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

a way to store & deal with data.

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

Data structure

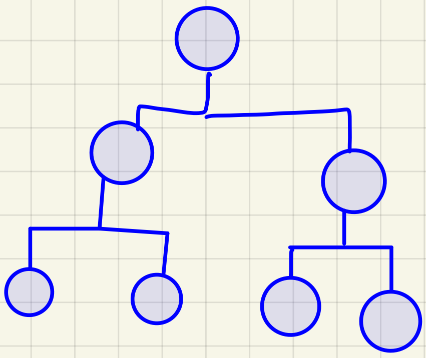
Linear data struction

- elements following each other
- example is
  - array
  - Linked List
- Linearly reffering to each other

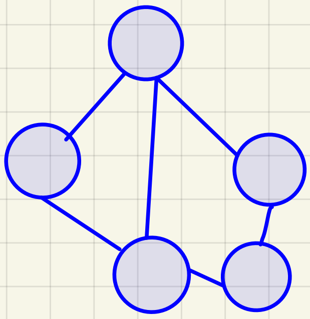
non Linear data structure

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)

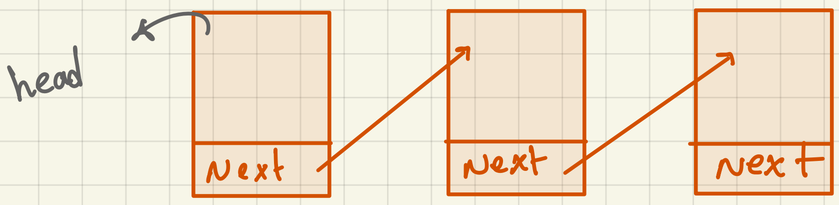
so you have the choice increase the memory or Lower the performance and time

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

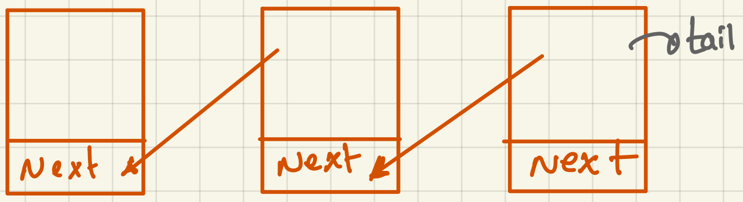
→ search in array is faster
insert & delete is better in Linked List

