**MEMORY MANAGEMENT**

Memory management in Python involves a private heap containing all Python objects and data structures. The management of this private heap is ensured internally by the Python memory manager. The Python memory manager has different components which deal with various dynamic storage management aspects, like sharing, segmentation, preallocation or caching.

**GARBAGE COLLECTION**

Garbage Collection is a process in which the interpreter frees up the memory when not in use to make it available for other objects. Assume a case where no reference is pointing to an object in memory i.e. it is not in use so, the virtual machine has a garbage collector that automatically deletes that object from the heap memory.

**REFERENCE COUNTING**

Reference counting works by counting the number of times an object is referenced by other objects in the system. When references to an object are removed, the reference count for an object is decremented. When the reference count becomes zero, the object is deallocated.

Eg: x = 10

When x = 10 is executed an integer object 10 is created in memory and its reference is assigned to variable x, this is because everything is object in Python.

|  |  |
| --- | --- |
| x = 10  y = x  if id(x) == id(y):      print("x and y refer to the same object")  **OUTPUT**  x and y refer to the same object  In the above example, y = x will create another reference variable y which will refer to the same object because Python optimizes memory utilization by allocation the same object reference to a new variable if the object already exists with the same value. Now, let’s change the value of x and see what happens.   |  | | --- | | x = 10  y = x  x += 1  if id(x) != id(y):      print("x and y do not refer to the same object") |   **MEMORY ALLOCATION**  There are two parts of memory:   1. Stack Memory 2. Heap Memory   **WORK OF STACK MEMORY**  The allocation happens on contiguous blocks of memory. We call it stack memory allocation because the allocation happens in the function call stack. The size of memory to be allocated is known to the compiler and whenever a function is called, its variables get memory allocated on the stack. It is the memory that is only needed inside a particular function or method call. When a function is called, it is added onto the program’s call stack. Any local memory assignments such as variable initializations inside the particular functions are stored temporarily on the function call stack, where it is deleted once the function returns, and the call stack moves on to the next task. This allocation onto a contiguous block of memory is handled by the compiler using predefined routines, and developers do not need to worry about it.  EG: def func():  #all these variables get memory  #allocated on stack  a = 20  b = [ ]  c = “ “ |

**OUTPUT**

x and y do not refer to the same object

**WORK OF HEAP MEMORY**

The memory is allocated during the execution of instructions written by programmers. Note that the name heap has nothing to do with the heap data structure. It is called heap because it is a pile of memory space available to programmers to allocated and de-allocate. The variables are needed outside of method or function calls or are shared within multiple functions globally are stored in Heap memory.

EG: #this memory for 10 integers

#is allocated on heap.

a = [0]\*10