Linked Lists

Thursday, 7 April 2022 23:05

Now we going to build our own data structures (advanced). We knew list, tuple, etc.

Linked Lists → "A sequence of things to do"
or Linear Collection of Data structures

Node
- A class that is an element of linked lists
- A node contains { a link to the next node
  some unit of data : cargo (int, str, etc.)
- Beginning of linked list is called head, the end is called tail
- Last node is None & does not provide a link to any other nodes

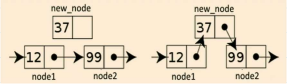
Why linked list ?
- Dynamically shrink or grow at run-time
- Faster insertion & deletion (no need to shift if we remove in the middle)
- Efficient memory management
- Implementation of data structures (queues and stacks)

but
- More memory for 1 element
- ☆ Random access : because storage not continuous, we must traverse through all nodes to access content at node X (not indexing like lists)
- No easy way to reverse traversal (one-way arrow. If ⟷, more memory)

- Remove a node from the beginning

⇒ "Point" the head to the second element ⇒ no shifting

- Insertion ⇒ pointing, no need to shift.



- Deletion ⇒ changing the pointing

The Node Class

Thursday, 7 April 2022 23:42

```
class Node:
    def __init__(self, cargo = None, next = None):
        self.cargo = cargo
        self.next = next
    def __str__(self):
        return str(self.cargo)
```

- Creating linked lists

+) Instantiating cargos for nodes (overwrite cargo)

+) Linking nodes together node1. next = node2
node2. next = node3

- Iterating through linked list (Traversing)

→ Take advantage of None ⇒ while loop !

```
head = node1
while head:
    print (head)
    head = head.next
```

← or while head != None

- If we start at node2, it will do the same but 2 onwards

Infinite list

+) When a node points to itself / previous nodes
⇒ Run forever

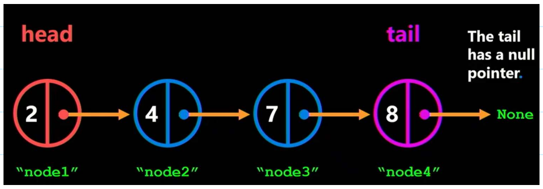
Modifying list

+) Modifying cargo ⇒ node1. cargo = new_cargo

+) Remove nodes ⇒ Simply changing previous_node . next = after_node

node_to_remove . next

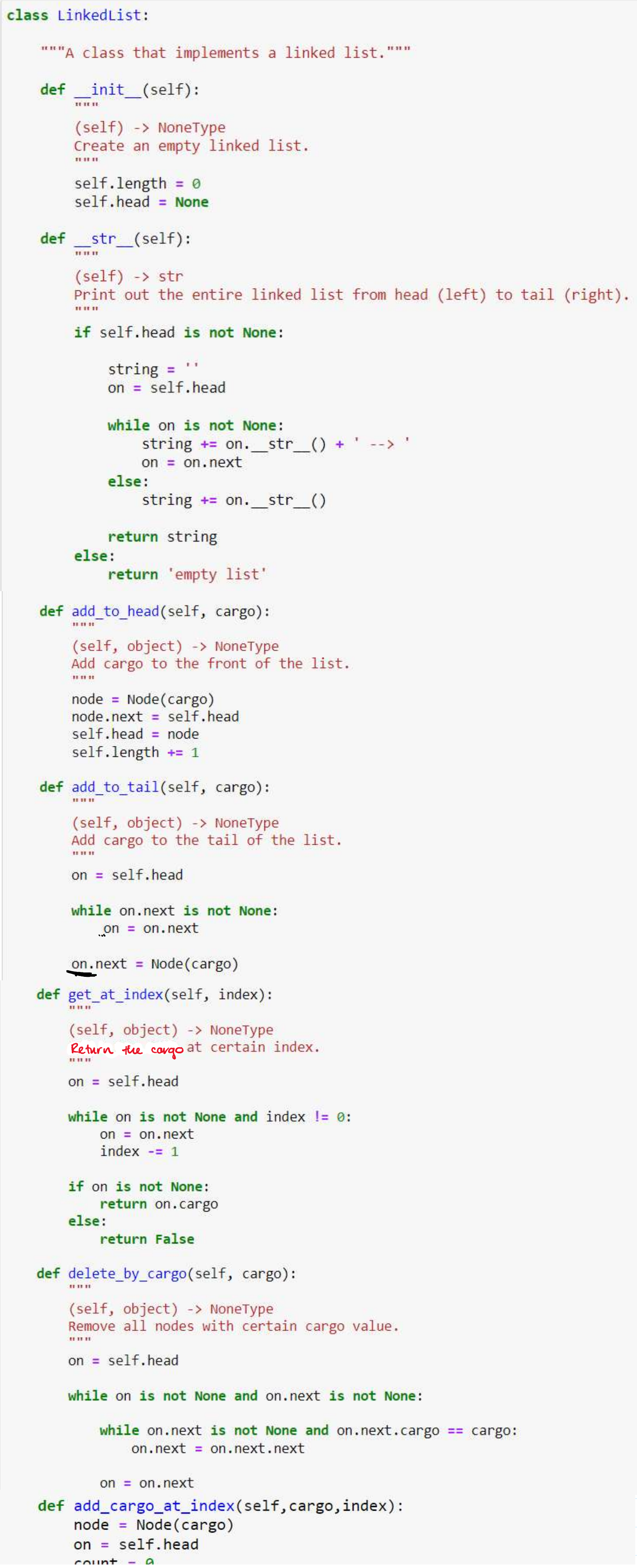
+) Adding nodes ⇒ Point last node to a new object
node4. next = Node (3) → node5