storage depends on the purpose

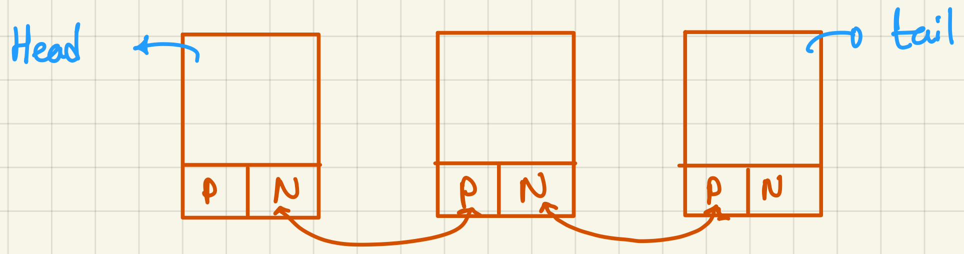
→ Linke List types



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



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



→ This is a double Linked List

⇒ 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

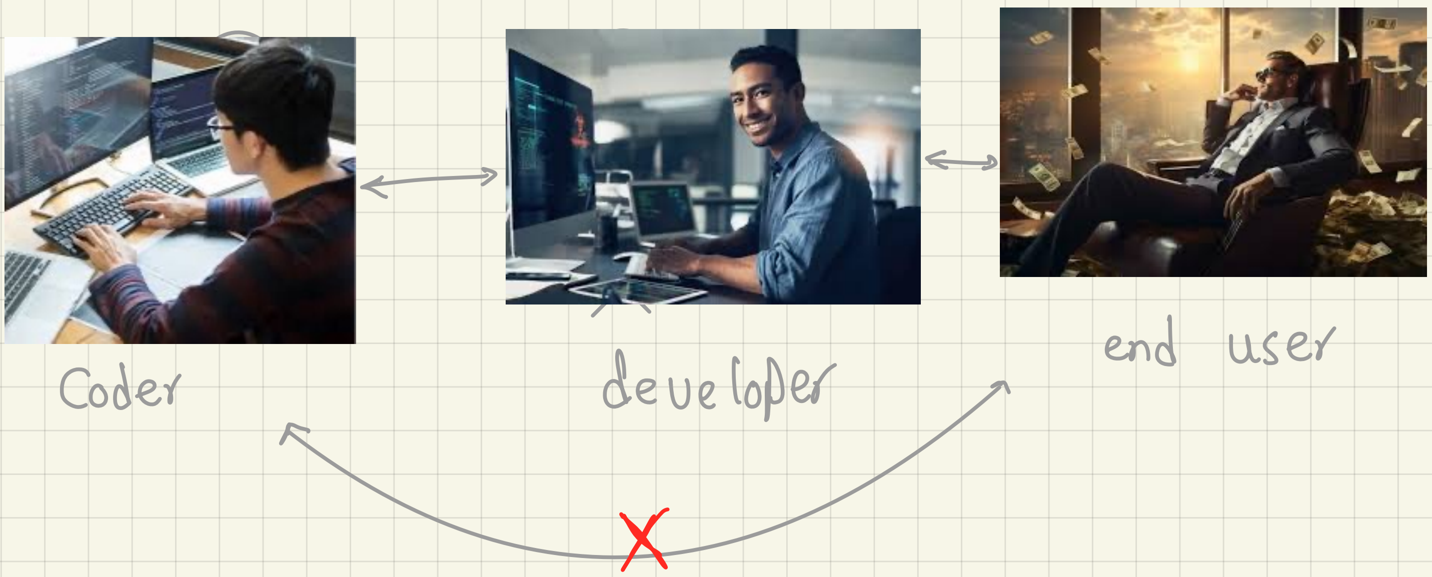
⇒ there's no fixed structure for Linked List

⇒ 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 you need to define the data type



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

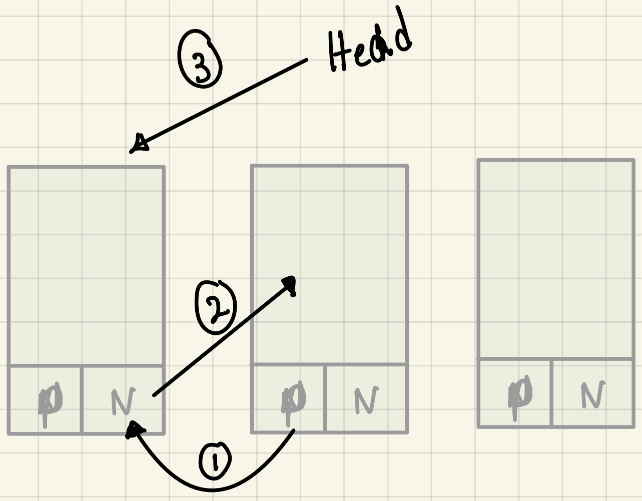
→ No addresses in Python just references.

