Data structure and Algorithms

Course

Created by : Israa Abdelghany
LinkedIn : www.linkedin.com/in/israa-abdelghany-4872b0222
GitHub : https://github.com/IsraaAbdelghany9

Data struction & Algorithms

what is it?

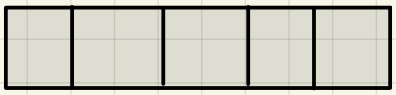
a way to store & deal with data.

- discussing the idea of Linked List each list in the Linked list called node

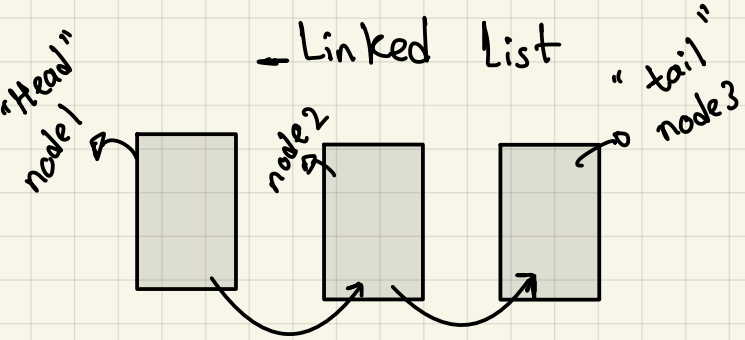
Data structure

Linear data struction

- elements following each other
- example is
  - array



  - Linked List

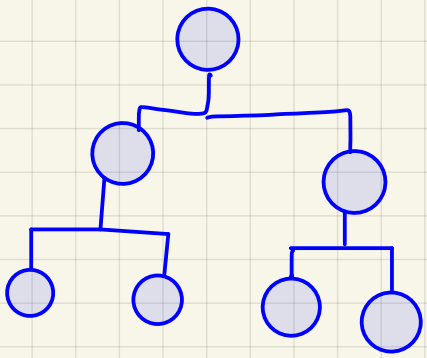


- Linearly reffering to each other

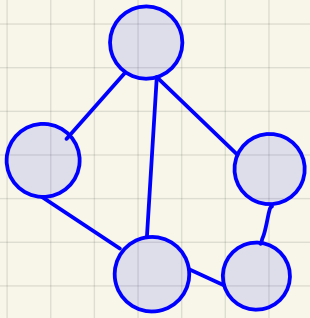
non Linear data structure

ex;-

- Tree



- Graph



- non linear reffering to each other

→ dynamic alocation vs data structure

→ functions are not the best from performance wise
(over head of function performance)

so you have the choice increase the memory or Lower the performance and time

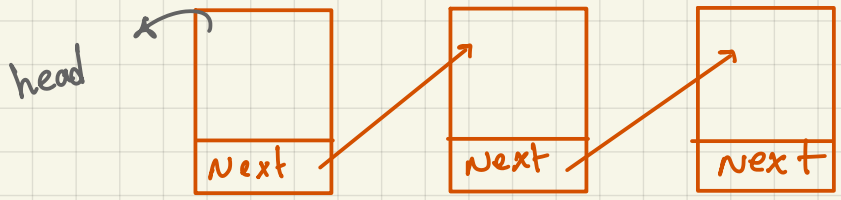
→ if you know the size you have or you need you Can use array
if not you can use Linked lists

→ search in array is faster
insert & delete is better in Linked List

storage depends on the purpose

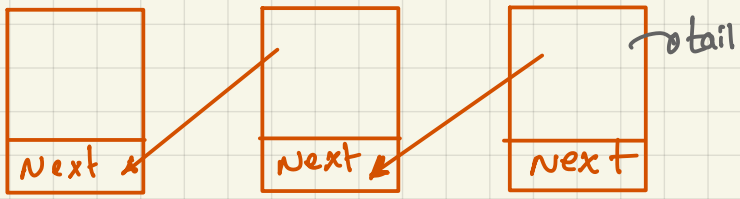
"Linked Lists types"

[1] A



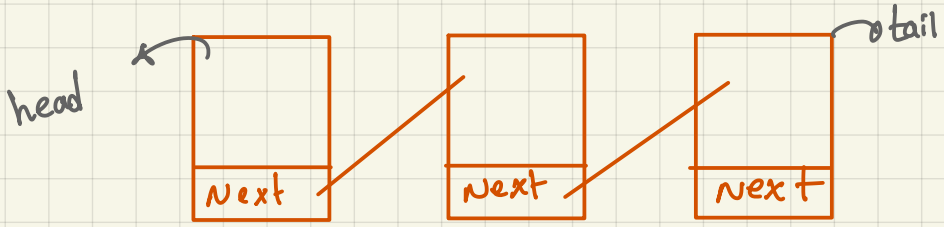
→ Called Single Linked List (forward only)
the most important part is the head

