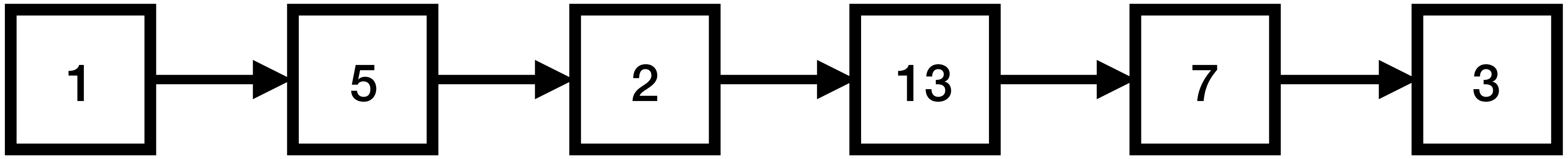
Linked List

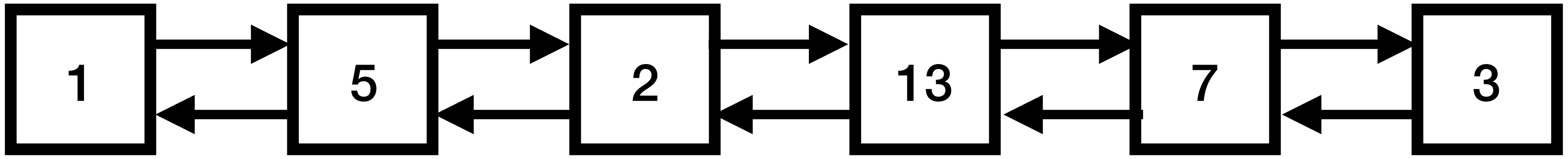
Jiao Li
08/19/2018

- A linked list is a data structure that represents a sequence of nodes.

- In a singly linked list, each node points to the next node in the linked list.



- A doubly linked list gives each nodes pointers to both the next node and the previous node.



- Unlike an array, a linked list does not provide constant time access to a particular “index” within the list. This means that if you’d like to find the Kth element in the list, you will need to iterate through K elements.

- The benefit of a linked list is that you can add and remove items in constant time.

Linked List vs Array

- Both Array and Linked List can be used to store linear data of similar types, but both have some advantages and disadvantages over each other.

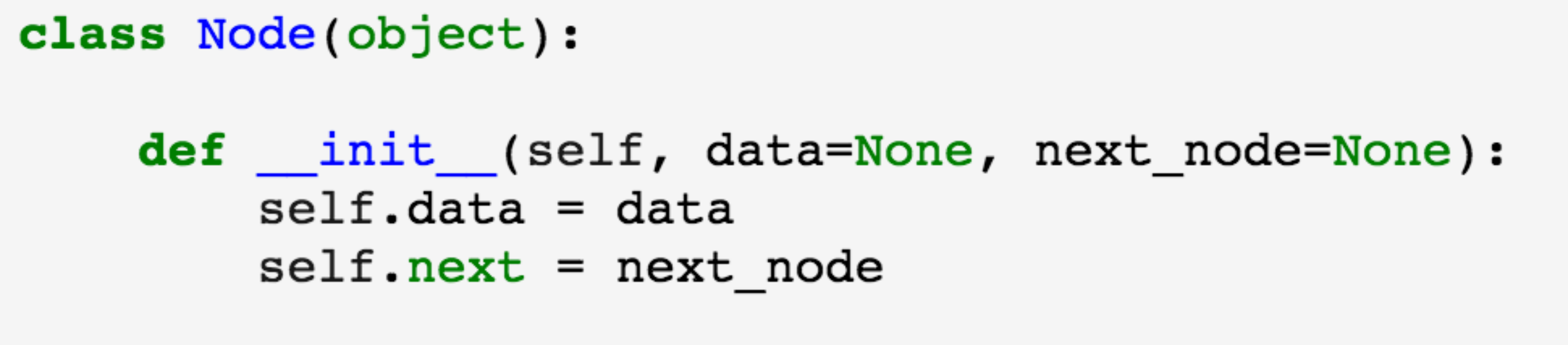
- Following are the points in favor of Linked Lists

    - The size of the arrays is fixed: So we must know the upper limit on the number of elements in advances.

