

Data structure and Algorithms

Course

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

Data structure & 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

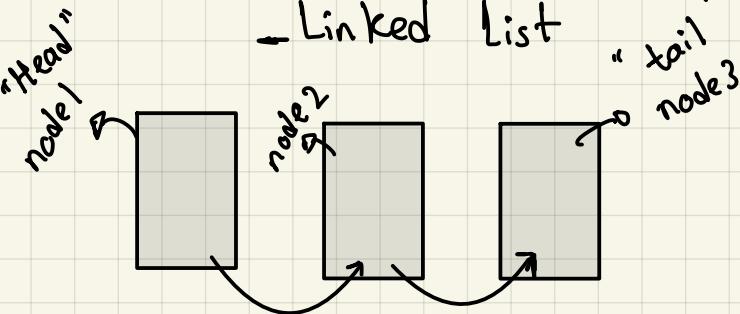
Linear data structure

- elements following each other
- example is

- array



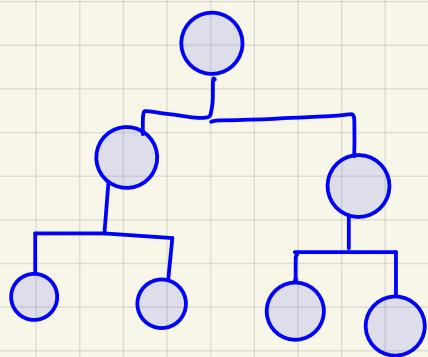
- Linked List



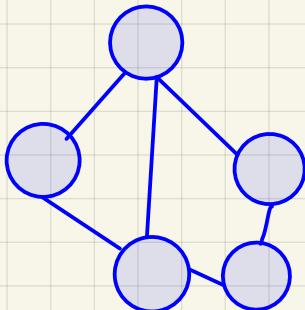
non Linear data structure

ex:-

- Tree



- Graph



- Linearly referring to each other

- non linear referring to each other

→ dynamic allocation 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

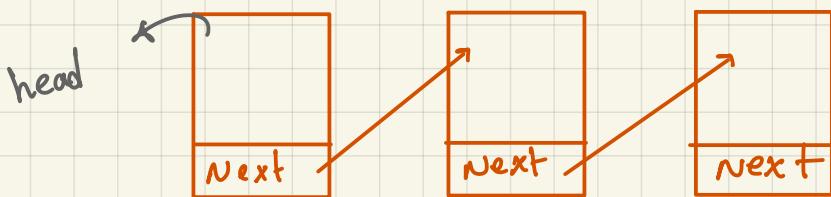
→ if you know the size you have or you need you can use array
if not you can use Linked Lists

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

storage depends on the purpose

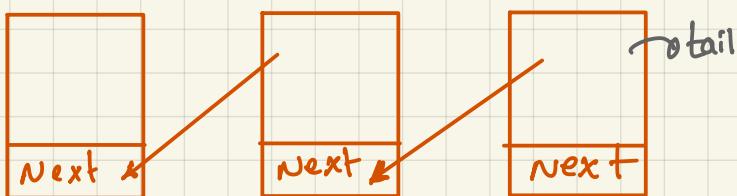
"Linked Lists types"

II A



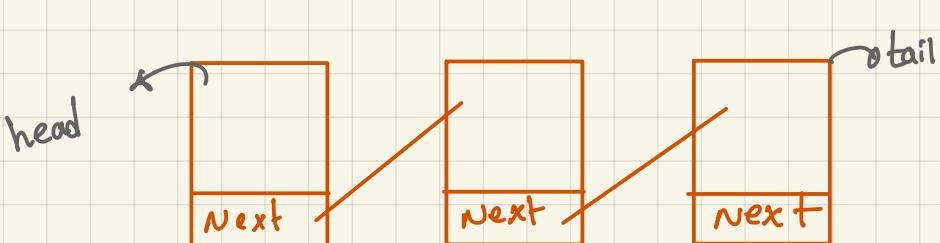
→ Called Single Linked List (forward only)
the most important part is the head

II B



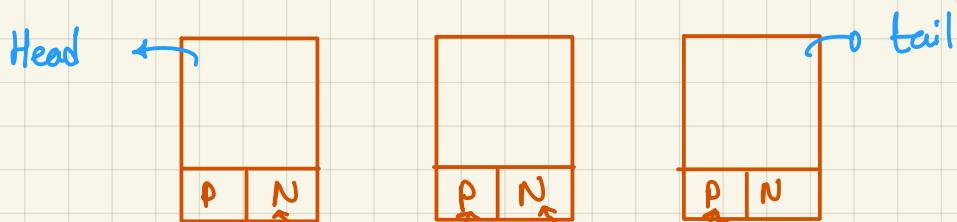
Single Linked List (backward only)
the most important is the tail

II 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 Python 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 you 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 occurrence. → will have return (location)
- Delete :- Delete a node by its location (Zero based) → will have return



