

# Data Structure and Algorithm

Data Structure Implementation

[Stack , Queue, Link List]

# Data Structure Design

- Draw data structure pictures
  - Choose based data type to implement (List , Dict ,...) or
  - Implement using Class
- Define usage method
  - Implement algorithm of each method

# Stack Implementation

# Stack Implementation

Logical ADT :

1. **Data** : ของมีลำดับ มีปลายบน

2. **Methods** :

- |                             |               |
|-----------------------------|---------------|
| 1. init empty stack         | init()        |
| 2. insert i ที่ top         | push(i)       |
| 3. เอาของที่ top ออก        | i = pop()     |
| 4. ดูของที่ top (ไม่เอาออก) | i = peek()    |
| 5. stack empty ?            | b = isEmpty() |
| 6. stack full ?             | b = isFull()  |
| 7. หาจำนวนของใน stack       | i = size()    |

Implementation ?

Python List

`S = []`

`S.append(i)` ใส่ท้าย

`i = S.pop()` อันท้าย

?

?

?

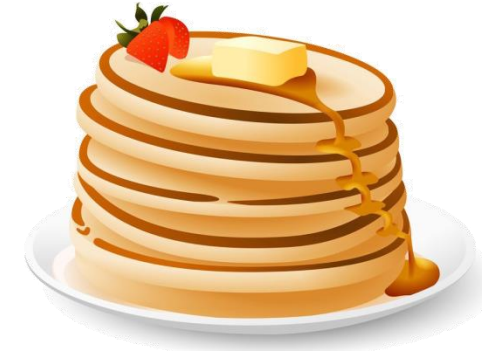
?



**LIFO**

Last in First out

# Stack Data Implementation



# 1. Data :

**`__init__()` : constructor ให้ค่าตั้งต้น**

## 2 underscores

## 2 underscores

# Data Implementation : `__init__()`

## 1. Data Implementation : Stack กองของซ้อนกัน ของมีลำดับ มีปลายด้านบน -> Python List

ทำใน constructor

`self` คือ object ที่เรียก method ในแต่ละครั้ง

เช่น `s = Stack()`

`self` หมายถึง `s` เสมือนเรียก `s = Stack(s)`

`self` จะถูก pass เป็น arg. ตัวแรก โดยอัตโนมัติ

docstring : ใน triple quote

`print(Stack.__doc__)`

→ docstring

constructor

ถูกเรียกโดยอัตโนมัติเมื่อ

instantiate instance ใหม่

```
class Stack:
    """ class Stack
        create empty stack
    """
    total = 0 # class data
    def __init__(self):
        self.items = []
        self.size = 0
        stack.total += 1
```

Class Data

สำหรับทุก stack

items , size:

Instance Attributes /data

สำหรับแต่ละ instance

items

[ ]

size

0

`s = Stack()`

`print(s.items)` [ ]

`print(s.size)` 0

`s2 = Stack()`

`print(s.total)` → 2

เรียกชื่อ class :

สร้าง object ใหม่ (instantiate instance/obj)

ไปเรียก constructor ฟังก์ชัน `__init__()`

# \_\_init\_\_() with Default Argument

```
class Stack:
    """ class Stack
        default : empty stack /
        Stack([list])
    """
    def __init__(self, list = None):
        if list == None:
            self.items = []
        else:
            self.items = list
```

default argument  
ถ้าไม่มีการ pass arg. มา  
list = None  
ถ้า pass arg. มา  
list = ตัวที่ pass มา

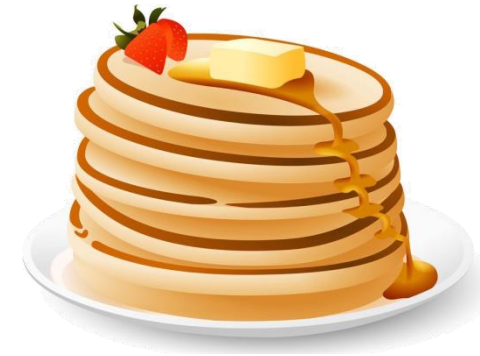
```
s = Stack()
```

```
s1 = Stack(['A', 'B', 'C'])
```

ไม่เหมือนกับ C++ & Java

ใน Python มี constructor ได้ตัวเดียว

# Stack Operation Implementation



## 1. Data :

`__init__()` : constructor ให้ค่าตั้งต้น

## 2. Methods (Operations) :

2. `push()` : ใส่ ด้านบน top

3. `pop ()` : เอาออก ด้านบน **top**

4. `peek()` : ดู top ไม่เอาออก

5. `isEmpty()` : stack ว่าง ?

