CSc 120
Introduction to Computer Programming II

Adapted from slides by Dr. Saumya Debray

10: Linked Lists
Python lists: reprise

L

L.insert: O(n)

L[i]: O(1)

L.append: O(1)

concatenating two lists: O(n)

**Question:** Can we do insertion and concatenation in O(1) time?

(complexity of other operations may change).

⇒ "Linked list"
Python lists: reprise

- Key feature: L[i] and L[i+1] are adjacent in memory
- This makes accessing L[i] very efficient
  - O(1)
- Insertion and concatenation require moving O(n) elements
  - O(n)
Linked lists

- To get $O(1)$ insertion and concatenation, we cannot afford to move $O(n)$ list elements.
- We have to relax the requirement that $i^{th}$ element is adjacent to $(i+1)^{st}$ element.
  - Any element can be anywhere in memory.
- Each element has to tell us where to find the next element.
linked lists
Linked lists

- Linked list:

A collection of ordered elements where each element has a value and a reference to the next element.

There is at least one variable that references the beginning of the list.
Linked lists

Each element of the list has a reference to the next list element.

This is how we draw a linked list.
Linked lists

With each element of the list, keep a reference to the next list element

"nodes"

each node in the list has a reference to the next node
EXERCISE

• Exploring linked lists using a piece of paper as a node

• Each piece of paper has
  – a value (a word)
  – a reference (where to go next)
Linked lists

With each element of the list, keep a reference to the next list element

Let's explore this idea using a file for a "node"
Linked lists

With each element of the list, keep a reference to the next list element

Files as nodes: each file has two lines
- the first line is a value
- the second line is a reference to the next "node" (file)
Linked lists

• Let's explore this idea using a file for a "node"
  - each file has two lines:
    ▪ the first line is a value
    ▪ the second line is a reference to the next "node" (file)

Sample file “node”: filename is 20.txt
  value: The
  next: 11.txt
EXERCISE

• Exploring linked lists using files as nodes

How would we add the word "total" to our linked lists of files so that the sentence reads:

*The expert in anything was once a total beginner.*
**EXERCISE**

• Exploring linked lists using files as nodes

How would we add the word "total" to our linked lists of files so that the sentence reads:

*The expert in anything was once a total beginner.*

Create a new "node" (a new file)

The first line is "total"
The second line is 17.txt
What else do we have to do?
   - modify the file node for the value "a" to change its reference
Linked lists

References are addresses in memory.

Here is the diagram with explicit addresses (simplified).
Insertion

Consider inserting a new node into the linked list

```
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>96</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>address 96</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;aaa&quot;</td>
<td>432</td>
</tr>
<tr>
<td></td>
<td>next</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>address 432</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;bbb&quot;</td>
<td>64</td>
</tr>
<tr>
<td></td>
<td>next</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>address 64</td>
<td>132</td>
</tr>
<tr>
<td></td>
<td>next</td>
<td></td>
<td>None</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;ccc&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>next</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;ddd&quot;</td>
<td></td>
</tr>
</tbody>
</table>
```
Insertion

Specifically, add a new node between "bbb" and "ccc". What do we change?
Specifically, add a new node between "bbb" and "ccc". What do we change?
We want to add a new node between "bbb" and "ccc". What do we change?
Insertion

Set the next references appropriately. What is the complexity of insertion?

*assuming we have a reference to the node of insertion
To insert an element (which can be a linked list) into a linked list: set next references appropriately

O(1)
Insertion

To insert an element into a linked list: set next references appropriately

O(1)
Concatenation

To concatenate two linked lists: set next reference of end of first list to refer to beginning of second list

* once we have a reference to the end of the first list
Concatenation

To concatenate two linked lists: set next reference of end of first list to refer to beginning of second list

* once we have a reference to the end of the first list
implementation
Nodes: Implementation

class Node:
    def __init__(self, value):
        self._value = value  # reference to the object at that node
        self._next = None    # reference to the next node in the list

Getters:
    def value(self):
        return self._value
    def next(self):
        return self._next

