Python Track ←

Basic Data Structures

15-March

⇓

23-april

5weeks

⇓

4classes per week

→ Basic level data structures

↳ linear data structure → arrays
Stacks
Queue
Deques
LL, DLL

↳ Hierarchal data structu → Trees
↓
Binary Tree
BST

↳ HashMaps

Heaps

Mix problem solving

Data Structure ⇒ It is particular way of storing & organizing your data.

Arrays

→ Memory Management

→ Scoping

→ Basic Arrays

→ Dynamic Arrays → lists → How to make own lists

→ Problem Solving

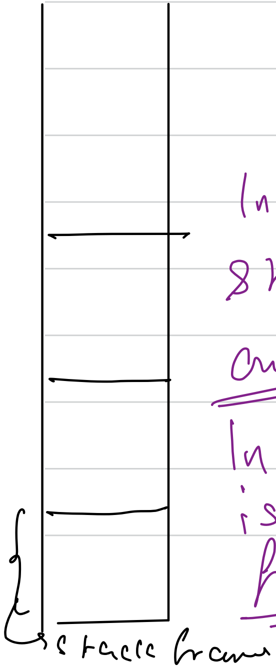
Memory Management → 2 major partitions

Call Stack → linear memory space

→ follows LIFO structure

In python → call stack stores function calls only

In call stack one entry is called one stack frame



→ Heap area

→ Big pool of memory

→ no specific orientation

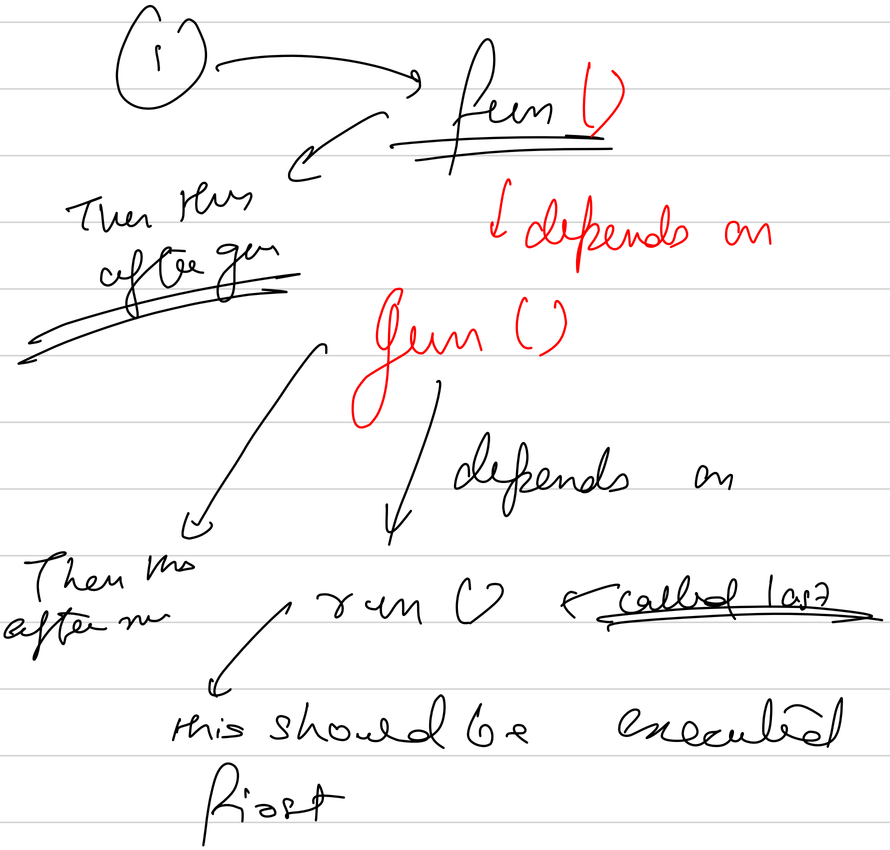
whenever we call a function, a stack frame enters the call stack.

