CPE111HDS — Module1 — Linear Data Structures

Lecture3: Stacks/Queues/Deques

Dr. Prapong Prechaprapranwong

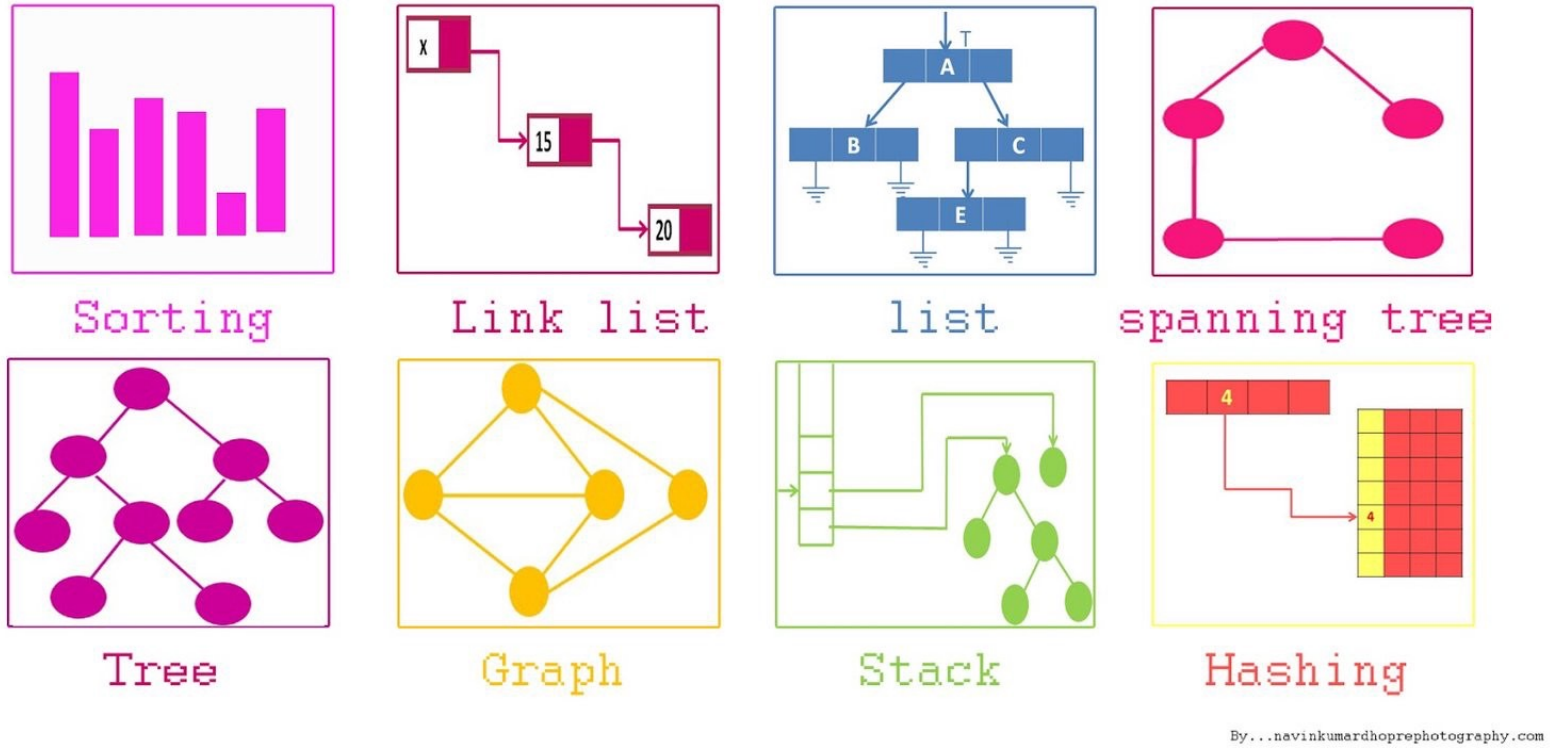
Computer Engineering

King Mongkut's University of Technology Thonburi

Abstract Data Type (ADT) =

A definition for a data type

- a set of values
- a set of Operations allowed on data type



Content

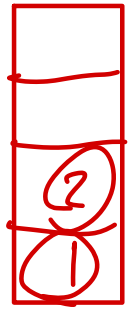
- Stack ADT
  - Implement Stack with Python’s List
- Queue ADT
  - Implement Queue with Python’s List
  - Implement Queue with Circular Array
  - Priority Queue ADT
- Deque ADT
  - Implement Deque with Python’s List

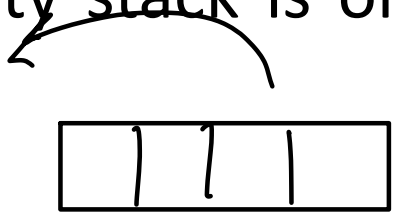
STACK ADT

→ ใช้ List

Stack Array

A *stack* is a data structure that stores a linear collection of items with access limited to a *last-in first-out (LIFO)* order. Adding and removing items is restricted to one end known as the *top* of the stack. An empty stack is one containing no items.





Stack():	Creates a new empty stack
isEmpty():	Returns a Boolean value indicating if the stack is empty
length ():	Returns the number of items in the stack
pop():	Removes and returns the top item of the stack, if the stack is not empty. Items cannot be popped from an empty stack. The next item on the stack becomes the new top item
peek():	Returns a reference to the item on top of a non-empty stack without removing it. Peeking, which cannot be done on an empty stack, does not modify the stack contents
push(item):	Adds the given item to the top of the stack

list.append()

Stack overflow is an unwanted condition that occur when a computer program tries to use more memory space than the call stack has available.