6. `size()` : มีของกี่อัน



# push()

```
class Stack:
    def __init__(self, list = None):
        if list == None:
            self.items = []
        else:
            self.items = list
        self.size = len(self.items)

    def push(self, i):
        self.items.append(i)
        self.size += 1
```

Python list automatically expanding size

`list.append (i)`: insert i ที่ท้าย list

```
s = Stack()
s.push('A')
s.push('B')
s.push('C')
```

s.items

```
[]
['A']
['A', 'B']
['A', 'B', 'C']
```



# pop()

```
class Stack:
    def __init__(self, list = None):
        if list == None:
            self.items = []
        else:
            self.items = list
```

```
def pop(self):    # remove & return อันบนสุด
    return self.items.pop()
```

อย่าลืม return !!!

list.pop() : delete ตัวสุดท้ายของ list

list.pop(i) : delete ตัวที่ index i ของ list

```
print(s.items)
print(s.pop())
print(s.pop())
s.pop()
```

['A', 'B']

B

A

# error Stack Underflow

B

A

# peek()

```
class Stack:
    def __init__(self, list = None):
        if list == None:
            self.items = []
        else:
            self.items = list
```

```
def peek(self):    # return อันบนสุด
    return self.items[ -1]
```

-1 : last index

```
print(s.items)
```

['A', 'B']

```
print(s.peak())
```

B

```
print(s.items)
```

['A', 'B']

B

A

# isEmpty()

```
class Stack:  
    def __init__(self, list = None):  
        if list == None:  
            self.items = []  
        else:  
            self.items = list
```

```
def isEmpty(self):  
    return self.items == []
```

```
return len(self.items) == 0
```

```
print(s.items)
```

```
print(s.isEmpty())
```

['A', 'B']

false

B

A

# size()

```
class Stack:  
    def __init__(self, list = None):  
        if list == None:  
            self.items = []  
        else:  
            self.items = list
```

```
def size(self):  
    return len(self.items)
```

```
print(s.items)
```

['A', 'B']

```
print(s.size())
```

2

B

A

# Stack Implementation

```
class Stack:
    """ class Stack
        default : empty stack / Stack([...])
    """
    def __init__(self, list = None):
        if list == None:
            self.items = []
        else:
            self.items = list

    def __str__(self):
        s = 'stack of ' + str(self.size()) + ' items : '
        for ele in self.items:
            s += str(ele) + ' '
        return s
```

`__str__()` ต้อง return string

```
def push(self, i):
    self.items.append(i)

def pop(self):
    return self.items.pop()

def peek(self):
    return self.items[-1]

def isEmpty(self):
    return self.items == []

def size(self):
    return len(self.items)
```

```
s1 = Stack([1,2,3])
```

```
print(s1.items) [1, 2, 3]
```

```
print(s1)
```

```
stack of 3 items : 1 2 3
```



## Activity

ให้นักศึกษาเขียนโปรแกรมสร้าง Stack แล้ว  
Push เก็บข้อมูล ABCDEF  
Pop ออกจน empty

# Queue Implementation



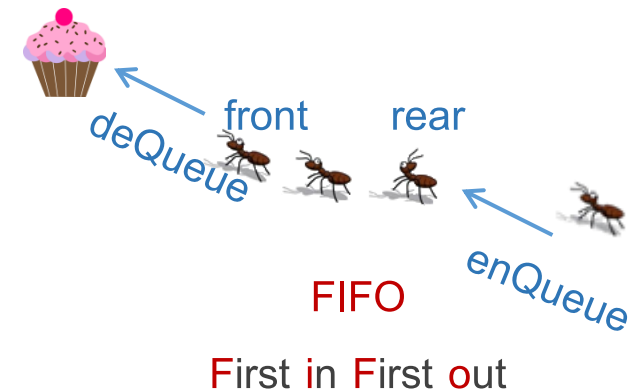
# Logical Abstract Data Type

Logical ADT :

1. Data : ของมีลำดับ มีปลาย หัว **front** ท้าย **rear**

2. Methods :

1. `init()`                      init empty queue
2. `enqueue(i)`            insert i ที่ **rear** / **tail**
3. `i = dequeue()` return + เอาของที่ **front** / **head** ออก
4. `b = isEmpty()`    queue empty ?
5. `b = isFull()`        queue full ?
6. `i = size()`            return จำนวนของใน queue



# Queue Implementation

Logical ADT :

1. Data : ของมีลำดับ มีปลายหัว ท้าย

2. Methods :

1. init()                  init empty Q

2. enqueue(i)          insert i ที่ rear

3. i = dequeue() return + เอาของที่ front ออก    i = Q.pop(0) อันแรก

4. b = isEmpty()    Q empty ?

5. b = isFull()      Q full ?

6. i = size()          return จำนวนของใน Q

Data Implementation ?

