Try-Catch

Definition:

- `try` and `except` blocks (Exception Handling) are used for handling exceptions or errors that might occur during the execution of a program.

Exceptions:
Errors coming from execution are exception errors:

1. `ZeroDivisionError (e.g., 4/0)`
2. `NameError (name is not defined, such as 4 + name*3)`
3. `TypeError (e.g., cant convert 'int' object to str implicitly such as '2' +2 )`
4. `Value Error (e.g., int('one')`

To handle: add exceptions

1. Include `Try Clause`
2. If an `exception occurred`, skip the rest of the try clause, to a `matching except clause`
3. If no exception occurs, the except clause is skipped (go to the else clause, if it exists)
4. The `finally clause` is always executed before leaving the try statement, whether an exception has occurred or not.

Example:

```
try:
    # code that may raise an exception
    result = 10 / 0
except ZeroDivisionError as e:
    # code to handle the exception
    print(f"Error: {e}")
else:
    # code to be executed if no exception occurs
    print("No error occurred.")
finally:
    # code to be executed regardless of whether an exception occurred or not
    print("This will always be executed.")
```

```
try:
    # code that may raise an exception
    result = 10 / 5
except ZeroDivisionError as e:
    # code to handle the exception
    print(f"Error: {e}")
else:
    # code to be executed if no exception occurs
    print("No error occurred.")
finally:
    # code to be executed regardless of whether an exception occurred or not
    print("This will always be executed.")
```

Explanation:

- The `try` block contains the code that may raise an exception.
- The `except` block catches and handles the specified exception.
- The `else` block is executed if no exception occurs in the `try` block.
- The `finally` block contains code that will always be executed, whether an exception occurred or not.

Inside the scope

In Python, the `try` block will catch an exception if it occurs anywhere within its scope, including inside the `while` loop. The `while` loop continues to execute until the condition `n > 0` is true. Once `n` becomes 0, the loop exits, and the program proceeds to the `except` block.

```
n = 2
x = 0
try:
        while True:
                x += 10//n
                n-=1
                print("x:", x, "and", "n:", n)
except:
        print(x)
```

the `try` block catches any exceptions that may occur during its execution, and the loop runs until `n` becomes 0. When `n` reaches 0, the `while` loop exits, and the program goes to the `except` block

Assert Statement

Definition:

- `assert` is used for debugging purposes to check whether a given condition is `True`, and if not, it raises an `AssertionError` exception.
- If the statement following in the assertion is `False` then `Exception` will be called.

Example:

```
x = 5
assert x > 0, "x should be a positive number"
print("Assertion passed!")
```

```
x = 5
assert x > 10, "x should be a positive number"
print("Assertion passed!")
```

Explanation:

- The `assert` statement checks if the given condition is `True`.
- If the condition is `False`, it raises an `AssertionError` with an optional error message.

```
while True:
        try:
                pos = int(input("Input: "))
                assert 0 < pos < 10
                break
        except AssertionError:
                print("OI WRONG LAH CB")
        except:
                print("Wrong")
```