โค้ดเราเรียกใช้ memory ที่จองมามันเต็ม

not just a developer community's site

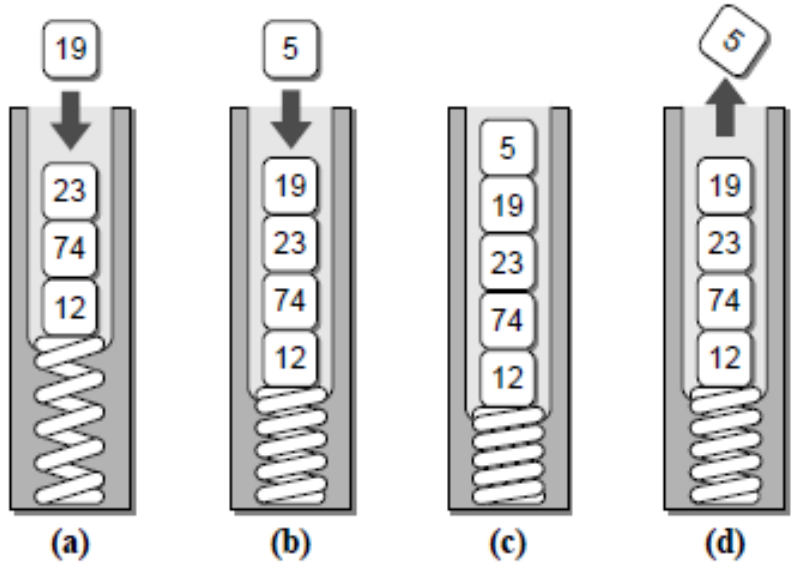
Question: What does Stack underflow mean?

↳ เรียกใช้แล้ว แต่ stack empty. (ไม่เจอข้อมูล)

Implementing a Stack with a Python List

Implementing a stack with a python list is the easiest way. The end of the list is used as the top. Other python list's methods are adapted to stack's method as in table.

S.push(19) S.push(5) e=S.peek() e=S.pop()



สิ่งที่มันทำจริงๆ

Stack Method	Implement with Python List
S.push(item)	L.append(item)
S.pop()	L.pop()
S.peek()	L[-1]
S.isEmpty()	len(L) == 0
len(S)	len(L)

Question: Do we still have Stack overflow situation?

↳ ไม่เกิด เพราะ List มันจองเรื่อยๆ

```python
# Implementation of the Stack ADT using a Python list.
class Stack :
   # Creates an empty stack.
  def __init__( self ):
    self._theItems = list()

   # Returns True if the stack is empty or False otherwise.
  def isEmpty( self ):
    return len( self ) == 0

   # Returns the number of items in the stack.
  def __len__ ( self ):
    return len( self._theItems )

   # Returns the top item on the stack without removing it.
  def peek( self ):
    assert not self.isEmpty(), "Cannot peek at an empty stack"
    return self._theItems[-1]
```

self._theItems.pop()

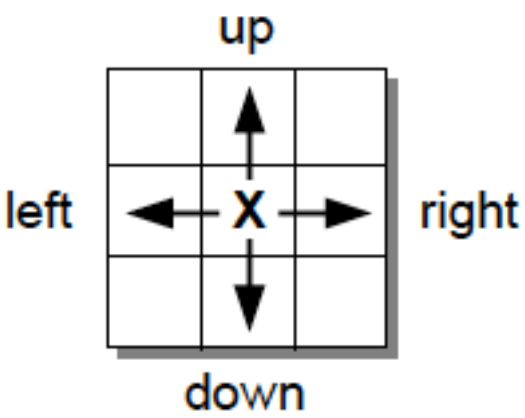
```
20     # Removes and returns the top item on the stack.
21    def pop( self ):
22      assert not self.isEmpty(), "Cannot pop from an empty stack"
23      return [redacted]
24
25     # Push an item onto the top of the stack.
26    def push( self, item ):
27      [redacted]
```

Can you guess ?

self._theItems.append(item)

Application: Maze solution with Stacks

Back Track Algorithms



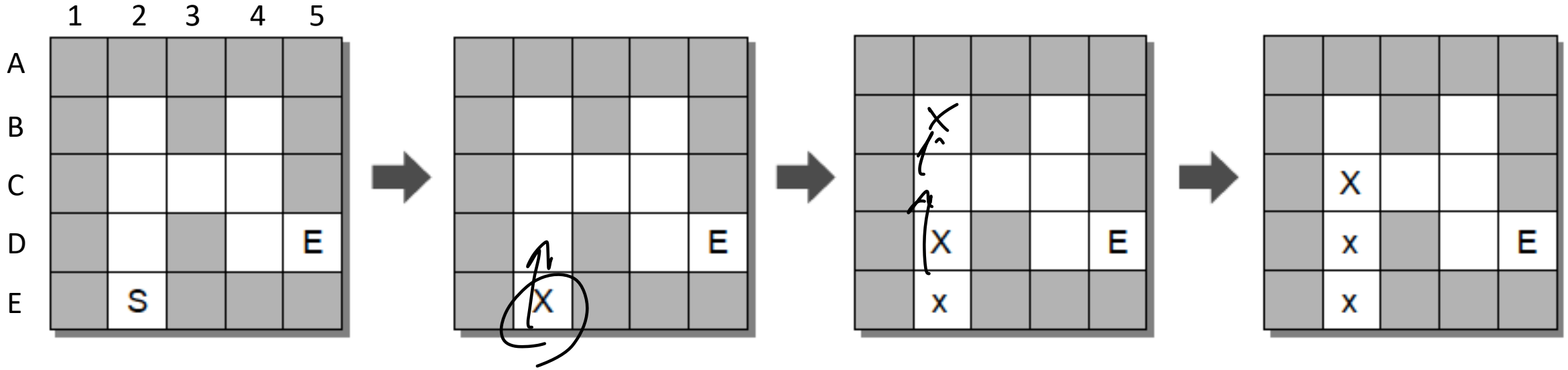
movement order:

1. up
2. down
3. left
4. right

Rules:

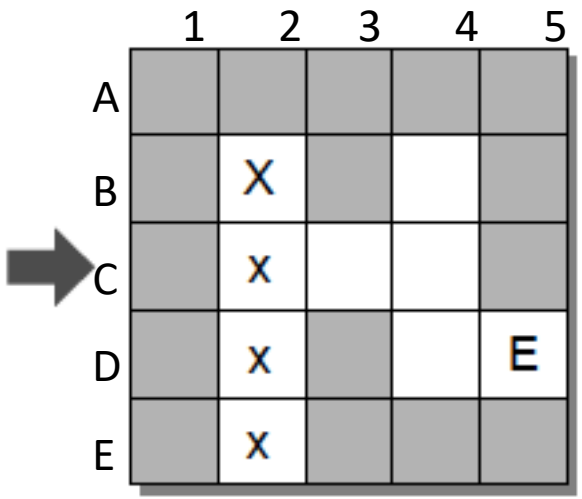
1. move to open cell
2. move in order "up, down, left, right"
3. mark "x" to moved cell Path.push(cell)
4. when hit the dead end go back and mark "o" Path.pop()

