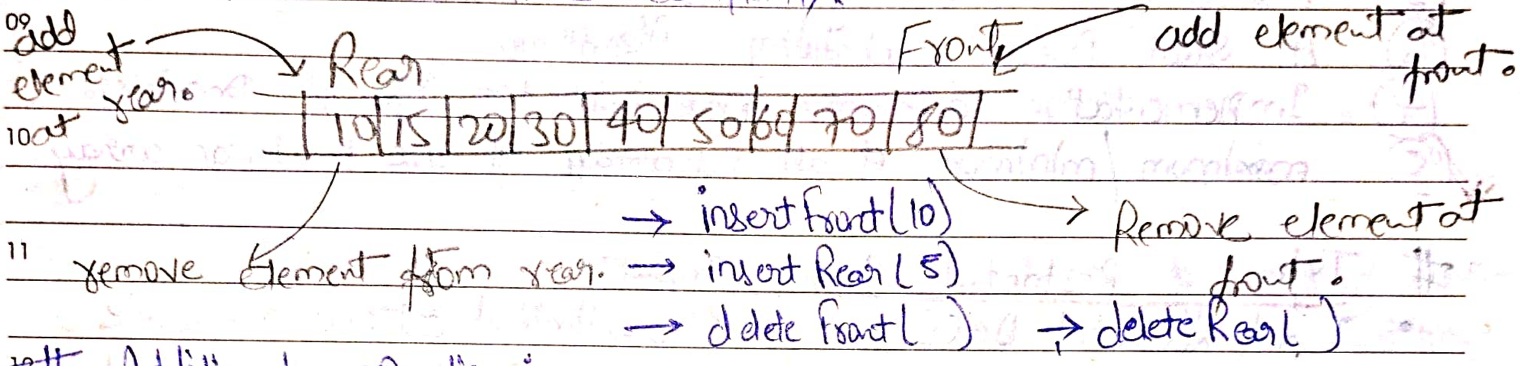
Deque

→ Deque(Double ended queue) in python is implemented using the module 'collections. Deque is preferred over a list in the cases where we need quicker append and pop operation as compared to a list that provide O(n) time complexity.



→ insertFront(10)

→ insertRear(5)

→ deleteFront()   → deleteRear()

Additional operation:-

getFront() — gets the front item from queue.

getRear() — gets the last item from queue.

is Full() —

is Empty() — check whether deque is empty (or) not.

size() — gets number of element in deque.

insertFront(10)

insertFront(20)

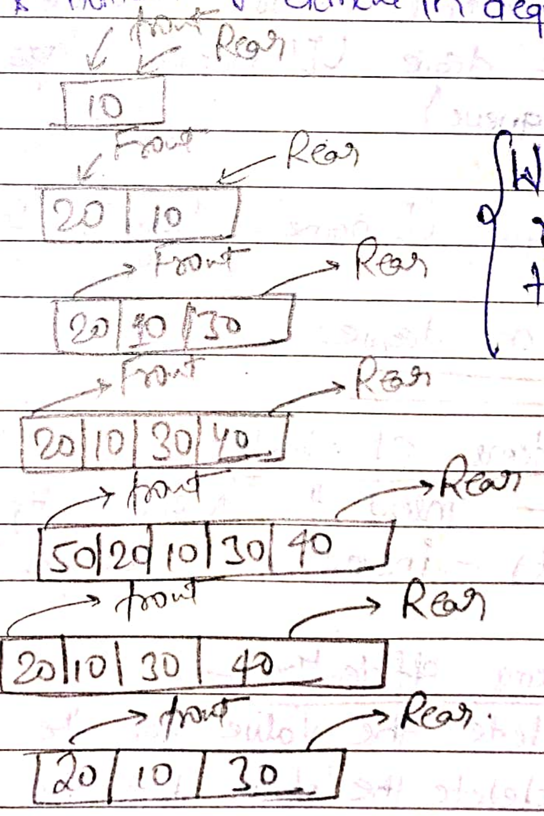
insertRear(30)

insertRear(40)

insertFront(50)

deleteFront()

deleteRear()



We can add (or) remove from both the End.

It allows insertion and deletion at both ends.

It make use of double linked list

Applications

08 (1) A Deque can be used as both stack and queue.
(2) Maintaining History of actions.
09 (3) A Steal Process scheduling Algorithms.
(4) Implementation a Priority queue with two types of Priorities.
10 (5) maximum/minimum of all subarray of size K in an array.

Types of Restricted Deque Input

- Input Restricted Deque : input is limited at one end while deletion is Permitted at both ends.

12

01
- Output Restricted Deque : Output is limited at one end but insertion is Permitted at both ends.

02

eg
03
```
from collections import deque
queue = deque (['name', 'age', 'DOB'])
print (queue)
```
04
```
o/p — deque (['name', 'age', 'DOB'])
```
05

operation on deque

06

⇒ Appending items Efficiently —

07
- append () — insert the value in its right end of deque.
- append left() — insert the value in its left end of deque.

} — O(1)

⇒ popping items efficiently —

- pop() — delete the value from the right end of deque.
- popleft() — delete the value from the left end of deque.