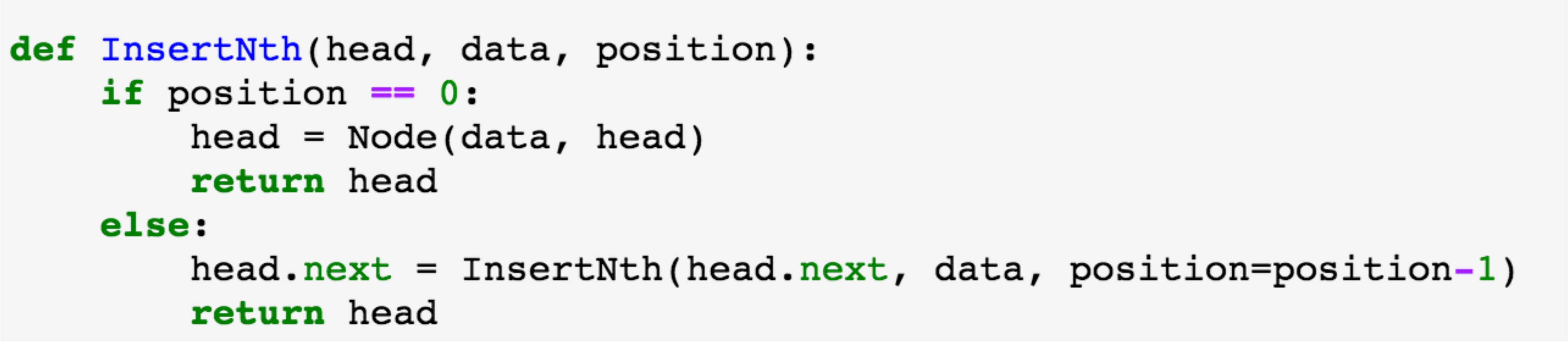
    - Inserting a new element in an array of elements is expensive, because the room has to be created for the new elements and to create room existing elements have to shifted.

- Advantages over arrays
  - Dynamics size
  - Ease of insertion/deletion
- Drawbacks:
  - Random access is not allowed. We have to access elements sequentially starting from the first node. So we cannot do binary search with linked lists.
  - Extra memory space for a pointer is required with each elements of the list.
  - Arrays have better cache locality that can make a pretty big difference in performance.

- Insert a node at a specific position in a linked list (Recursion)
    - E.g. if your list starts as 1->2->3 and you want to insert a node at position 2 with data “4”, you need list should be 1->2->4->3



