

S	M	T	W	T	F	S
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

## Queue

JULY

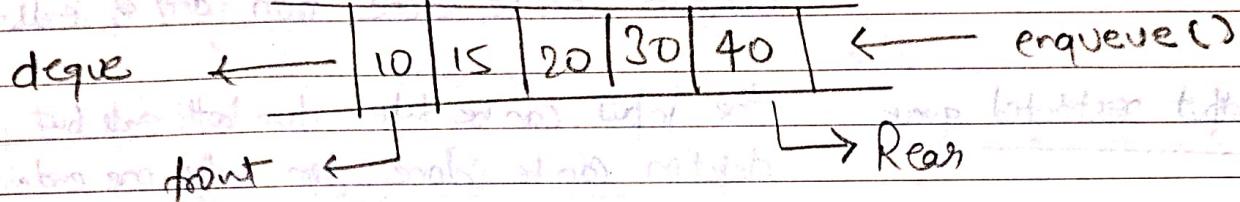
'24

29th Week  
199-167

17

WEDNESDAY

→ Queue is a linear data structure that is open at both ends and the operations are performed in first-in-first-out (FIFO) order.



### Operations:

enqueue (x) — to insert the element

dequeue () — to delete the element

getFront () — to get front element

getRear () — to get rear element

Size () — to know the size

isEmpty () — to know whether it is empty or not (True or False)

Queue q

q.enqueue (10)

10

q.enqueue (20)

10 20

q.enqueue (30)

10 20 30

q.enqueue (40)

10 20 30 40

0/b

print (q.dequeue ())

20 30 40

→ 10

print (q.dequeue ())

30 40

→ 20

print (q.getFront ())

→ 30

print (q.getRear ())

→ 40

JULY

18

'24

29th Week  
200-166

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

THURSDAY

- # Types of Queue :-
- ① Input restricted queue — the input can be taken from only one end but deletion can be done from any of ends.
  - ② Output restricted queue — the input can be taken from both ends but deletion can be done from only one end.
  - ③ Circular queue — In these where the last position is connected back to the first position.
  - ④ Double ended queue — insertion & deletion operations, both can be performed from both ends.
  - ⑤ Priority queue — Special queue where the elements are accessed based on the priority assigned to them.

#### # Application of Queue :-

- ① Single Resources and multiple consumers [CPU scheduling & Disk scheduling]
- ② Synchronization b/w slow and fast devices.
- ③ In operating system (semaphores, FCFS scheduling, spooling, buffers for devices like keyboard).
- ④ In computer network (Routers / switches and mail queues)
- ⑤ Variations : Deque, priority queue and Doubly ended Priority queue.

Implementation in python.

- a) using list
- b) using collections.deque
- c) using queue.Queue → Multithreaded Environment
- d) our own implementation.

S	M	T	W	T	F	S
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

JULY

'24

29th Week  
201-165

19

FRIDAY

## # Using List

08 q = []

q.append(10) # [10]

09 q.append(20) # [10, 20]

q.append(30) # [10, 20, 30]

10 print(q) [10, 20, 30]

print(q.pop(0)) # [20, 30]

11 q.append(40) # [20, 30, 40]

print(q.pop(0)) # [30, 40]

12 print(len(q))

print(q[0])

01 print(q[-1])

if q == []:

02 # Using Deque

03 os informed collections implement deque

04 q = deque() # deque([])

05 q.append(10) # deque([10])

q.append(20) # deque([10, 20])

06 q.append(30) # deque([10, 20, 30])

print(q)

print(q.popleft()) # deque([20, 30])

q.append(40) # deque([20, 30, 40])

print(q.popleft()) # deque([20, 30, 40])

print(len(q))

print(q[0])

print(q[-1])

TC = O(1)

O/P - [10, 20, 30]

30

40

20

30

40

20

30

40

20

30

40

20

30

40

20

30

40

20

30

40

JULY

20

'24

29th Week  
202-164

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				

JULY 2024

SATURDAY

TC = O(1)

using linked list

DB

Class Node:

09 def \_\_init\_\_(self, k):  
 self.key = k

10 self.next = None

def enqueue(self, n):

temp = Node(n)

if self.rear == None:

self.front = temp

else:

self.rear.next = temp

11 class MyQueue:

def \_\_init\_\_(self):

12 self.front = None

self.rear = None

01 self.s2 = 0

def dequeue(self):

if self.front == None:

return None

else:

res = self.front.key

self.front = self.front.next

if self.front == None:

self.rear = None

self.s2 = self.s2 - 1

return res

02 def size(self):

return self.s2

03

def isEmpty(self):

04 return (self.s2 == 0)

05 def getFront(self):

return self.front.key

06

def getRear(self):

07 return self.rear.key

using List

S	M	T	W	T	F	S
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

AUG 2024

'24  
30th Week  
203-163

JULY

21

SUNDAY

## Implement Stack using Queue:

(b) We will take 2 queues:

q<sub>1</sub>: To keep the actual items.

push(10): q<sub>1</sub> = [10]

q<sub>2</sub>: To be used as an auxiliary queue.

q<sub>2</sub> = []

push(n): (a) Move everything from q<sub>1</sub> to q<sub>2</sub>. push(20):

(b) Enqueue n to q<sub>1</sub>

(a) q<sub>1</sub> = [], q<sub>2</sub> = [10]

(c) move everything back from q<sub>2</sub> to q<sub>1</sub>

(b) q<sub>1</sub> = [20], q<sub>2</sub> = [10]

(c) q<sub>1</sub> = [20, 10], q<sub>2</sub> = []

pop(n): Remove front of q<sub>1</sub>

push(30):

top(): return front of q<sub>1</sub>

(a) q<sub>1</sub> = [], q<sub>2</sub> = [20, 10]

size(): return size of q<sub>2</sub>

(b) q<sub>1</sub> = [30], q<sub>2</sub> = [20, 10]

(c) q<sub>1</sub> = [30, 20, 10], q<sub>2</sub> = []

(b) In optimize solution of Queue implementation is

Instead of this we use

push(n): (a) Enqueue n to q<sub>2</sub>.

(b) move everything from q<sub>1</sub> to q<sub>2</sub>.

(c) Swap the variable q<sub>1</sub> and q<sub>2</sub>.

Cook

class Stack:

def \_\_init\_\_(self):

self.q1 = deque()

self.q2 = deque()

def pop(self):

if self.q1:

self.q1.pop()

def push(self, n):

self.q2.append(n)

while self.q1:

self.q2.append(self.q1.pop())

self.q1, self.q2 = self.q2, self.q1

~~Tc = O(n)~~

def top(self):

if self.q1:

return self.q1[0]

else:

return None

def size(self):

O(1)

return len(self.q1)

JULY

22

'24

30th Week  
204-162

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				

MONDAY

## # Reverse a Queue

08

i/p:  $q = [20, 10, 15, 30]$ 

Iterative Solution

o/p:  $q = [30, 15, 10, 20]$ 

① Create an empty stack.

M-1 from collections import deque

10 def reverseQueue(q):

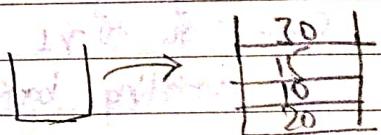
② move all item of q to stack.

st = []

③ put all items back to the q.

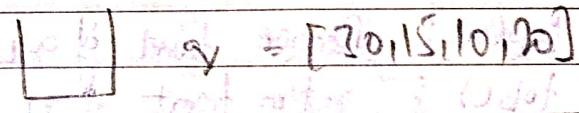
11 while q:

st.append(q.popleft())



12 while st:

13 q.append(st.pop())



M-2 def revD(q):

Naive Sol?

02 if len(q) == 0: return q

① Traverse through all natural No. while we have not generated the n numbers

T=O(n)

SC=O(1)

revD(q)

q.append(n)

② for every traversed no. check if it has digit as 5 and 6 only. If yes, print the no. and increasing the count.