Path:[E2] Path:[E2,D2] Path:[E2,D2,C2]

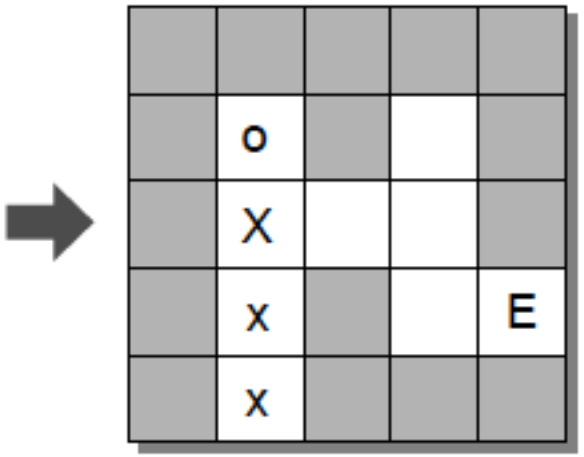


Path.pop() and mark dead end

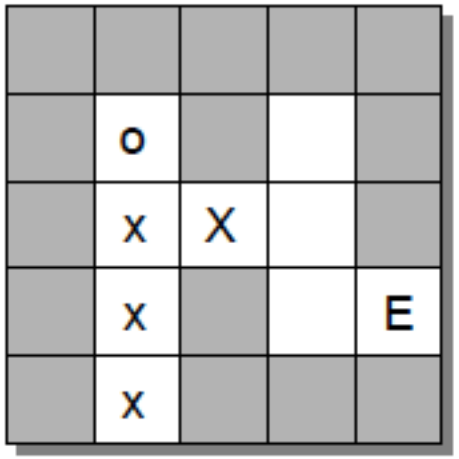
Path:[E2,D2,C2,B2]



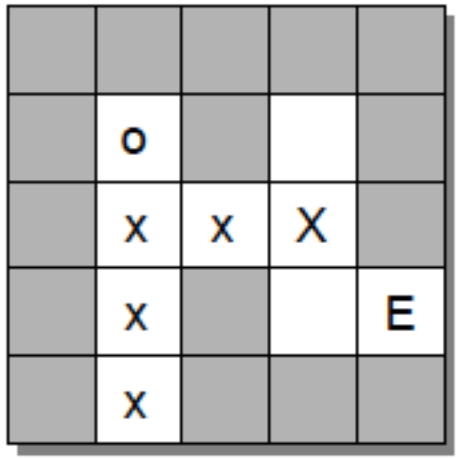
Path:[E2,D2,C2]



Path:[E2,D2,C2,C3]

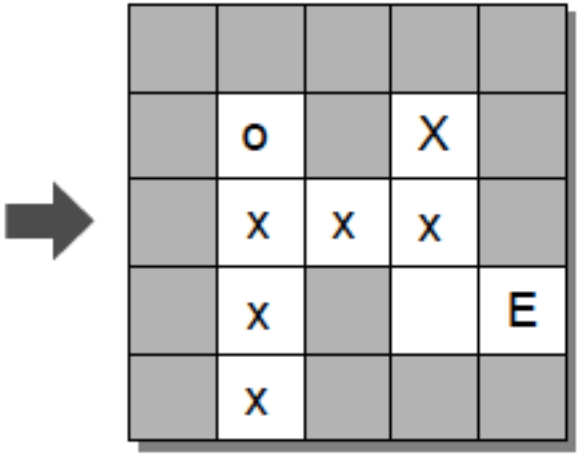


Path:[E2,D2,C2,C3,C4]

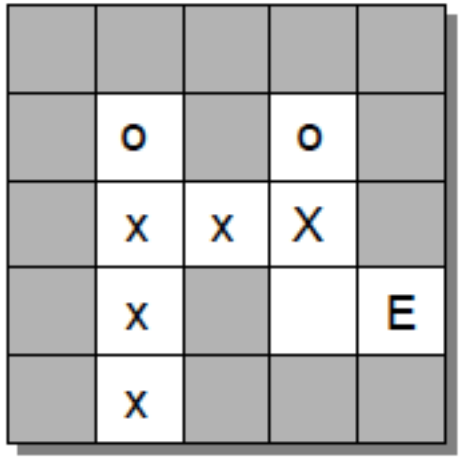


Path.pop() and mark dead end

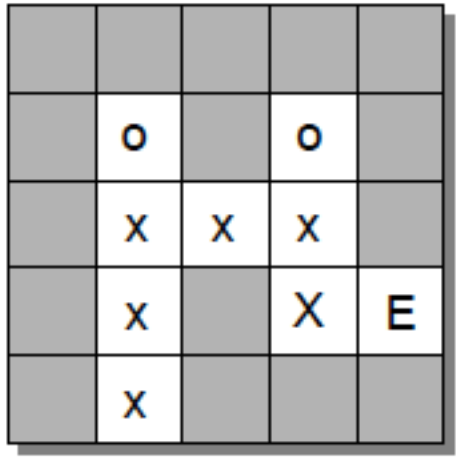
Path:[E2,D2,C2,C3,C4,B4]



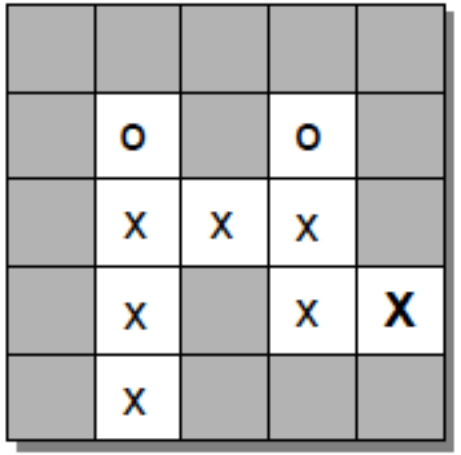
Path:[E2,D2,C2,C3,C4]