1 stack frame ⟶ 1 functi call

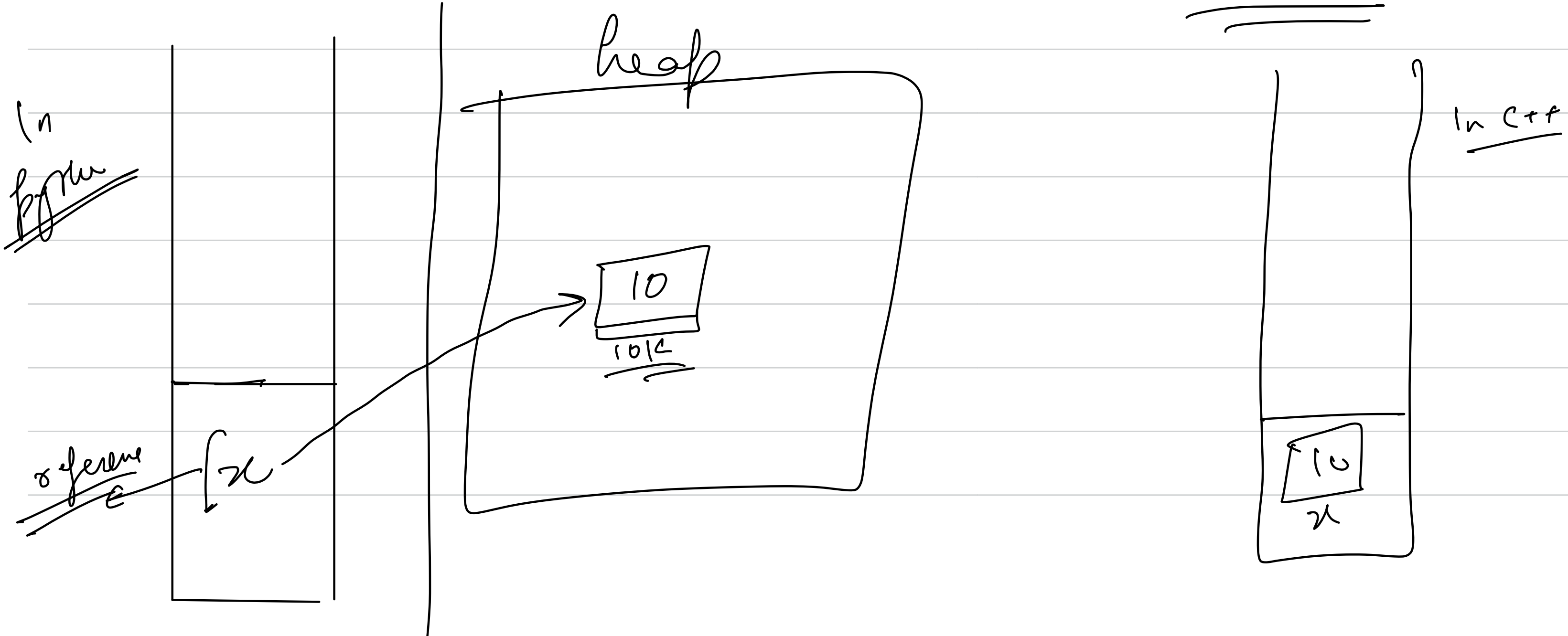
```
'''
def fun():
    print("Having fun")

fun()
'''
```

when function call is completed or we hit return the stack frame is removed.

```
def run()

def gen()
    run()

def fun()
    gen()

→ fun()
```



In python a Stack frame only stores references to the variables wheras in C++ they actually sometimes store the variable values.



