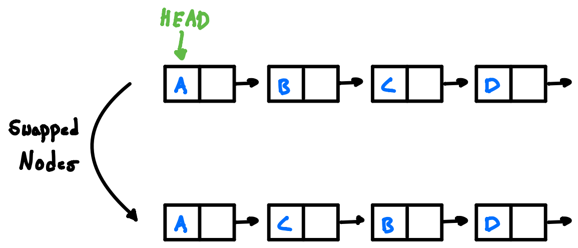
Singly Linked Lists- Node Swap

Node Swap. Two Cases: (Assume data entries are unique)

1. Node 1 and Node 2 are not head Nodes

2. Either Node 1 or Node 2 are head Nodes



Swap Nodes Function

```python
def swap_nodes(self, key_1, key_2):

    if key_1 == key_2:
        return
```

if Given same node to swap, **return**

```python
    previous_1 = None
    current_1 = self.head

    while current_1  and current_1 .data != key_1:
        previous_1 = current_1
        current_1  = current_1.next
```

Search for Node 1

loop through linked list while keeping track of the current node and previous node.

```python
    previous_2 = None
    current_2 = self.head

    while current_2 and current_2.data != key_2:
        previous_2 = current_2
        current_2 = current_2.next
```

For both nodes we like to swap

Search for Node 2

```python
    if not current_1 or not current_2:
        return
```

node doesnt exist

```python
    if  previous_1:
         previous_1.next = current_2
    else:
        self.head = current_2

    if previous_2:
       previous_2.next = current_1
    else:
        self.head = current_1

    current_1.next, current_2.next = current_2.next, current_1.next
```

On next Page

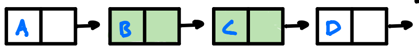
Case 1: Node 1 and Node 2 are not head Nodes

```
① if previous_1:
       previous_1.next = current_2
   else:
       self.head = current_2

② if previous_2:
       previous_2.next = current_1
   else:
       self.head = current_1

③ current_1.next,current_2.next = current_2.next,  current_1.next
```

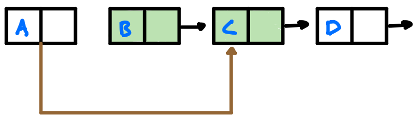
We check if the previous node has a node. If True

If it has a node it essentially telling us it's not a head node.

Were trying to swap node B and C



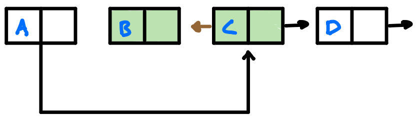
①



previous_1.next = Current_2 ~ Change Pointer A points to C

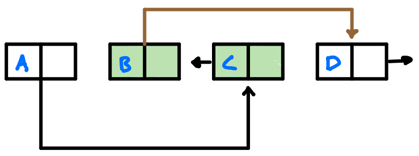
★ .next : Where node is pointing to ★

②



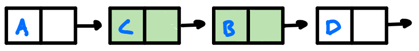
previous_2.next = Current_1 ~ Change Pointer, C points to B

③



~ Swapping the next nodes of node 1 and node 2 (B and C).

Current_1.next, Current_2.next = Current_2.next, Current_1.next



Can also be Coded like this:

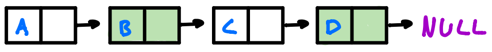
```
temp = current_1.next
current_1.next = current_2.next
current_2.next = temp
```

Swap Nodes B and D
(another case 1 example)

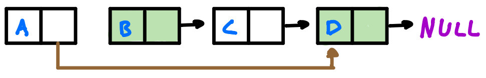
```
1  if previous_1:
       previous_1.next = current_2
   else:
       self.head = current_2

2  if previous_2:
       previous_2.next = current_1
   else:
       self.head = current_1

3  current_1.next,current_2.next = current_2.next,  current_1.next
```

We

While loop kept track of our previous and current node

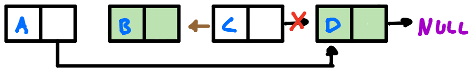


1



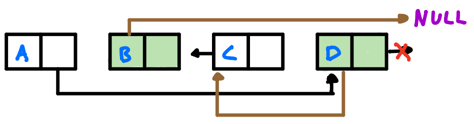
previous_1.next = current_2

2



previous_2.next = current_1

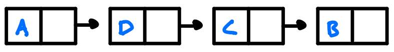
3



current_1.next, current_2.next = current_2.next, current_1.next

$B \longrightarrow$ NULL

$D \rightarrow C$

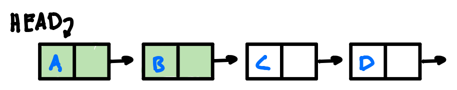


- Swapped

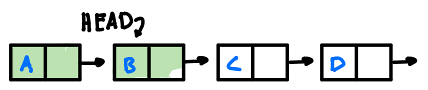
Case 2: Either Node 1 or Node 2 are head Nodes