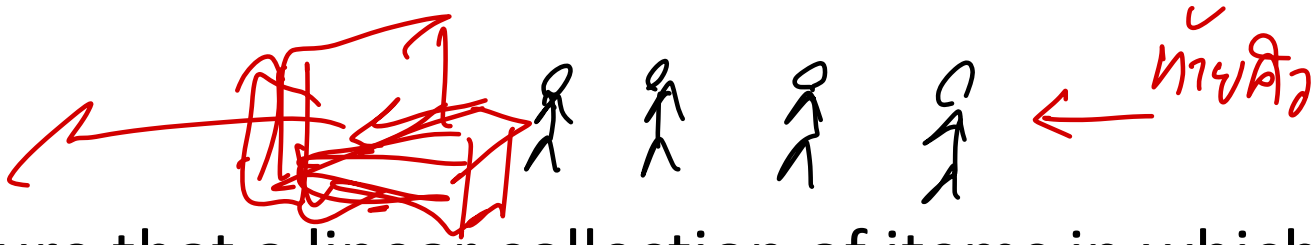
Path:[E2,D2,C2,C3,C4,D4]



Path:[E2,D2,C2,C3,C4,D4,D5]



QUEUE ADT

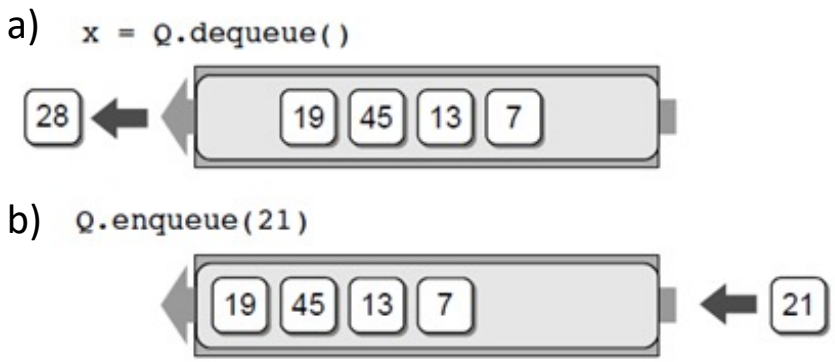


A queue is a data structure that a linear collection of items in which access is restricted to a first-in first-out (FIFO) basis. New items are inserted at the back and existing items are removed from the front. The items are maintained in the order in which they are added to the structure.

Queue():	Creates a new empty queue, which is a queue containing no items
isEmpty():	Returns a Boolean value indicating whether the queue is empty
length ():	Returns the number of items currently in the queue
enqueue(item):	Adds the given item to the back of the queue
dequeue():	Removes and returns the front item from the queue. An item cannot be dequeued from an empty queue

Implementing a Queue with a Python List

Implementing a Queue with a python list is also simple. By using append() method for enqueue, and pop(0) method for dequeue()



Queue Method	Implement with Python List
• Q.enqueue(item)	• L.append(item)
• Q.dequeue()	• L.pop(0)
• Q.isEmpty()	• len(L) == 0
• len(Q)	• len(L)

Implementation of the Queue ADT using a python list

```
# Implementation of the Queue ADT using a Python list.
class Queue :
   # Creates an empty queue.
  def __init__( self ):
    self._qList = list()

   # Returns True if the queue is empty.
  def isEmpty( self ):
    return len( self ) == 0

   # Returns the number of items in the queue.
  def __len__( self ):
    return len( self._qList )

   # Adds the given item to the queue.
  def enqueue( self, item ):
    [redacted]
            self._qList.append(item)
   # Removes and returns the first item in the queue.
  def dequeue( self ):
    assert not self.isEmpty(), "Cannot dequeue from an empty queue."
    return [redacted]
            self._qList.pop(0)
```

List

.append()

.pop()

.insert(0)

```
Q = Queue()
Q.enqueue( 28 )
Q.enqueue( 19 )
Q.enqueue( 45 )
Q.enqueue( 13 )
Q.enqueue( 7 )
```

ลองคิดดูว่ามันย้าย 1000 ตัว

ตอนใส่ไม่มีปัญหา

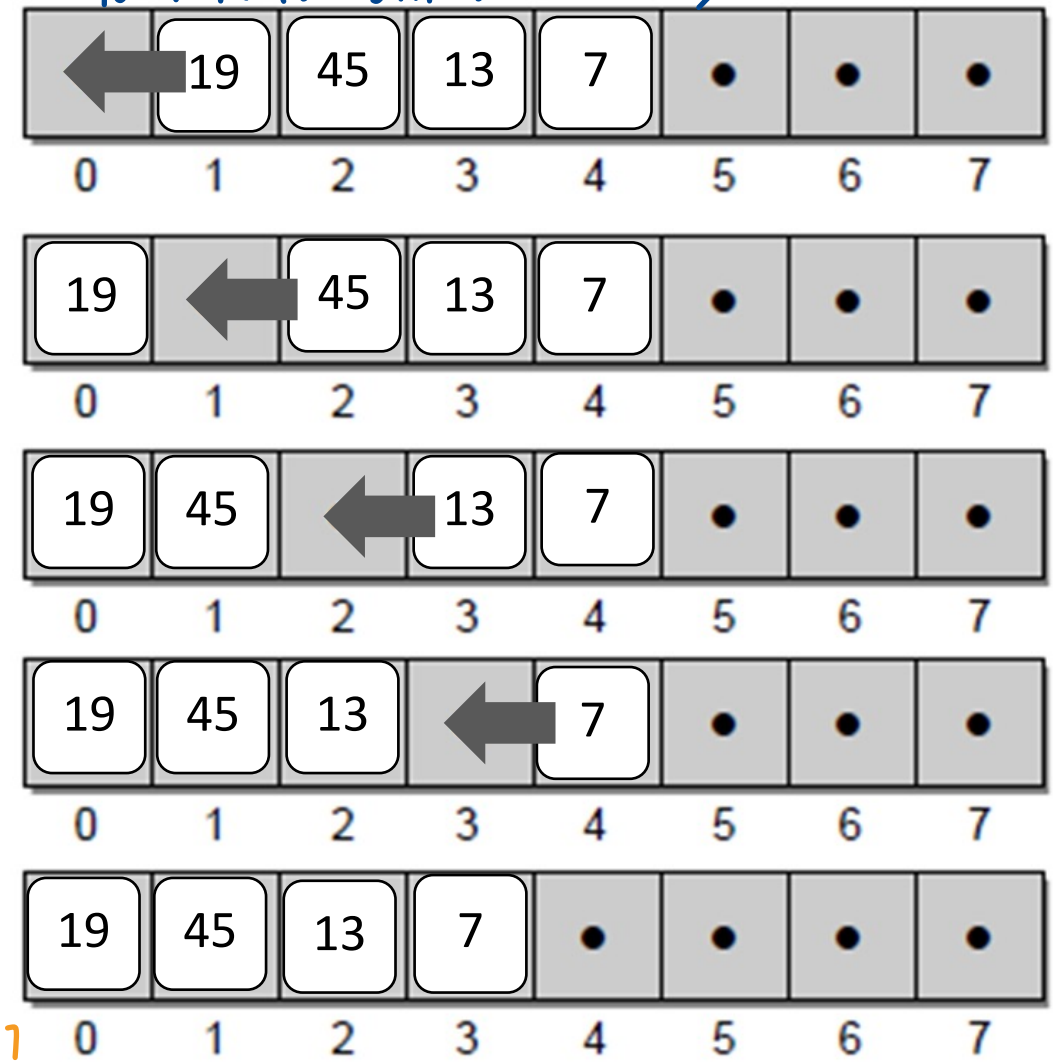
```
x = Q.dequeue()
```


```
Q.enqueue(21)
```


```
Q.enqueue(74)
```

