Linked_List

February 23, 2021

0.0.1 Class for a Single Node

[1]:
```
class Node:
    def __init__(self, data):# constructor
        self.item = data # data sub part
        self.ref = None  # ref - next subpart
```

[2]:
```
Node(10)
```

[2]:
```
<__main__.Node at 0x25dd9ae2b38>
```

0.0.2 Class for Linked List

Interfaces

- Insert at start
- Insert at end
- Insert at given index
- Traverse
- Get count
- Delete at start
- Delete at end

[11]:
```
class LinkedList:
    def __init__(self):
        self.start_node = None # head variable



    def insert_at_start(self, data): # interface for adding data to LL
        new_node = Node(data)  # node are entity ..create a node ..
        new_node.ref = self.start_node  # copy address of current head/first␣
↪node to new node ref
        self.start_node= new_node  # makes the newly created node as head...


    def insert_at_end(self, data):# interface to add data to last of LL
        new_node = Node(data) # create a node
```

```
        if self.start_node is None: # checking if the ll is empty
            self.start_node = new_node
            return
        n = self.start_node # temp variable to hold the starting address
        while n.ref is not None: #  this check if the current node is last node␣
↪or not
            n= n.ref
        n.ref = new_node;


    def traverse_list(self):
        if self.start_node is None:# list is empty or not
            print("List has no element")
            return
        else:
            n = self.start_node #temp var
            while n is not None:
                print(n.item , " ")
                n = n.ref



    def get_count(self):
        if self.start_node is None:
            return 0;
        n = self.start_node
        count = 0;
        while n is not None:
            count = count + 1
            n = n.ref
        return count




    def delete_at_start(self):
        if self.start_node is None:# ll isempty or not
            print("The list has no element to delete")
            return
        self.start_node = self.start_node.ref


    def delete_at_end(self): # delete at end
        if self.start_node is None: # LL is empty or not
            print("The list has no element to delete")
            return
```

```
        n = self.start_node # same logic as traversal
        while n.ref.ref is not None: # you need to stop one node before the␣
↪last node. ie y n.ref.ref is used
            n = n.ref
        n.ref = None



    ### additional interfaces

    def insert_at_index (self, index, data):
        if index == 1:
            new_node = Node(data)
            new_node.ref = self.start_node
            self.start_node = new_node

        i = 1
        n = self.start_node
        while i < index-1 and n is not None:
            n = n.ref
            i = i+1
        if n is None:
            print("Index out of bound")
        else:
            new_node = Node(data)
            new_node.ref = n.ref
            n.ref = new_node


    def reverse_linkedlist(self):
        pass

    def search_item(self, x):
        pass

    def insert_after_item(self, x, data):
        pass

    def insert_before_item(self, x, data):
        pass


    def delete_element_by_value(self, x):
        pass
```

[12]:
```
new_linked_list = LinkedList()
```

[13]:
```
new_linked_list.insert_at_end(929)
new_linked_list.insert_at_end(100)
new_linked_list.insert_at_end(1800)
```

[18]:
```
new_linked_list.traverse_list()
```

```
201
929
80
100
1800
12
```

[15]:
```
new_linked_list.insert_at_end(12)
```

[16]:
```
new_linked_list.insert_at_start(201)
```

[17]:
```
new_linked_list.insert_at_index(3,80)
```

[19]:
```
new_linked_list.get_count()
```

[19]: 6

[ ]:
```

```