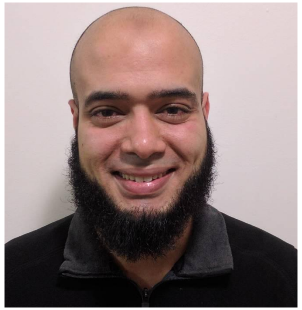
Data Structures

Singly Linked List

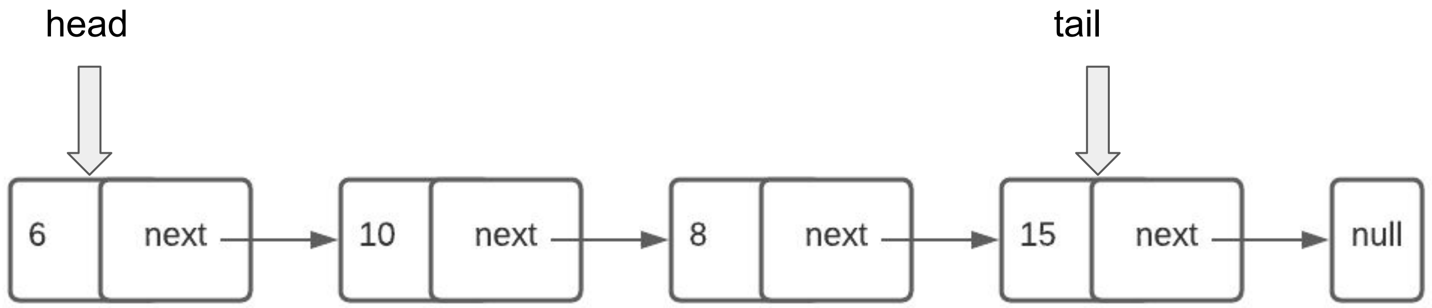
Mostafa S. Ibrahim
Teaching, Training and Coaching for more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)



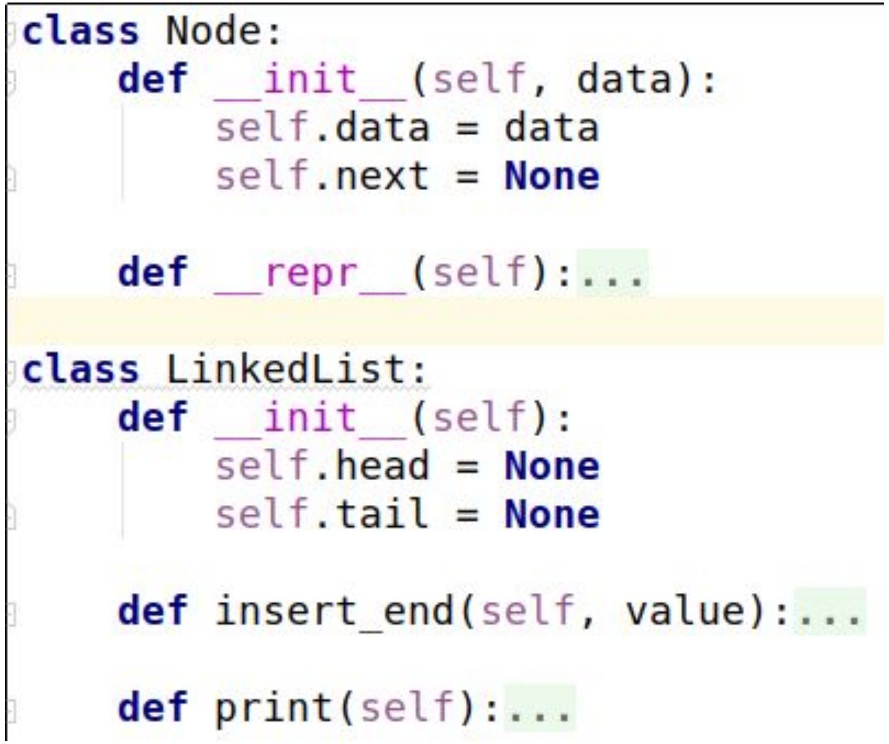
Singly Linked List

- Linked List is a **sequence of nodes**, where each node contains data and a link to the next node, creating a dynamic connected chain of values
  - We can easily expand its data with a new element in **O(1) memory**
  - Several further operations are possible
- The first node is called **head** and the last one (**optional**) is called **tail**