After break I

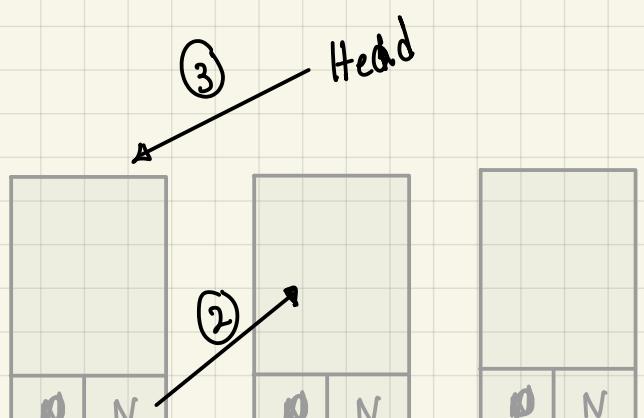
→ No addresses in Python just references.

→ object in Python are reference type.

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

insert method :-

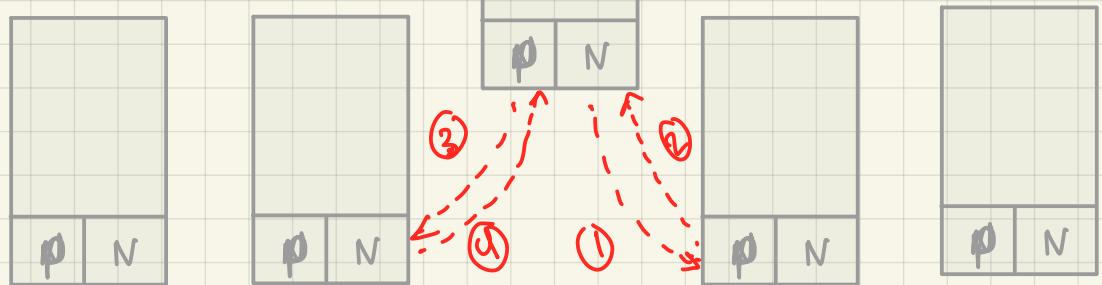
```
def Insert (data, loc)
    nd = Node (data)
    if (self .Head == None):
        nd .Head = nd
        nd .tail = nd
    else if (loc == 0): # insert at the beginning
        self .head .prev = nd => ①
        nd .next = self .head => ②
        self .head = nd => ③
```



Note

I could use the add if empty but this will lead to overhead of method call to delete 2 lines and write only one

Loc = 4



We have some cases:-

→ the code:-

$i = 0$
current = Head

while($i < \text{Loc}-1$ and current != None):

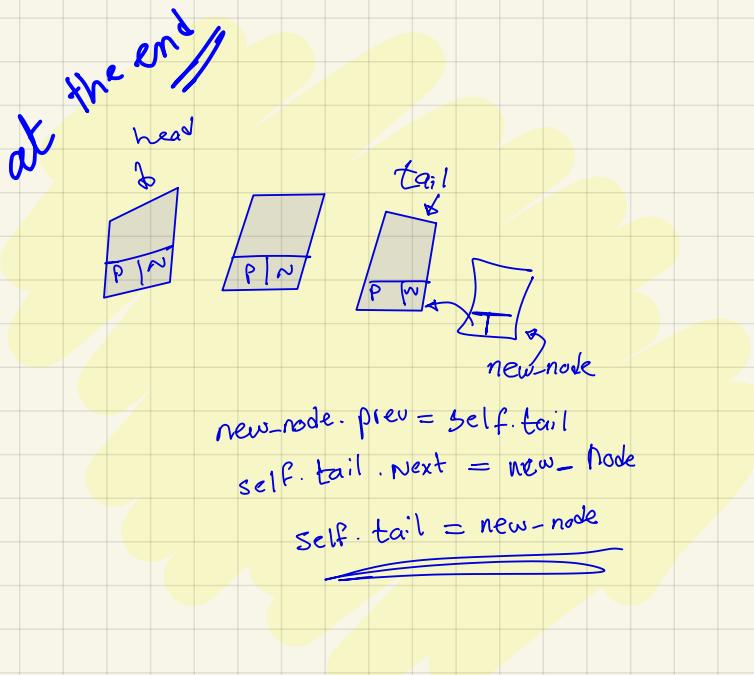
 current = current.next

$i = i + 1$

if (current = self.tail or current = None):