head
tail
The tail has a null pointer.
2
4
7
8
None
"node1"
"node2"
"node3"
"node4"

The LinkedList Class

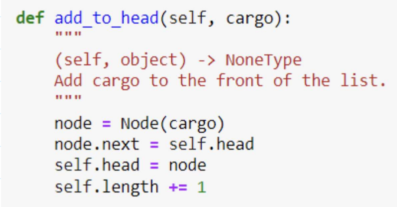
Friday, 8 April 2022 14:05

Is a new class that helps us keep track of all Nodes



```python
class LinkedList:

    """A class that implements a linked list."""

    def __init__(self):
        """
        (self) -> NoneType
        Create an empty linked list.
        """
        self.length = 0
        self.head = None

    def __str__(self):
        """
        (self) -> str
        Print out the entire linked list from head (left) to tail (right).
        """
        if self.head is not None:

            string = ''
            on = self.head

            while on is not None:
                string += on.__str__() + ' --> '
                on = on.next
            else:
                string += on.__str__()

            return string
        else:
            return 'empty list'

    def add_to_head(self, cargo):
        """
        (self, object) -> NoneType
        Add cargo to the front of the list.
        """
        node = Node(cargo)
        node.next = self.head
        self.head = node
        self.length += 1

    def add_to_tail(self, cargo):
        """
        (self, object) -> NoneType
        Add cargo to the tail of the list.
        """
        on = self.head

        while on.next is not None:
            on = on.next

        on.next = Node(cargo)
    def get_at_index(self, index):
        """
        (self, object) -> NoneType
        Return the cargo at certain index.
        """
        on = self.head

        while on is not None and index != 0:
            on = on.next
            index -= 1

        if on is not None:
            return on.cargo
        else:
            return False

    def delete_by_cargo(self, cargo):
        """
        (self, object) -> NoneType
        Remove all nodes with certain cargo value.
        """
        on = self.head

        while on is not None and on.next is not None:

            while on.next is not None and on.next.cargo == cargo:
                on.next = on.next.next

            on = on.next
    def add_cargo_at_index(self,cargo,index):
        node = Node(cargo)
        on = self.head
        count = 0
```

1) add_to_head [linked_list . add_ to _ head (cargo)]

