# Ch20-LinkedLists

September 10, 2025

## 1 Linked Lists

http://openbookproject.net/thinkcs/python/english3e/linked_lists.html ## forward list - made up of nodes linked to each other such that a not contains a reference to the next node in the list - first node is often called head and last node is also called tail - each node also contains a data field also called cargo - recursive definition: - a linked list is either: 1. the empty list, represented by None, or 2. a node that contains a cargo object and a reference to a linked list - this definition of linked-list is also called forward list or singly linked list as opposed to doubly linked list - drawbacks: - can't directly access nodes by their position as opposed to built-in data structure list - consume some extra memory to keep the linking information associated to each element (may be an important factor for large lists of small-sized elements)

### 1.1 The Node class

[1]:
```
class Node:
    def __init__(self, cargo=None, next=None):
        self.cargo = cargo
        self.next = next

    def __str__(self):
        return str(self.cargo)
```

[2]:
```
node = Node("test")
print(node)
```

```
test
```

[3]:
```
node1 = Node(1)
node2 = Node(2)
node3 = Node(3)
```

### 1.2 visualize with Pythontutor.com

https://goo.gl/u1vePS

[11]:
```
from IPython.display import IFrame
src = """
```

```
http://pythontutor.com/iframe-embed.
  ↪html#code=class%20Node%3A%0A%20%20%20%20def%20__init__%28self,%20cargo%3DNone,%20next%3DNon
  ↪cargo%20%3D%20cargo%0A%20%20%20%20%20%20%20%20self.
  ↪next%20%3D%20next%0A%20%20%20%20%20%20%20%20%0A%20%20%20%20def%20__str__%28self%29%3A%0A%20
  ↪cargo%29%0A%20%20%20%20%20%20%20%20%0Anode1%20%3D%20Node%281%29%0Anode2%20%3D%20Node%282%29
  ↪js&py=3&rawInputLstJSON=%5B%5D&textReferences=false
"""
IFrame(src, width=900, height=600)
```

[11]: `<IPython.lib.display.IFrame at 0x1049da908>`

### 1.2.1 link the nodes to form linked list

[6]:
```
node1.next = node2
node2.next = node3
```

### 1.2.2 visualize the linked-list with pythontutor

https://goo.gl/tNhz4a

[34]:
```
from IPython.display import IFrame
src = """
http://pythontutor.com/iframe-embed.
  ↪html#code=class%20Node%3A%0A%20%20%20%20def%20__init__%28self,%20cargo%3DNone,%20next%3DNon
  ↪cargo%20%3D%20cargo%0A%20%20%20%20%20%20%20%20self.
  ↪next%20%3D%20next%0A%20%20%20%20%20%20%20%20%0A%20%20%20%20def%20__str__%28self%29%3A%0A%20
  ↪cargo%29%0A%20%20%20%20%20%20%20%20%0Anode1%20%3D%20Node%281%29%0Anode2%20%3D%20Node%282%29
  ↪next%20%3D%20node2%0Anode2.
  ↪next%20%3D%20node3&codeDivHeight=400&codeDivWidth=350&cumulative=false&curInstr=16&heapPrim
  ↪js&py=3&rawInputLstJSON=%5B%5D&textReferences=false
"""
IFrame(src, width=900, height=700)
```

[34]: `<IPython.lib.display.IFrame at 0x1049daac8>`

### 1.2.3 list as collection

- first node, node1 of the list serves as a reference to the entire list; also called root/first node
- to pass list as a parameter, we only have to pass a reference to the first node

[7]:
```
def print_list(node):
    while node is not None:
        print(node, end=" ")
        node = node.next
    print()
```

[8]:
```
print_list(node1)
```

```
1 2 3
```

### 1.3 Lists and recursion

- because of recursive definition, it naturally lends to many recursive operations
- print list backward
    1. General case:
        1. separate the lists into two pieces: the first node (head) and the rest (tail)
        2. recursive print the tail
        3. print the head

[31]:
```python
def print_backward(alist):
    if alist is None:
        return
    head = alist
    tail = alist.next
    print_backward(tail)
    print(head, end=" ")
```

[32]:
```python
print_backward(node1)
```

```
3 1
```

### 1.4 Modifying lists

- change cargo/data of a node
- add a new node
- delete an existing node
- re-order nodes

[17]:
```python
# function that removes the second node in the list and returns reference to␣
 ↪the removed node
def remove_second(alist):
    if alist is None: return
    first = alist
    second = alist.next
    # Make the first node point to the third
    first.next = second.next
    # Separate the second node from the rest of the list
    second.next = None
    return second
```