→ 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 &

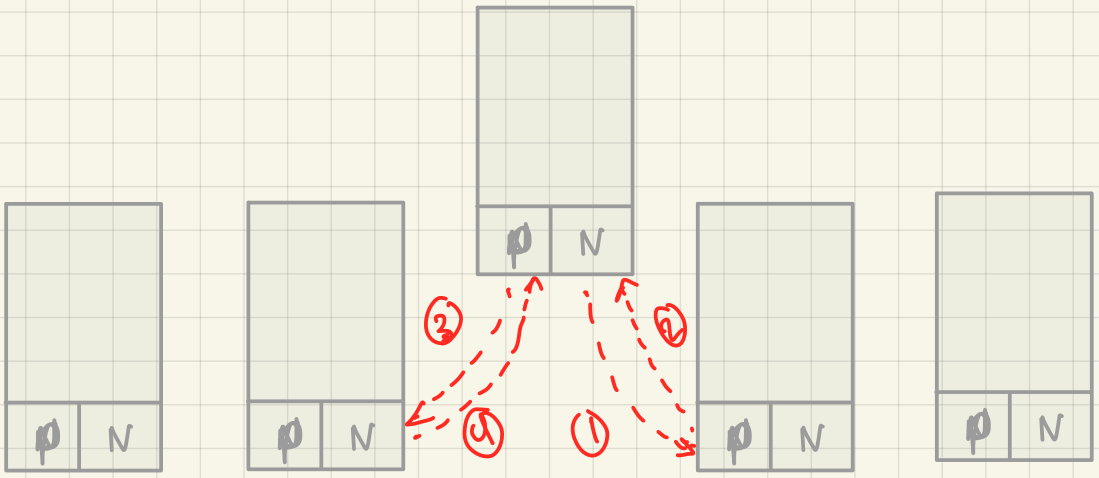
insert method :-

```
def Insert (data, loc)
    nd = Node (data)
    if (self.Head = None):
        nd.Head = nd    }
        nd. tail = nd   }  if empty
    else if (loc == 0):
        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 delete 2 Lines 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 ):
```

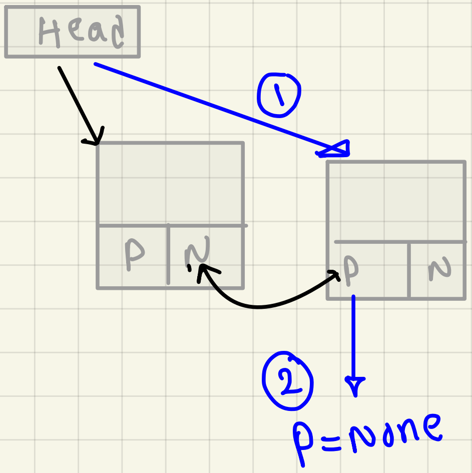
same thing as you will add node
   at the end of the Linked List

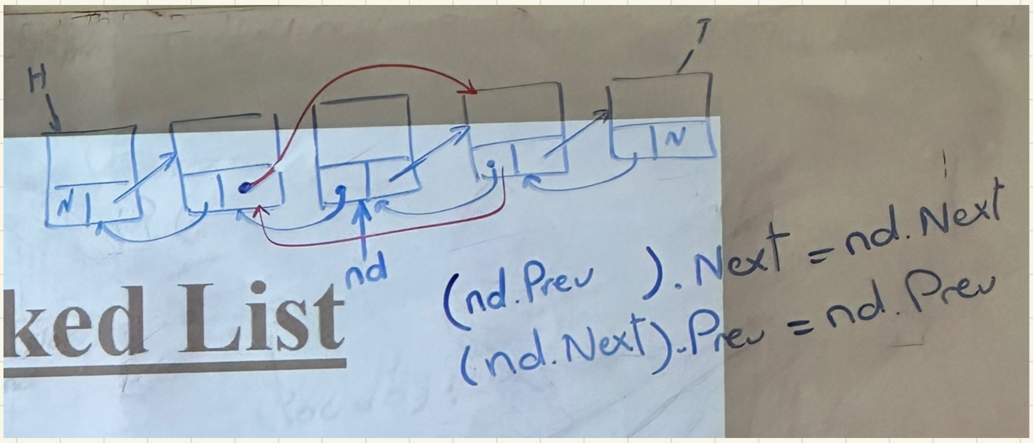
After break 2:

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 & != None

---

Last in first out (LIFO)

First in Last out (FILA)