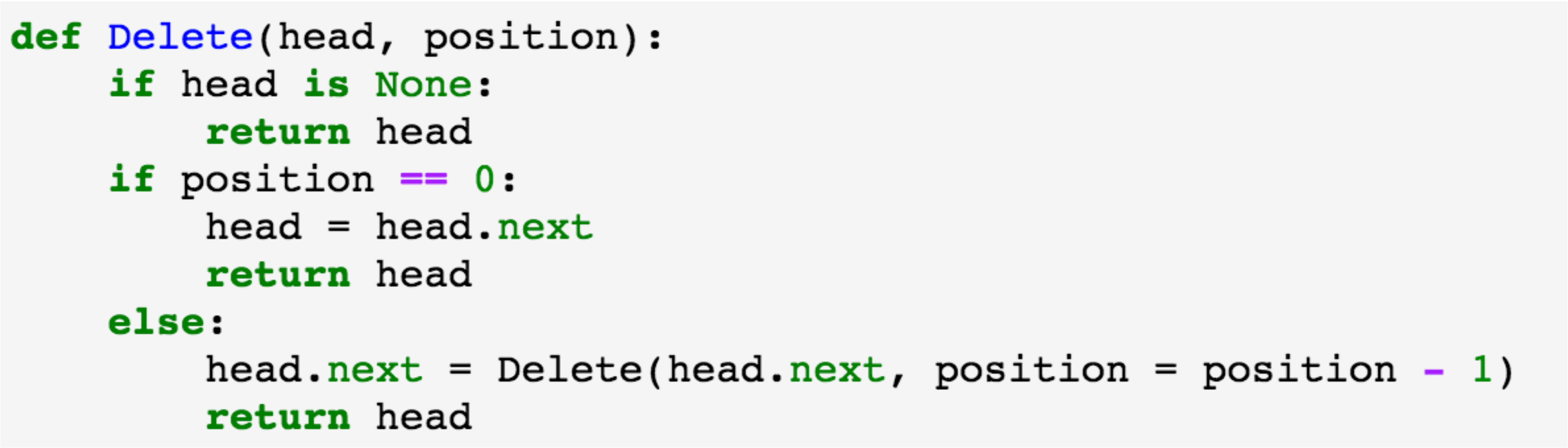
```
class Node(object):

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