Python List

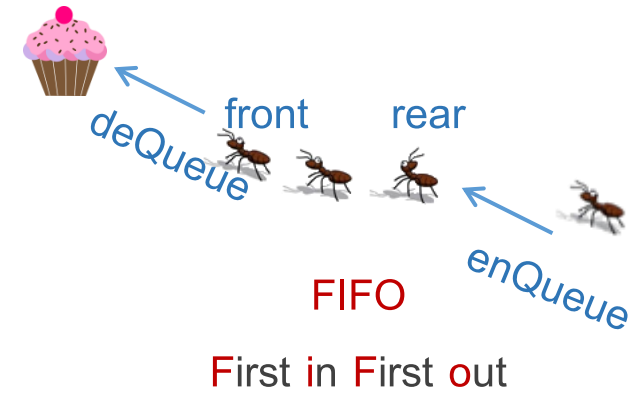
Q = []

Q.append(i) ใส่ท้าย

Q == [] ?

list expansion -> Python List implementation

i = len(Q)



# Queue Data Implementation



## 1. Data :

        init         () : constructor ให้ค่าตั้งต้น

2 underscores

2 underscores

# Data Implementation : `__init__()`

## 1. Data Implementation : Queue แลวคอย มีปลาย หัว ท้าย -> Python List ทำใน constructor `__init__()`

`self` คือ object ที่เรียก method ในแต่ละครั้ง  
เช่น `q = Queue()`  
`self` หมายถึง `q` เสมือนเรียก `q = Queue(q)`  
`self` จะถูก pass เป็น arg. ตัวแรก โดยอัตโนมัติ

docstring : ใน triple quote  
`print(Queue.__doc__)`  
→ docstring

constructor function  
ถูกเรียกโดยอัตโนมัติเมื่อ  
instantiate instance ใหม่

instantiate instance (object) ใหม่  
โดยไม่ pass argument

```
class Queue:
    """ class Queue
    create empty Queue
    """
    def __init__(self):
        self.items = []
```

constructor :  
ใช้ define  
Instance Attributes ได้ ใน  
ตัวอย่างนี้มี attribute เดียวคือ  
items

items :  
Instance  
Attributes  
สำหรับแต่ละ  
instance

```
q = Queue()
```

```
print(q.items) []
```

# Default Argument

```
class Queue:
    """ class Queue
        default : empty Queue/
        Queue([list])
    """
    def __init__(self, list = None):
        if list == None:
            self.items = []
        else:
            self.items = list
```

default argument  
ถ้าไม่มีการ pass arg. มา  
list = None  
ถ้า pass arg. มา  
list = ตัวที่ pass มา

Object None  
ใช้เช็ค obj identity

```
q = Queue()
```

```
print(q.items)
```

```
[]
```

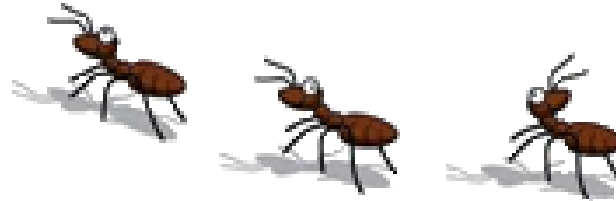
```
q1 = Queue(['A', 'B', 'C'])
```

```
print(q1.items)
```

```
['A', 'B', 'C']
```

ไม่เหมือนกับ C++ & Java ใน Python ไม่สามารถมี constructor หลายตัวได้

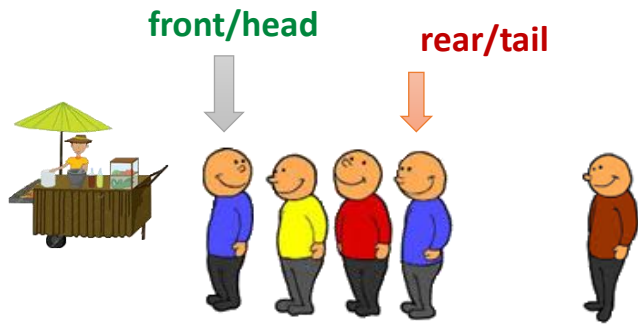
# Queue Operation Implementation



## 2. Methods :

1. `__init__()` : ให้ค่าตั้งต้น
2. ใส่ `enqueue()` : ด้านท้าย **rear**
3. เอาออก `dequeue()` : ด้านหัว **front**
4. `isEmpty()` : queue ว่าง ?
5. `size()` : มีของกี่อัน

# enqueue()



```
q = Queue()  
print(q.items)  
q.enqueue('A')  
print(q.items)  
q.enqueue('B')  
print(q.items)  
q.enqueue('C')  
print(q.items)
```

[ ]

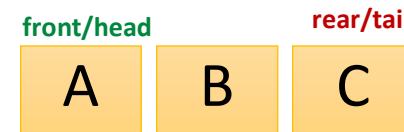
[ 'A' ]

[ 'A', 'B' ]

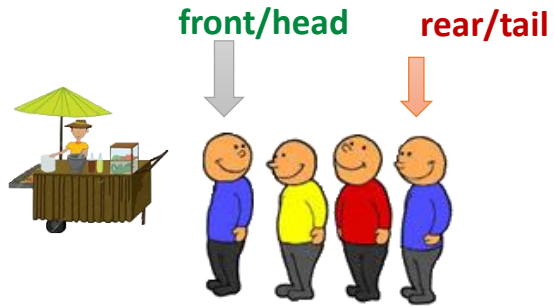
[ 'A', 'B', 'C' ]

ใน class Queue:

```
def enqueue(self, i):  
    self.items.append(i) # insert i ที่ท้าย list
```