# Generate number with Given digits.

05

→ Given a value K, generate all well-ordered numbers of length k.

→ Well ordered means that digit should be in increasing order.

eg. i/p: [5, 6], n=10

o/p: 5 6 55 56 6566

555 556 565 566

(1) well order must

(2) well order must

(3) well order must

(4) well order must

(5) well order must

(6) well order must

(7) well order must

(8) well order must

(9) well order must

(10) well order must

(11) well order must

(12) well order must

(13) well order must

(14) well order must

(15) well order must

(16) well order must

(17) well order must

(18) well order must

(19) well order must

(20) well order must

(21) well order must

(22) well order must

(23) well order must

(24) well order must

(25) well order must

(26) well order must

(27) well order must

(28) well order must

(29) well order must

(30) well order must

(31) well order must

(32) well order must

(33) well order must

(34) well order must

(35) well order must

(36) well order must

(37) well order must

(38) well order must

(39) well order must

(40) well order must

(41) well order must

(42) well order must

(43) well order must

(44) well order must

(45) well order must

(46) well order must

(47) well order must

(48) well order must

(49) well order must

(50) well order must

(51) well order must

(52) well order must

(53) well order must

(54) well order must

(55) well order must

(56) well order must

(57) well order must

(58) well order must

(59) well order must

(60) well order must

(61) well order must

(62) well order must

(63) well order must

(64) well order must

(65) well order must

(66) well order must

(67) well order must

(68) well order must

(69) well order must

(70) well order must

(71) well order must

(72) well order must

(73) well order must

(74) well order must

(75) well order must

(76) well order must

(77) well order must

(78) well order must

(79) well order must

(80) well order must

(81) well order must

(82) well order must

(83) well order must

(84) well order must

(85) well order must

(86) well order must

(87) well order must

(88) well order must

(89) well order must

(90) well order must

(91) well order must

(92) well order must

(93) well order must

(94) well order must

(95) well order must

(96) well order must

(97) well order must

(98) well order must

(99) well order must

(100) well order must

(101) well order must

(102) well order must

(103) well order must

(104) well order must

(105) well order must

(106) well order must

(107) well order must

(108) well order must

(109) well order must

(110) well order must

(111) well order must

(112) well order must

(113) well order must

(114) well order must

(115) well order must

(116) well order must

(117) well order must

(118) well order must

(119) well order must

(120) well order must

(121) well order must

(122) well order must

(123) well order must

(124) well order must

(125) well order must

(126) well order must

(127) well order must

(128) well order must

(129) well order must

(130) well order must

(131) well order must

(132) well order must

(133) well order must

(134) well order must

(135) well order must

(136) well order must

(137) well order must

(138) well order must

(139) well order must

(140) well order must

(141) well order must

(142) well order must

(143) well order must

(144) well order must

(145) well order must

(146) well order must

(147) well order must

(148) well order must

(149) well order must

(150) well order must

(151) well order must

(152) well order must

(153) well order must

(154) well order must

(155) well order must

(156) well order must

(157) well order must

(158) well order must

(159) well order must

(160) well order must

(161) well order must

(162) well order must

(163) well order must

(164) well order must

(165) well order must

(166) well order must

(167) well order must

(168) well order must

(169) well order must