```
def InsertNth(head, data, position):
    if position == 0:
        head = Node(data, head)
        return head
    else:
        head.next = InsertNth(head.next, data, position=position-1)
        return head
```

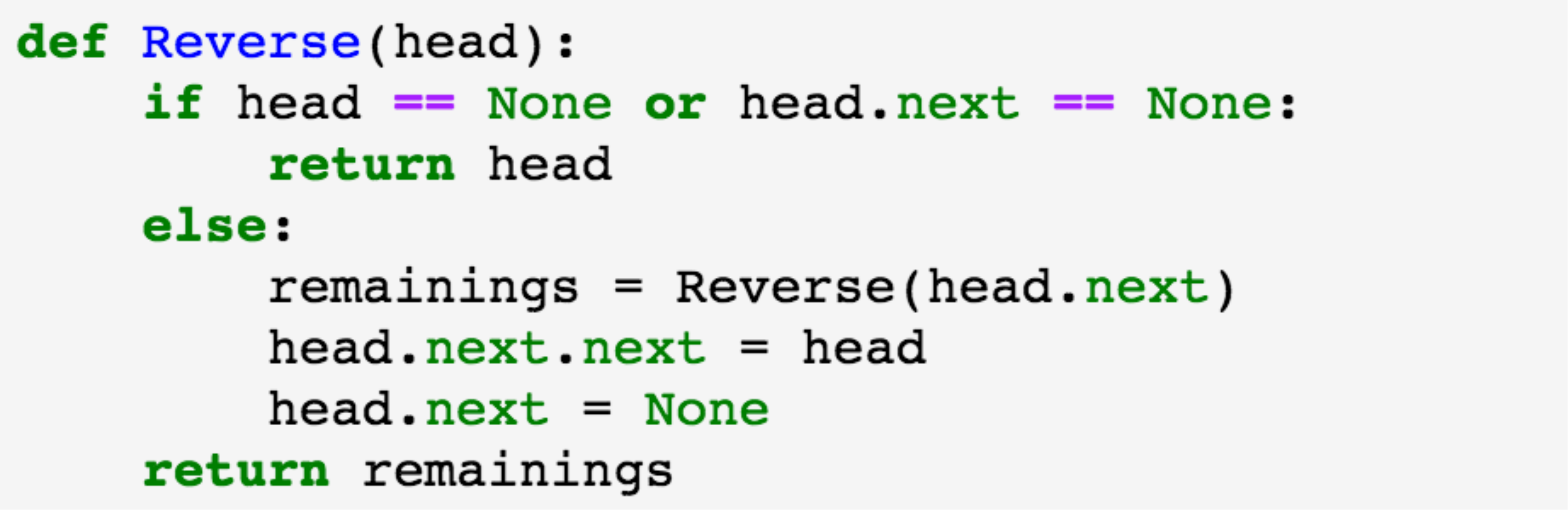
- Delete a Node

  - E.g. You're given the pointer to the head node of a linked list and the position of a node to delete. Delete the node at the given position and return the head node.



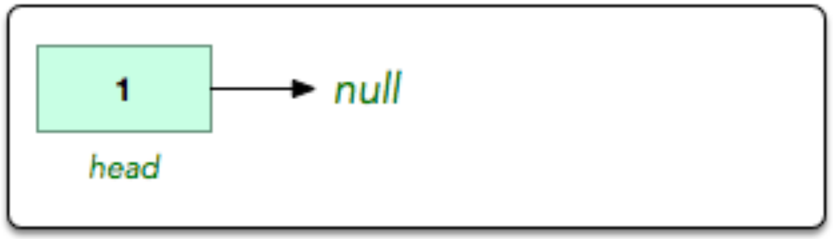
```python
def Delete(head, position):
    if head is None:
        return head
    if position == 0:
        head = head.next
        return head
    else:
        head.next = Delete(head.next, position = position - 1)
        return head
```

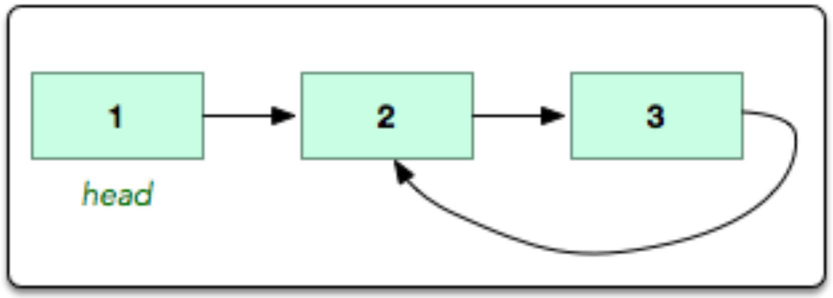
- Reverse a linked list
  - The initial linked list is: 1->2->3->4->5->Null
  - The reversed linked list is: 5->4->3->2->1->Null
  - Hint: assume 5->4->3->2<-1, we want 3's next node (2) to point to 3, so 3.next.next=3. And 5.next must be None.

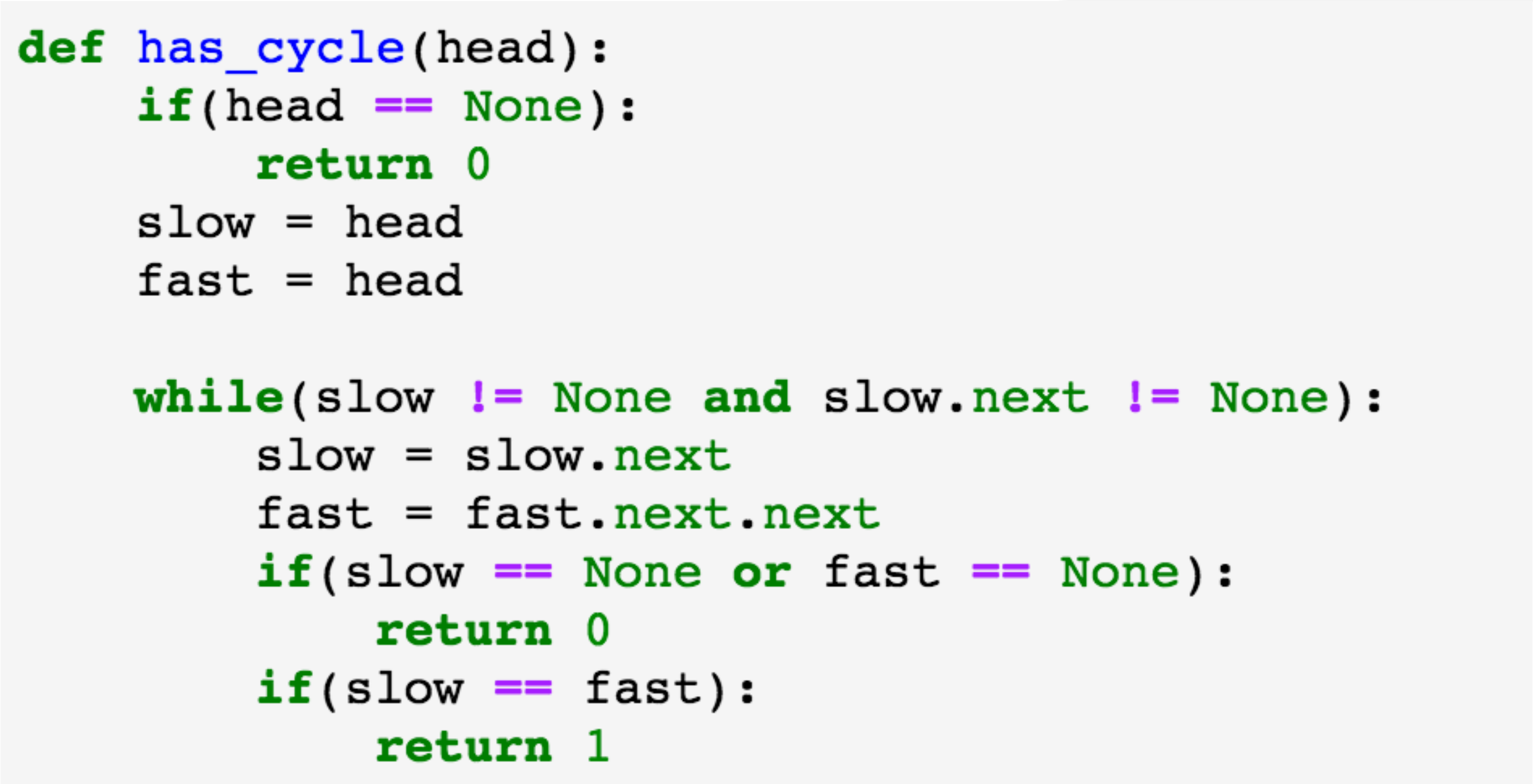