# deQueue()



```
print(q.items)
print(q.deQueue())
print(q.items)
print(q.deQueue())
print(q.items)

q.deQueue()
```

class Queue:

```
def deQueue(self):
    return self.item.pop(0) # pop out index 0
```

deQueue must check  
Queue Underflow

['A', 'B']

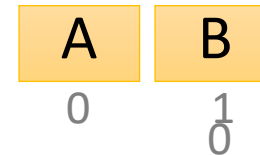
A

['B']

B

[]

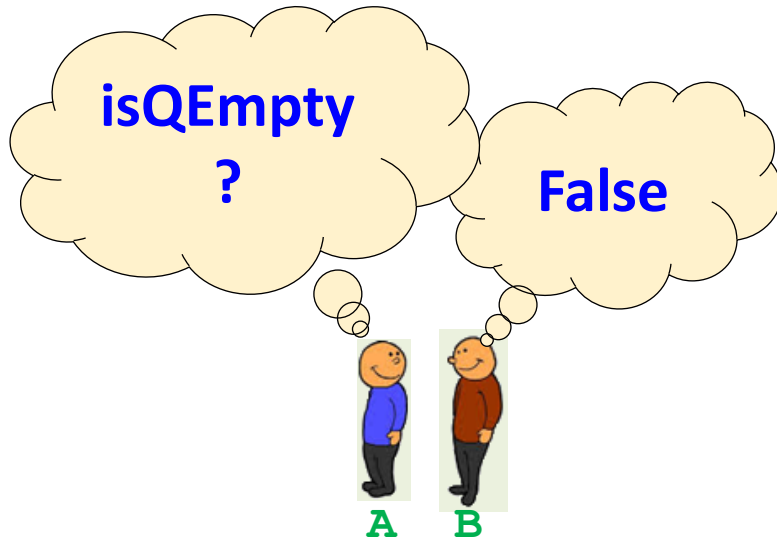
front/head rear/tail



```
# error
# Queue Underflow
```



# isEmpty()



ใน class Queue:

```
def isEmpty(self):
```

```
    return self.items == []
```

```
    #return len(self.items) == 0
```

Q Empty  
เมื่อไหร่ ?

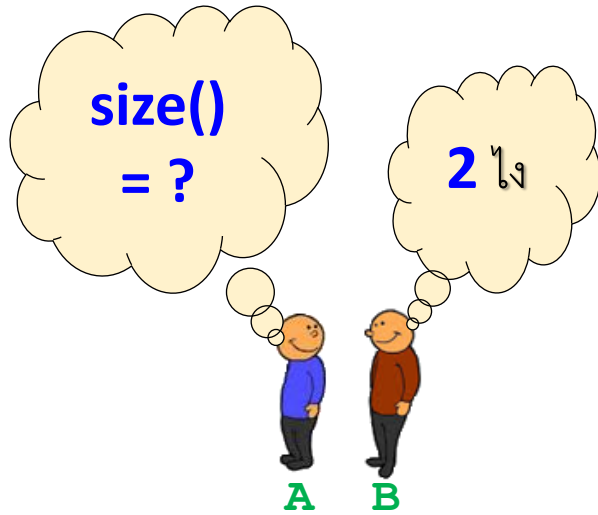
```
print(q.items)
```

```
print(q.isEmpty())
```

['A', 'B']

false

# size()



```
print(q.items)  
print(q.size())
```

```
ใน class Queue:
```

```
def size(self):  
    return len(self.items)
```

```
['A', 'B']
```

```
2
```

## Activity

ให้นักศึกษาเขียนโปรแกรมสร้าง Queue แล้ว  
เก็บข้อมูล ABCDEF  
หลังจากนั้นให้แสดงผล ขนาดของข้อมูลที่เก็บใน Queue

# Link List Implementation

- Node Class / List Class



### 1. Data :

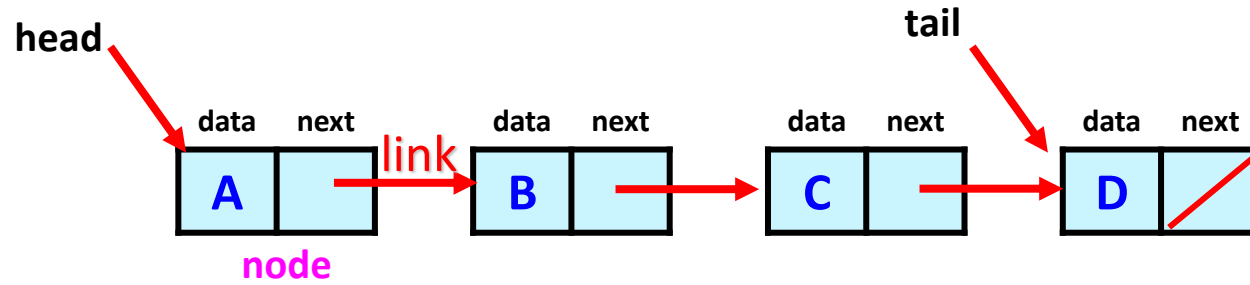
`__init__()` : constructor ให้ค่าตั้งต้น



