Assignment 10

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

   Ans:

   The **"try"** and **"except"** blocks in Python are used for handling exceptions, which are unexpected or erroneous situations that may occur during the execution of a program. The main role of the **"try"** and **"except"** blocks is to catch and handle exceptions in a structured and controlled manner, allowing the program to gracefully handle errors and prevent crashes or unexpected behavior.

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

   Ans:

```
try:
    # Code that may raise an exception
    # ...
except ExceptionType:
    # Code to handle the exception
    # ...
```

   Example:

```
try:
    num1 = int(input("Enter a number: "))
    num2 = int(input("Enter another number: "))
    result = num1 / num2
    print("Result:", result)
except ZeroDivisionError:
    print("Error: Cannot divide by zero.")
```

```
Enter a number:  0
Enter another number:  0
Error: Cannot divide by zero.
```

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

Ans:

If an exception occurs inside a **“try”** block and there is no matching **“except”** block to handle that exception, the exception is not caught or handled within the **“try”** block. Instead, the exception propagates up the call stack to the next outer **“try”** block, or if there is no outer **“try”** block to catch the exception, it results in an unhandled exception error.

```
try:
    result = 10 / 0
except ValueError:
    print("Caught ValueError")  # This except block will not catch the ZeroDivisionError
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
Cell In [40], line 2
      1 try:
----> 2     result = 10 / 0
      3 except ValueError:
      4     print("Caught ValueError")

ZeroDivisionError: division by zero
```

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

Ans:
The difference between using a bare except block and specifying a specificexception type are-

Bare except block	Specific exception type

• A bare **"except"** block catches and handles any type of exception that occurs within the corresponding **"try"** block.
• It does not specify the type of exception to be caught, which means it will catch all exceptions, including built-in exceptions and custom exceptions.
• While a bare **"except"** block can be convenient for catching any exception, it may make it harder to identify and handle specific exceptions appropriately.
• Using a bare **"except" block** is generally discouraged, as it can lead to unintended consequences and make it difficult to debug and maintain code.
 	
`try:`
`  # Code that may raise an exception`
`  # ...`
`except:`
`  # Code to handle any exception`
`  # ...`
	• Specifying a specific exception type in an **"except"** block allows you to catch and handle only that particular type of exception, providing more control and specificity in error handling.
	• By specifying the exception type, you can handle different exceptions differently based on their specific characteristics or requirements.
	• It helps in providing targeted error messages, custom handling logic, or appropriate actions for specific exceptions.
	• It also allows other exceptions that are not specified to propagate up the call stack for potential handling in outer **"try"** blocks or by a default exception handler.
	
	`try:`
	`  # Code that may raise an exception`
	`  # ...`
	`except ValueError:`
	`  # Code to handle a specific ValueError`
	`  # ...`

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

Ans:

Yes, it is possible to have nested try-except blocks in Python. This means that you can place one try-except block inside another try or except block. Nesting try-except blocks allows for handling exceptions at different levels of the code, providing more fine-grained control over exception handling.

```
try:
    print("Outer try block")
    try:
        print("Inner try block")
        result = 10 / 0
    except ValueError:
        print("Caught ValueError in inner except block")
except ZeroDivisionError:
    print("Caught ZeroDivisionError in outer except block")
```

```
Outer try block
Inner try block
Caught ZeroDivisionError in outer except block
```

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

Ans:

Yes, we can use multiple **"except"** blocks to handle different types of exceptions in Python. Each **"except"** block can specify a different exception type to handle, allowing for specific error handling based on the type of exception that occurs.

```
try:
    x = int(input("Enter a number: "))
    result = 10 / x
    print("Result:", result)
except ValueError:
    print("Invalid input. Please enter a valid number.")
except ZeroDivisionError:
    print("Error: Cannot divide by zero.")
except Exception as e:
    print("An error occurred:", str(e))
```

```
Enter a number:  0
Error: Cannot divide by zero.
```

Error as we have enter Zero"0" in either numerator and denominator

```
try:
    x = int(input("Enter a number: "))
    result = 10 / x
    print("Result:", result)
except ValueError:
    print("Invalid input. Please enter a valid number.")
except ZeroDivisionError:
    print("Error: Cannot divide by zero.")
except Exception as e:
    print("An error occurred:", str(e))
```

```
Enter a number:  a
Invalid input. Please enter a valid number.
```

Error as we have enter any alphabet in either numerator and denominator

7. **Write the reason due to which following errors are raised:**
    a. **EOFError**
    b. **FloatingPointError**
    c. **IndexError**
    d. **MemoryError**
    e. **OverflowError**
    f. **TabError**
    g. **ValueError**

Ans:
The reason due to which following errors are raised are--

a. **EOFError:**

Raised when there is an attempt to read beyond the end of a file or input stream (End-of-File Error).
Occurs when an input operation is performed, but no more data is available to be read.

b. **FloatingPointError:**

Raised when a floating-point operation fails to execute correctly.
Can occur due to various reasons such as division by zero, invalid operation, or overflow during floating-point arithmetic.

c. **IndexError:**

Raised when a sequence (such as a list or string) is accessed using an invalid index.
Occurs when an index is used to access an element that is outside the range of valid indices for the given sequence.