[1] B



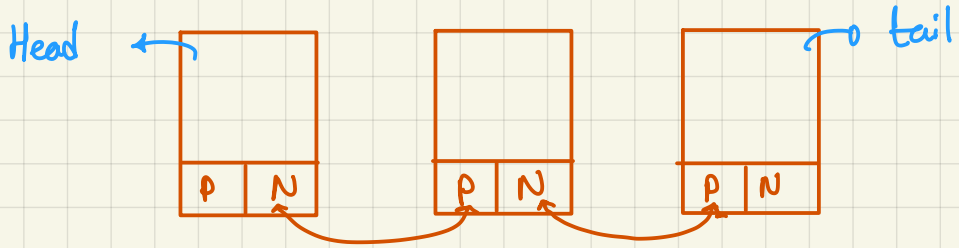
Single Linked List (backward only)
the most important is the tail

[1] C



Some Causes are Single Linked Lists but work in both directions so it needs both head & tail.

[2]



→ This is a double Linked List

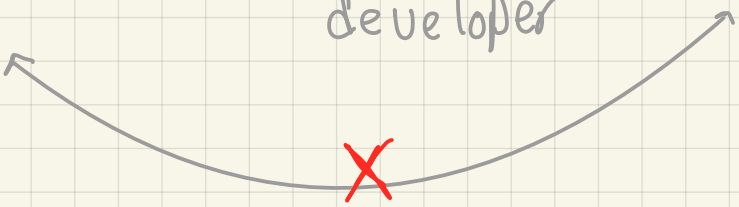
Notes :-

- ⇒ if you have head and tail that doesn't mean it's a double Linked list → you may don't need both
- ⇒ name of the array is address to the first element
- ⇒ there's no fixed structure for Linked list
- ⇒ if you need a function to be implemented Like insert or search you need to provide the full information
- ⇒ the head or the tail is reference in C#, Java. and Py pointer in C , C++ referring to the first or last node
- ⇒ the Linked list in C , C++ can have different data types using null ptr
- ⇒ in Python no need to define the data type

Coder

developer

end user



Job description;-

- add :- create new node at the first of the List
- Insert:- // node and add it to specific position (zero based or you can choose)

  if location is greater than the number of nodes add at end

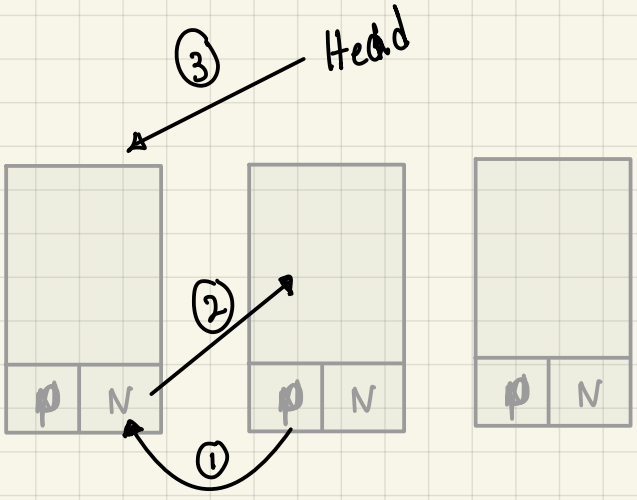
- Search:- find the location of first occurance. → will have return (location)
- Delete :- Delete a node by its location (zero based) → will have return

After break 1

→ No addresses in Python just references.

→ object in python are reference type.

⇒ over head of method call. it's when you call function you need to understand the over head of method call that it will reduce the performance &