2 underscores

2 underscores

# Node Class



```
p = node('A', None)
```

```
class node:

    def __init__(self, data, next = None):

        self.data = data

        if next is None:
            self.next = None
        else:
            self.next = next

    def __str__(self):
        return str(self.data)
```

# List Class

class list:

```
""" unordered singly linked list
    with head
"""
```

```
def __init__(self):  
    self.head = None
```

```
""" unordered singly linked list
    with head & tail
"""
```

```
def __init__(self):  
    self.head = self.tail = None
```

```
def __init__(self, head = None):
```

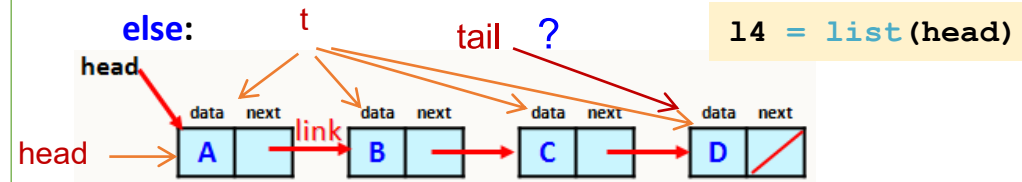
```
""" unordered singly linked list
    can set default list
    with head, tail & size
"""
```

```
if head == None:
```

```
    self.head = self.tail = None
```

```
    self.size = 0
```

```
else:
```



```
    self.head = head
```

```
    t = self.head
```

```
    self.size = 1
```

```
    while t.next != None: # locating tail & find size
```

```
        t = t.next
```

```
        self.size += 1
```

```
    self.tail = t
```

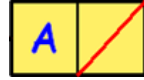


# Methods

1. `__init__()` : ให้ค่าตั้งต้น
2. `size()`:
3. `isEmpty()`:
4. `append ()` : add at the end
5. `__str__()`:
6. `addHead()` : ให้ค่าตั้งต้น
7. `remove(item)`:
8. `removeTail()`:
9. `removeHead()` :
10. `isIn(item)`: / `search(item)`
11. . . .



# Creating a List



`node('A', None)`

```
class list:
```

```
    """ unordered singly linked list
    with head """
```

```
    def __init__(self):
        self.head = None
```

```
    def append(self, data):
```

```
        """ add at the end of list """
```

```
        p = node(data)
        if self.head == None: # null list
            self.head = p
        else:
            t = self.head
            while t.next != None :
                t = t.next
            t.next = p
```

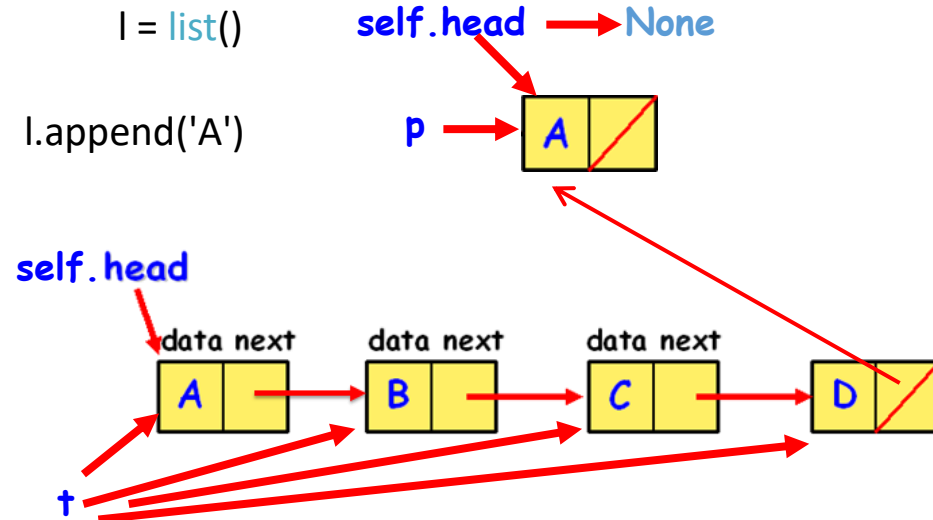
```
class node:
```

```
    def __init__(self, data, next = None):
```