→ There are some other minor facilities for memory of a process also

- → Space for global variable
- → Kernel stack

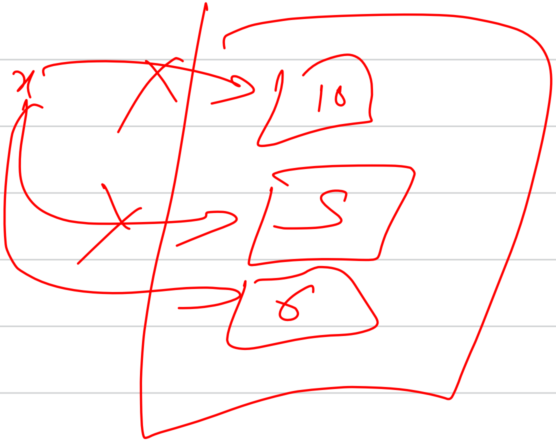
We have a limited memory

x = 10

x = 5

x = 6

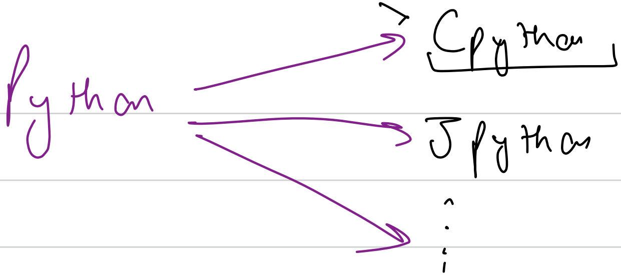
x = 7

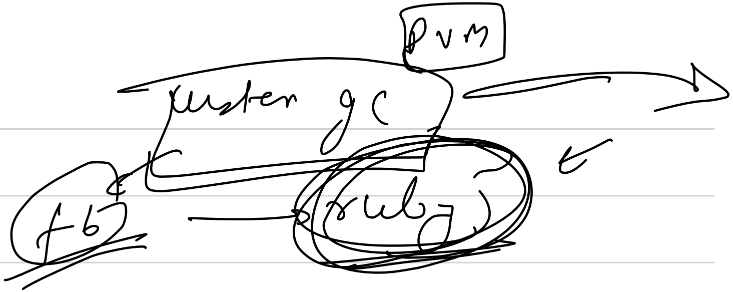


Garbage collector

This is an automatic process, which frees unused memory location

Java





Cpython → It keeps track of all the allocated memory

It maintains generations of the memory

Spot

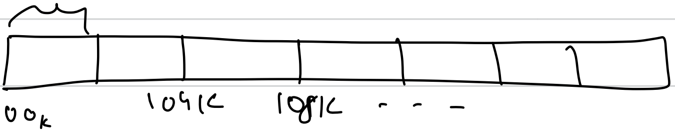
1. first generation → newly allocated memory
2. older generation → garbage collector ran but any memory was not cleaned, then thats an older generation

→ It maintains no. of allocated memory spots & checks if they exceed a threshold or not. When they exceed the threshold GC triggers

Reference Counting

Arrays → Arrays are linear data structure, which stores homogenous data in contigious memory location

int → 4 bytes

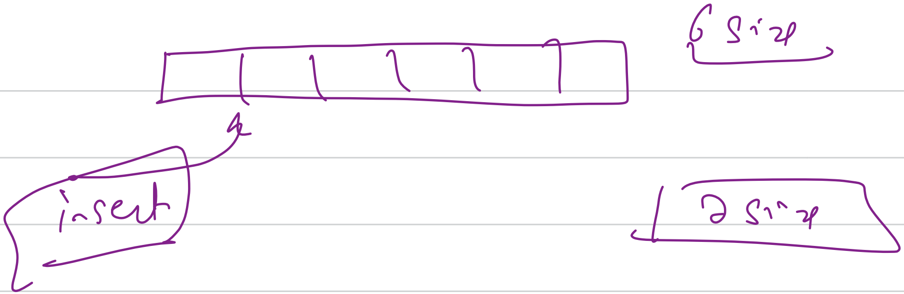


C++

Java

Pyth

For arrays we need to specify the size as they have fixed length always



generally → create new arr of bigger size & copy elements

In python for arrays we have 0 based indexing.