1
```
if  previous_1:
     previous_1.next = current_2
else:
    self.head = current_2

if previous_2:
    previous_2.next = current_1
else:
    self.head = current_1

current_1.next,current_2.next = current_2.next,  current_1.next
```

Variable Explorer: prev_1.data = none, Curr_1.data = A , prev_2.data = A , Curr_2.data = B

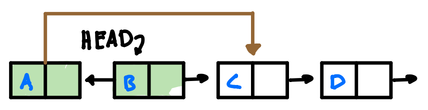


1



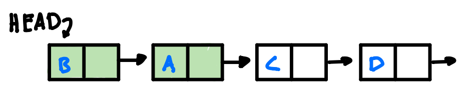
Self.head = Current_2

Self.head = B



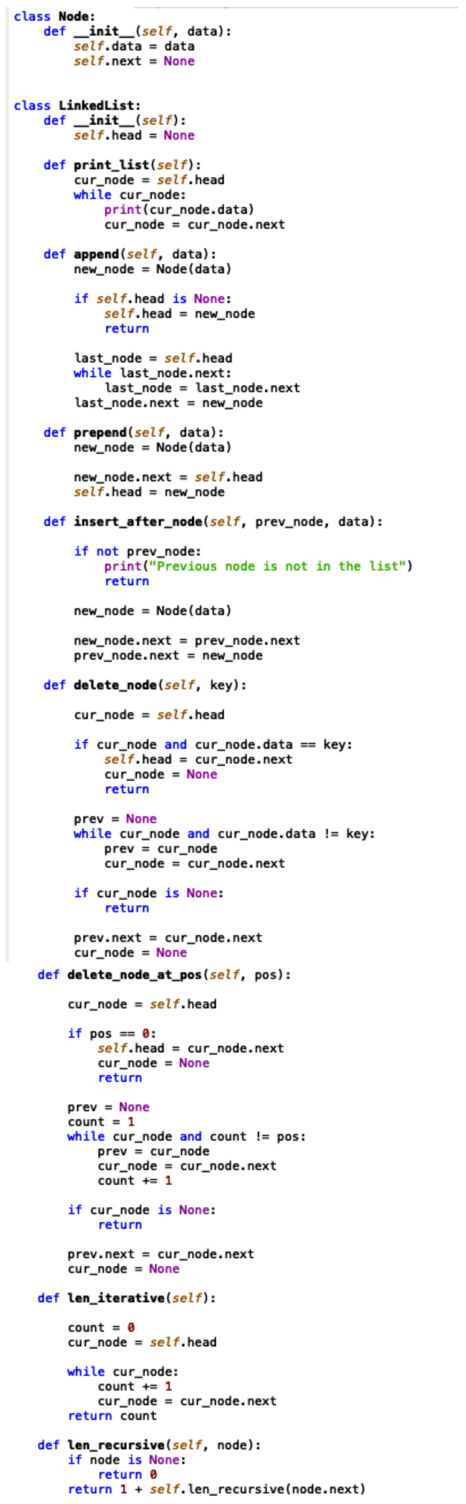
prev_2.next = curr_1
prev_2.next = A

Curr_1.next, curr_2.next = Curr_2.next, curr_1.next

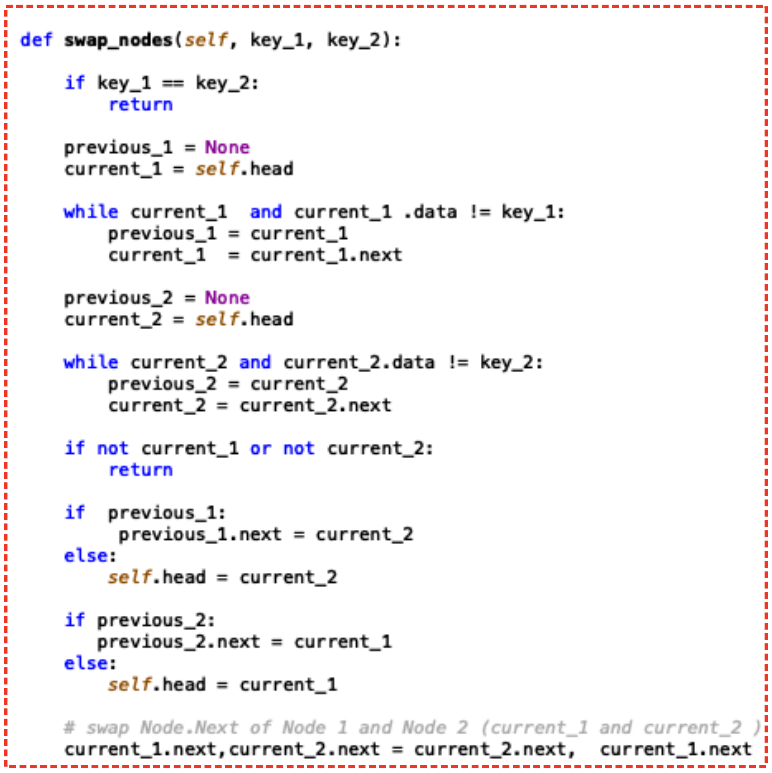


Swapped ↺

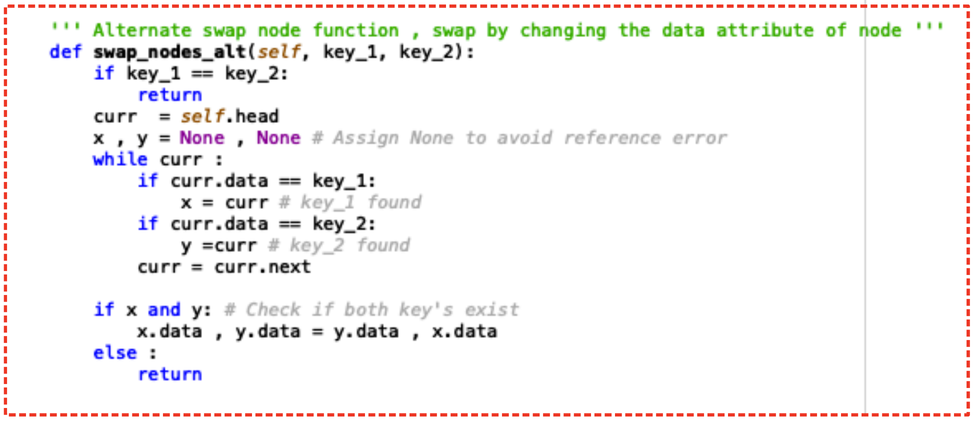
Full Code:



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


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

    def print_list(self):
        cur_node = self.head
        while cur_node:
            print(cur_node.data)
            cur_node = cur_node.next

    def append(self, data):
        new_node = Node(data)

        if self.head is None:
            self.head = new_node
            return

        last_node = self.head
        while last_node.next:
            last_node = last_node.next
        last_node.next = new_node

    def prepend(self, data):
        new_node = Node(data)

        new_node.next = self.head
        self.head = new_node

    def insert_after_node(self, prev_node, data):

        if not prev_node:
            print("Previous node is not in the list")
            return

        new_node = Node(data)

        new_node.next = prev_node.next
        prev_node.next = new_node

    def delete_node(self, key):

        cur_node = self.head

        if cur_node and cur_node.data == key:
            self.head = cur_node.next
            cur_node = None
            return

        prev = None
        while cur_node and cur_node.data != key:
            prev = cur_node
            cur_node = cur_node.next

        if cur_node is None:
            return

        prev.next = cur_node.next
        cur_node = None
    def delete_node_at_pos(self, pos):