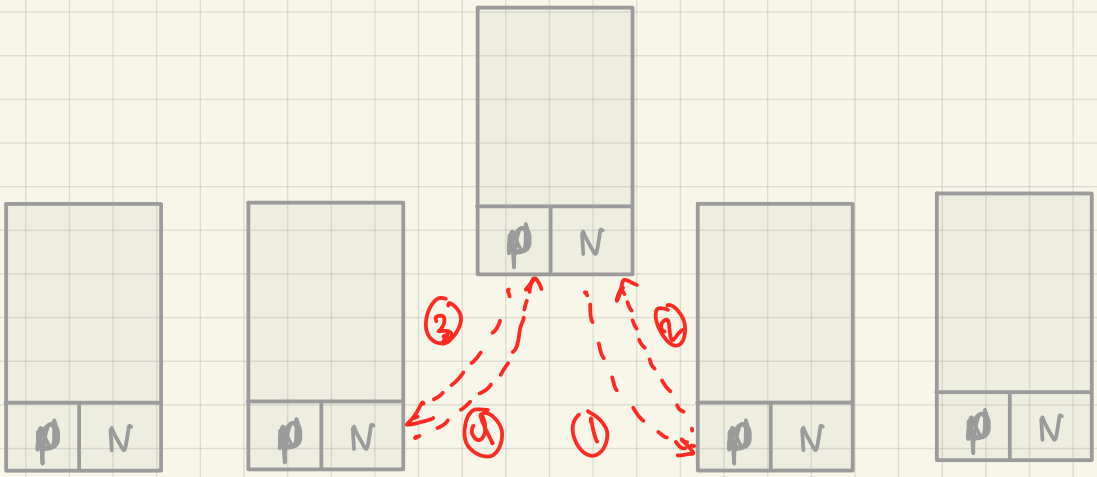
insert method :-

```
def Insert (data, loc)
    nd = Node (data)
    if (self.Head = None):
        nd.Head = nd
        nd.tail = nd          } if empty
    else if (loc == 0):       # insert at the beginning
        self.head.prev = nd   ⇒ ①
        nd.next = self.head   ⇒ ②
        self.head = nd        ⇒ ③
```

Note

I could use the add if empty but this will lead to over head of method call to detete 2 Liner and write only one

Loc = 4

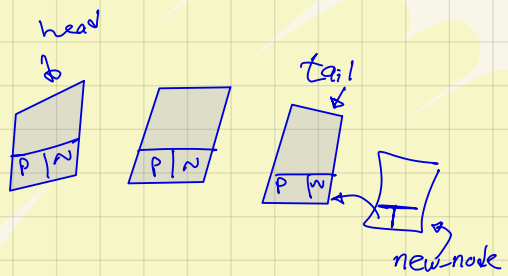


we have some cases:-

⇒ the code:-

```
i = 0
current = Head
while( i < Loc-1 and current != None):
    current = current.next
    i = i+1
if (current = self.tail or current=None):
    same thing as you will add node
    at the end of the Linked List
```

at the end



new_node.prev = self.tail

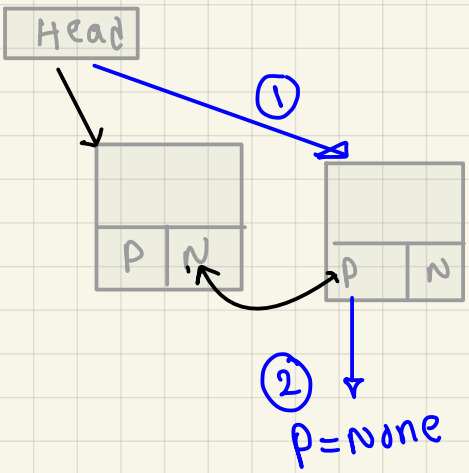
self.tail.next = new_Node

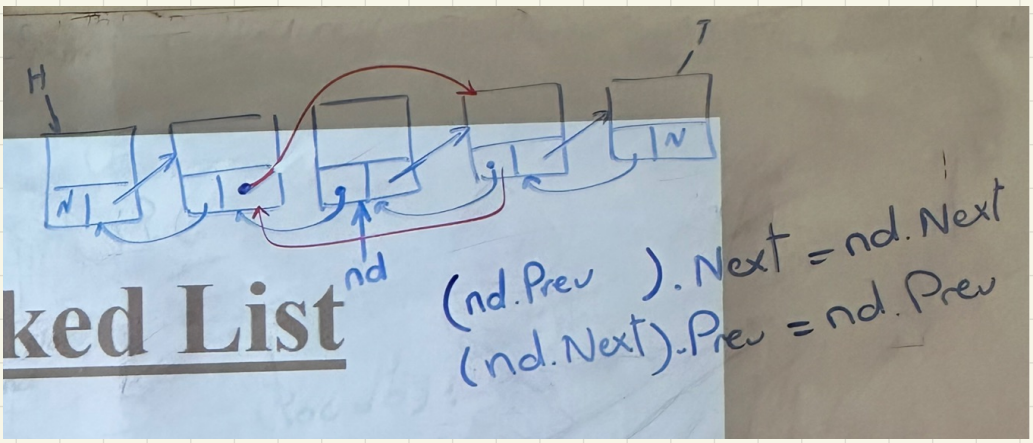
self.tail = new-node

After break 2:

the delete function:-

```
def delete (self, loc):
    Deleted = false
    if (self.head != None):          # to check if there's already empty
        if (loc == 0):
            if (self.head == self.tail):
                self.head = None
                 .tail = None
            else:
                self . head = self .head .next
                self . head . prev = None
            deleted = true
        else :
            nd = self . head
            i = 0
            while (nd != None && i < loc):
                nd = nd . next
                i = i + 1
            if nd != None
                if (nd == self . tail)
                :
                else :
```





this for the delete if the location is between the head & tail & != None

---

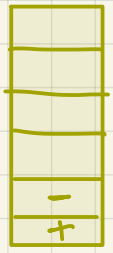
Last in first out (LIFO)

First in Last out (FILA)

- stack is to use data
- stack is a concept applied and performed on data so it can be called as data structure.
-

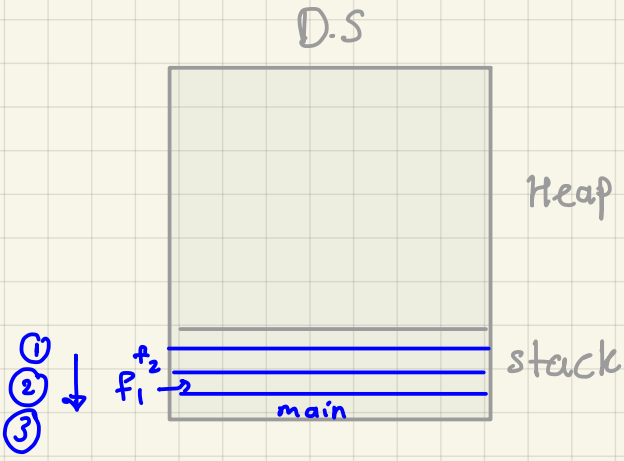
Line calculator

3+4*5-2



4*5 because * is higher than - behind it

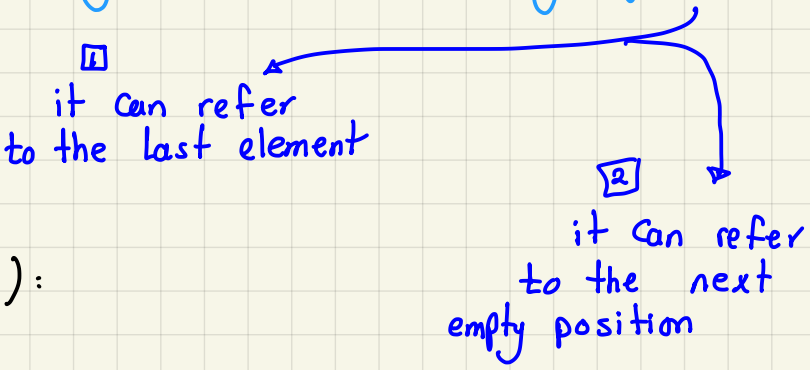
• so it needs stack.



main entered at first and left the last

Stack → array, Linked Lists (Linear Data structure)

- array of numpy and List are Linked List (in python)
- to deal & implement stack you need the array & tos



Code:-

```
class stack :

    def __init__(self,size):

        self.Ar = [ ]
        self.tos = 0
        self.size = size

    def __init__(self):  → over loading.

        self.Ar = [ ]
        self.tos = 0
        self.size = 10


    def push (self , data):
        pushed = false
        if (self.tos < self.size
        |     self.Ar.append (data) X self.Ar[self.tos] = data
        |     tos = tos + 1
        |     pushed = true
        |
        return pushed
```

```
def pop (self):
    popped = -1
    if ( self.tos > 0) :
        tos = tos -1
        popped = self. Ar [self.tos]
    return popped
```

think as a stack :-

⇒ you only have push and pop

⇒ if you want to count the number of elements in the stack you can create another stack pop from the stack 1 and increase the counter then push in another one

⇒ don't deal with stack with other functions only push & pop.

Queue

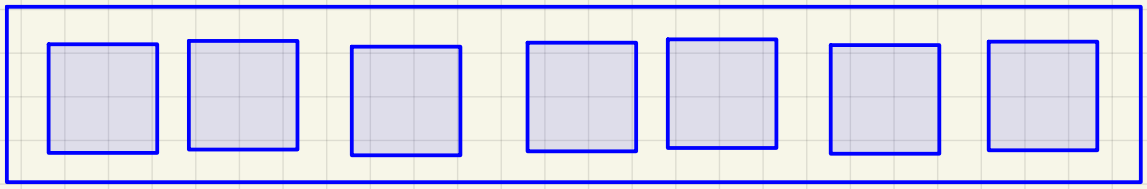
Linear Queue

elements are moving to the casheir

Circular Queue

Casheir is moving to elements

Windows has message Queue.