 some thing as you will add node

at the end of the Linked List

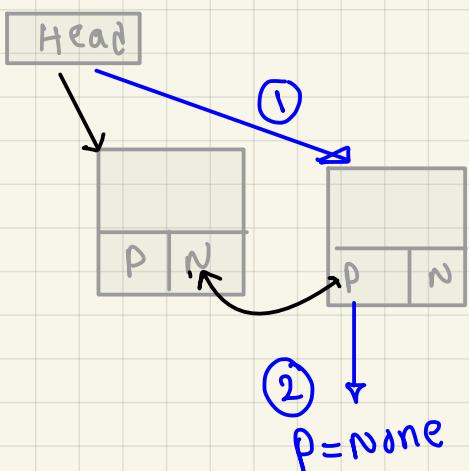


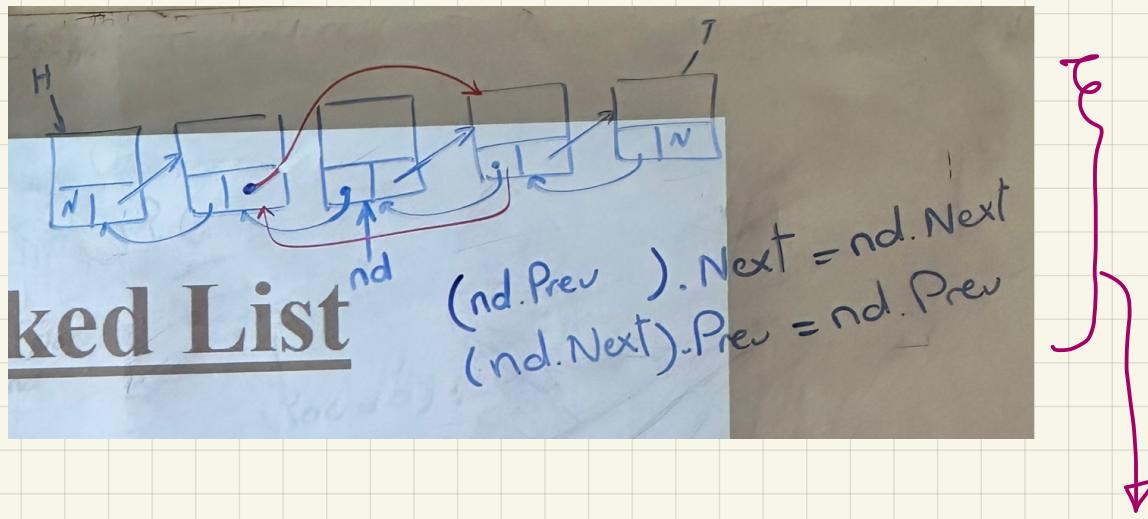
After break 2:

the delete function:-

```
def delete (self, loc):  
    Deleted = false  
    if (self.head != None):  
        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:
```

to check if there's already empty





this for the delete if the location
is between the head & tail & $! = \text{None}$

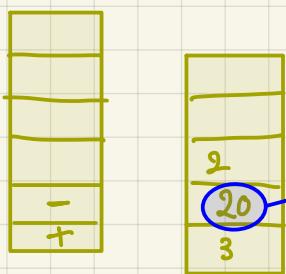
Last in first out (LIFO)

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



main entered at first
and left the last

• So it needs stack.



(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
```

[1] it can refer
to the last element

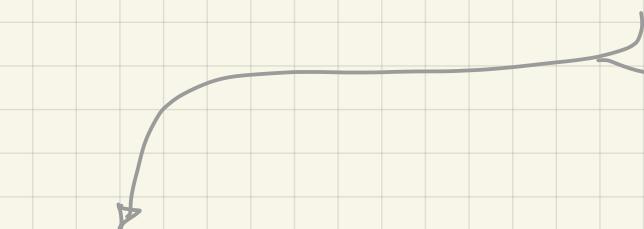
[2] it can refer
to the next
empty position

```
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



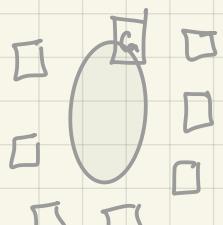
Linear Queue

$a \rightarrow b \rightarrow c \dots$ 1st in 1st out



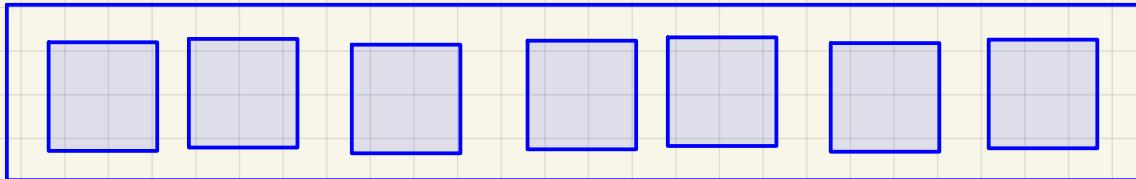
elements
are moving to the casheir

Circular Queue



Casheir is moving to elements

Windows has message Queue.



Queue

→ Array
→ Linked List

- In retrieving 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 ~~11~~

Code of Queue:-

```
class Node:  
    def __init__(self, data):  
        self.data = data  
        self.next = None  
  
class Queue:  
    def __init__(self):  
        self.head = None  
        self.tail = None  
  
    def EnQueue(self, data):  
        nd = Node(data)  
        if (self.head == None):  
            self.head = nd  
            self.tail = nd  
        else:  
            self.tail.next = nd  
            self.tail = nd
```



def DeQueue(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