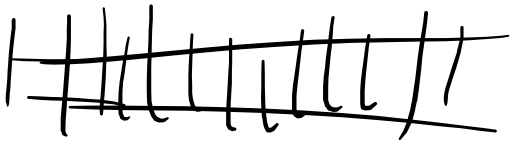
2025-02-20 - To develop program for data structure using built in functions in python

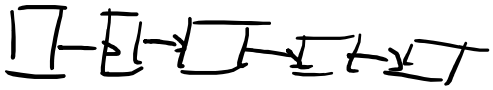
20 February 2025 08:14

Linked Lists :

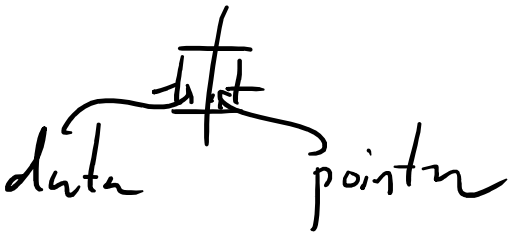
List <=> Array



Function oriented



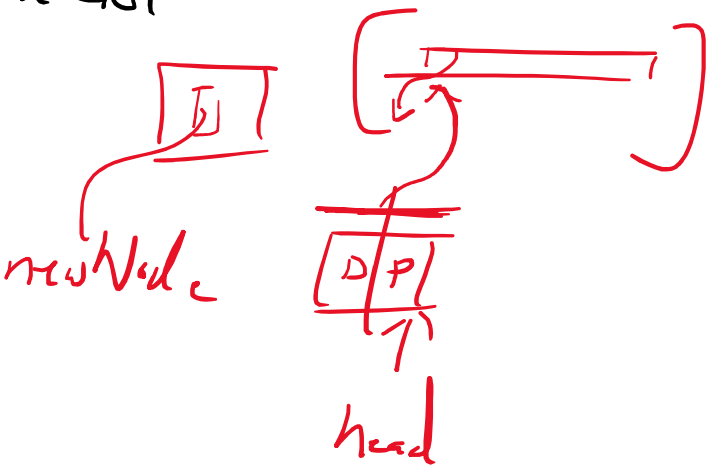
Object oriented Space



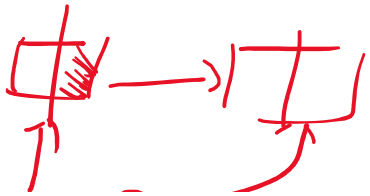
Null ≈ ϕ

Node => Linked List

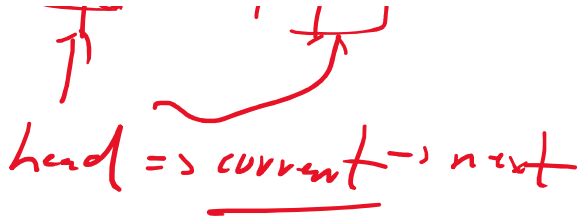
```
def insertion (self, data):
        newNode = Node (data)
        newNode.next = self.head
        self.head = newNode
```

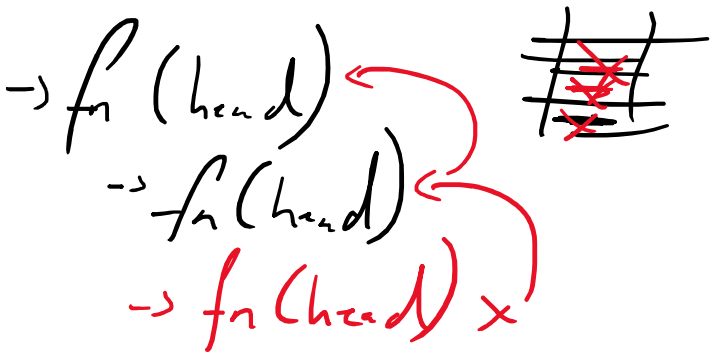


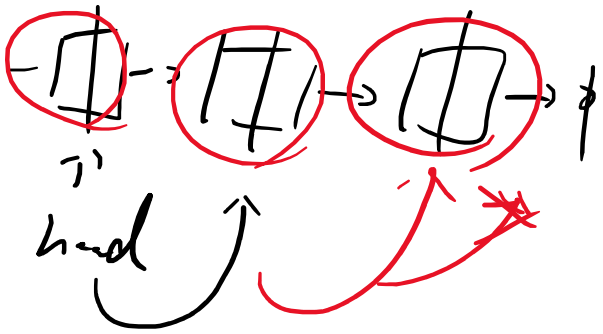

```
    def insertion (self, data):
            newNode = Node (data)
            if not self.head:
                self.head = newNode
            else:
                current = self.head
                while current.next:
                    current = current.next
```



```
current = self.head
while current.next:
    current = current.next
current.next = newNode
```



