17th June Assignment

Q1. What is the role of try and exception block?

Ans: Try block: The code within the try block contains the statements that may potentially raise an exception. It allows you to specify the section of code that you want to monitor for exceptions.

Exception block: If an exception occurs within the try block, the corresponding except block(s) are executed. The except block allows you to define the actions or code that should be executed when a specific exception is raised. You can have multiple except blocks to handle different types of exceptions.

```
#e.g: normal function error

print(x)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[1], line 3
      1 #e.g: normal function error
----> 3 print(x)

NameError: name 'x' is not defined
```

```
#try and exception handling

try:
  print(x)
except:
  print("There is no x")
```

```
There is no x
```

Q2. What is the syntax for a basic try-except block?

```
try:  #finds error
  print(x)
except:  #handles error
  print("Some issue with x")
```

```
Some issue with x
```

Q3. What happens if an exception occurs inside a try block and there is no matching except block?

Ans: In Python, if an exception occurs inside a try block and there is no matching except block, the exception will propagate up to the next higher level that can handle it. If no handler is found, it becomes an unhandled exception and execution stops with a traceback message.

```
#Example

try:
    x = 5 / 0  # This will raise a ZeroDivisionError
except ValueError:
    print("This is a ValueError handler.")
```

```
---------------------------------------------------------------------------
```

```
ZeroDivisionError                         Traceback (most recent call last)
Cell In[4], line 4
      1 #Example
      3 try:
----> 4     x = 5 / 0  # This will raise a ZeroDivisionError
      5 except ValueError:
      6     print("This is a ValueError handler.")

ZeroDivisionError: division by zero
```

In this example, the try block attempts to divide 10 by 0, which raises a ZeroDivisionError. However, the except block specifies ValueError, which does not match the raised exception. Since there is no matching except block, the program will terminate with an unhandled exception error.

Q4. What is the difference between using a bare except block and specifying a specific exception type?

Ans:Bare Except Block: This catches all exceptions, including system exceptions such as SystemExit, KeyboardInterrupt, and even exceptions that are typically used for non-error messaging like StopIteration. It's generally not recommended to use a bare except block because it can catch unexpected exceptions and hide programming errors.

```
#Example

try:
    x = 3 / 0  # This will raise a ZeroDivisionError
except:
    print("An error occurred.")
```

```
An error occurred.
```

Specifying a specific exception type: This only catches exceptions of the specified type or its subclasses. It allows you to have different handlers for different types of exceptions, which can make your error handling code more precise and easier to understand.

```
#Example

try:
    x = 1 / 0  # This will raise a ZeroDivisionError
except ZeroDivisionError:
    print("You can't divide by zero!")
```

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

Q5. Can you have nested try-except blocks in Python? If yes, then give an example.

Ans: Yes, you can have nested try-except blocks in Python. This means you can place one try-except block inside another. Nested try-except blocks allow you to handle exceptions at different levels of your code, providing more fine-grained control over error handling.

```
#Example

x = 10
y = 0
try:
    print("Outer try block.")
    try:
        print("Nested try block.")
        print(x / y)
```

```
    except TypeError:
        print("Nested except block.")
except ZeroDivisionError:
    print("Outer except block.")
```

```
Outer try block.
Nested try block.
Outer except block.
```

Q6. Can we use multiple exception blocks, if yes then give an example.

Ans: Yes, you can use multiple except blocks in Python to handle different types of exceptions. Each except block will catch exceptions of the specified type and its subclasses.

```
#Example

try:
    num = int(input("Enter a number: "))
    result = 10 / num
    print("The answer is ", result)
except ValueError:
    print("Invalid input: Please enter a valid integer.")
except ZeroDivisionError:
    print("Cannot divide by zero.")
except Exception as e:
    print(f"An error occurred: {e}")
```

```
Enter a number: 5
The answer is  2.0
#Again running the above program with different input

try:
    num = int(input("Enter a number: "))
    result = 10 / num
    print("The answer is ", result)
except ValueError:
    print("Invalid input: Please enter a valid integer.")
except ZeroDivisionError:
    print("Cannot divide by zero.")
except Exception as e:
    print(f"An error occurred: {e}")
```

```
Enter a number: d
Invalid input: Please enter a valid integer.
#Again running the above program with different input

try:
    num = int(input("Enter a number: "))
    result = 10 / num
    print("The answer is ", result)
except ValueError:
    print("Invalid input: Please enter a valid integer.")
except ZeroDivisionError:
    print("Cannot divide by zero.")
except Exception as e:
    print(f"An error occurred: {e}")
```