} — O(1)

⇒ Accessing item in a deque —

- index (ele, beg, end) — return the index of Value mentioned
- insert (i, a) — insert the Value at index i.
- remove () — remove first occurrence of Value.
- count () — count the No. of occurrence of Value.

O(n)

⇒ Different operation on deque —

O(len(y))
- extend (iterable) — add multiple Value at the right end of deque.
- extendleft (iterables) — add multiple Value at the left end of deque.
- reverse () — reverse the order of deque element.

O(abs(r))
- rotate () — The rotation will be done at specified in argument. If the number specified is negative rotation occur to the left. Else rotation is to right.

Examples

```
from Collections import deque
d= deque ()            # []
d. append (10)         # [10]
d. append (20)         # [10, 20]
d. append (30)         # [10, 20, 30]
d. appendleft (40)     # [40, 10, 20, 30]
Print (d)
Print (d. pop())
Print (d. popleft())        # [40, 10, 20]
Print (d)                   # [10, 20]
```

o/p —
```
deque([40, 10, 20, 30])
30
40
deque ([10, 20]).
```

JULY

14 '24

29th Week
196-170

SUNDAY

rotate (r) — O [abs(r)] , extend
extend left(L) } O (len(L))

S	M	T	W	T	F	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

JUL 2024

→ Sliding is not allowed in deque

eg

```
   d= deque ([10,20 ,30,40])       # [10,20,30,40]
08 d.insert (2,10)   # [10,20, 10,30,40]
   print (d.count (10))
09 d.remove (10)              # [20,10, 30,40]
   print (d)
10 d.extend ([50,60])         # [20,10, 30,40,50,60]
   print (d)
11 d.extend left ([15,25]) # [25,15,20,10,30,40,50,60]
12 o/p—  2
        deque ([20,10,30,40])
01      deque ([20,10,30,40,50,60])
        deque ([25,15,20,10,30,40,50,60])
02
```

eg

```
03 from collections import deque
    d= deque ([10,20,30,40,50])
04  d.rotate (2)     # [40,50,10,20,30])
     print (d)
05   d.rotate(-2)    # [10,20,30,40,50]
     print (d)
06    d.reverse()      # [50,40,30,20,10]
      print (d)
07
o/p—  deque  ([40,50,10,20,30])
      deque ([10,20,30,40,50])
      deque ([50,40,30,20,10])
```

append(x)
appendleft(x)
pop()
popleft() } — O(1)

d[i]
count(x)
insert(i,x) } — O(n)

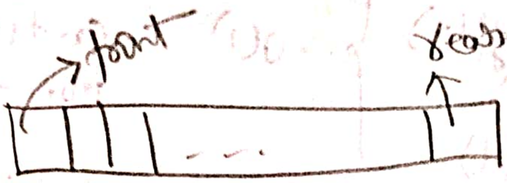
Linked List implementation of Deque.

```
Class Node :
    def __init__(self, K):
        Self.key = K
        Self.Next = NONE
        Self.Prev = NONE
```

ये line MyDeque class में ही होगा।

```
def deletefront (self) :
    if Self.front == None :
        return None
    else :
        res = self.front.key
        self.front = self.front.next
        if self.front == NONE :
            self.rear = NONE
        else :
            Self.front.Prev = NONE
        Self.s2 = self.s2 - 1
        return res

def get_front (self) :
    if self.front :
        return self.front.key

def getRear (self):
    if self.rear :
        return self.rear.key
```

```
Class MyDeque :
    def __init__ (self, c):
        Self.front = NONE
        Self.rear = None
        Self.s2 = 0

    def size (self):
        return self.s2

    def isempty (self):
        return Self.s2 == 0

    def insertRear (self, x):
        temp = Node(x)
        if Self.rear == NONE:
            Self.front = temp
        else :
            Self.rear.next = temp
            temp.Prev = Self.rear
        Self.rear = temp
        Self.s2 = Self.s2 + 1
```



Operation on deque Over List :—

- insertFront() : Adds an item at the front of deque.
- insertRear() : Adds an item at the rear of Deque.
- deleteFront() : Deletes an item from front of deque.
- deleteRear() : Deletes an item from rear of deque.
- getFront() : gets the front item from queue.
- getRear() : gets the last item from queue.
- size () : gets the size of queue.
- isEmpty () : Check whether it is empty (or) not.

} O(1)

```
Class MyDeque:
    def __init__(self,c):
        self.l = [None]*c
        self.cap = c
        self.size = 0
        self.front = 0

    def deleteFront(self):
        if self.size == 0:
            return None
        else:
            res = self.l[self.front]
            self.front = (self.front +1)% self.cap.
            self.size = self.size -1
            return res.
    def insertRear(self,x):
        if self.size == self.cap: return
        new_rear = (self.front + self.size)% self.cap
        self.l[new_rear] = x
        self.size = self.size + 1
```

```
    def insertFront(self,x):
        if self.size == self.cap: return
        else: self.front = (self.front -1)%self.ca
              self.l[self.front] = x
              self.size = self.size +1

    def deleteRear(self):
        s2 = self.size
        if s2==0: return NONe
        else: rear = (self.front +s2-1)%self.ca
              self.size = s2-1
              return self.l [rear]
    def frontEle(self):
        return self.l [self.front]

    def rearEle(self):
        rear = (self.front +self.size-1)%self.cap
        return self.l[rear].
```