Setters:
    def set_value(self, value):
        self._value = value
    def set_next(self, next):
        self._next = next
Linked Lists: Implementation

A linked list is just (a reference to) a sequence of nodes
Linked Lists: Implementation

A linked list is just (a reference to) a sequence of nodes

```python
class LinkedList:
    def __init__(self):
        self._head = None
```

nodes
Linked Lists: Implementation

A linked list is just (a reference to) a sequence of nodes

```python
class LinkedList:
    def __init__(self):
        self._head = None
```

*head of the list*  
*tail of the list*
class LinkedList:
    def __init__(self):
        self._head = None

    def is_empty(self):
        return self._head == None

    def head(self):
        return self._head
addition at the head of the list
Adding a node at the head

```
new +

 L _head 'aa' 'bb'
     _next

   +

 new 'dd'
     _next

 L _head 'aa' 'bb'
     _next

 new 'dd'
     _next
```
Adding a node at the head

Sequence of operations for an add method:
1. new._next = L._head
2. L._head = new
Adding a node at the head

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

# add a node new at the head of the linked list
def add(self, new):
    new._next = self._head
    self._head = new
Creating a linked list: Example 1

class Node:
    def __init__(self, value):
        self._value = value
        self._next = None

    ...

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

    def add(self, new):
        new._next = self._head
        self._head = new

>>> my_list = LinkedList()
>>> this_node = Node(3)
>>> my_list.add(this_node)
>>> this_node = Node(20)
>>> my_list.add(this_node)
Creating a linked list: Example 1

class Node:
    def __init__(self, value):
        self._value = value
        self._next = None

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

    def add(self, new):
        new._next = self._head
        self._head = new

>>> my_list = LinkedList()
>>> this_node = Node(3)
>>> my_list.add(this_node)
>>> this_node = Node(20)
>>> my_list.add(this_node)
Creating a linked list: Example 1

```python
class Node:
    def __init__(self, value):
        self._value = value
        self._next = None

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

    def add(self, new):
        new._next = self._head
        self._head = new

>>> my_list = LinkedList()
>>> this_node = Node(3)
>>> my_list.add(this_node)
>>> this_node = Node(20)
>>> my_list.add(this_node)

my_list

LinkedList

_ head

None

Node

3

None

this_node
```
Creating a linked list: Example 1

class Node:
    def __init__(self, value):
        self._value = value
        self._next = None

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

    def add(self, new):
        new._next = self._head
        self._head = new

>>> my_list = LinkedList()
>>> this_node = Node(3)
>>> my_list.add(this_node)
>>> this_node = Node(20)
>>> my_list.add(this_node)
Creating a linked list: Example 1

class Node:
    def __init__(self, value):
        self._value = value
        self._next = None

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

    def add(self, new):
        new._next = self._head
        self._head = new

>>> my_list = LinkedList()
>>> this_node = Node(3)
>>> my_list.add(this_node)
>>> this_node = Node(20)
>>> my_list.add(this_node)
Creating a linked list: Example 1

class Node:
    def __init__(self, value):
        self._value = value
        self._next = None

... 

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

    def add(self, new):
        new._next = self._head
        self._head = new

>>> my_list = LinkedList()
>>> this_node = Node(3)
>>> my_list.add(this_node)
>>> this_node = Node(20)
>>> my_list.add(this_node)
Creating a linked list: Example 1

class Node:
    def __init__(self, value):
        self._value = value
        self._next = None

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

    def add(self, new):
        new._next = self._head
        self._head = new

>>> my_list = LinkedList()
>>> this_node = Node(3)
>>> my_list.add(this_node)
>>> this_node = Node(20)
>>> my_list.add(this_node)
Creating a linked list: Example 1

class Node:
    def __init__(self, value):
        self._value = value
        self._next = None

... 