        cur_node = self.head

        if pos == 0:
            self.head = cur_node.next
            cur_node = None
            return

        prev = None
        count = 1
        while cur_node and count != pos:
            prev = cur_node
            cur_node = cur_node.next
            count += 1

        if cur_node is None:
            return

        prev.next = cur_node.next
        cur_node = None

    def len_iterative(self):

        count = 0
        cur_node = self.head

        while cur_node:
            count += 1
            cur_node = cur_node.next
        return count

    def len_recursive(self, node):
        if node is None:
            return 0
        return 1 + self.len_recursive(node.next)
```



```python
def swap_nodes(self, key_1, key_2):

    if key_1 == key_2:
        return

    previous_1 = None
    current_1 = self.head

    while current_1  and current_1 .data != key_1:
        previous_1 = current_1
        current_1  = current_1.next

    previous_2 = None
    current_2 = self.head

    while current_2 and current_2.data != key_2:
        previous_2 = current_2
        current_2 = current_2.next

    if not current_1 or not current_2:
        return

    if  previous_1:
         previous_1.next = current_2
    else:
        self.head = current_2

    if previous_2:
       previous_2.next = current_1
    else:
        self.head = current_1

    # swap Node.Next of Node 1 and Node 2 (current_1 and current_2 )
    current_1.next,current_2.next = current_2.next,  current_1.next
```



```python
''' Alternate swap node function , swap by changing the data attribute of node '''
def swap_nodes_alt(self, key_1, key_2):
    if key_1 == key_2:
        return
    curr  = self.head
    x , y = None , None # Assign None to avoid reference error
    while curr :
        if curr.data == key_1:
            x = curr # key_1 found
        if curr.data == key_2:
            y =curr # key_2 found
        curr = curr.next

    if x and y: # Check if both key's exist
        x.data , y.data = y.data , x.data
    else :
        return
```