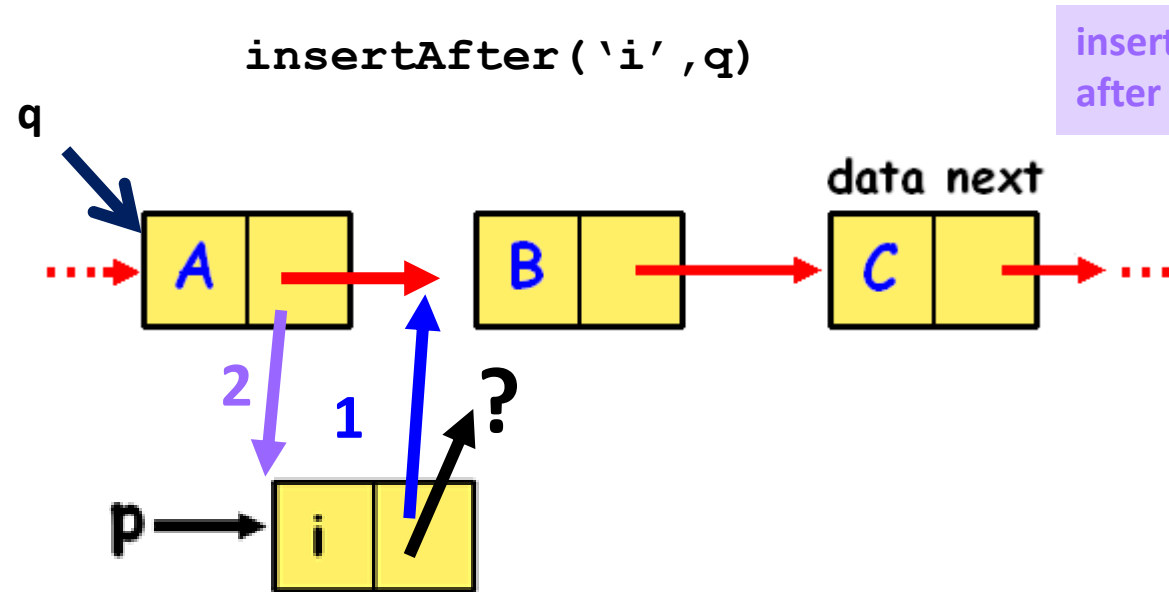
```
        self.data = data
```

```
        if next == None:
            self.next = None
```

```
        else:
            self.next = next
```



# Insert After



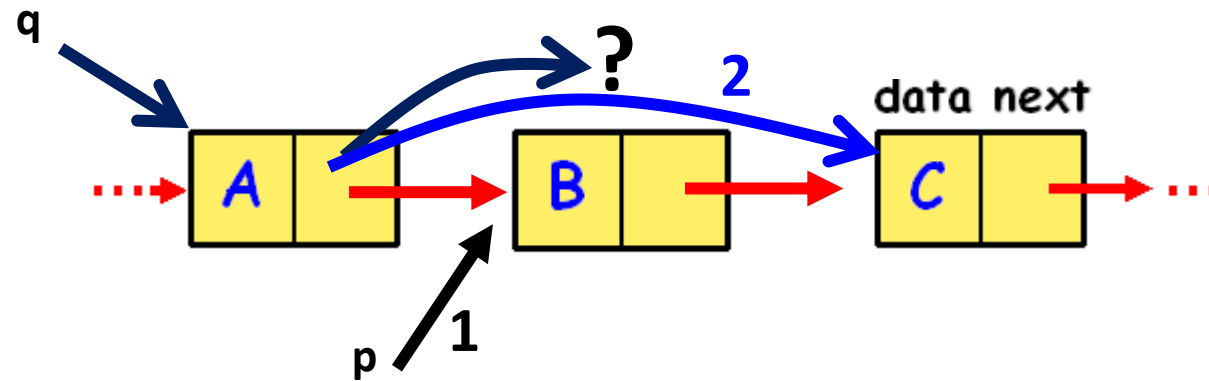
insert node data  
after a node pointed by q

Why insert after ?  
Can you insert before ?