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

    def add(self, new):
        new._next = self._head
        self._head = new

>>> my_list = LinkedList()
>>> this_node = Node(3)
>>> my_list.add(this_node)
>>> this_node = Node(20)
>>> my_list.add(this_node)
Creating a linked list: Example 1

```python
class Node:
    def __init__(self, value):
        self._value = value
        self._next = None

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

    def add(self, new):
        new._next = self._head
        self._head = new

my_list = LinkedList()
this_node = Node(3)
my_list.add(this_node)
this_node = Node(20)
my_list.add(this_node)
```

```>>> my_list = LinkedList()
>>> this_node = Node(3)
>>> my_list.add(this_node)
>>> this_node = Node(20)
>>> my_list.add(this_node)```

```graph
digraph G {
    rankdir=LR;
    node [shape=box, style=filled, fillcolor=white ];
    my_list [label= "LinkedList"]; 
    self [label= ""]; 
    _head [label= ""];
    new [label= ""];
    this_node [label= ""];
    Node [label= "Node"]; 
    None [label= "None"]; 
    20 [label= ""];
    3 [label= ""];
    new [label= ""];
    this_node [label= ""];
    my_list [label= ""];
    self [label= ""];
    _head [label= ""];
    new [label= ""];
    this_node [label= ""];
}
```
Creating a linked list: Example 1

```python
class Node:
    def __init__(self, value):
        self._value = value
        self._next = None

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

    def add(self, new):
        new._next = self._head
        self._head = new

>>> my_list = LinkedList()
>>> this_node = Node(3)
>>> my_list.add(this_node)
>>> this_node = Node(20)
>>> my_list.add(this_node)
```

```
my_list

self

_linked

this_node

new

Node

Node

Node

None

20

3
```
Creating a linked list: Example 1

```python
class Node:
    def __init__(self, value):
        self._value = value
        self._next = None

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

    def add(self, new):
        new._next = self._head
        self._head = new

>>> my_list = LinkedList()
>>> this_node = Node(3)
>>> my_list.add(this_node)
>>> this_node = Node(20)
>>> my_list.add(this_node)
```

![Diagram of linked list structure](image_url)
Creating a linked list: Example 1

```python
class Node:
    def __init__(self, value):
        self._value = value
        self._next = None

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

    def add(self, new):
        new._next = self._head
        self._head = new

>>> my_list = LinkedList()
>>> this_node = Node(3)
>>> my_list.add(this_node)
>>> this_node = Node(20)
>>> my_list.add(this_node)
```

Integers are not compound objects. Can simplify the diagram as shown.
Exercise

• Do problem 1 in the Exercise worksheet.
Adding a node at the head

⚠ Changing the order of assignments does not work:

```python
def add(self, new):
    new._next = self._head
    self._head = new

def broken_add(self, new):
    self._head = new
    new._next = self._head
```

```
  _head                     _head
    'aa'                   'aa'
  +----------------------
  ^                       ^
  |                       |
new 'dd'                  new 'dd'
  _next                  _next
```

```
  'bb'
```

```python
def add(self, new):
    new._next = self._head
    self._head = new
```

```
  'bb'
```

```
  _head
    'aa'
  +----------------------
  ^                       ^
  |                       |
new 'dd'                  new 'dd'
  _next                  _next
```

```
  'bb'
```

Creating a linked list: Example 2

class Node:
    def __init__(self, value):
        self._value = value
        self._next = None

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

    def add(self, new):
        new._next = self._head
        self._head = new

infile = open("infile.txt")
my_list = LinkedList()
for line in infile:
    this_node = Node(line)
    my_list.add(this_node)
Creating a linked list: Example 2

class Node:
    def __init__(self, value):
        self._value = value
        self._next = None

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

    def add(self, new):
        new._next = self._head
        self._head = new

infile = open("infile.txt")
my_list = LinkedList()
for line in infile:
    this_node = Node(line)
    my_list.add(this_node)
Creating a linked list: Example 2

class Node:
    def __init__(self, value):
        self._value = value
        self._next = None

...  

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

    def add(self, new):
        new._next = self._head
        self._head = new

infile = open("infile.txt")
my_list = LinkedList()
for line in infile:
    this_node = Node(line)
    my_list.add(this_node)
Creating a linked list: Example 2

