**MEMORY MAMAGEMENT IN PYTHON**

Memory allocation can be defined as allocating a block of space in the computer memory to a program. In Python memory allocation and deallocation method is automatic as the Python developers created a [garbage collector](https://www.geeksforgeeks.org/garbage-collection-python/) for Python so that the user does not have to do manual garbage collection.

**Garbage Collection**

[Garbage collection](https://www.geeksforgeeks.org/garbage-collection-python/) 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.

**PYTHON PROGRAM:**

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.

'''

y = x

print(id(x))

print(id(y))

if id(x) == id(y):

    print("\nx and y refer to the same object")

#Here, x and y refer to the same object

#Now, let’s change the value of x and see what happens.

x += 1

print()

print(id(x))

print(id(y))

if id(x) == id(y):

    print("x and y refer to the same object")

else:

    print("x and y do not refer to the same object")

#Here, x and y do not refer to the same object

#So now x refer to a new object x and the link between x and 10 disconnected but y still refer to 10.

**MEMORY ALLOCATION IN PYTHON**

There are two parts of memory:

* Stack Memory
* Heap Memory

The methods/method calls and the references are stored in stack memory and all the values objects are stored in a private heap.

**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.

**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.

**PYTHON PROGRAM:**

def func():

    # All these variables get memory allocated on stack

    a = 20

    b = []

    c = ""

# This memory for 10 integers is allocated on heap.

a = [0]\*10

print(a)

GIT Link for python code (Memory Management): <https://github.com/devanshtyagi26/DPyWorld/blob/978a0728c07f5a2d321c8834a248ff1333d3f5cd/Memory%20Management.py>

- Documented by

Devansh Tyagi

20221416