```
def Reverse(head):
    if head == None or head.next == None:
        return head
    else:
        remainings = Reverse(head.next)
        head.next.next = head
        head.next = None
    return remainings
```

- Cycle Detection (runner technique)

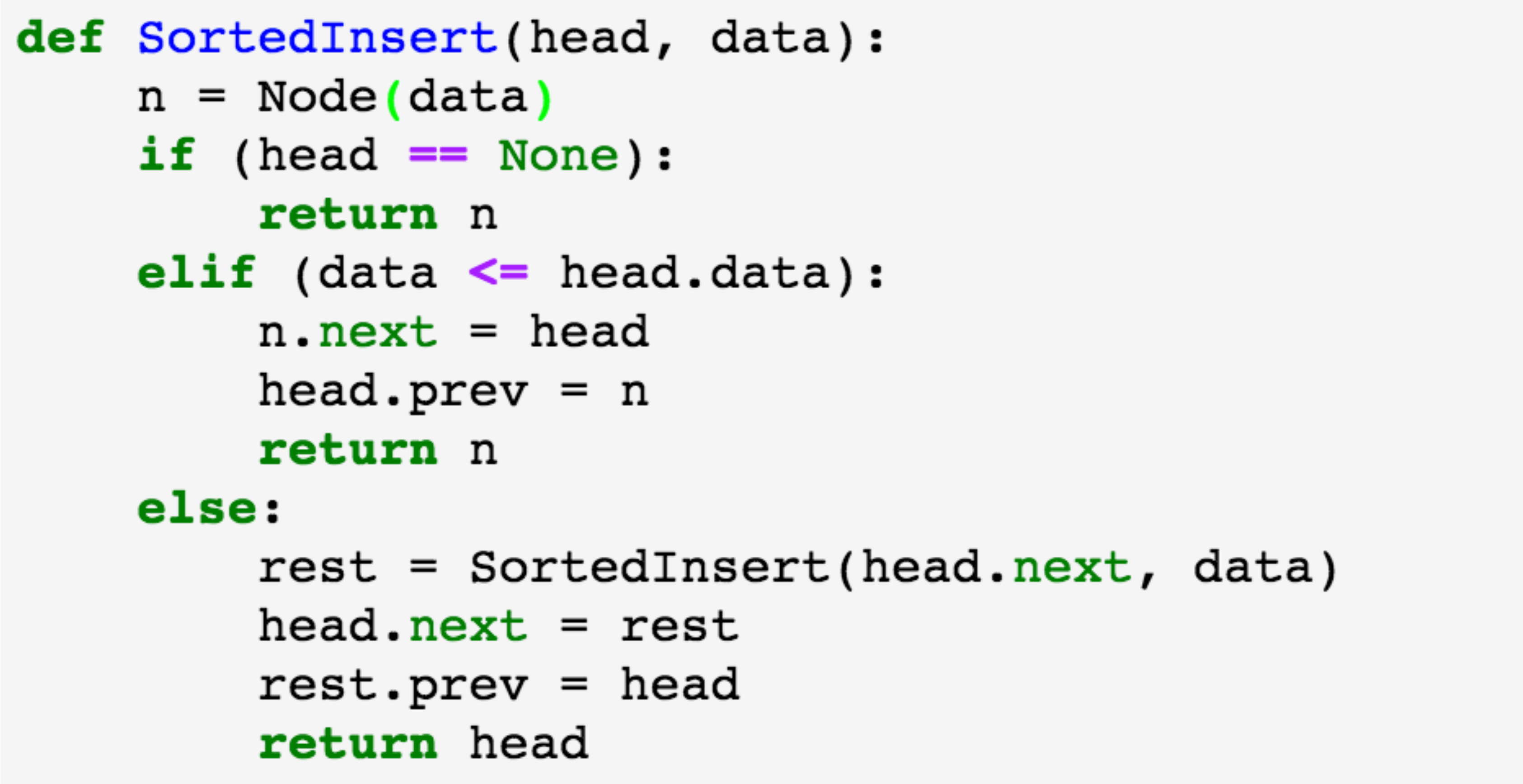