# Delete After

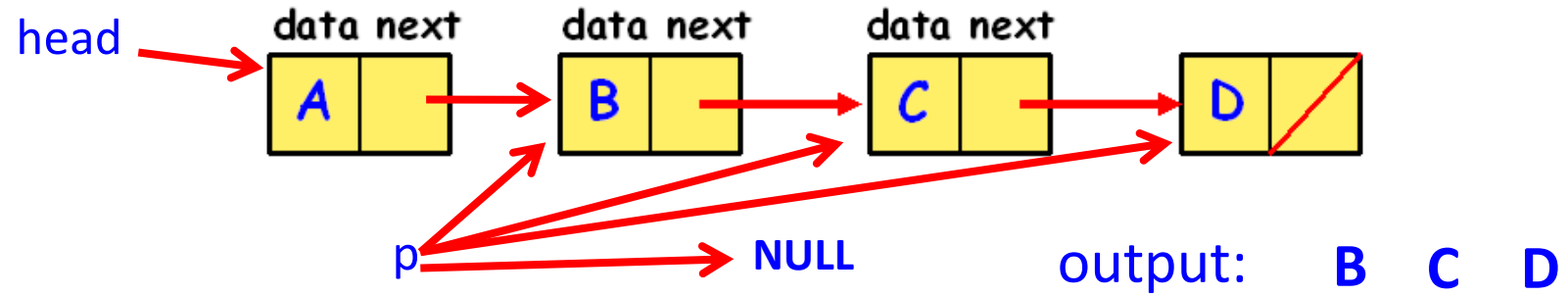
`deleteAfter(q)`

delete a node  
after a node pointed by q



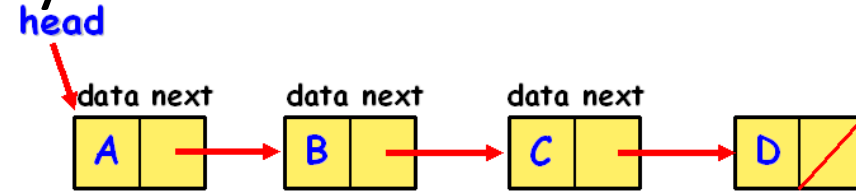
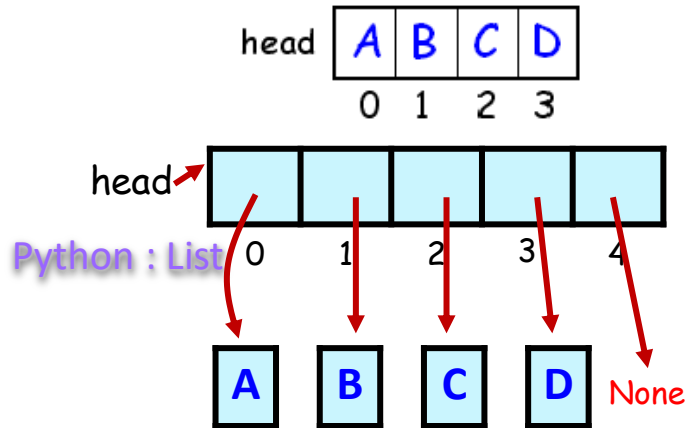
# print list

Design how to call.



```
    p is not None:
while p != None:
    print(p.data)
    p = p.next
}
```

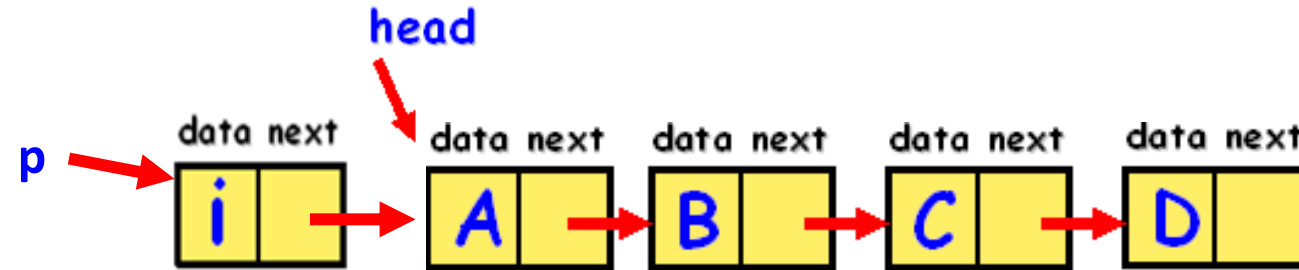
# Linked List VS Sequential Array



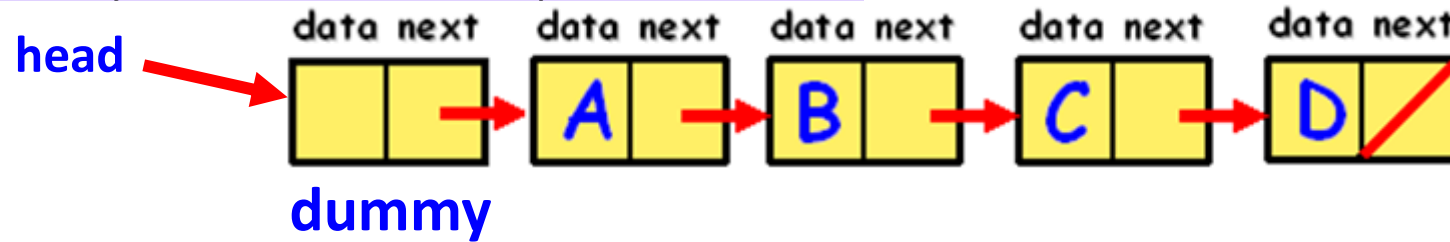
Sequential Array	Linked List
<ul style="list-style-type: none"><li>• Insertion / Deletion    Shifting Problem.</li><li>• Random Access.</li><li>• C array : Automatic Allocation. Python List array : Dynamic Allocation</li><li>• Lifetime : C-array, Python List<ul style="list-style-type: none"><li>• from defined until its scope finishes.</li></ul></li><li>• Only keeps data.</li></ul>	<ul style="list-style-type: none"><li>• Solved.</li><li>• Sequential Access.</li><li>• Node : Dynamic Allocation.</li><li>• Node Lifetime : from allocated (C : malloc()/new, python: instantiate obj) until C: deallocated by free()/delete, Python : no reference.</li><li>• Need spaces for linkes.</li></ul>

# Dummy Node

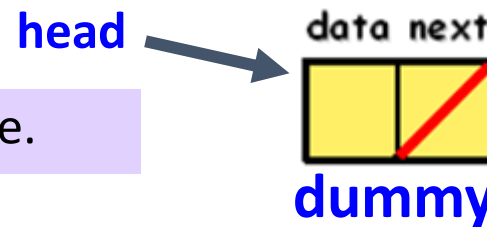
To insert & delete at 1<sup>st</sup> position change head ie. make special case.