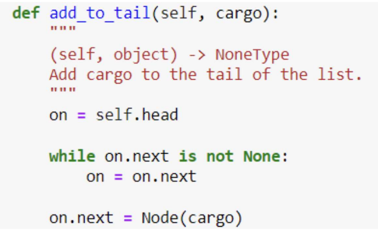


```python
def add_to_head(self, cargo):
    """
    (self, object) -> NoneType
    Add cargo to the front of the list.
    """
    node = Node(cargo)
    node.next = self.head
    self.head = node
    self.length += 1
```

- Create new node
- Point node to current head
- Change self.head to new node
- Add length

2) add_to_tail [linked_list . add_ to_tail (cargo)]

Since we don't know the tail, we have to travese the list ⇒ create variable on to travel .



```python
def add_to_tail(self, cargo):
    """
    (self, object) -> NoneType
    Add cargo to the tail of the list.
    """
    on = self.head

    while on.next is not None:
        on = on.next

    on.next = Node(cargo)
```

- Set on to start at head
- While loop to find tail. We know it's tail when on.next is None
- When we find tail, connect on to new node

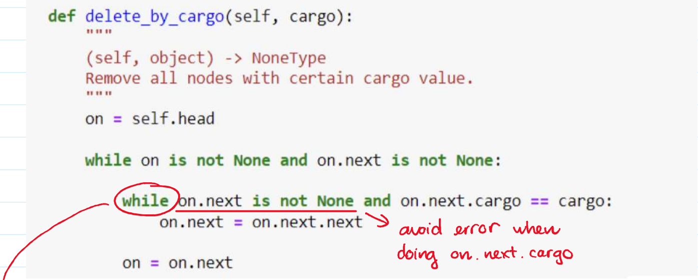
NOTE When we are doing on = node1 , it is aliasing
⇒ If we do on.next = Node(3) , we are changing the next attribute of node 1 ⇒ connect
- Inside while loop, we are doing on = on.next
⇒ Assigning node2 (current on.next) to update variable on

3) get_at_index [linked_list. get_at_index (index)] :

```
def get_at_index (self, index)
on = self. head
count = 0
while count != index or on is not None :
    on = on . next
    count += 1
if on is not None :
    return on . cargo
else :
    return False
```

⇒ also have to traverse

4) delete_by_cargo [linked_list . delete_by_cargo (cargo)] :

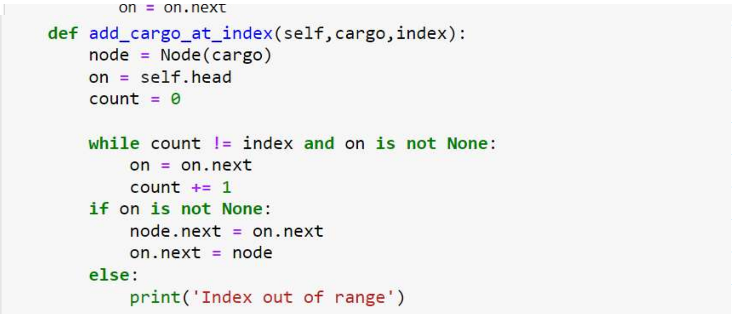


```python
def delete_by_cargo(self, cargo):
    """
    (self, object) -> NoneType
    Remove all nodes with certain cargo value.
    """
    on = self.head

    while on is not None and on.next is not None:

        while on.next is not None and on.next.cargo == cargo:
            on.next = on.next.next

        on = on.next
```

→ avoid error when doing on.next.cargo

- Why do we need both on & on.next is not None ?

We need a while loop because on.next.next might also contain



```
            on = on.next
    def add_cargo_at_index(self,cargo,index):
        node = Node(cargo)
        on = self.head
        count = 0

        while count != index and on is not None:
            on = on.next
            count += 1
        if on is not None:
            node.next = on.next
            on.next = node
        else:
            print('Index out of range')
```

- Why do we need both on & on.next is not None?

We need a while loop because on.next.next might also contain the cargo we want to delete

5) add_cargo_at_index (add after the index)

- Use on, find the index
- link node.next = on.next, then on.next = node