**Python variables and memory allocation**.

In Python, variables are not containers that directly store values, but rather references or pointers to objects in memory. When a variable is assigned a value, it essentially points to the memory location where that object is stored.

Memory Allocation in Python:

Python utilizes a private heap for object storage and data structures, managed primarily by the Python memory manager. This manager handles the allocation and deallocation of memory for Python objects. Key aspects of memory allocation include:

* **Heap Memory:**

This is where Python objects and data structures reside. Memory is dynamically allocated on the heap during program execution as objects are created. This allows for flexible memory usage, as objects can be of varying sizes and their lifetime is not necessarily tied to function calls.

* **Stack Memory:**

The stack is used for managing function calls and local variables. When a function is called, a stack frame is created, and local variables within that function are allocated on the stack. This memory is automatically deallocated when the function finishes execution.

* **Reference Counting:**

Python employs a reference counting mechanism to track how many variables are referencing a particular object. When an object's reference count drops to zero, it means no variables are pointing to it, and the object becomes eligible for garbage collection.

* **Garbage Collection:**

Python's automatic garbage collector reclaims memory occupied by objects that are no longer referenced. This prevents memory leaks and simplifies memory management for the programmer.

Variable Behavior and Memory:

* **Assignment:**

When you assign a value to a variable (e.g., x = 10), a new object (if it doesn't already exist and isn't interned) is created in the heap, and the variable x is made to reference this object.

* **Reassignment:**

If you reassign a variable to a new value (e.g., x = 20), x will now reference the new object (20), and the old object (10) may become eligible for garbage collection if no other variables reference it.

* **Sharing References:**

Multiple variables can reference the same object in memory. For example, y = x would make y point to the same object that x references.

* **Immutability:**

For immutable objects like integers, strings, and tuples, any operation that appears to modify them actually creates a new object in memory, and the variable is then made to reference this new object. The original object remains unchanged (unless no longer referenced).

* **Mutability:**

For mutable objects like lists and dictionaries, operations that modify the object (e.g., my\_list.append(5)) change the object in place in memory, and the variable continues to reference the same object.