Variable Scope & Lifetime

Variable scope and lifetime are important concepts in programming languages, including Python. They determine where and for how long a variable can be accessed and used in your code. Let's explore these concepts in detail:

Variable Scope:

- Variable scope refers to the region of code where a variable is visible and accessible.
- In Python, variables can have one of the following scopes:
    - Global Scope: Variables defined outside of any function or class have global scope. They can be accessed from anywhere in the code.
    - Local Scope: Variables defined within a function have local scope. They can only be accessed within that function.
    - Enclosing (Nonlocal) Scope: Variables defined in an enclosing function have enclosing scope. They can be accessed by nested functions but not from outside the enclosing function.
- Python follows the LEGB (Local, Enclosing, Global, Built-in) rule to resolve variable names.

Lifetime of Variables:

- The lifetime of a variable is the duration for which it exists in memory during program execution.
- In Python, the lifetime of variables depends on their scope:
    - Global Variables: Exist until the program terminates or until explicitly deleted.
    - Local Variables: Exist only within the function where they are defined and are destroyed when the function returns.
    - Enclosing Variables: Exist as long as the enclosing function is executing and are destroyed when the function completes execution.
- Memory occupied by variables is automatically reclaimed by Python's garbage collector when they go out of scope.

Let's illustrate variable scope and lifetime with an example:

```
# Global variable
global_var = 10

def outer_function():
 # Enclosing variable
 enclosing_var = 20

 def inner_function():
 # Local variable
 local_var = 30
 print("Inner function:", global_var, enclosing_var, local_var)

 inner_function()
 print("Outer function:", global_var, enclosing_var)

outer_function()
print("Global scope:", global_var)
```

In this example:

- `global_var` is a global variable accessible from all parts of the code.
- `enclosing_var` is an enclosing variable accessible within `outer_function` and its nested functions.
- `local_var` is a local variable accessible only within `inner_function`.
- Each variable has a different scope and lifetime based on where it is defined.

Understanding variable scope and lifetime is essential for writing correct and maintainable code in Python. It helps prevent naming conflicts, manage memory efficiently, and ensure proper encapsulation of data.

Exercises and Answers for Variable Scope & Lifetime

Exercise 1:

Define a global variable called `global_var` with an initial value of 10. Write a function called `modify_global_var` that tries to modify the value of `global_var` by assigning it a new value of 20. Inside the function, print the value of `global_var` before and after the assignment. Finally, call the `modify_global_var` function and observe the output.

Answer 1:

```
global_var = 10

def modify_global_var():
 global global_var
 print("Before modification:", global_var)
 global_var = 20
 print("After modification:", global_var)

# Test the function
modify_global_var()
print("Global variable outside function:", global_var)
```

Exercise 2:

Write a function called `outer_function` that defines a local variable called `outer_var` with an initial value of 10. Inside `outer_function`, define another function called `inner_function` that tries to access the `outer_var` variable. Print the value of `outer_var` from within `inner_function`. Finally, call the `outer_function` and observe the output.

Answer 2:

```
def outer_function():
 outer_var = 10

 def inner_function():
```

```
 print("Inner function accessing outer variable:", outer_var)

 inner_function()

# Test the function
outer_function()
```

Exercise 3:

Create a function called `nested_functions` that defines a variable called `enclosing_var` with an initial value of 5. Inside `nested_functions`, define two nested functions: `first_inner_function` and `second_inner_function`. Each inner function should try to access the `enclosing_var` variable and print its value. Finally, call the `nested_functions` function and observe the output.

Answer 3:

```
def nested_functions():
 enclosing_var = 5

 def first_inner_function():
 print("First inner function accessing enclosing variable:", enclosing_var)

 def second_inner_function():
 print("Second inner function accessing enclosing variable:", enclosing_var)

 first_inner_function()
 second_inner_function()

# Test the function
nested_functions()
```