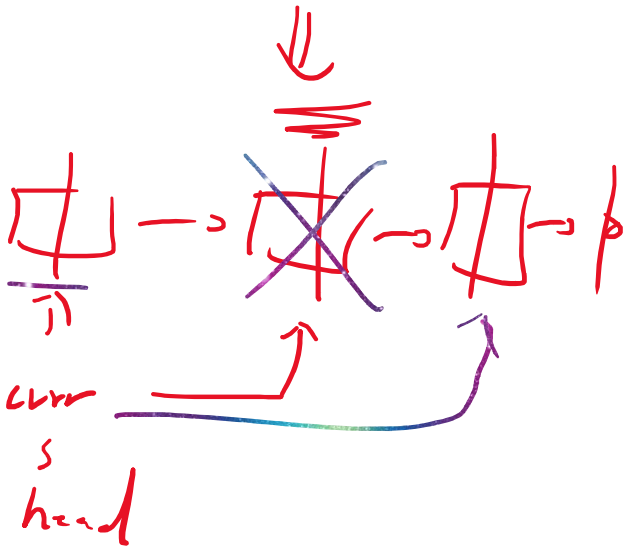




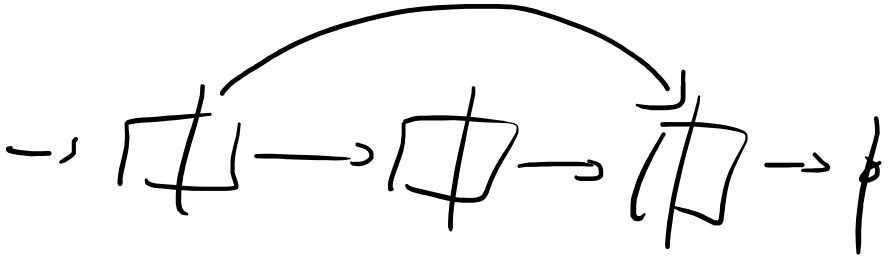
```
def deletion (self, element):
    if not self.head:
        return

    if self.head == element:
        self.head = self.head.next
        return

    current = self.head
    while current.next:
        if current.next.data == element:
            current.next = current.next.next
            return
        current = current.next
```



⇒ Detect a cycle in LL :



Hashmap based

Floyd Cycle
(Hare & Tortoise Alg)

=> Hashmap :

-> Traverse the list individually & keep putting node addresses in a Hashtable.

-> At any point if NULL is reached return False.

-> If the next of current points to any prev, nodes in Hashtable return true.

-> Time Complexity O(N)

-> Auxillary Space O(N)

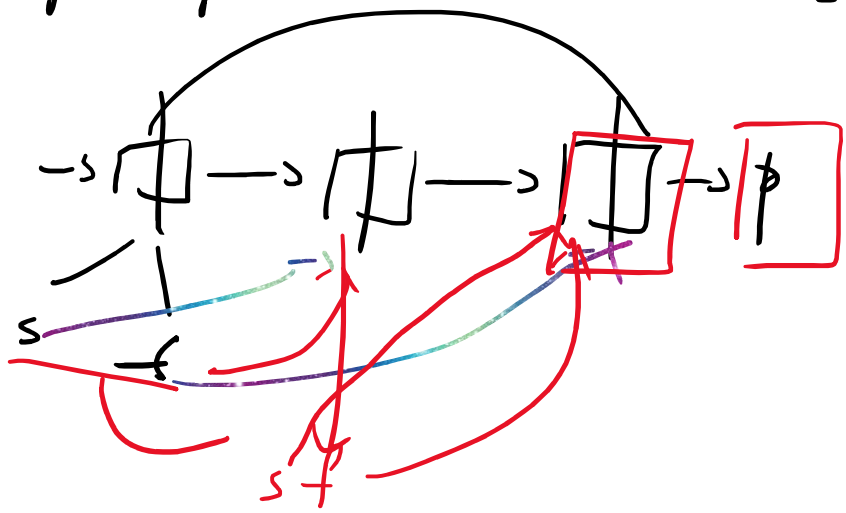
=> Floyd Cycle ≈ Hare & Tortoise Algorithm.

-> Traverse LL using 2 pointers.

-> Move one pointer slow & another by 2.

-> Move one pointer slow & another by 2.

-> If 2 pointers meet there is a cycle.



=> Reverse Linked List :

↳ Iterate through 3 pointers : prev, current & next

↳ Iterate through LL :

↳ Before changing next of curr, store it

-> next = curr -> next

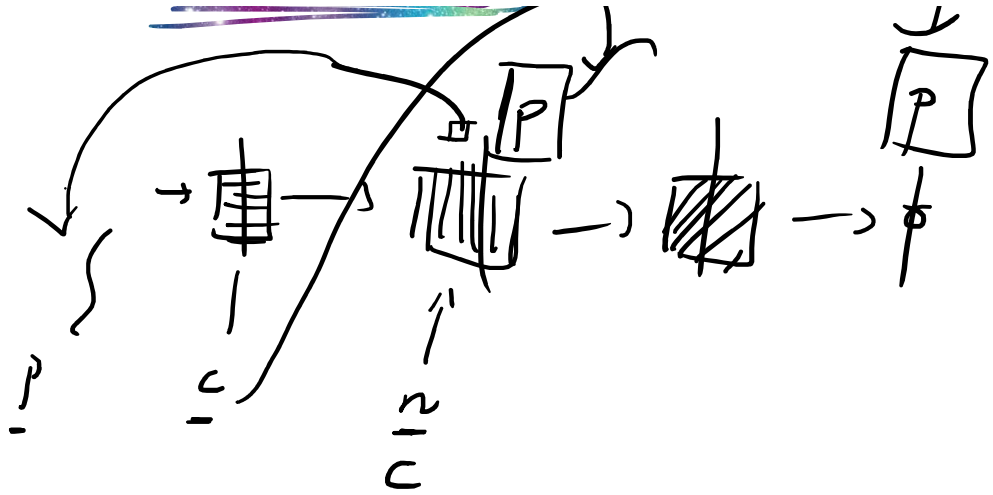
↳ Now update the next pointer of curr to prev

-> curr -> next = prev

↳ Update prev as curr & curr as next

-> prev = curr

-> curr = next



Time Complexity $O(N)$

Auxiliary Space $O(1)$