Singly Linked List: Data Structure

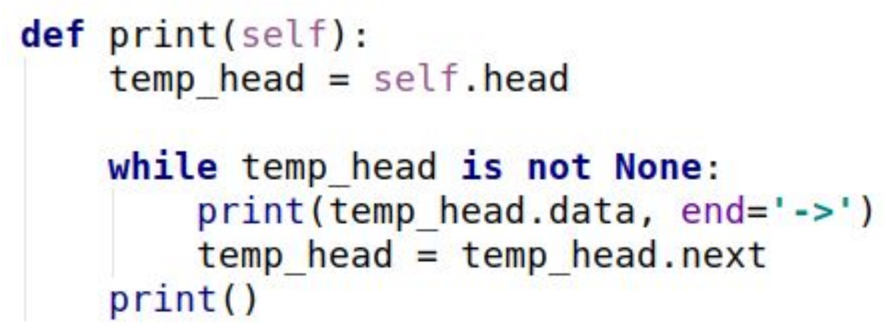
- The linked list data structure is simply 2 nodes; head and tail
- To manipulate the content, several operations can be implemented



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

    def __repr__(self):...

class LinkedList:
    def __init__(self):
        self.head = None
        self.tail = None

    def insert_end(self, value):...

    def print(self):...
```

Singly Linked List: print

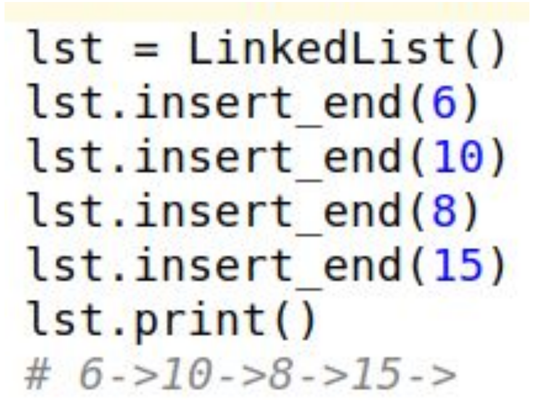
- Let’s rewrite the print function
- The key difference: we MUST take a **copy** of the head node
- This node will trace the list and end up being None, so head must remain there pointing to the first node
- Tip: Don’t corrupt your DS



