Module-2

Que: How memory is managed in python?

Ans:

- Understanding Memory allocation is important to any software developer as writing efficient code means writing a memory-efficient code.
- 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 for Python so that the user does not have to do manual garbage collection.
- At the lowest level, a raw memory allocator ensures that there is enough room in the private heap for storing all Python-related data by interacting with the memory manager of the operating system.
- On top of the raw memory allocator, several object-specific allocators operate on the same heap and implement distinct memory management policies adapted to the peculiarities of every object type.
- Raw Memory Interface:-

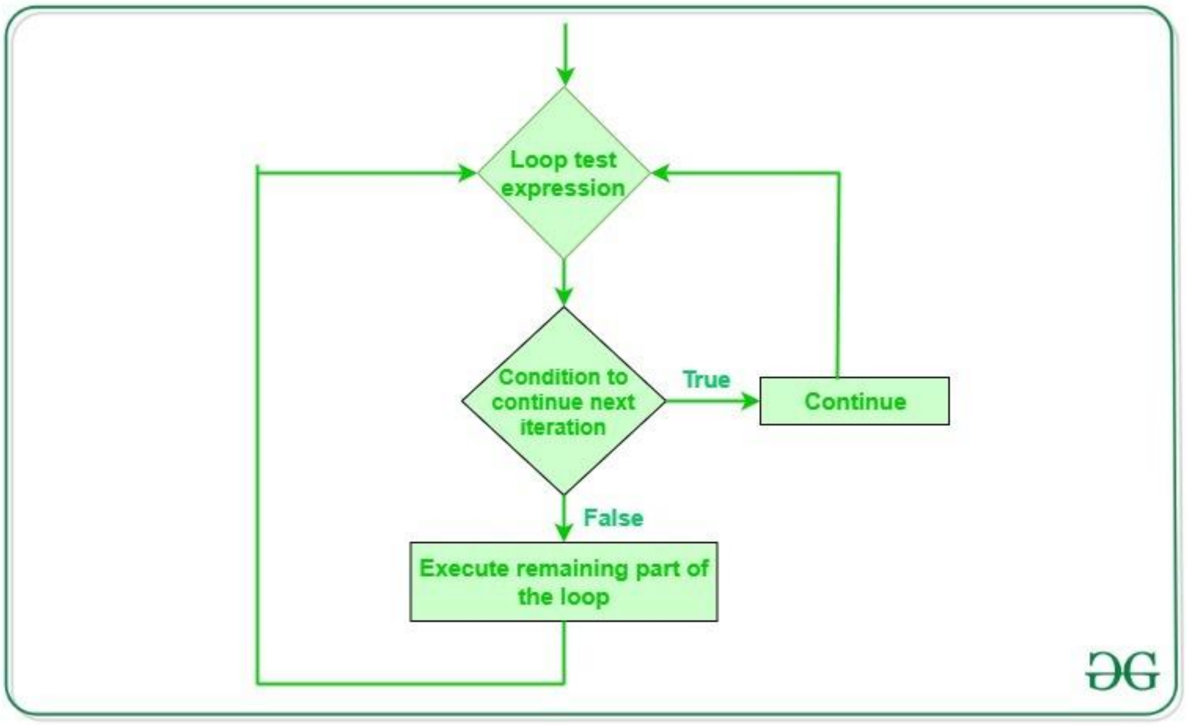
  The default raw memory allocator uses the following functions: Malloc(), calloc(), realloc(), and free() call malloc(1) (calloc(1,1)) when requesting zero bytes.

- Memory Interface:
  - pyMem_MALLOC(size)
  - pyMem_NEW(type,size)
  - pyMem_REALLOC(ptr,size)
  - pyMem_RESIZE(ptr, type, size)
  - pyMem_FREE(ptr)
  - pyMem_DEL(ptr)

Que: what is the purpose continue statement in python?

Ans:

- The continue statement in Python returns the control to the beginning of the while loop.
- The continue statement rejects all the remaining statements in the current iteration of the loop and moves the control back to the top of the loop.
- The continue statement can be used in both while and for loops.


Loop test expression
Condition to continue next iteration
True
Continue
False
Execute remaining part of the loop

Que: What are negative indexes and why are they used?

Ans:

- Negative indexing is used in Python to manipulate sequence objects such as lists, arrays, strings, etc.
- Negative indexing retrieves elements from the end by providing negative numbers as sequence indexes.

Ex:

- Slicing from index start to index stop-1
  Arr[start:stop]
- Slicing from index start to the end
  Arr[start:]
- Slicing from the beginning to index stop-1
  Arr[:stop]
- Slicing from index start to index stop, by skipping step
  Arr[start:stop:step]