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

Q7. 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: **a. EOFError:** An EOFError (End of File Error) in Python occurs when an input operation, such as reading from a file or the standard input, reaches the end of the input unexpectedly. It indicates an attempt to read more data than is available, often caused by premature termination of input.

b. FloatingPointError: A FloatingPointError in Python refers to an exception that arises when there is an issue with floating-point arithmetic operations, such as division by zero or numerical inaccuracies. This error usually occurs when performing calculations with floating-point numbers that result in undefined or no representable values.

c. IndexError: An IndexError in Python occurs when attempting to access an index of a sequence (like a list or string) that is outside the valid range of indices. This error suggests that the specified index does not exist in the sequence, and it commonly arises when trying to access an element beyond the bounds of the sequence.

d. MemoryError: A MemoryError in Python occurs when the program runs out of available memory while trying to allocate memory for an object. This error indicates that the system's memory resources are exhausted, preventing the program from creating or storing additional data structures.

e. OverflowError: An OverflowError in Python occurs when a numerical operation exceeds the maximum representable value for a numeric type. This error typically happens when performing arithmetic operations that result in a value beyond the range that the data type can handle.

f. TabError: A TabError happens when you use tabs and spaces inconsistently for indentation in your Python code, making it difficult for Python to understand the structure of your program. Python requires either tabs or spaces for indentation but not both together.

g. ValueError: A ValueError in Python occurs when a function or operation receives an argument that has the correct type but an inappropriate value. It means the input data doesn't make sense or is out of the expected range for the given context.

Q8. 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

a. Program to divide two numbers

```
#Try part

try:
    num= float(input("Enter Dividend: "))
    den= float(input("Enter Divisor: "))
    result= num/den
    print("Result: ", result)

except ZeroDivisionError:
    print("Division by zero is not allowed")
```

```
Enter Dividend: 8
Enter Divisor: 2
Result:  4.0
```

```
#Except part

try:
    num= float(input("Enter Dividend: "))
    den= float(input("Enter Divisor: "))
    result= num/den
    print("Result: ", result)

except ZeroDivisionError:
    print("Division by zero is not allowed")
```

```
Enter Dividend: 4
Enter Divisor: 0
Division by zero is not allowed
```

b. Program to convert a string to an integer

```
#try part

try:
    value= input("Enter integer: ")
    result= int(value)
    print("Converted value to integer is:", result)

except ValueError:
    print("Error: Entered Value is incorrect.")
```

```
Enter integer: 23
Converted value to integer is: 23
```

```
#Except part

try:
    value= input("Enter integer: ")
    result= int(value)
    print("Converted value to integer is:", result)

except ValueError:
    print("Error: Entered Value is incorrect.")
```

```
Enter integer: 2d
Error: Entered Value is incorrect.
```

c. Program to access an element in a list

```
#Try part
```

```
try:
    lst= [1,2,3,4,5,6,7,8]
    idx= int(input("Enter index: "))
    res= lst[idx]
    print("Element at index ", idx, "is ", res)

except IndexError:
    print("Index out of range, please enter valid index.")
```

```
Enter index: 6
Element at index  6 is  7
```

```
#Except part

try:
    lst= [1,2,3,4,5,6,7,8]
    idx= int(input("Enter index: "))
    res= lst[idx]
    print("Element at index ", idx, "is ", res)

except IndexError:
    print("Index out of range, please enter valid index.")
```

```
Enter index: 9
Index out of range, please enter valid index.
```

d. Program to handle a specific exception

```
#Try part

try:
    age = int(input("Enter age: "))
    if age < 0:
        raise ValueError("Invalid age: Age cannot be negative.")
    print("Age is ", age)

except ValueError as e:
    print("Error: ", str(e))
```

```
Enter age: 23
Age is  23
```

```
#Except part

try:
    age = int(input("Enter age: "))
    if age < 0:
        raise ValueError("Invalid age: Age cannot be negative.")
    print("Age is ", age)

except ValueError as e:
    print("Error: ", str(e))
```

```
Enter age: -21
Error:  Invalid age: Age cannot be negative.
```

e. Program to handle any exception

```
#Try Part

try:
    # Code that may raise exceptions
    num1 = int(input("Enter first number: "))
    num2 = int(input("Enter second number: "))
```

```
    result = num1/num2
    print("Result:", result)

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

```
Enter first number: 9
Enter second number: 3
Result: 3.0
```

```
#Except Part

try:
    # Code that may raise exceptions
    num1 = int(input("Enter first number: "))
    num2 = int(input("Enter second number: "))
    result = num1/num2
    print("Result:", result)

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

```
Enter first number: 9
Enter second number: a
An error occurred:  invalid literal for int() with base 10: 'a'
```