(170) well order must

(171) well order must

(172) well order must

(173) well order must

(174) well order must

(175) well order must

(176) well order must

(177) well order must

(178) well order must

(179) well order must

(180) well order must

(181) well order must

(182) well order must

(183) well order must

(184) well order must

(185) well order must

(186) well order must

(187) well order must

(188) well order must

(189) well order must

(190) well order must

(191) well order must

(192) well order must

(193) well order must

(194) well order must

(195) well order must

(196) well order must

(197) well order must

(198) well order must

(199) well order must

(200) well order must

(201) well order must

(202) well order must

(203) well order must

(204) well order must

(205) well order must

(206) well order must

(207) well order must

(208) well order must

(209) well order must

(210) well order must

(211) well order must

(212) well order must

(213) well order must

(214) well order must

(215) well order must

(216) well order must

(217) well order must

(218) well order must

(219) well order must

(220) well order must

(221) well order must

(222) well order must

(223) well order must

(224) well order must

(225) well order must

(226) well order must

(227) well order must

(228) well order must

(229) well order must

(230) well order must

(231) well order must

(232) well order must

(233) well order must

(234) well order must

(235) well order must

(236) well order must

(237) well order must

(238) well order must

(239) well order must

(240) well order must

(241) well order must

(242) well order must

(243) well order must

(244) well order must

(245) well order must

(246) well order must

(247) well order must

(248) well order must

(249) well order must

(250) well order must

(251) well order must

(252) well order must

(253) well order must

(254) well order must

(255) well order must

(256) well order must

(2

S	M	T	W	T	F	S
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

JULY

'24

30th Week  
205-161

23

TUESDAY,

Steps

- (1) Create an empty queue, q
- (2) Add 5 and 6 to q
- (3) Run a loop n times.
- (4) → Take out an item from q.
- (5) → Print the item.
- (6) → Append 5 to the item and add the new number to q.
- (7) → Append 6 to the item and add the new number to q.
- (8) → ...
- (9) → ...
- (10) → ...
- (11) → ...

```

def printFirst(n):
    q = deque()
    q.append("5")
    q.append("6")
    for i in range(n):
        curr = q.popleft()
        print(curr, end=" ")
        q.append(curr + "5")
        q.append(curr + "6")
    
```

# Design a Data Structure with max/min operation.

→ Design a data structure that support following operation in O(1) Time complexity.

- (1) insertMin(n)
- (2) insertMax(n)
- (3) getMin()
- (4) getMax()
- (5) extractMin()
- (6) extractMax()

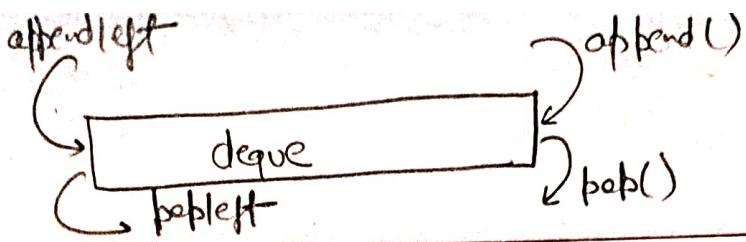
s/p: insertMin(5)      insertMax(10)      insertMin(3)      insertMax(15)      insertMin(2)      getMin()      getMax()      insertMin(1)      getMin()

Each one sees what he carries in his heart

insertMax(20)  
getMax()

JULY  
24  
30th  
20

30th Week  
206-160



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	

## WEDNESDAY

from collections import deque

## 08 Class Myps:

def \_\_init\_\_(self):

09 self. day = que ( )

def insertMin [self, n]:  
    self.dgr.appendleft(n)

def insertman (self, n):  
    self.data.append (n)

def extractMin(self):  
 return self.dq.popLeft()

def extractMax (self):

01                  return self.\_obj.pop()

def \_\_str\_\_(self):

def getMin():

02      return self.dq[0]

def getMax():

03 return self.dg[-1]

```

d = MyDS()
deque[ ]
d.insertMin(10)
deque[10]
d.insertMin(5)
deque[5, 10]
d.insertMax(20)
deque[5, 10, 20]
d.insertMin(3)
deque[3, 5, 10, 20]
print(d.extractMin())
# 3
print(d.extractMax())
# 20
print(d.getMin())
# 5
print(d.getMax())
# 10

```

#<sup>04</sup> Maximum in all Subarrays of size K.