แต่ตอน Pop (Dequeue) จะมีตน.ว่าง

เราก็จะต้อง Shift ไปเรื่อยๆ



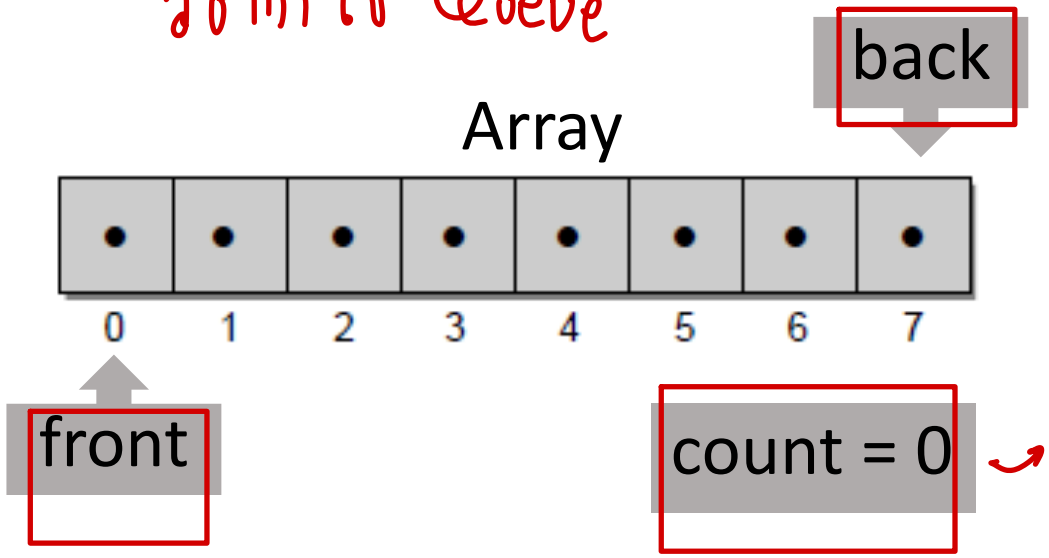
ปัญหาคือ ช่องเวลาที่ต้อง shift ค่า

linear time operation penalty for dequeue()

Linear Priority

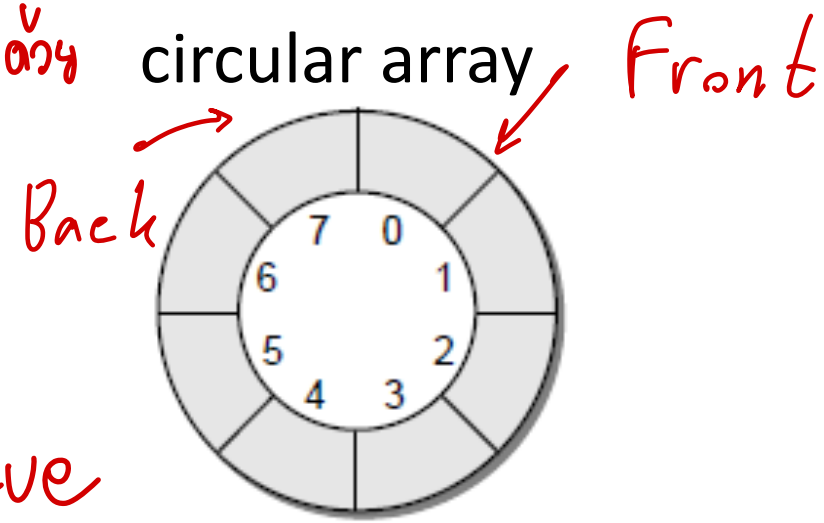
Implementation of the Queue ADT using a circular array

วิธีแก้ไข Queue



เราจะ Implement ด้วย Circular Array

ใช้นับจำนวน Queue



Q = Queue(8)

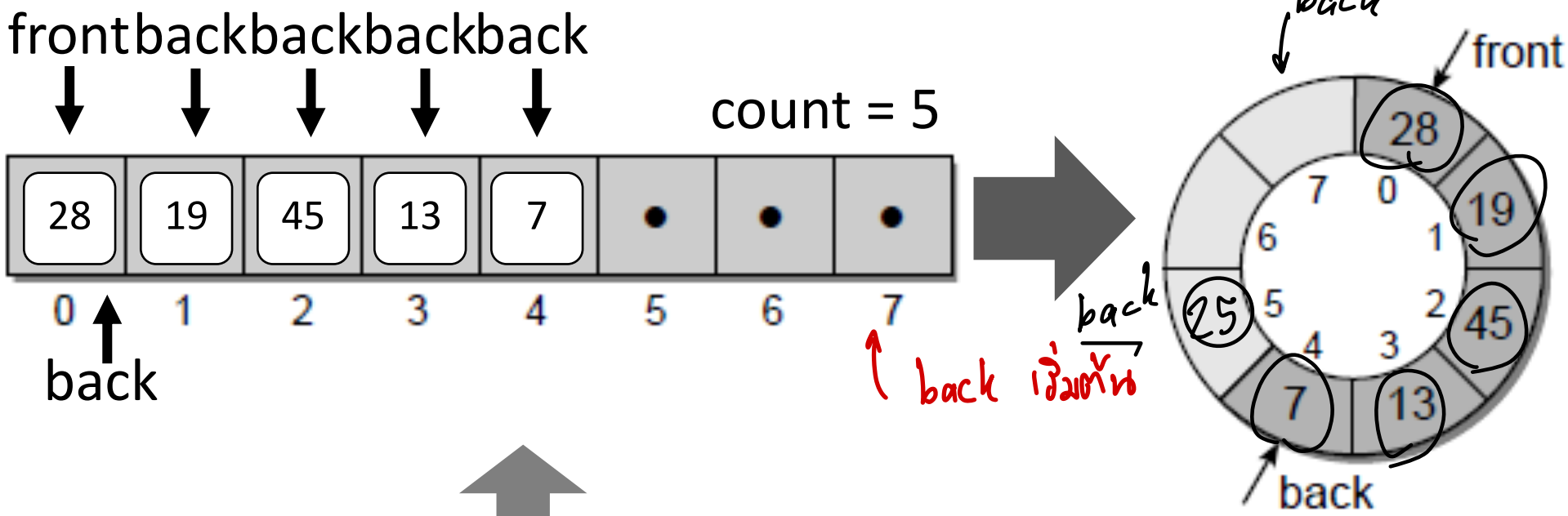
```
def __init__( self, maxSize ) :
    self._count = 0
    self._front = 0
    self._back = maxSize - 1
    self._qArray = Array( maxSize )
```

