Memory Management and Garbage Collection

Memory Management

- Using the Big data writing the Memory management in python is important now a days
- Not managing Memory will be making the slowness in the Python
- Python memory management is been divided into two parts.
  - Stack memory
  - Heap memory
- Methods and variables are created in Stack memory.
- Objects and instance variables values are created in Heap memory.
- In stack memory - a stack frame is created whenever methods and variables are created.
- These stacks frames are destroyed automatically whenever functions/methods returns.
- Python has mechanism of Garbage collector, as soon as variables and functions returns, Garbage collector clear the dead objects.

Static Memory Allocation

- memory allocated at compile time
- Like C and c++ we can declare static arrays only with fixed sizes
- The memory is allocated at the time of compilation. Stack is used for implementing static allocation. In this case, memory can not be reused.

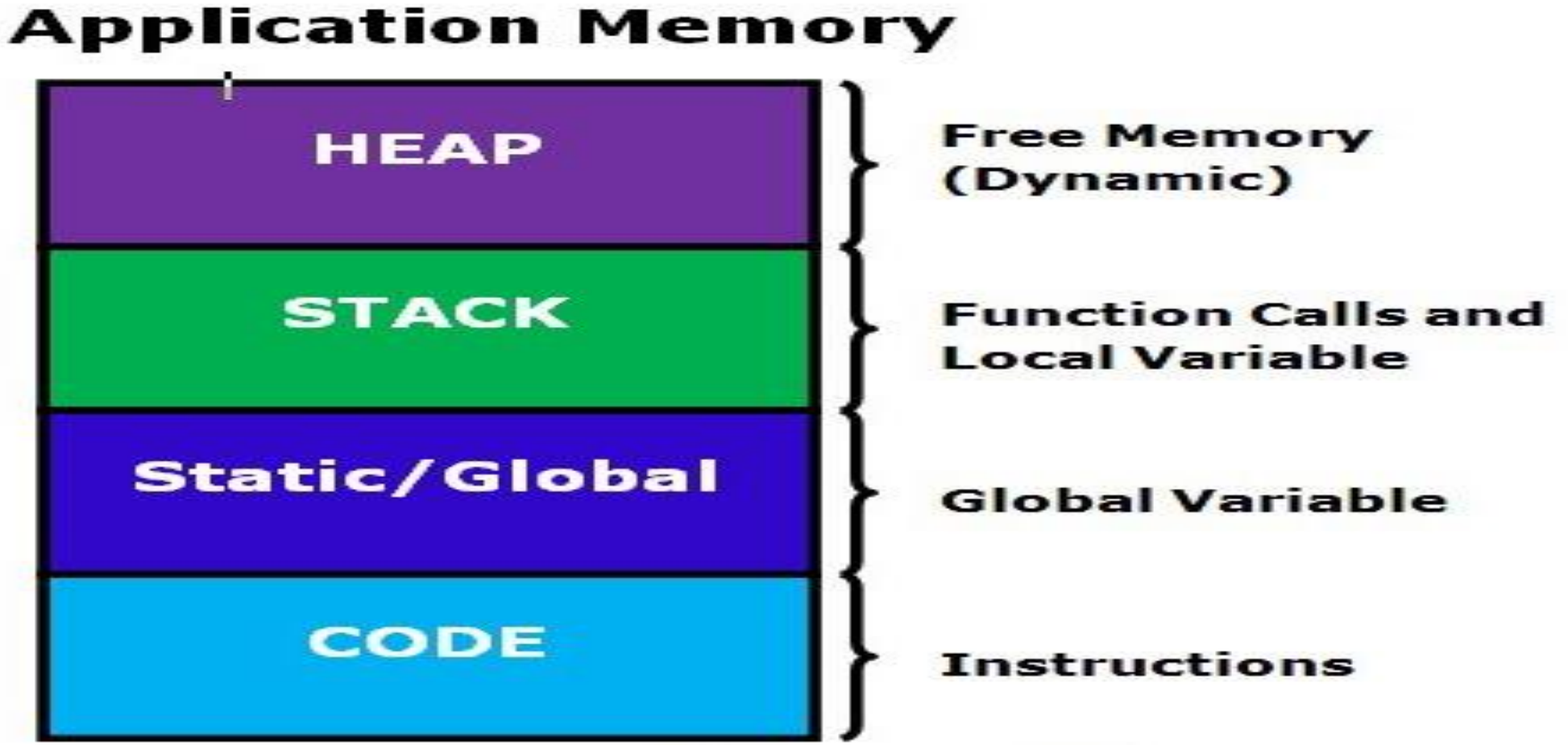
Dynamic Memory Allocation

- memory allocated at Runtime time
- declare arrays with the unary operator new
- The memory is allocated at runtime. ***Heap*** is used for implementing dynamic allocation
- memory can be freed and reused when not required.

Python Memory

- The good thing about Python is that everything in Python is an object. This means that Dynamic Memory Allocation underlies Python Memory Management. When objects are no longer needed, the Python Memory Manager will automatically reclaim memory from them.

Application Memory



Memory Manager

- The Python memory manager has object-specific allocators to allocate memory distinctly for specific objects such as int, string, etc… Below that, the raw memory allocator interacts with the memory manager of the operating system to ensure that there’s space on the private heap.

Memory Management

- The Python memory manager manages chunks of memory called “Blocks”. A collection of blocks of the same size makes up the “Pool”. Pools are created on Arenas, chunks of 256kB memory allocated on heap=64 pools. If the objects get destroyed, the memory manager fills this space with a new object of the same size.
- Methods and variables are created in ***Stack memory***. A stack frame is created whenever methods and variables are created. These frames are destroyed automatically whenever methods are returned.
-

Memory Management

- It is important to note that the Python memory manager doesn’t necessarily release the memory back to the Operating System, instead memory is returned back to the python interpreter. Python has a small objects allocator that keeps memory allocated for further use. In long-running processes, you may have an incremental reserve of unused memory.

Memory Management

```
>>> M="Memory Management\n"
>>> M+="Dont do this"
>>> print(M)
Memory Management
Dont do this
>>>
```

```
>>> M=["Memeory","Use this","Best Result"]
>>> print("\n".join(M))
Memeory
Use this
Best Result
>>>
```

Best Practice

- Don’t use “+” for string, use format method’s
- Try to use Decorators, List Comprehensives and Lambda Functions

```
>>> import sys
>>> sys.getsizeof(x)
50
>>> sys.getsizeof(3.14)
24
>>> sys.getsizeof(2**63)
36
>>> sys.getsizeof(12)
28
>>>
```

Size in bytes

Request in bytes	Size of allocated block
1-8	8
9-16	16
17-24	24
25-32	32
33-40	40
41-48	48
49-56	56
57-64	64
65-72	72