Local Variables

Local variables are those defined within a function. They are only accessible within the function and cannot be accessed outside of it. To help visualize this, consider the example of a classroom whiteboard:

- Classroom Whiteboard: The content written on it can only be seen by those within the classroom during the session. This represents a local variable, as its scope is limited to that particular function (or classroom).

Example:

```
def my_function():
    y = 5  # Local variable inside the function
    print("Inside function: x =", x, "y =", y)

x = 10  # Global variable outside the function
my_function()  # Calling the function
```

In this case, the function can access the global variable x but y can only be accessed within my_function().

Global Variables

Global variables are defined outside of functions and are accessible throughout the entire program. In our analogy, the school's notice board can be seen by everyone in the school, not just those in a specific class. Similarly, a global variable can be accessed by any function in the program.

Example:

```
x = 10  # Global variable
def my_function():
    y = 5  # Local variable
    print("Inside function: x =", x, "y =", y)

my_function()  # Calling the function
print("Outside function: x =", x)
```

Here, the variable x is global, so it can be accessed both inside and outside the function.

Practical Example: Local vs. Global Variables

In the example below, we try to access a global variable inside a function, which works fine. However, attempting to access a local variable outside the function results in an error because local variables are confined to the function's scope.

Example:
```
x = 10  # Global variable
def my_function():
    y = 5  # Local variable
    print("Inside function: x =", x, "y =", y)
my_function()
print("Outside function: x =", x)

# Trying to access the local variable 'y' outside the function will result in an error.
# print("Outside function: y =", y)  # This will give an error: name 'y' is not defined
```

Understanding Variable Scope

Local Variables:

- Defined inside a function.
- Only accessible within that function.

Global Variables:

- Defined outside any function.
- Accessible from any part of the code.

Nested Functions and Enclosing Variables

When one function is defined inside another, the variables of the inner function are considered local to that function. If a variable is defined in the outer function but is accessed by the inner function, it is called an enclosing variable.

Example:
```
x = 10  # Global variable
def outer_function():
    y = 5  # Enclosing variable
    def inner_function():
        z = 3  # Local variable
        print("Inside inner function: x =", x, "y =", y, "z =", z)
    inner_function()
```

```
outer_function()
```

Modifying Global Variables from Within Functions

To modify a global variable inside a function, we need to use the global keyword. Without this keyword, Python treats the variable as a local one within the function.

Example:

```
counter = 0  # Global variable
def increment_counter():
    global counter
    counter += 1  # Modifying global variable
increment_counter()
print("Counter value:", counter)  # Output: Counter value: 1
```

Without the global keyword, the function would not be able to modify the global counter variable.

Comparing Memory Locations: is vs ==

In Python, we can check if two variables refer to the same object in memory using the is operator, and we can check if their values are equal using ==.

Example:

```
a = [1, 2, 3]
b = [1, 2, 3]
# Comparing values
print(a == b)  # Output: True
# Comparing memory locations
print(a is b)  # Output: False
```

Even though the lists a and b have the same values, they are stored at different memory locations. Therefore, a is b returns False.

Conclusion

In this lecture, we learned the distinctions between local and global variables, how Python handles variable scopes, and how we can modify global variables using the global keyword. Additionally, we explored nested functions, enclosing variables, and how to check for object equality and memory location using == and is operators.

Final Thoughts:

It's crucial to understand the difference between local and global variables, especially when dealing with larger projects where variable scope plays an important role in

managing and maintaining code. Practice these concepts by experimenting with different types of variables and functions.
Next, make sure to review the provided examples and ensure that you understand how variables interact within different scopes. This foundational knowledge will help you better grasp more advanced topics in Python.