05) p: arr[] = {10, 8, 5, 12, 15, 7, 6}, k=3, | {10, 8, 5, 12, 15, 7, 6}

General formula =  $(n - k + 1)$

def printmaxK (arr, K):

for  $i$  in range( $n-k+1$ ):

yes =  $\text{ans}$  [i]

for j in range(l+1, i+k):

$$yes = \max(yes, err_{[T]})$$

Print (yes, f) end = " "

Error tolerates, truth condemns

AUG 2024

S	M	T	W	T	F	S
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

JULY

'24

30th Week  
207-159

25

THURSDAY

Efficient : O(n) K=3, {10, 8, 5, 12, 15, 7, 6}

08	$i = 0, dq$	<table border="1"><tr><td>10</td></tr></table>	10		
10					
	$i = 1, dq$	<table border="1"><tr><td>10</td><td>8</td></tr></table>	10	8	
10	8				
09	$i = 2, dq$	<table border="1"><tr><td>10</td><td>8</td><td>15</td></tr></table>	10	8	15
10	8	15			
	$i = 3, dq$	<table border="1"><tr><td></td><td></td><td>-10</td></tr></table>			-10
		-10			

$i = 4$	$dq$	<table border="1"><tr><td>15</td><td></td><td></td></tr></table>	15			$-15$
15						
$i = 5$	$dq$	<table border="1"><tr><td>15</td><td>7</td><td></td></tr></table>	15	7		$-15$
15	7					
$i = 6$	$dq$	<table border="1"><tr><td>15</td><td>7</td><td>6</td></tr></table>	15	7	6	$-15$
15	7	6				

10

→ Whenever we see an item we remove all the smaller than it from deque.

11  
Def point KMax (arr, k):

12 dq = deque()

for i in range (k):

01 while dq and arr[i] &gt;= arr[dq[-1]]: K=3

dq.pop()

i=0 : 

0	
---	--

02 dq.append(i)

i=1 : 

1	
---	--

print (arr[dq[0]], end="")

i=2 : 

1	2	1
---	---	---

 printing

03 for i in range (K):

while dq and dq[0] &lt;= i-K:

Inside the second loop:

04 dq.popleft()

i=3 : 

1	3
---	---

, print(4)

while dq and arr[i] &gt;= arr[dq[-1]]:

i=4 : 

4	
---	--

 print(6)

dq.pop()

dq.append(i)

06 print (arr[dq[0]], end="")

07 # First CircularTour.

→ Given information about N Petrol pumps (says arr[]) that are present in a circular path. The information consists of the distance of the next petrol pump from the current one (in arr[i][1]) and the amount of petrol stored in that petrol pump (in arr[i][0]). Consider a truck with infinite capacity that consumes 1 unit of petrol to travel 1 unit distance. The task is to find the index of the first starting point such that the truck can visit all the petrol pumps & come back to that starting point.

Even a broken pot might contain sugar

JULY

26

'24

30th Week  
208-158

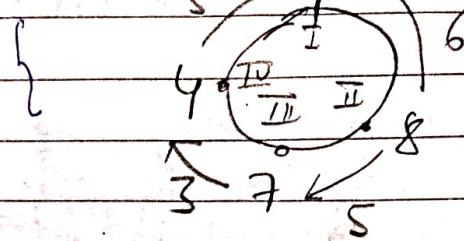
S	M	T	W	T	F
1	2	3	4	5	
7	8	9	10	11	12
14	15	16	17	18	19
21	22	23	24	25	26
28	29	30	31		

FRIDAY

$$\text{def PointTour}(\text{pet}, \text{dirt}):$$

08       $\text{dist}[] = \{6, 5, 3, 5\}$

$$\{ \text{op: } 2 \}$$



09      def PointTour(pet, dirt):

10      n2 len(pet)

11      for i in range(n):

12      curr-pet += (pet[i] - dirt[i])

13      if curr-pet < 0:

14      break

15      end = (end + 1) % n

16      if end == start:

17      return start + 1

18      return -1

03

04

05

06

07

During the harvest time a rat keeps five wives

and each wife has five children. Each child has five sons and five daughters. All the sons and daughters have five wives each. How many rats are there in all?