# Understanding Namespaces and Scopes in Python

## 1. What are Namespaces?

- A **namespace** is a mapping from names (identifiers) to objects (values or functions).
- Think of it as a dictionary where names are keys and objects are values.
- Types of namespaces:
  - **Module namespace**: Holds the attributes of a module (e.g., functions, classes, variables).
  - **Local namespace**: Created within a function and contains local variables.
  - **Global namespace**: Contains variables declared at the outermost level of a script.

## 2. What is Scope?

- **Scope** refers to the region of a program where a namespace is directly accessible.
- Python's scope follows the **LEGB rule**:
  i. **Local**: Variables defined within the current function.
  ii. **Enclosing**: Variables in the enclosing scope of nested functions.
  iii. **Global**: Variables defined at the script level or module level.
  iv. **Built-in**: Names in Python's built-in modules.

## 3. The LEGB Rule: Example

```python
# Global scope
name = "Python"

def outer_function():
    # Enclosing scope
    name = "Java"

    def inner_function():
        # Local scope
        name = "C++"
        print("Inner Function:", name)

    inner_function()
    print("Outer Function:", name)

outer_function()
print("Global Scope:", name)
```

### Output:

```
Inner Function: C++
Outer Function: Java
Global Scope: Python
```

## 4. Special Keywords: `global` and `nonlocal`

- `global` : Allows access to global variables from within a function.
- `nonlocal` : Allows access to variables in an enclosing (non-global) scope.

## 5. Example of `global`

```
# Global variable
counter = 0

def increment():
    global counter
    counter += 1
    print("Inside Function:", counter)

increment()
print("Outside Function:", counter)
```

### Output:

```
Inside Function: 1
Outside Function: 1
```

## 6. Example of `nonlocal`

```
def outer_function():
    count = 10  # Enclosing variable

    def inner_function():
        nonlocal count
        count += 5
        print("Inside Inner Function:", count)

    inner_function()
    print("Inside Outer Function:", count)

outer_function()
```

**Output**:

```
Inside Inner Function: 15
Inside Outer Function: 15
```

## 7. Practical Example: Animal Scope

```python
# Global variable
animal = "camel"

def d():
    # Local variable in d
    animal = "elephant"

    def e():
        nonlocal animal  # Refers to the enclosing variable in d
        animal = "giraffe"
        print("Inside Nested Function:", animal)

    print("Before Nested Function:", animal)
    e()
    print("After Nested Function:", animal)

# Calling functions
d()
print("Global Variable:", animal)
```

**Output**:

```
Before Nested Function: elephant
Inside Nested Function: giraffe
After Nested Function: giraffe
Global Variable: camel
```

## 8. Key Observations

- **Local and Enclosing Scope**:
  - Variables in the enclosing scope remain unchanged unless modified explicitly using `nonlocal`.
- **Global Variables**:
  - Avoid modifying global variables unless absolutely necessary; it leads to "spaghetti code."
- **Unbound Error**:
  - Using `nonlocal` without a matching enclosing variable results in an error.

### 9. Good Practices

1. **Avoid global variables**: Use function parameters and return values to pass data.
2. **Limit the use of `nonlocal`**: It is useful for specific cases like nested functions but can reduce code readability.
3. **Use meaningful scopes**: Organize your code to minimize dependencies on outer scopes.
4. **Debugging Tip**: Use `locals()` and `globals()` functions to inspect the current scope.

By practicing these concepts, you will develop a solid understanding of Python namespaces and scopes, crucial for writing efficient and maintainable code.