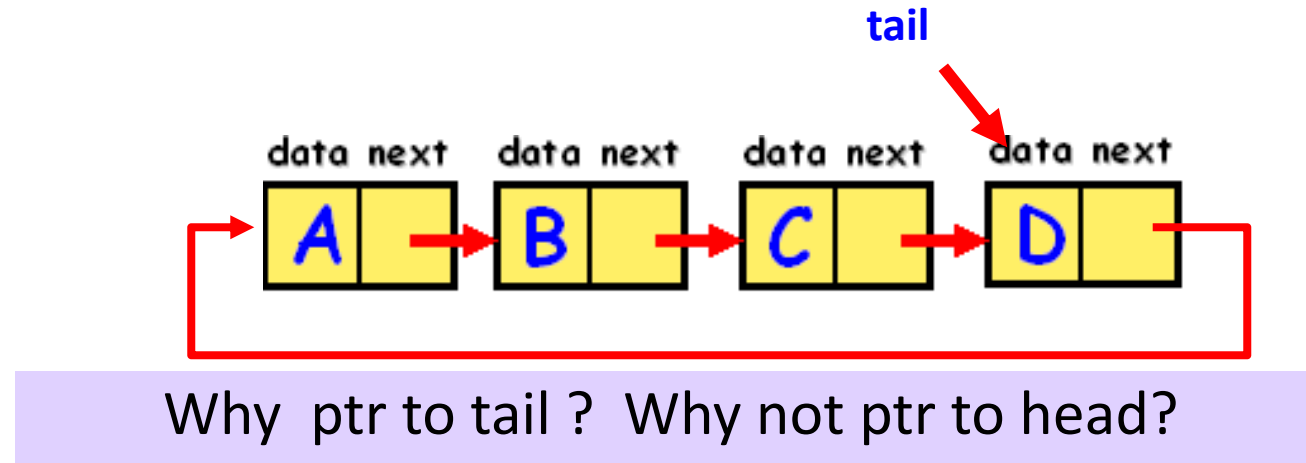
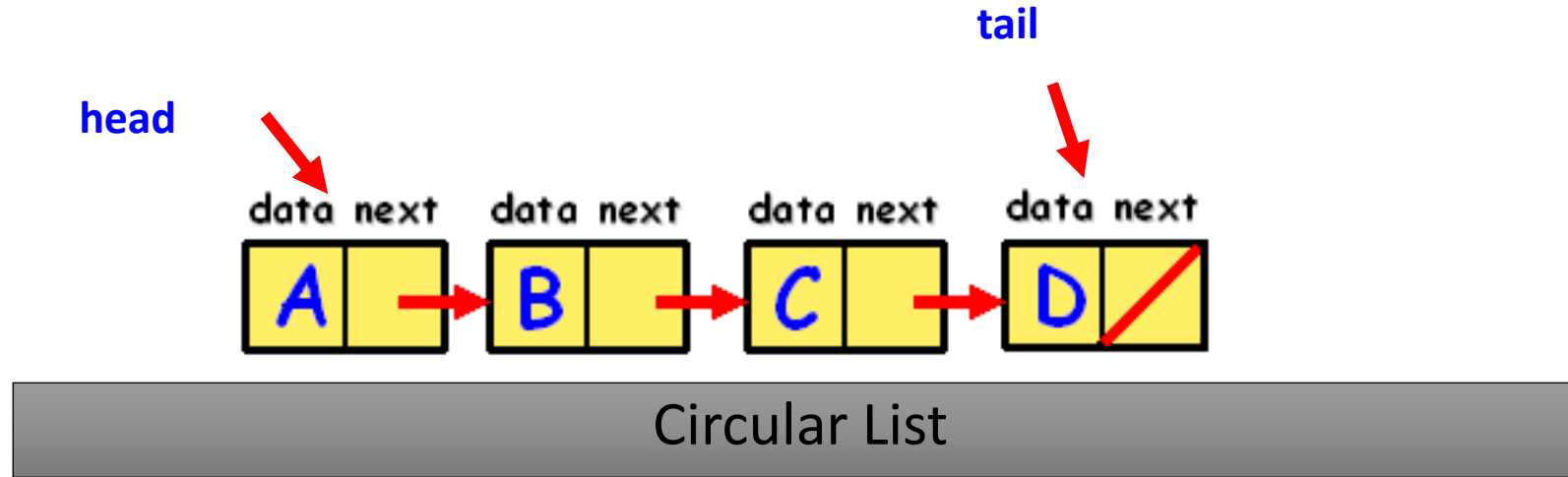
“Dummy Node” solves the problem.



Empty List has a dummy node.



# Head & Tail Nodes



## Activity

ให้นักศึกษาเขียนโปรแกรมสร้าง Link List แล้ว  
เก็บข้อมูล ABCDEF  
หลังจากนั้นให้ แทรกข้อมูล XYZ ระหว่าง C,D