arrays are mutable

C++

arrays ⟷ Vector

Java

arrays ⟷ Arraylist

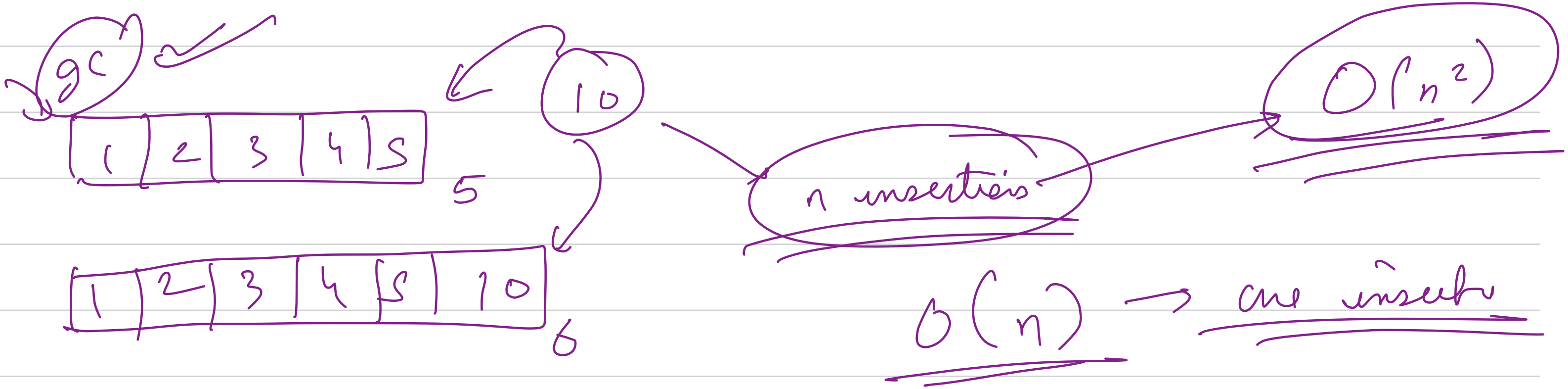
Python

Arrays ⟷ Lists

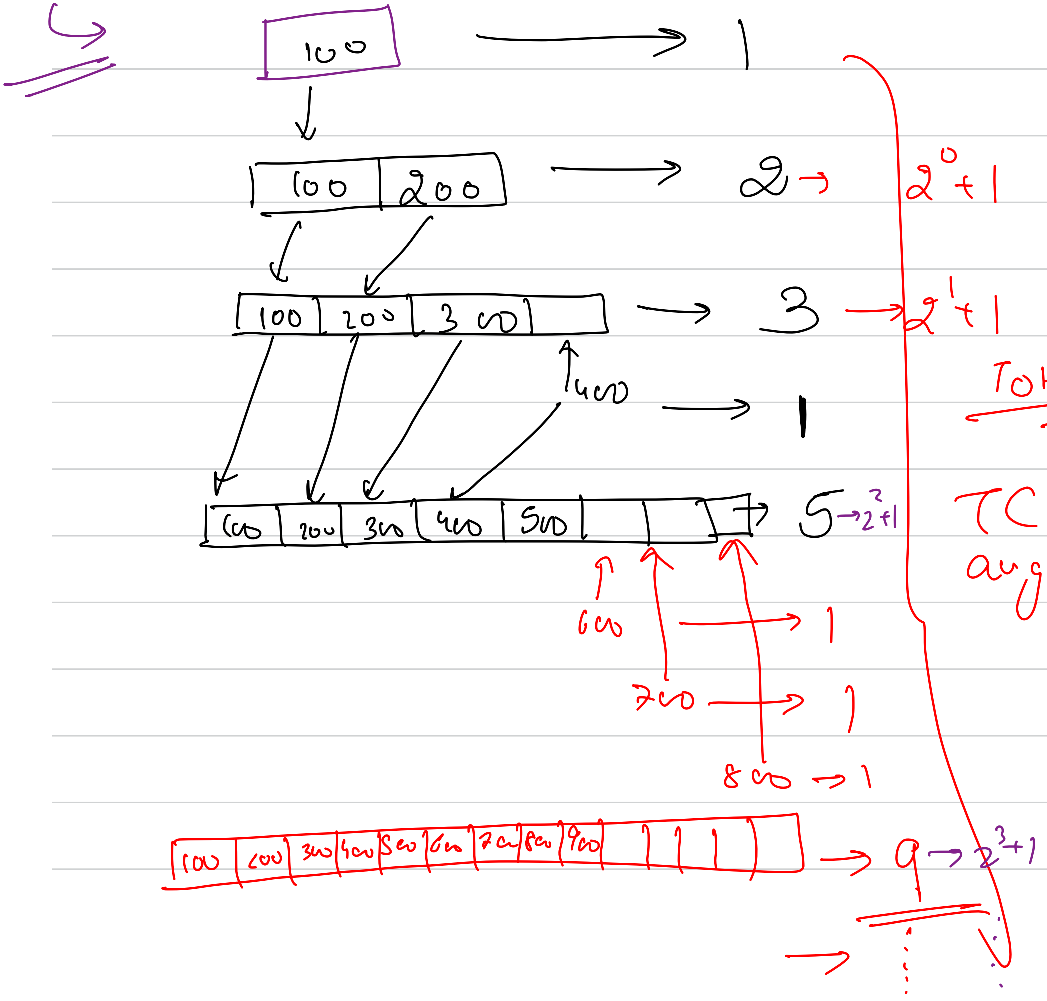
lists → resizable data structure, containing heterogenous elements stored in continuous memory loc.

How lists are as effecient as an array

→ at runtime we can resize. So internally lists/vectors/arraylists are implemented using constant size arrays only.



Lists don't inc size everytime. They double the size when they are full.



100, 200, 300, 400, 500 - - 900

$\underbrace{\qquad\qquad}_{n \text{ ele}}$

Total n insertion

$$TC_{avg} = \frac{\text{Total operation}}{n}$$

$\Downarrow$

avg complexity per insertion

$$\rightarrow \frac{(1 + 2 + 3 + 1 + 5 + 1 + 1 + 1 + 9 \ldots\ldots)}{n}$$