```python
class Node:
    def __init__(self, value):
        self._value = value
        self._next = None

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

    def add(self, new):
        new._next = self._head
        self._head = new

infoline = open("infoline.txt")
my_list = LinkedList()
for line in infoline:
    this_node = Node(line)
    my_list.add(this_node)
```

```plaintext
aa
bb
cc
```
Creating a linked list: Example 2

class Node:
    def __init__(self, value):
        self._value = value
        self._next = None

class LinkedList:
    def __init__(self):
        self._head = None
    def add(self, new):
        new._next = self._head
        self._head = new

infile = open(“infile.txt”)  
my_list = LinkedList()  
for line in infile:
    this_node = Node(line)  
    my_list.add(this_node)
Creating a linked list: Example 2

```python
class Node:
    def __init__(self, value):
        self._value = value
        self._next = None

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

    def add(self, new):
        new._next = self._head
        self._head = new

infile = open("infile.txt")
my_list = LinkedList()
for line in infile:
    this_node = Node(line)
    my_list.add(this_node)
```

```
infile.txt
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>aa</td>
<td></td>
</tr>
<tr>
<td>bb</td>
<td></td>
</tr>
<tr>
<td>cc</td>
<td></td>
</tr>
</tbody>
</table>
```

```
my_list
LinkedList
_head
this_node
```

```
line
"aa"
Node
None
```
Creating a linked list: Example 2

class Node:
    def __init__(self, value):
        self._value = value
        self._next = None

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

    def add(self, new):
        new._next = self._head
        self._head = new

in_file = open("infile.txt")
my_list = LinkedList()
for line in in_file:
    this_node = Node(line)
    my_list.add(this_node)
Creating a linked list: Example 2

class Node:
    def __init__(self, value):
        self._value = value
        self._next = None

    ...

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

    def add(self, new):
        new._next = self._head
        self._head = new

    infile = open("infile.txt")
    my_list = LinkedList()
    for line in infile:
        this_node = Node(line)
        my_list.add(this_node)
Creating a linked list: Example 2

class Node:
    def __init__(self, value):
        self._value = value
        self._next = None

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

    def add(self, new):
        new._next = self._head
        self._head = new

infile = open("infile.txt")
my_list = LinkedList()
for line in infile:
    this_node = Node(line)
    my_list.add(this_node)
Creating a linked list: Example 2

class Node:
    def __init__(self, value):
        self._value = value
        self._next = None

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

    def add(self, new):
        new._next = self._head
        self._head = new

infile = open("infile.txt")
my_list = LinkedList()
for line in infile:
    this_node = Node(line)
    my_list.add(this_node)
Creating a linked list: Example 2

class Node:
    def __init__(self, value):
        self._value = value
        self._next = None

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

    def add(self, new):
        new._next = self._head
        self._head = new

infile = open("infile.txt")
my_list = LinkedList()
for line in infile:
    this_node = Node(line)
    my_list.add(this_node)
Creating a linked list: Example 2

class Node:
    def __init__(self, value):
        self._value = value
        self._next = None


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

    def add(self, new):
        new._next = self._head
        self._head = new

infile = open("infile.txt")
my_list = LinkedList()
for line in infile:
    this_node = Node(line)
    my_list.add(this_node)
Creating a linked list: Example 2

class Node:
    def __init__(self, value):
        self._value = value
        self._next = None

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

    def add(self, new):
        new._next = self._head
        self._head = new

infile = open("infile.txt")
my_list = LinkedList()
for line in infile:
    this_node = Node(line)
    my_list.add(this_node)
class Node:
    def __init__(self, value):
        self._value = value
        self._next = None

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

    def add(self, new):
        new._next = self._head
        self._head = new

infile = open("infile.txt")
my_list = LinkedList()
for line in infile:
    this_node = Node(line)
    my_list.add(this_node)
Creating a linked list: Example 2

class Node:
    def __init__(self, value):
        self._value = value
        self._next = None

class LinkedList:
    def __init__(self):
        self._head = None
    def add(self, new):
        new._next = self._head
        self._head = new

infile = open("infile.txt")
my_list = LinkedList()
for line in infile:
    this_node = Node(line)
    my_list.add(this_node)
Creating a linked list: Example 2

class Node:
    def __init__(self, value):
        self._value = value
        self._next = None
    ...

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

    def add(self, new):
        new._next = self._head
        self._head = new

infile = open("infile.txt")
my_list = LinkedList()
for line in infile:
    this_node = Node(line)
    my_list.add(this_node)

Can simplify the diagram as shown.
Visiting all of the nodes

Suppose we want to do something to each node of a list.

How do we loop through the nodes (elements)?
  - with a built-in list, we would use a for or while loop
Consider a Python list

Suppose we want to do something to each node of a list.

In a Python list, use a for or while loop:
```
alist = [20, 6, 17, 4, 28, 5]
i = 0  # start at the beginning
while i < len(alist):  # stop when you hit the end
    <do something with alist[i]>
i = i + 1  # go to the next element
```

How do we do each of these with a linked list?
- start at the beginning
- go to the next element
- stop when you hit the end
Visiting all of the nodes

Suppose we want to do something to each node of a list:
Visiting all of the nodes

Suppose we want to do something to each node of a list:

Start at the beginning: use the head of the list

\[ \text{current} = \text{self._head} \]
Visiting all of the nodes

Suppose we want to do something to each node of a list:

Start at the beginning: use the head of the list

\[ \text{current} = \text{self}_\text{.}_{\text{head}} \]

Go to the next element: use the next attribute

\[ \text{current} = \text{current}_\text{.}_{\text{next}} \]
Visiting all of the nodes

Suppose we want to do something to each node of a list:

Start at the beginning: use the head of the list
current = self._head
Go to the next element: use the next attribute
current = current._next
Stop when you hit None:
current == None
Visiting all of the nodes

Suppose we want to do something to each node of a list:

```
progress through the list until you hit the end:
current = self._head
while current != None:
    do something with current._value
    current = current._next
```
Visiting all of the nodes

Suppose we want to do something to each node of a list:

Progress through the list until you hit the end:

current = self._head

while current != None:
    <do something with current._value>
    current = current._next
Visiting all of the nodes

Suppose we want to do something to each node of a list:

Progress through the list until you hit the end:

current = self._head
while current != None:
    <do something with current._value>
    current = current._next
Visiting all of the nodes

Suppose we want to do something to each node of a list:

```python
progress through the list until you hit the end:
current = self._head
while current != None:
    <do something with current._value>
current = current._next
```
Visiting all of the nodes

Suppose we want to do something to each node of a list:

Progress through the list until you hit the end:

```python
current = self._head
while current != None:
    <do something with current._value>
    current = current._next
```
Example: print each element

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

....

def print_elements(self):
    current = self._head
    while current != None:
        print(str(current._value))
        current = current._next

O(n)
• Consider a linked list whose value attributes consist of integers.

• Write a method `double(self)` that doubles the value attributes of all nodes in a linked list.
• Consider a linked list whose value attributes consist of integers.

• Write a method `first_even(self)` that returns the first node in the linked list whose value is even. If there are no even values, the method returns `None`. 
EXERCISE

• Consider a linked list whose value attributes consist of strings.

• Write a method \texttt{replace}(\texttt{self}, \texttt{s1}, \texttt{s2}) that replaces the value attributes of all nodes that equal \texttt{s1} with \texttt{s2}.
adding to the end (tail) of the list
Adding a node to the tail

Suppose we want to add a node to the end of a list:
Adding a node to the tail

Suppose we want to add a node to the end of a list:

To add a node new at the end (i.e., tail) of a list L:

1. find the last element Y of L
2. Y._next = new
Adding a node to the tail

To add a node new at the end (i.e., tail) of a list L:

1. find the last element Y of L
2. Y._next = new
Adding a node to the tail

To add a node new at the end (i.e., tail) of a list L:

1. find the last element Y of L $\quad O(n)$
2. Y._next = new $\quad O(1)$
Adding a node to the tail

Suppose we want to add a node to the end of a list:

```
L
"aaa"
"bbb"
"ccc"
None
```

Progress through the list until you hit the end:

```python
current = self._head
while current != None:
    current = current._next
```
Adding a node to the tail

Suppose we want to add a node to the end of a list:

Progress through the list until you hit the end:

```python
current = self._head
while current != None:
    current = current._next
```

Any issues with this code?
Adding a node to the tail

Suppose we want to add a node to the end of a list:

To add a node new at the end (i.e., tail) of a list L:

1. Traverse in a loop
2. Keep track of the previous node
Adding a node to the tail

Suppose we want to add a node to the end of a list:

To add a node new at the end (i.e., tail) of a list $L$:

1. Traverse in a loop
2. Keep track of the previous node in prev
Adding to the end

class LinkedList:

def add_to_end(self, new):
    current = self._head
    prev = None
    while current != None:
        prev = current   # keep track of previous node
        current = current._next
    prev._next = new   # add to the end

O(n)
Adding to the end

class LinkedList:

def add_to_end(self, new):
    current = self._head
    prev = None
    while current != None:
        prev = current
        current = current._next
        prev._next = new
Adding to the end

class LinkedList:

    def add_to_end(self, new):
        current = self._head
        prev = None
        while current != None:
            prev = current
            # keep track of previous node
            current = current._next
        prev._next = new
        # add to the end

The list might be empty!
Adding to the end

class LinkedList:

    def add_to_end(self, new):
        if self._head == None:  # the list is empty
            self._head = new  # the list now has one node
        else:
            current = self._head
            prev = None
            while current != None:
                prev = current  # keep track of previous node
                current = current._next
            prev._next = new  # add to the end

O(n)
Adding to the end (review)

class LinkedList:

    def add_to_end(self, new):
        if self._head == None:  # the list is empty
            self._head = new  # add the new node
        else:
            current = self._head  # Look-ahead method
            while current._next != None:
                current = current._next
            current._next = new  # add to the end
finding the \( n^{\text{th}} \) element
Finding the $n^{th}$ element

class LinkedList:

    # return the node at position $n$ of the linked list
    def get_element(self, n):
        elt = self._head
        while elt != None and n > 0:
            elt = elt._next
            n -= 1
        return elt

$O(n)$
insertion
Inserting a node

Suppose we want to insert a node X into a list here:

```
... "aaa"          "bbb"          "ccc"
```

Then we have to adjust the next-node reference on the node Y just before that position.
Inserting a node

Suppose we want to insert a node X into a list here:

Then we have to adjust the next-node reference on the node Y just before that position.
Inserting a node

The order of operations is important:

1. $X._\text{next} = Y._\text{next}$
Inserting a node

The order of operations is important:

1. \( X._\text{next} = Y._\text{next} \)
2. \( Y._\text{next} = X \)
Inserting a node

Inserting a node $X$ at position $n$ in a list $L$:

1. find the node $Y$ at position $n-1$
   - iterate $n-1$ positions from the head of the list*

2. insert $X$ after $Y$
   - adjust next-node references as in previous example

* do something sensible if the list has fewer than $n-1$ nodes

Y = L._head
for i in range(n-1):
  Y = Y._next
X._next = Y._next
Y._next = X

$O(n)$

$O(1)$
Inserting a node

class LinkedList:

    # insert a node new at position n

def insert(self, new, n):
    if n == 0:
        self.add(new)
    else:
        prev = self.get_element(n-1)
        new.next = prev.next
        prev.next = new
Exercise

Consider a linked list whose value attributes consist of integers. Using the Node and LinkedList class, write a method