→ ตัวบอกว่าอยู่ที่ไหน

→ ตัวหน้าสุด

→ ตัวหลังสุด

Q.enqueue(28)
Q.enqueue(19)
Q.enqueue(45)
Q.enqueue(13)
Q.enqueue(7)



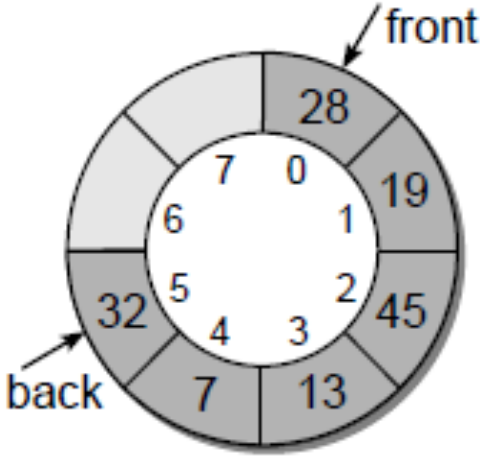
Key : Enqueue จะอ้างอิงตำแหน่ง back

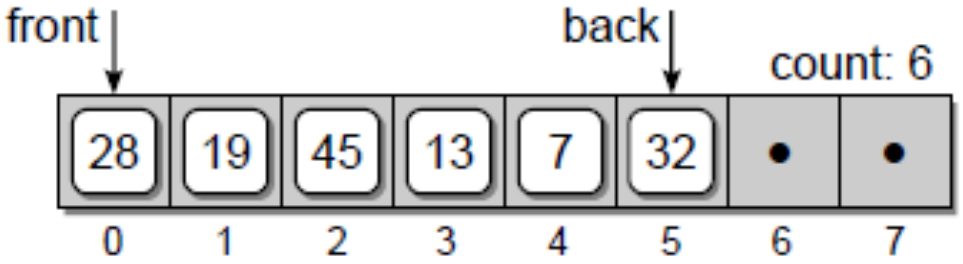
```
def enqueue( self, item ):
    assert not self.isFull(), "Cannot enqueue to a full queue."
    maxSize = len(self._qArray)
    self._back = (self._back + 1) % maxSize
    self._qArray[self._back] = item
    self._count += 1
```

เซ็ตตำแหน่งใหม่ของ back

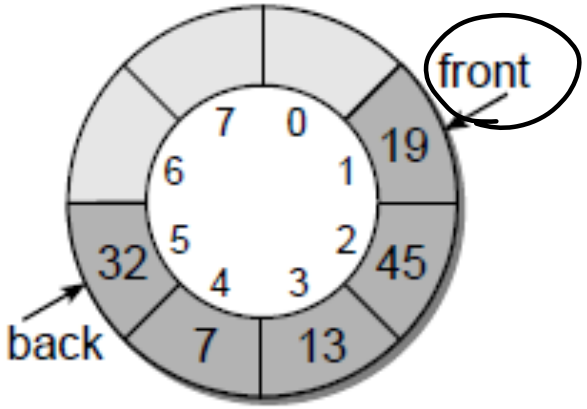
ทำให้มันวนกลับที่วงกลมเดิม

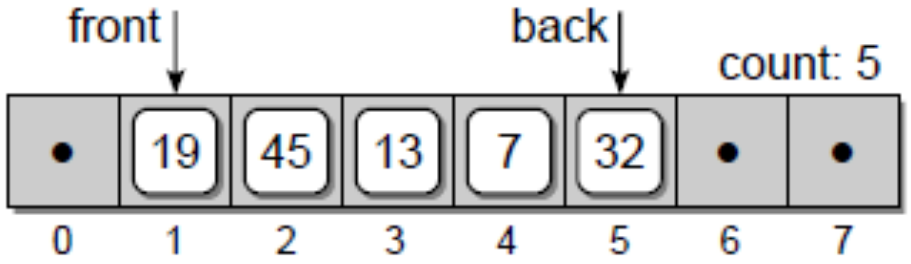
Q.enqueue(32)





q = Q.dequeue()






```
def dequeue( self ):
  assert not self.isEmpty(), "Cannot dequeue from an empty queue."
  item = self._qArray[ self._front ]
  maxSize = len(self._qArray)
  self._qArray[self._front] = None
  self._front = (self._front + 1) % maxSize
  self._count -= 1
  return item
```

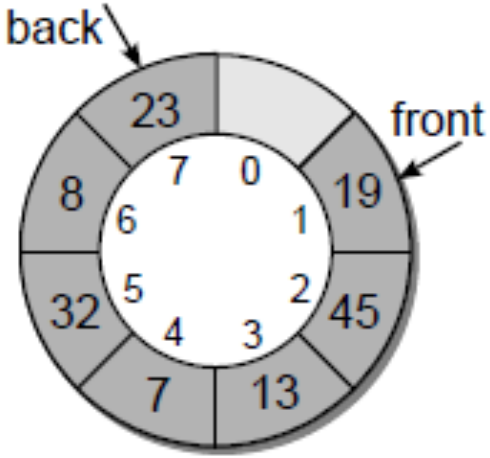
เอาตัว front

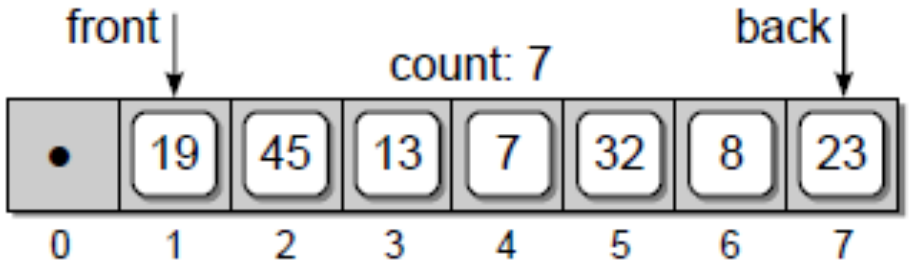
เคลียร์ค่า (ตัวค่า)

