Scoping in Python

- **What is Scoping?**
  - Scoping in Python refers to the concept of controlling the visibility and accessibility of variables within different parts of the program. It helps prevent accidental or unwanted changes to variables by confining their usage to specific areas of the code.
- **The Four Types of Scope (LEGB)**
  - Python has four main types of scope, which are ordered in terms of coverage:
    a. **Local Scope**: Variables declared inside a function are local to that function. They cannot be accessed outside.
    b. **Enclosing Scope**: Variables from an enclosing function (a function that contains another function) are accessible to the inner function.
    c. **Global Scope**: Variables declared outside any function are accessible anywhere in the code, including inside functions.
    d. **Built-in Scope**: This scope includes Python's built-in functions and exceptions (e.g., `print`, `len`, etc.) that are accessible throughout the program.
  - Together, these scopes are referred to as **LEGB** (Local, Enclosing, Global, Built-in).
- **Global Scope Example**
  - A variable `my_global` is declared in the global scope with a value of 10.
  - Inside a function `fn1`, a local variable `local_v` is declared with a value of 5.
  - The global variable `my_global` is accessible inside `fn1` and can be printed.
  - If you try to print the local variable `local_v` outside `fn1`, it will result in a **NameError** because `local_v` is only accessible inside the function.
    - Example:

```
my_global = 10
def fn1():
    local_v = 5
    print(my_global)  # prints 10
print(local_v)  # results in NameError
```

- **Enclosing Scope Example**
  - A second function `fn2` is declared inside `fn1`, making `fn2` have access to the variables of `fn1` due to enclosing scope.
  - The variable `enclosed_v` is declared in `fn1`, and `fn2` can access it, but `fn1` cannot access the local variables of `fn2`.
  - The global variable `my_global` is also accessible from both `fn1` and `fn2`.

- Example:

```
def fn1():
    enclosed_v = 8
    def fn2():
        print(enclosed_v)  # prints 8
    fn2()
fn1()
```

- **Built-in Scope**
  - Python has built-in functions and exceptions (e.g., `print`, `len`, `def`, etc.) that are available globally, regardless of where you are in the code.
  - These built-in functions can be accessed from any scope (local, enclosing, global).
- **Why Scoping Matters**
  - Scoping ensures that variables are not unintentionally changed by different parts of the code. It helps maintain the integrity of the program and makes the code more predictable.
  - Using **global scope** extensively is discouraged because it can lead to mistakes and unexpected behaviors.
- **Conclusion**
  - Scoping in Python provides greater control over variables and helps avoid errors related to variable accessibility. By understanding the four types of scope (LEGB), you can write more organized and error-free code.

Here's a detailed explanation with examples for each type of scope in Python (Local, Enclosing, Global, and Built-in):

1. Local Scope

- **Local scope** refers to variables declared inside a function. These variables are only accessible within that function and cannot be accessed outside.

Example:

```
def local_scope_example():
    local_var = "I'm a local variable"
    print(local_var)  # Accessible inside the function

local_scope_example()  # Prints: I'm a local variable

# Outside the function, local_var is not accessible
print(local_var)  # Results in NameError: name 'local_var' is not defined
```

Explanation:
The variable `local_var` is created inside the function `local_scope_example`. It cannot be accessed outside the function, resulting in an error when trying to print it.

2. Enclosing Scope

- **Enclosing scope** is the scope of a function that contains another function. Variables from the enclosing function are accessible to the inner function but not vice versa.

Example:

```
def enclosing_scope_example():
    enclosing_var = "I'm from enclosing scope"

    def inner_function():
        print(enclosing_var)  # Can access enclosing scope variable

    inner_function()

enclosing_scope_example()  # Prints: I'm from enclosing scope
```

Explanation:
The `inner_function` has access to the variable `enclosing_var` from the enclosing scope of `enclosing_scope_example`. However, `enclosing_var` cannot be accessed directly from outside the enclosing function.

3. Global Scope

- **Global scope** refers to variables declared outside of any function. These variables are accessible from any part of the code, including inside functions, unless shadowed by a local variable.

Example:

```
global_var = "I'm a global variable"  # Global scope variable

def global_scope_example():
    print(global_var)  # Accessing global variable inside the function

global_scope_example()  # Prints: I'm a global variable

print(global_var)  # Also accessible here, Prints: I'm a global variable
```

Explanation:
The variable `global_var` is declared outside the function, making it accessible globally. It can be

accessed both inside the function `global_scope_example` and outside.

Note: If you try to modify a global variable inside a function, Python requires the use of the `global` keyword.

Example with modification:

```
global_var = 10

def modify_global():
    global global_var  # Declare global variable to modify it
    global_var = 20  # Modify the global variable

print(global_var)  # Prints: 10
modify_global()
print(global_var)  # Prints: 20 after modification
```

4. Built-in Scope

- **Built-in scope** refers to functions and exceptions that are always available in Python, regardless of the scope you're in. These include functions like `print()`, `len()`, and keywords like `def`.

Example:

```
def built_in_scope_example():
    print("Hello from built-in scope!")  # print() is a built-in function
    print(len("Hello"))  # len() is another built-in function

built_in_scope_example()  # Prints: Hello from built-in scope! and 5
```

Explanation:
The `print()` and `len()` functions are part of Python's built-in scope and can be called from anywhere in the code. They don't need to be declared or imported.

Summary of Scopes in Action:

- **Local Scope:** Variables defined within a function.
- **Enclosing Scope:** Variables in an enclosing function, accessible by nested functions.
- **Global Scope:** Variables defined outside of functions, accessible globally.
- **Built-in Scope:** Python's predefined functions and exceptions, accessible everywhere.

By understanding these scopes, you can control the visibility and accessibility of variables more effectively in your Python programs.