RAM: random access memory
随机存储器

insert function: (Index, value)

remove function: remove (Index)

In python, list is implemented as dynamic array.

Double linked list

```
class Node (self, data = None, next = None)
    self.data = data
    self.next = next

class LinkedList (self)
    self.head = None (head point)

    def insert_at_begining (self, data):
        node = Node (data, self.head)
        self.head = node

    def print (self):
        if self.head is None:
            print ("linked is empty")
            return

        itr = itr.head
        llstr = ''
        while itr:
            llstr += str(itr.data) + '-->'
            itr = itr.next
        print (llstr)

if __name__ == '__main__'
    ll = LinkedList()
    ll.insert_at_begining(5)
    ll.insert_at_begining(89)
    ll.print
```

→ 让这个链条有一个初始的head.

下一个指针

相当于第一个方块

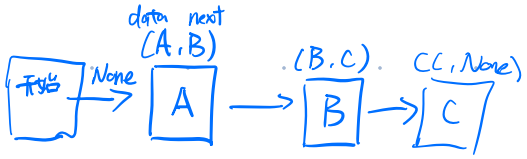
相当于第一个 point

explanation:

在一个node里面，存在两个值
第一个是data，代表本身
第二个是next，代表下一个变量的值.

这一步是让itr = 原本的值

这一步就可以移到下一个变量的值，实现数值的转送




```
def insert_at_end (self, data):
    if self.head is None:
        self.head = Node (data, None)
        return
```

第一个head如果碰上None
说明碰到链条末尾了

```
itr = self.head
while itr.next:                    → 查到最后一个
    itr = itr.next
itr.next = Node(data, None)        → 如果到最后一个 就给他赋值
```

```
def insert_values(self, data_list):
    self.head = None
    for data in data_list:
        self.insert_at_end(data)   → 一直接下去
```

```
def get_lenght(self):
    count = 0
    itr = self.head
    while itr:
        count += 1
        itr = itr.next
    return count
```

```
def remove_at(self, index)
    if index < 0 or index >= self.get_length():
        raise Exception("Invaild index")
    if index == 0
        self.head = self.head.next
        return
    count = 0
    itr = self.head
    while itr:
        if count == index - 1:
            itr.next = itr.next.next      → 让第一个直接连第三个，跳过第二个
            break
        itr = itr.next                    → 继续走下面的元素
        count += 1
```