```python
def print(self):
    temp_head = self.head

    while temp_head is not None:
        print(temp_head.data, end='->')
        temp_head = temp_head.next
    print()
```

Singly Linked List: insert_end

- In the previous lectures, we created and inserted the nodes manually
- The Insert_end method should do this automatically
- Take 10 minutes to implement it
- There are clearly two distinct cases to consider:
- **Case 1**: the head/tail is None initially
- **Case 2**: we have an existing list of items we want to add to
    - Tip. DRAW the list BEFORE and AFTER



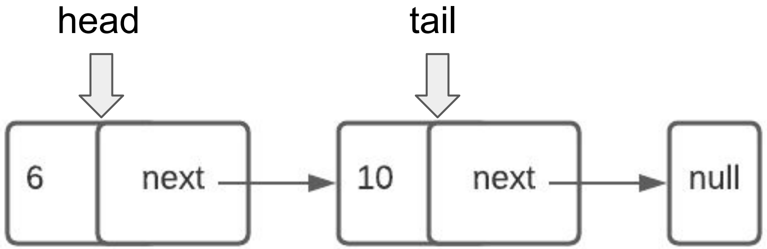
```
lst = LinkedList()
lst.insert_end(6)
lst.insert_end(10)
lst.insert_end(8)
lst.insert_end(15)
lst.print()
# 6->10->8->15->
```

Singly Linked List: insert_end

- First of all, we create a node with the requested value
- Case 1: the head/tail are both Node
  - Just set head = tail = this new node
- Case 2: we have 1+ item(s) in the list; i.e. head and tail will already have been defined - and cannot be Node!
  - Head won’t be changed
  - We need to link the tail node to the new item
  - We then set our tail node to be that new item, as the new item should be our tail now (hint: the function is called insert_end)

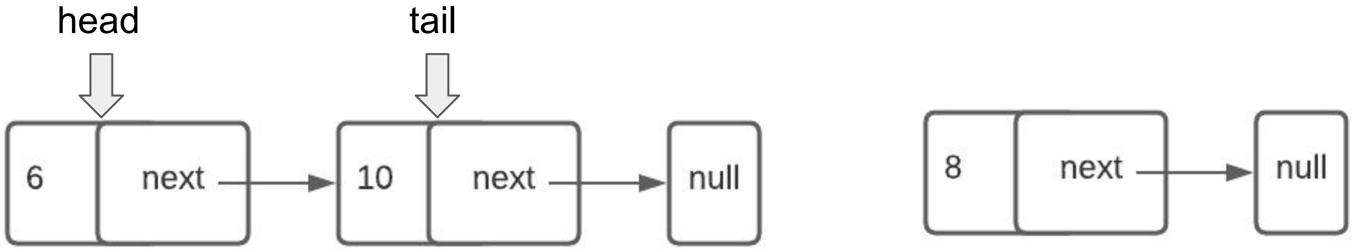
Singly Linked List: insert_end

- Assume the current list is (6, 10)
- Our goal is to add value 8



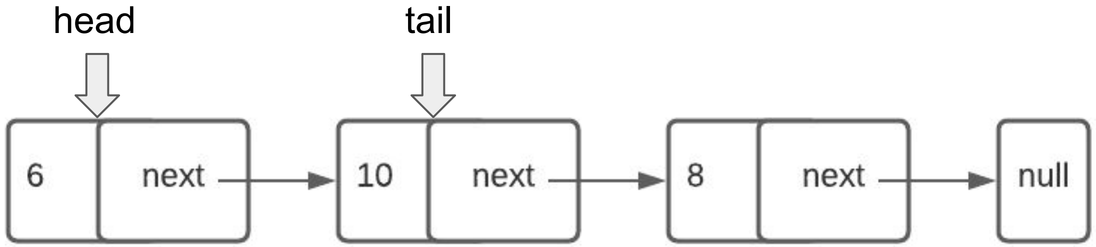
Singly Linked List: insert_end

- Assume the current list is (6, 10)
- Our goal is to add value 8
- First: let’s create a node with value 8 (item)



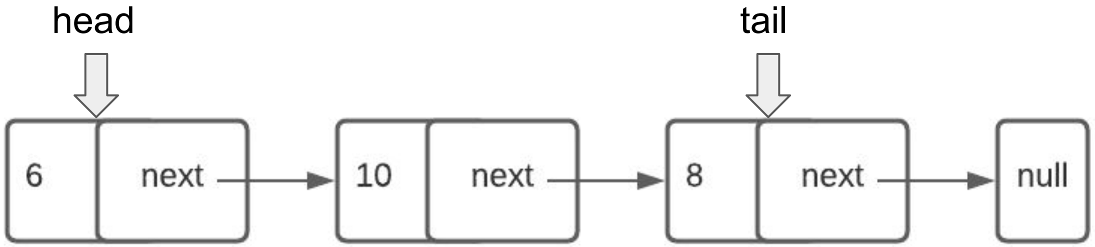
Singly Linked List: insert_end

- Assume the current list is (6, 10)
- Our goal is to add value 8
