It typically occurs when a function or operation expects a certain range or format for input values and receives a value that does not comply with those requirements.

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("Please Enter the numerator:"))
    b =  int(input("Please Enter the denominator:"))
    result = a/b
    print(f"Result after division is {result}")
except ZeroDivisionError:
    print("cannot divide by zero")
except ValueError:
    print("Enter correct value")
```

```
Output➔ Please Enter the numerator:12
Please Enter the denominator:2.1
Enter correct value
```

    b. Program to convert a string to an integer

```
def string_conversion(string):
    try:
        integer_value = int(string)
        return integer_value
    except ValueError:
        print("Enter the valid string")
```

Output1➔ string_conversion("three")

```
Enter the valid string
```

Output2➔ string_conversion("13")

```
13
```

    c. Program to access an element in a list

```
try:
    lst1=["Animal",1, 45, "pen",23,6.5,True]
    index= int(input("Enter the index value "))
    print(f"The {index} index value is {lst1[index]}")
except IndexError:
    print("Enter index within the range")
except ValueError:
    print("Enter the valid value")
```

```
Output1➔ Enter the index value jef
Enter the valid value

Output2➔ Enter the index value 4
The 4 index value is 23
```

d. Program to handle a specific exception

```
def divide_numbers(a, b):
    try:
        result = a / b
        return result
    except ZeroDivisionError:
        print("Error: Division by zero is not allowed.")
        return None

# Example usage
num1 = int(input("Enter the numerator: "))
num2 = int(input("Enter the denominator: "))

division_result = divide_numbers(num1, num2)

if division_result is not None:
    print("The division result is:", division_result)
```

```
output➔ Enter the numerator: 12
Enter the denominator: 0
Error: Division by zero is not allowed.
```

e. Program to handle any exception

```
def perform_operation(a, b):
    try:
        result = a / b
        return result
    except Exception as e:
        print("An error occurred:", str(e))
        return None

# Example usage
num1 = int(input("Enter the first number: "))
num2 = int(input("Enter the second number: "))

operation_result = perform_operation(num1, num2)

if operation_result is not None:
    print("The result of the operation is:", operation_result)
```

```
Output➔ Enter the first number: 12
Enter the second number: 0
An error occurred: division by zero
```