เปลี่ยนค่า front ++;

ลบความยาวข้อมูล

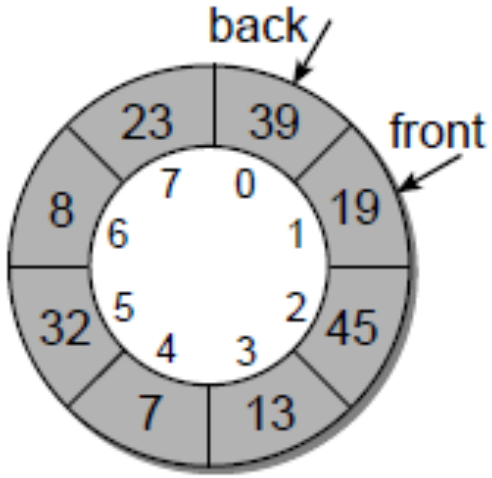
Q.enqueue(8)
Q.enqueue(23)

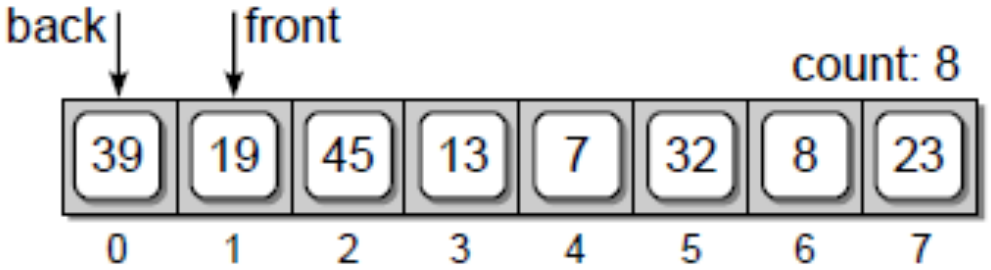




back = 7

Q.enqueue(39)





back = (back + 1) % maxSize
= 8 % 8 = 0

```
def enqueue( self, item ):
  assert not self.isFull(), "Cannot enqueue to a full queue."
  maxSize = len(self._qArray)
  self._back = (self._back + 1) % maxSize
  self._qArray[self._back] = item
  self._count += 1
```

Summary – Queue with circular array

ดีกว่า ในกรณี n element เลย

- The circular array implementation provides a more efficient solution than the Python list
- All operations have a **time-complexity of O(1)** → the array items never have to be shifted
- The circular array's drawback is **maximum-capacity** ข้อเสีย ขนาดจำกัด Memory จำกัด
- The queue with circular array is well suited for some applications → Round Robin scheduler → เจอในวิชา OS (จองเวลาเป็นวงกลม)

เริ่มแรก - ปัญหาของ Queue (List) คือเวลา Dequeue จะต้อง shift ซ้ายทุกตัว -1

แต่ถ้าเราใช้ Circular Array เราจะ shift แค่ index แค่ตัวแปรเดียว

ไม่กระทบกับข้อมูลจำนวนมาก

Practice I) Circular Array

fill the outputs and collect values in the queue

(10 mins)

Operation	Return Output	Queue[0..4]
Q = Queue(5)		[_ _ _ _ _]
Q.enqueue(9)		[9 _ _ _ _]
Q.enqueue(6)		[9 6 _ _ _]
len(Q)	2	[_ _ _ _ _]
Q.enqueue(3)		[9 6 3 _ _]
Q.enqueue(3)		[9 6 3 3 _]
Q.enqueue(7)		[9 6 3 3 7]
Q.enqueue(1)	Error, stack overflow	[_ _ _ _ _]
Q.isFull()		[_ _ _ _ _]
q = Q.dequeue()		[_ _ _ _ _]
Q.enqueue(5)		[_ _ _ _ _]
q = Q.dequeue()		[_ _ _ _ _]
Q.enqueue(4)		[_ _ _ _ _]
len(Q)		[_ _ _ _ _]

Priority Queues

รู้จักแค่ ทฤษฎี ก่อน (รอเรียน sorting)
บอกว่า แต่ละตัว มี ความสำคัญ แค่ไหน ถ้าสำคัญจะทำงานก่อน

A priority queue is simply an extended version of the basic queue with the exception that a *priority p* must be assigned to each item at the time it is enqueued. There are two basic types of priority queues: bounded and unbounded.

- The bounded priority queue assumes a small limited range of p priorities over the interval of integers [0 … p).
- The unbounded priority queue places no limit on the range of integer values that can be used as priorities.

Priority ที่สูงสุดคือ เลข 1
(ยิ่งค่า Priority ที่น้อย จะสำคัญมาก)

Priority Queue ADT

หลักการของมันคือ priority สูง มันจะให้ไปอยู่คิวข้างหน้า

A priority queue is a queue in which each item is assigned a priority and items with a higher priority are removed before those with a lower priority, irrespective of when they were added. Integer values are used for the priorities with **a smaller integer value having a higher priority.**

- ***bounded priority queue:*** priorities are int values between 0 and a predefined upper limit *p*
- ***unbounded priority queue:*** places no limits on the range of priorities.