```
    sum_even_pos(self)
```

that returns the sum of the values of the elements at even positions in the linked list. Node positions begin at 0. Return 0 if the list is empty.
Adding to the end (review)

class LinkedList:

    def add_to_end(self, new):
        if self._head == None:  # the list is empty
            self._head = new  # the list now has one node
        else:
            current = self._head
            prev = None
            while current != None:
                prev = current  # keep track of previous node
                current = current._next
            prev._next = new  # add to the end
Adding to the end (review)

class LinkedList:

    def add_to_end(self, new):
        if self._head == None:  # the list is empty
            self._head = new  # add the new node
        else:
            current = self._head  # Look-ahead method
            while current._next != None:
                current = current._next
            current._next = new  # add to the end
deletion
Deleting a node

Suppose we want to delete this node:
Deleting a node

Suppose we want to delete this node:
Deleting a node

Suppose we want to delete this node:

1. find the node Y just before X (i.e., Y._next == X) \[O(n)\]
2. Y._next = X._next \[O(1)\]
3. X._next = None
Deleting a node

What if the list has one element?

L

"aaa"
None

X
Deleting a node

What if the list has one element?

There is no Y before X.

Must check for that condition first.
Deleting a node

class LinkedList:
    
    # delete a node X
    def delete(self, X):
        if self._head == X:  # X is the head of the list
            self._head = X._next
        else:
            Y = self._head
            while Y._next != X:  # look-ahead method
                Y = Y._next
            Y._next = X._next
            X.next = None
Remove from the front

Removing from the front is simpler:
Removing a node from the front

Removing from the front is simpler:

```
L

<table>
<thead>
<tr>
<th>&quot;aaa&quot;</th>
<th>&quot;bbb&quot;</th>
<th>&quot;ccc&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>None</td>
</tr>
</tbody>
</table>
```
concatenation
# Concatenating two linked lists

```python
class LinkedList:
    # concatenate list2 at the end of the list
    def concat(self, list2):
        if self._head == None:  # list is empty
            self._head = list2._head
        else:
            tail = self._head
            while tail._next != None:  # O(n)
                tail = tail._next
            tail._next = list2._head
```

O(n)  
O(1)
maintaining a tail reference
Maintaining a tail reference

A variation is to also maintain a reference to the tail of the list

```python
class LinkedList:
    def __init__(self):
        self._head = None
        self._tail = None
```

*LinkedList*
Tail references and concatenation

list1

list2
Tail references and concatenation

list1

_list_head

_list_tail

“aa”

“bb”

“cc”

list2

_list_head

_list_tail

“dd”

“ee”

“ff”
Tail references and concatenation

list1

_list1_head
_list1_tail

"aa" "bb" "cc"

list2

_list2_head
_list2_tail

"dd" "ee" "ff"
Tail references and concatenation

list1

list2

"aa" "bb" "cc"

"dd" "ee" "ff"
Maintaining a tail reference

- Concatenation and append become $O(1)$:
  ```python
def concat(self, list2):
    if self._head == None:
      self._head = list2._head
      self._tail = list2._tail
    else:
      self._tail._next = list2._head
      self._tail = list2._tail
  ```

- All linked list operations must now make sure that the tail reference is kept properly updated.
Exercise

Given the following LinkedList definition:

```python
class LinkedList:
    def __init__(self):
        self._head = None
        self._tail = None
```

Write the append(self, new) method for the class.
**Linked lists: summary**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Without tail reference</th>
<th>With tail reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>add to front of list</td>
<td>O(1)</td>
<td></td>
</tr>
<tr>
<td>append to end of list</td>
<td>O(n)</td>
<td>O(1)</td>
</tr>
<tr>
<td>find nth element</td>
<td>O(n)</td>
<td></td>
</tr>
<tr>
<td>insert</td>
<td>O(1) if prev. node is available</td>
<td>O(n) otherwise</td>
</tr>
<tr>
<td>delete</td>
<td>O(1) if prev. node is available</td>
<td>O(n) otherwise</td>
</tr>
<tr>
<td>concatenate</td>
<td>O(n)</td>
<td>O(1)</td>
</tr>
</tbody>
</table>
Exercise

Using the Node and LinkedList class, write a method `remove_first(self)` that removes the first element of a linked list and returns the node removed.

If the list is empty, the method returns None.