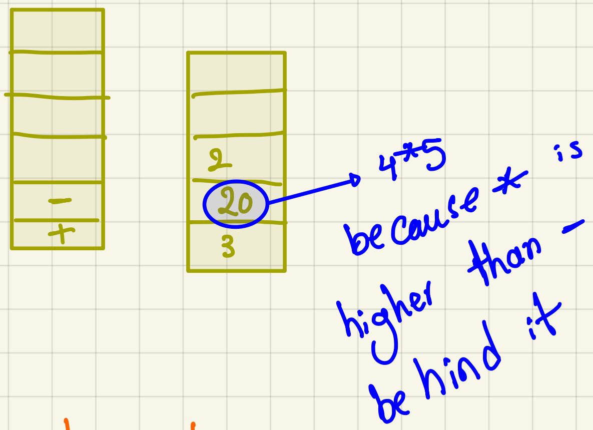
_ stack is to use data

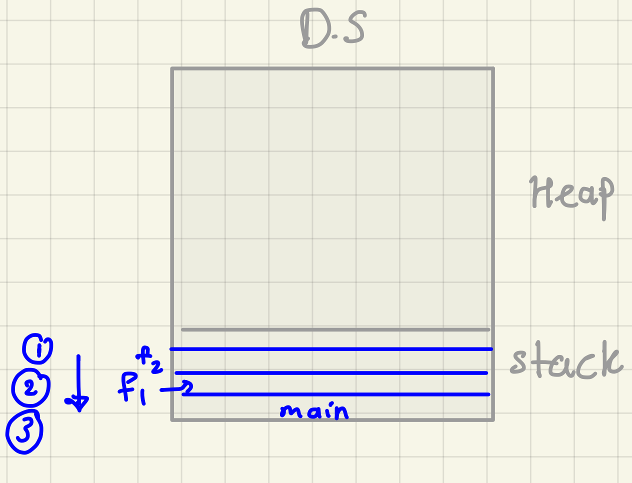
_ stack is a Concept applied and performed on data so it can be called as data structure.

_

Line calculator

$3+4*5-2$





main entered at first and left the last

• So it needs stack.

$$\begin{tikzcd}[row sep=small] & \text{array} \\ \text{Stack} \arrow[ur] \arrow[dr] & \\ & \text{Linked Lists} \end{tikzcd}$$ (Linear

- array of numpy and List are Linked List

- to deal & implement stack you need the array & tos

$\boxed{1}$ it can refer to the last element

$\boxed{2}$ it can refer to the next empty position

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

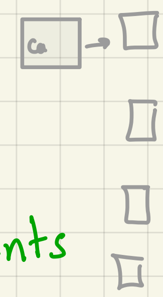
⇒ if you want to count the number of elements in the stack you can create another stack pop from the stack 1 and increase the counter then push in another one

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

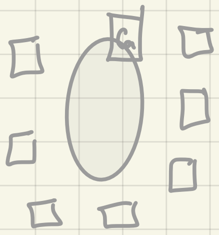
Queue

Linear Queue

Circular Queue



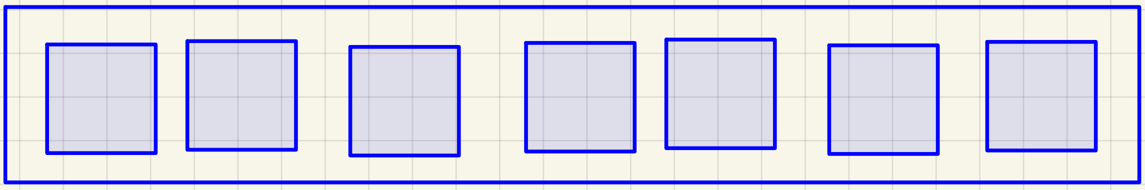
1^{st} in 1^{st} out



elements are moving to the cashair

Cashair is moving to elemen

Windows has message Queue .



$$\begin{tikzcd} & \text{Array} \\ \text{Queue} \arrow[ur] \arrow[dr] & \\ & \text{Linked List} \end{tikzcd}$$

