Q:-1 How memory is manged in python?

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.

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. For example, integer objects are managed differently within the heap than strings, tuples or dictionaries because integers imply different storage requirements and speed/space tradeoffs. The Python memory manager thus delegates some of the work to the object-specific allocators, but ensures that the latter operate within the bounds of the private heap.

It is important to understand that the management of the Python heap is performed by the interpreter itself and that the user has no control over it, even if they regularly manipulate object pointers to memory blocks inside that heap. The allocation of heap space for Python objects and other internal buffers is performed on demand by the Python memory manager through the Python/C API functions listed in this document.

To avoid memory corruption, extension writers should never try to operate on Python objects with the functions exported by the C library: malloc(), calloc(), realloc() and free(). This will result in mixed calls between the C allocator and the Python memory manager with fatal consequences, because they implement different algorithms and operate on different heaps.

How memory is managed in Python

According to the Python memory management documentation, Python has a private heap that stores our program's objects and data structures. Python memory manager takes care of the bulk of the memory management work and allows us to concentrate on our code.

Types of memory allocation

There are two types of memory allocation in Python, static and dynamic.

1. Static memory

The stack data structure provides static memory allocation, meaning the variables are in the stack memory. Statically assigned variables, as the name implies, are permanent; this means that they must be allocated in advance and persist for the duration of the program. Another point to remember is that we cannot reuse the memory allocated in the stack memory. Therefore, memory reusability is not possible.

2. Dynamic memory

The dynamic memory allocation uses heap data structures in its implementation, implying that variables are in the heap memory. As the name suggests, dynamically allocated variables are not permanent and can be changed while a program is running. Additionally, allotted memory can be relinquished and reused. However, it takes longer to complete because dynamic memory allocation occurs during program execution. Furthermore, after utilizing the allocated memory, we must release it. Otherwise, problems such as memory leaks might arise.

Q:-2 what is purpose continue statement in python?

Overview

The continue statement in Python is used to stop an iteration that is running and continue with the next one.

How it works

The continue statement cancels every statement running in the iteration of a loop and moves the control back to the top of the loop. In simple words, it continues to the next iteration when a certain condition is reached.

We can use the continue statement in the while and for loops.

Continue Statement

In python continue statement terminates all the remaining iteration and move the control back to the beginning of the loop for the next iteration.

The continue statement can be used in both while and for loops. Here in this article we have explained continue statement examples with for and while lo

Python continue statement is used to skip the execution of the current iteration of the loop.

We can’t use continue statement outside the loop, it will throw an error as “SyntaxError: ‘continue’ outside loop“.

We can use continue statement with for loop and while loops.

If the continue statement is present in a nested loop, it skips the execution of the inner loop only.

The “continue” is a reserved keyword in Python.

Generally, the continue statement is used with the if statement to determine the condition to skip the current execution of the loop.

Q:-3 what are negative indexes and why are they used ?

Basically indexes used in array or list starts from indexing 0 in most of the programming languages.

But in Python, we have an extra feature of negative indexing. The negative indexing is the act of indexing from the end of the list with indexing starting at -1 i.e. -1 gives the last element of list, -2 gives the second last element of list and so on.

The use of negative indexing can be done to use or display data from the end of the list and can also be used to reverse a number or string without using other functions.

```
>>> msg = "Cat"
>>> msg[-1]
't'
>>> msg[-2]
'a'
>>> msg[-3]
'C'
```

- Python arrays & list items can be accessed with positive or negative numbers (also known as index).

- For instance our array/list is of size n, then for positive index 0 is the first index, 1 second, last index will be n-1. For negative index, -n is the first index, -(n-1) second, last negative index will be – 1.

- A negative index accesses elements from the end of the list counting backwards.