[18]:
```python
print_list(node1)
```

```
1 2 3
```

[19]:
```python
removed = remove_second(node1)
```

[20]:
```python
print(removed)
```

```
2
```

[21]:
```
print_list(node1)
```

```
1 3
```

## 1.5 wrappers and helpers

[27]:
```
def print_backward_nicely(alist):
    print("[", end=" ")
    print_backward(alist)
    print("]")
```

[33]:
```
print_backward_nicely(node1)
```

```
[ 3 1 ]
```

## 1.6 LinkedList container class

### 1.6.1 better approach

- define LinkeList class that keeps track of all the meta-data and methods to work with linked lists such as traversing and printing, adding, deleting, etc.

[88]:
```
class LinkedList(object):
    def __init__(self):
        self.length = 0
        self.head = None
        self.tail = None

    def append(self, data):
        node = Node(data)
        if not self.head: # empty linked list
            # make the first and last point to the new node
            self.head = node
            self.tail = node
        else:
            self.tail.next = node # make the current tail's next node point to␣
 ↪the new node
            self.tail = node # node is the last node
        self.length += 1

    def __str__(self):
        """
        traverse linked list and return all the cargos/data
        """
        llist = list()
        current = self.head
        while current:
            llist.append(current.cargo)
            current = current.next
```

```python
        return str(llist)

    def print(self):
        current = self.head
        while current is not None:
            print(current, end=" ")
            current = current.next
        print()

    def print_reverse(self):
        current = self.head
        if not current:
            return
        print_backward(current.next)
        print(current, end=" ")

    def find(self, data):
        # find and return the node with given data
        current = self.head
        found = False
        while (current and not found):
            if current.cargo == data:
                found = True
            else:
                current = current.next
        return current


    def remove(self, data):
        """
        We need to consider several cases:
        Case 1: the list is empty - do nothing
        Case 2: The first node is the node with the given cargo/data, we need␣
↪to adjust head and may be tail
        Case 3: The node with the given info is somewhere in the list.
            i. find the node and delete
            ii. If the node to be deleted is the tail,
                we must adjust tail.
        Case 4: The list doesn't contain a node with the given info - do nothing
        """
        # case 1
        if not self.head:
            return # done
        # case 2
        if self.head.cargo == data:
            self.head = self.head.next # 2nd node becomes the head
            # if list becomes empty; update tail as well
            if not self.head:
                self.tail = None
            self.length -= 1
        else:
            # search the list for the node with given data
            found = False
            trailCurrent = self.head # first node
            current = self.head.next # second node
            while(current and not found):
                if current.cargo == data:
                    found = True
                else:
                    trailCurrent = current
                    current = current.next
            if found: #case 3
                trailCurrent.next = current.next
                if self.tail is current:
                    self.tail = trailCurrent
                self.length -= 1
            else: # case 4
                return

    def clear(self):
        self.length = 0
        self.head = None
        self.tail = None

    def __len__(self):
        return self.length
```

[89]:
```
alist = LinkedList()
alist.append(10)
alist.append(5)
alist.append(15)
alist.append('a')
alist.append('ball')
print(alist, len(alist))
```

```
[10, 5, 15, 'a', 'ball'] 5
```

[90]:
```
alist.remove(15)
print(alist)
print(len(alist))
```

```
[10, 5, 'a', 'ball']
4
```

[91]:
```
alist.remove(10)
print(alist)
```
```
[5, 'a', 'ball']
```
[92]:
```
alist.remove('ball')
print(alist)
```
```
[5, 'a']
```
[93]:
```
alist.append('cat')
print(alist)
assert len(alist) == alist.length
```
```
[5, 'a', 'cat']
```
[94]:
```
print(alist.length)
```
```
3
```
[95]:
```
alist.print_reverse()
```
```
cat a 5
```
[96]:
```
alist.print()
```
```
5 a cat
```
[97]:
```
node = alist.find('cat')
```
[98]:
```
print(node)
```
```
cat
```
[99]:
```
node.cargo = 'Cat'
```
[100]:
```
alist.print()
```
```
5 a Cat
```
[102]:
```
alist.remove('dog')
```
[103]:
```
alist.print()
```
```
5 a Cat
```

## 1.7 exercises

1. By convention, lists are often printed in brackets with commas between the elements, as in [1, 2, 3]. Modify print_list function so that it generates output in this format.
2. By convention, lists are often printed in brackets with commas between the elements, as in [1, 2, 3]. Modify print method of LinkeList class so that it generates output in this format.

```
[ ]:
```