In [ ]:
```
Q.1 : What is the role of try and exception block?
Ans : The try and except blocks in Python are used for handling exceptions or errors that may occur during the
      execution of a program. The primary role of these blocks is to gracefully handle unexpected situations and
      prevent the program from crashing.
```

In [ ]:
```
Q.2 :What is the syntax for a basic try-except block?
Ans :try:
    # Code that might raise an exception
    # ...
except SomeExceptionType:
    # Code to handle the specific exception (SomeExceptionType)
    # ...
except AnotherExceptionType as variable:
    # Code to handle another specific exception (AnotherExceptionType)
    # The exception instance is assigned to the variable
    # ...
except:
    # Code to handle any other exceptions that were not caught by the previous blocks
    # ...
else:
    # Code to execute if no exceptions were raised in the try block
    # ...
finally:
    # Code that will be executed no matter what, whether an exception was raised or not
    # ...
```

In [ ]:
```
Q.3 :What happens if an exception occurs inside a try block and there is no matching
    except block?
Ans :

    try:
    x = 10 / 0
    except ValueError:
    # No except block for ZeroDivisionError
    print("This won't be executed.")
```

In [ ]:
```
Q.4 :What is the difference between using a bare except block and specifying a specific
     exception type?
Ans :
    In a try-except block in Python, we can either use a bare except block or specify a specific
    exception type. The main difference lies in the level of specificity and the type of exceptions you intend to catch.
```

In [ ]:
```
Q.5 :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. Nested try-except blocks allow WE
    to handle different types of exceptions at different levels of your code.
    try:
        # OUTER TRY BLOCK
    num1 = int(input("Enter the first number: "))
    num2 = int(input("Enter the second number: "))

    try:
        # Inner try block
        result = num1 / num2
        print("Result:", result)

    except ZeroDivisionError:
        print("Inner except block: Division by zero is not allowed.")

except ValueError:
    print("Outer except block: Please enter valid integers.")
except Exception as e:
    print("Outer except block: An error occurred -", e)
```

In [ ]:
```
Q.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 catch a specific type of
    exception, allowing you to tailor your error handling to different situations.

    try:
    num1 = int(input("Enter the first number: "))
    num2 = int(input("Enter the second number: "))

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

except ValueError:
    print("ValueError: Please enter valid integers.")

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

except Exception as e:
    print(f"An unexpected error occurred: {e}")
```

In [ ]:
```
Q.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 :a. EOFError: Stands for "End of File Error." It occurs when the input() function
        hits the end of the file unexpectedly. This can happen when the user is supposed
        to input something, but the end of the file is reached before any input is provided.

b. FloatingPointError: This error occurs when a floating-point operation (like division or modulo)
    results in an undefined or infinite value. For example, dividing by zero with floating-point numbers or
    trying to represent a value that is too large as a floating-point number can cause this error.

c. IndexError: Raised when you try to access an index that is outside the bounds of a list, tuple,
    or other iterable. For example, trying to access an element at an index that does not exist.

d. MemoryError: This error occurs when an operation runs out of memory. This could happen, for
    instance, when trying to allocate a large amount of memory using functions like range() or
    when the system has insufficient memory to perform an operation.

e. OverflowError: Raised when the result of an arithmetic operation is too large to be represented.
    This can happen with integer operations that exceed the maximum representable value.

f. TabError: Occurs when there is an issue with the indentation of code, especially when mixing tabs
    and spaces in an inconsistent manner. Python expects consistent indentation to define blocks of code.

g. ValueError: Raised when a built-in operation or function receives an argument of the correct type but an
    inappropriate value. For example, trying to convert a string to an integer where the string does not represent a valid integer.
```

In [ ]:
```
Q.8 :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)
    try:
    numerator = int(input("Enter the numerator: "))
    denominator = int(input("Enter the denominator: "))

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

except ZeroDivisionError:
    print("Error: Division by zero is not allowed.")
except ValueError:
    print("Error: Please enter valid integers.")
except Exception as e:
    print(f"An unexpected error occurred: {e}")

b)
try:
    input_str = input("Enter an integer: ")
    converted_int = int(input_str)
    print("Converted Integer:", converted_int)

except ValueError:
    print("Error: Please enter a valid integer.")
except Exception as e:
    print(f"An unexpected error occurred: {e}")


c)
try:
    my_list = [1, 2, 3, 4, 5]
    index = int(input("Enter the index to access: "))

    value = my_list[index]
    print("Value at index {}: {}".format(index, value))

except IndexError:
    print("Error: Index out of range. Please enter a valid index.")
except ValueError:
    print("Error: Please enter a valid integer as index.")
except Exception as e:
    print(f"An unexpected error occurred: {e}")

d)
try:
    # Your code that may raise a specific exception
    raise ValueError("This is a specific exception.")

except ValueError as ve:
    print(f"Caught a specific exception: {ve}")
except Exception as e:
    print(f"An unexpected error occurred: {e}")

e)
try:
    # Your code that may raise any exception
    x = 10 / 0

except Exception as e:
    print(f"An exception occurred: {e}")
```

In [ ]:
```

```