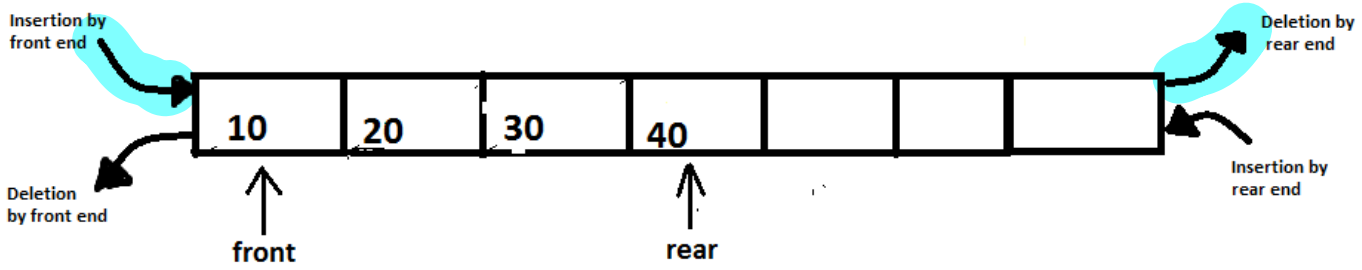
PriorityQueue():	Creates an empty **unbounded priority queue**
BPriorityQueue(numLevels):	Creates an empty **bounded priority queue** with priority in the range of 0 to *numLevels - 1*
isEmpty():	Returns a boolean value indicating whether the queue is empty
length ():	Returns the number of items currently in the queue
enqueue(item, priority):	Adds the item to the queue by inserting in the position based on the priority. The priority value must be within the range for a **bounded priority queue**
dequeue():	Removes and returns the front item from the queue with *the highest priority*. If two items have the same priority, then they are removed in a FIFO order. An item cannot be dequeued from an empty queue

ยกตัวอย่างในชีวิต เวลาหมอเจอเคสฉุกเฉิน Priority เขาสูง

Double-Ended Queue ADT

คิวที่มีทั้งหัวและท้าย

มันจะเข้าทางหัวหรือทางหางก็ได้



เหมือน stack กับ queue มารวมกันครึ่งทาง

A **doubled-ended queue** is a queue-like data structure that supports insertion and deletion at both the front and the back of the queue.

Deque(): Create a new empty deque, which no item with in

AddFirst(item): Add an item to the front of deque

AddRear(item): Add an item to the back of deque

DeleteFirst( ): Remove and return the first item from deque; an error occurs if the deque is empty

DeleteRear ( ): Remove and return the last item from deque; an error occurs if the deque is empty

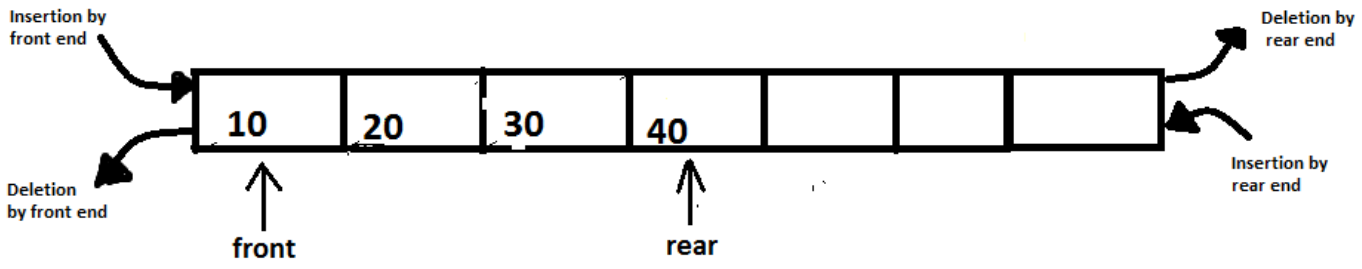
First(): Return (but do not remove) the first item of deque; an error occurs if the deque is empty

Rear(): Return (but do not remove) the last item of deque; an error occurs if the deque is empty

isEmpty( ): Return **True** if deque does not contain any items

length( ): Return the number of items in deque

Implementing a Deque with a Python List



rear

Deque Method

- D.addFirst(item)
- D.addRear(item)
- D.deleteFirst( )
- D.deleteRear( )
- D.first()
- D.rear()
- D.isEmpty( )
- len(D)

Implement with Python's List

- L.insert(0,item)
- L.append(item)
- L.pop(0)
- L.pop( )
- L[0]
- L[-1]
- len(L) == 0
- len(L)

ข้อดีของ Deque คือ
เป็นอิสระสองทาง

Practice II) Deques: fill the outputs and collect values in the deque (10 mins)

จริงๆ Deque มีอยู่ใน module collections

Operation	Return Output	Deque
D = Deque()		
D.AddRear(5)		
D.AddFirst(3)		
D.AddFirst(7)		
d = D.First()		
d = D.DeleteRear()		
l = length(D)		
d = D.DeleteRear()		
d = D.DeleteRear()		
D.AddFirst(6)		
d = D.Rear()		
D.AddFirst(8)		
D.isEmpty()		
d = D.Rear()		

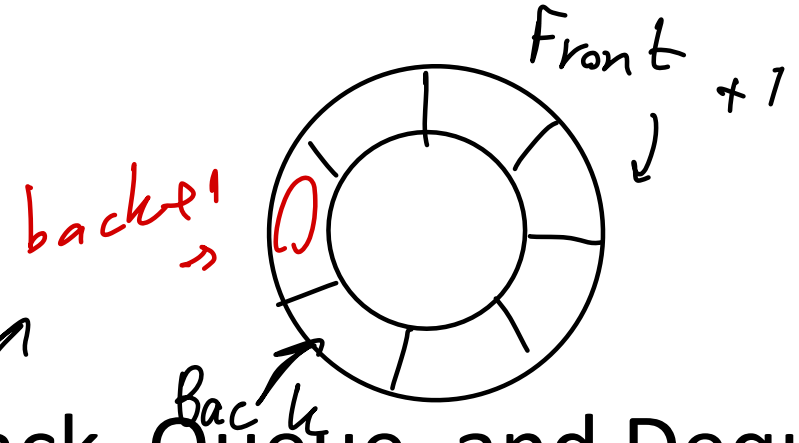
Deque can be implemented in python using the module "**collections**". Deque is preferred over list in the cases where we need quicker append and pop operations from both the ends of container, as deque provides an **O(1)** time complexity for append and pop operations as compared to list which provides O(n) time complexity.

https://www.geeksforgeeks.org/deque-in-python/

```
import collections

# initializing deque
de = collections.deque([1,2,3])
```

What should you know



- Understand the operations of Stack, Queue, and Deque
- How to implement Stack class with Python's List
- How to implement Queue class with Python's List
- How to implement Queue class with circular array
- How to implement Deque class with Python's List