d. **MemoryError:**

Raised when an operation fails due to insufficient memory allocation.
Occurs when the program tries to allocate more memory than the system can provide.

e. **OverflowError:**

Raised when the result of an arithmetic operation exceeds the maximum representable value for a numeric type.
Occurs when the value computed is too large to be stored within the available memory or data type.

f. **TabError:**

Raised when inconsistent use of tabs and spaces is encountered in indentation.
Occurs when there is a mix of tabs and spaces or incorrect indentation in the code.

g. **ValueError:**

Raised when an operation receives an argument of the correct type but an inappropriate value.
Occurs when a function or operation is called with an argument that is of the correct type but is outside the acceptable range or does not meet the required conditions.

h. **Write code for the following given scenario and add try-exception block to it.**
   a. **Program to divide two numbers**
   b. **Program to convert a string to an integer**
   c. **Program to access an element in a list**
   d. **Program to handle a specific exception**
   e. **Program to handle any exception**

Ans:

a. Program to divide two numbers

```python
#Divide two numbers

try:
    numerator = float(input("Enter the numerator: "))
    denominator = float(input("Enter the denominator: "))

    result = numerator / denominator
    print("Result:", result)

except ZeroDivisionError:
    print("Error: Cannot divide by zero.")
except ValueError:
    print("Error: Invalid input. Please enter numeric values.")
```

```
Enter the numerator:  12
Enter the denominator:  3
Result: 4.0
```

b. Program to convert a string to an integer

```python
#Convert a string to an integer

try:
    string_num = input("Enter a number: ")
    num = int(string_num)
    print("Converted integer:", num)
except ValueError:
    print("Error: Invalid input. Please enter a valid integer.")
```

```
Enter a number:  6
Converted integer: 6
```

c. Program to access an element in a list

```python
#Access an element in a list

my_list = [1, 2, 3, 4, 5]

try:
    index = int(input("Enter the index of the element to access: "))
    element = my_list[index]
    print("Element at index", index, "is:", element)
except IndexError:
    print("Error: Index is out of range.")
except ValueError:
    print("Error: Invalid input. Please enter a valid integer.")
```

```
Enter the index of the element to access:  3
Element at index 3 is: 4
```

No Error as we enter numeric values present in the list

```python
#Access an element in a list

my_list = [1, 2, 3, 4, 5]

try:
    index = int(input("Enter the index of the element to access: "))
    element = my_list[index]
    print("Element at index", index, "is:", element)
except IndexError:
    print("Error: Index is out of range.")
except ValueError:
    print("Error: Invalid input. Please enter a valid integer.")
```

```
Enter the index of the element to access:  a
Error: Invalid input. Please enter a valid integer.
```

```python
#Access an element in a list

my_list = [1, 2, 3, 4, 5]

try:
    index = int(input("Enter the index of the element to access: "))
    element = my_list[index]
    print("Element at index", index, "is:", element)
except IndexError:
    print("Error: Index is out of range.")
except ValueError:
    print("Error: Invalid input. Please enter a valid integer.")
```

```
Enter the index of the element to access:  9
Error: Index is out of range.
```

Error as we have enter any alphabet or any other integer value which is not present in the list

d. Program to handle a specific exception

```
#Handle a specific exception
try:
    numerator = int(input("Enter the numerator: "))
    denominator = int(input("Enter the denominator: "))

    result = numerator / denominator
    print("Result:", result)

except ZeroDivisionError:
    print("Error: Cannot divide by zero.")
```

```
Enter the numerator:  4
Enter the denominator:  2
Result: 2.0
```

No Error as we enter numeric values in both numerator and denominator

```
#Handle a specific exception
try:
    numerator = int(input("Enter the numerator: "))
    denominator = int(input("Enter the denominator: "))

    result = numerator / denominator
    print("Result:", result)

except ZeroDivisionError:
    print("Error: Cannot divide by zero.")
```

```
Enter the numerator:  4
Enter the denominator:  0
Error: Cannot divide by zero.
```

Error as we have enter Zero"0" in either numerator and denominator

e. Program to handle any exception

```
# Handle any exception
try:
    numerator = int(input("Enter the numerator: "))
    denominator = int(input("Enter the denominator: "))

    result = numerator / denominator
    print("Result:", result)

except Exception as e:
    print("An error occurred:", str(e))
```

```
Enter the numerator:  6
Enter the denominator:  3
Result: 2.0
```

No Error as we enter numeric values in both numerator and

```
# Handle any exception
try:
    numerator = int(input("Enter the numerator: "))
    denominator = int(input("Enter the denominator: "))

    result = numerator / denominator
    print("Result:", result)

except Exception as e:
    print("An error occurred:", str(e))
```

```
Enter the numerator:  3
Enter the denominator:  a
An error occurred: invalid literal for int() with base 10: 'a'
```

Error as we have enter any alphabet in either numerator and denominator

```
# Handle any exception
try:
    numerator = int(input("Enter the numerator: "))
    denominator = int(input("Enter the denominator: "))

    result = numerator / denominator
    print("Result:", result)

except Exception as e:
    print("An error occurred:", str(e))
```

```
Enter the numerator:  3
Enter the denominator:  0
An error occurred: division by zero
```

Error as we have enter Zero"0" in either numerator and denominator