```
def has_cycle(head):
    if(head == None):
        return 0
    slow = head
    fast = head

    while(slow != None and slow.next != None):
        slow = slow.next
        fast = fast.next.next
        if(slow == None or fast == None):
            return 0
        if(slow == fast):
            return 1
```

- Inserting a Node into a Sorted Doubly Linked List
  - The initial doubly linked list is: 1<->3<->4<->10<->Null
  - The doubly linked list after insertion is: 1<->3<->4<->5<->10<->Null



```
def SortedInsert(head, data):
    n = Node(data)
    if (head == None):
        return n
    elif (data <= head.data):
        n.next = head
        head.prev = n
        return n
    else:
        rest = SortedInsert(head.next, data)
        head.next = rest
        rest.prev = head
        return head
```

- Pros/Cons
    - Inserts/Deletes - Constant Time
    - Random Assess -Linear Time
- Applications
    - Choice for underlying structure for other data structures (e.g. stack, queue, etc)
    - Anytime we need to have “fast” insertions and deletions, but random access less important

- Homework:
    - Remove Nth Node From End of List
    - Merge Two Sorted Lists
    - Remove Duplicates from Sorted List
    - Palindrome Linked List
    - Linked List Cycle
    - Linked List Cycle II
    - Reverse Linked List
    - Reverse Linked List II