- 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:-

$\boxed{1}$ at first I need to select the Largest shape to test

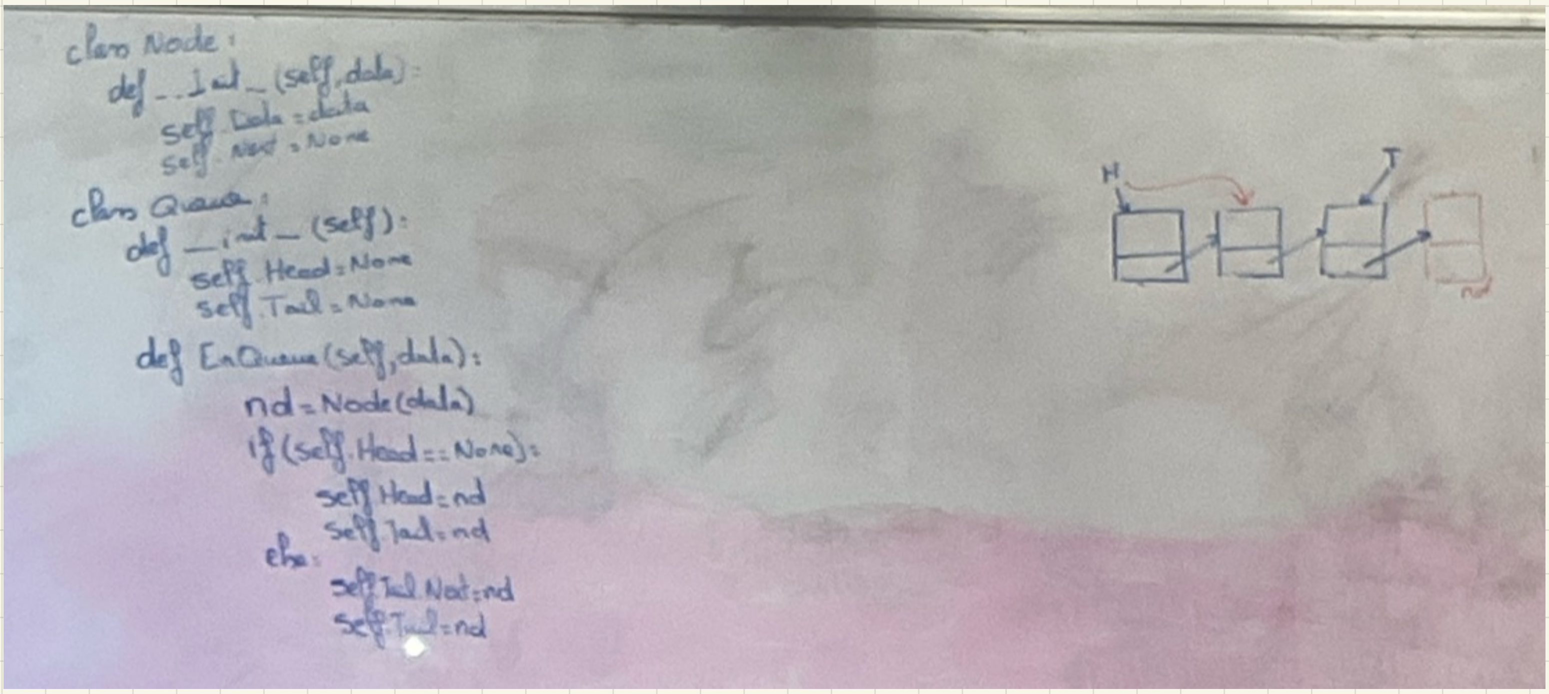
$\boxed{2}$ I found that I need only the Next previous is not used to the double Linked list is not important I can use only single Linked List

$\boxed{3}$ we finished the first and Large categorization Now we can choose to have head only or tail only or both

$\boxed{4}$ I need head to pop them FIFO and I need tail to add elements later

So I need both #

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 DeQueu(self):
    nd = None
    if (head.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