Queue
- Array
- Linked List

- In retreving I need Next only not previous so only single List needed.
- In adding I need the tail so I need both tail & head

→ this is a way of thinking :-

to implement Queues I don't know which Linked List I need to use so :-

[1] at first I need to select the most general type to start to test it which is double Linked list and find if I need all its features or can move to another one

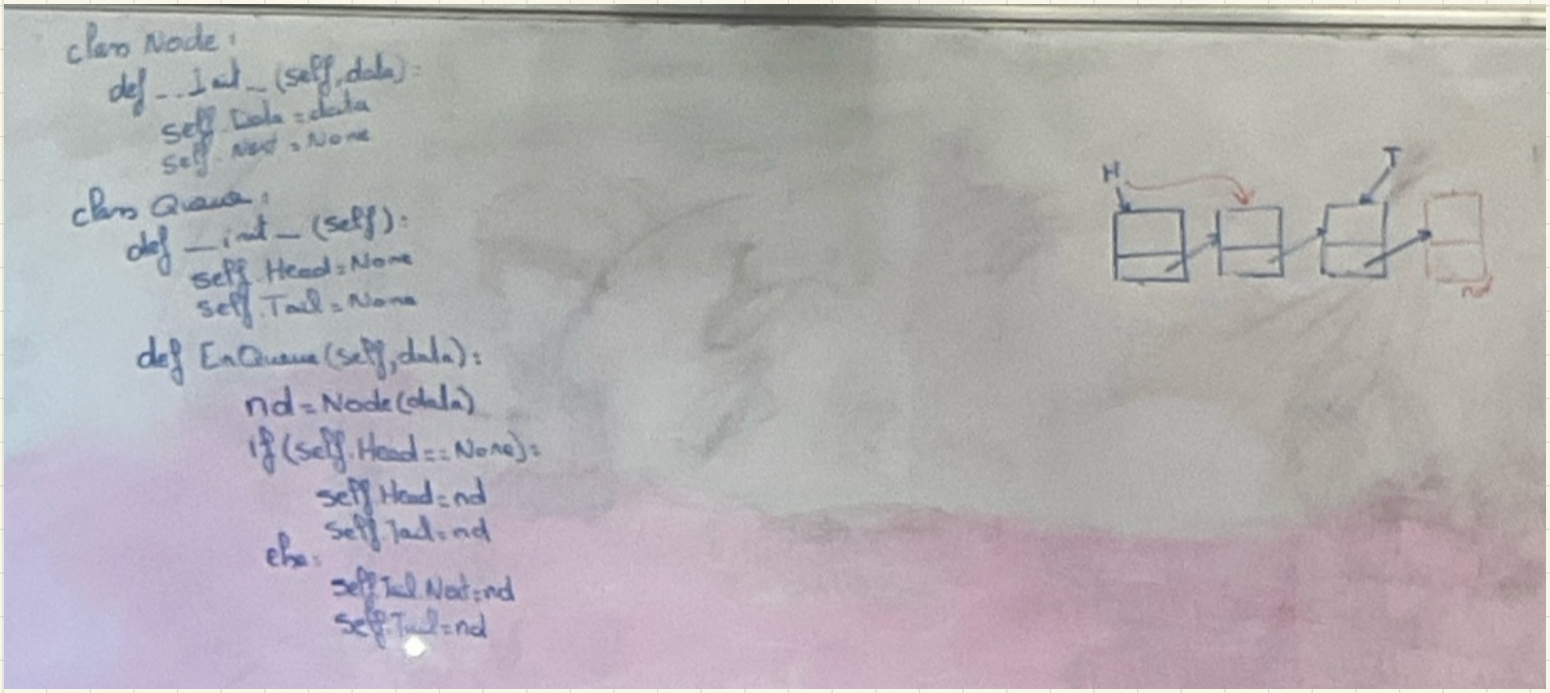
[2] I found that I need only the next. previous is not used to the double Linked list is not important I can use only single Linked list

[3] we finished the first and large categorization Now we can choose to have head only or tail only or both

[4] I need head to pop them FIFO and I need tail to add elements later

So I need both //

Code of Queue:-




```
def DeQueu(self):
    nd = None
    if ( self.head != None):
        nd = self.head
        self.head = self.head.next
        if (self.head ==None):
            self.tail = None
    return nd
```

if it was the last element then we need to free head & tail to indicate it's empty now