- Second, let's link our tail node (value 10) with the new node (8)
  - tail.next = item;



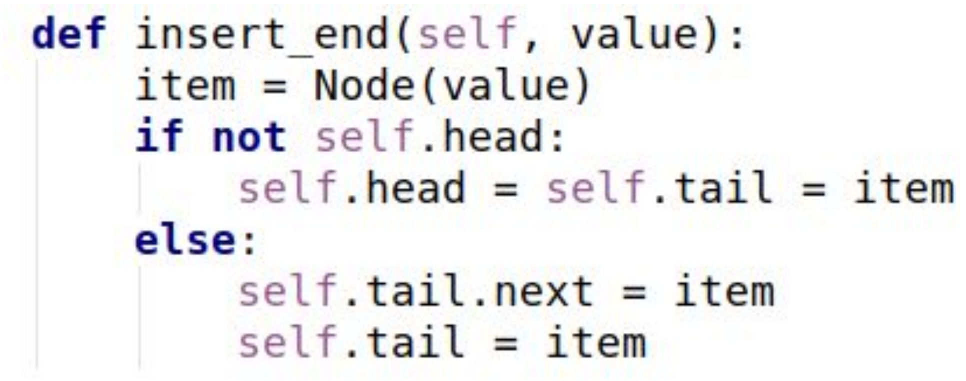
Singly Linked List: insert_end

- Assume the current list is (6, 10)
- Our goal is to add value 8
- Finally, update the tail to look to the new 'last' node
    - tail = item



Singly Linked List: insert_end

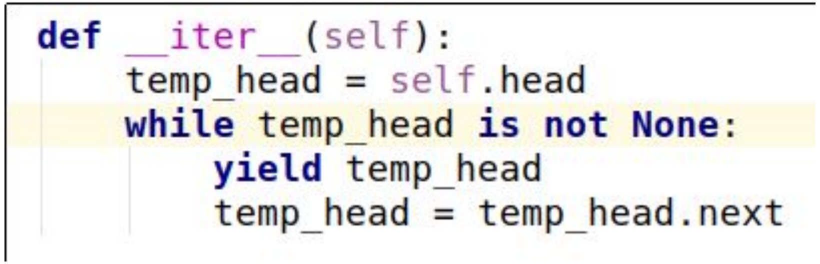
- Now, we can code this logic in this simple way!



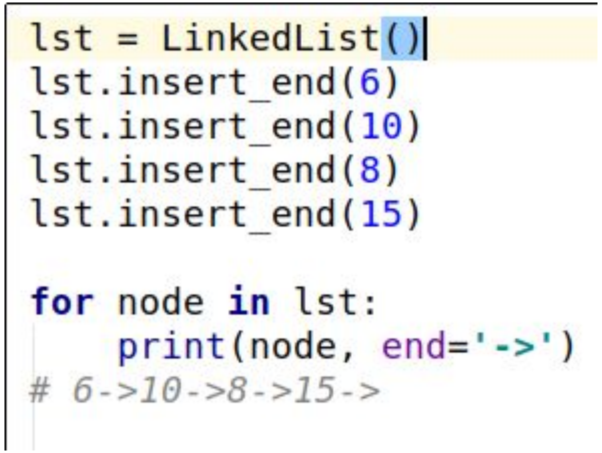
```python
def insert_end(self, value):
    item = Node(value)
    if not self.head:
        self.head = self.tail = item
    else:
        self.tail.next = item
        self.tail = item
```

Iterating with __iter__

- You can also support iter dunder to use the in operator
  - Skip if you don't know



```
def __iter__(self):
    temp_head = self.head
    while temp_head is not None:
        yield temp_head
        temp_head = temp_head.next
```



```
lst = LinkedList()
lst.insert_end(6)
lst.insert_end(10)
lst.insert_end(8)
lst.insert_end(15)

for node in lst:
    print(node, end='->')
# 6->10->8->15->
```

Easier coding

- Many courses exclude the tail node, using only the **head** node
  - With tail, some problems are shorter to code
  - Sometimes, it is a good challenge to try solving a problem without a tail
- Length variable
  - Introduce a length variable for convenience
  - Increment this length variable with each insertion - and
    decrement it with every deletion
  - It will make many coding validations much easier
  - Common mistake
    - Always remember to **maintain** the length variable;
      we'll be coding several insert/delete functions which will involve it

“Acquire knowledge and impart it to the people.”

“Seek knowledge from the Cradle to the Grave.”