$$\rightarrow \frac{(1 + (2^0+1) + (2^1+1) + 1 + (2^2+1) + 1 + 1 + 1 + (2^3+1) \ldots\ldots)}{n}$$

$$\rightarrow \frac{(\overbrace{1+1+1+1 \ldots\ldots}^{n \text{ items}} + \overbrace{(2^0+2^1+2^2 \ldots\ldots)}^{\log_2 n})}{n}$$

$$\frac{n + 1 \times 2^{\log_2 n} - 1}{n} \qquad \boxed{n \rightarrow 1}$$

$$\rightarrow \frac{2n-1}{1} \approx \underline{\text{constant}} \quad \underline{O(1)}$$

Q→ Given a list of n size, and a non-negative no. k. rotate the list by k steps.

$K > n$ $\quad$ $K = K \% n$

$[\overbrace{1, 2, 3, 4}^{n-k}, \underbrace{5, 6, 7}_{\text{last k elem}}]$

$n \leq 10^7$

$K = 3$

→ $[5, 6, 7, 1, 2, 3, 4]$ ans

O(1) space

→ We reverse the whole list

$[5, 6, 7, 1, 2, 3, 4]$

$[\overbrace{7, 6, 5}^{K}, \overbrace{4, 3, 2, 1}^{n-k}]$