Linked Structures

• Self-referential classes can be used to create linked data structures:

```python
class Node:
    def __init__(self, data, next):
        self.data = data
        self.next = next
    def getNext(self):
        return self.next
    def getData(self):
        return self.next
```

• `next` holds a reference to a Node object
• through the next reference, can link `Node` objects together:


# Linked List

- Ordered Collection of data
- Need a single variable which is reference to 1ˢᵗ node on list
- Nodes are linked together in-order by following `next` references

```python
# an empty list:
head = None

# a list with one node:
head = Node(25, None)

# add a second Node to the end:
head.setNext(Node(99, None))
```

List of length 1:

```
<table>
<thead>
<tr>
<th>data</th>
<th>25</th>
</tr>
</thead>
<tbody>
<tr>
<td>next</td>
<td></td>
</tr>
</tbody>
</table>
```

List of length 2:

```
<table>
<thead>
<tr>
<th>data</th>
<th>25</th>
</tr>
</thead>
<tbody>
<tr>
<td>next</td>
<td></td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>data</th>
<th>99</th>
</tr>
</thead>
<tbody>
<tr>
<td>next</td>
<td></td>
</tr>
</tbody>
</table>
```
Operations on a List

• All start at Node referred to by head reference, and traverse next references to access other nodes in the list

• Accessing the ith node is O(n):
  – first access Node referred to by head, follow its next reference to access the 2\textsuperscript{nd} Node, follow its next reference to access the 3\textsuperscript{rd} Node, and so on
Insert at Head of List

head = None
For i in range(10): // make list of 10 Nodes
    val = input("Enter a value: ")
    tmp = Node(val, None)
    tmp.setNext(head)
    head = tmp

i == 0:

i == 1:
Resulting List of 10 nodes:
Traverse the List

tmp = head    # start at the 1st node
while(tmp != None):
    # print out value of the data field of curr node
    print (tmp.getData() + " ")
    # set tmp to point to the next Node in the list
    tmp = tmp.getNext()
# output:  23 44 35 77 88 683 21 55 63 25
Find Element In List

- Start at head Node, compare search value to data field
- traverse next refs until matching data field is found, or until no more list

```python
# found will pt to matching Node
found = None
tmp = head
while(tmp != None):
    if(tmp.getData() == val):
        found = tmp
    tmp = tmp.getNext()
```

```
stack

val

head

tmp

```

```

```

```

```

```

```

```
Insert in the middle

```python
def insert(node, value):
    new_node = Node(value)
    new_node.setNext(node.getNext())
    node.setNext(new_node)
```

```
new_node = Node(20, None)
tmp = head.getNext()  # lets just make tmp point
    # to some Node after head

# insert new_node after tmp
new_node.setNext(tmp.getNext())
tmp.setNext(new_node)
```
LinkedList Class

- Would really write and use a LinkedList class that encapsulates all data and method functions associated with a linked list:

```python
class LinkedList:
    def __init__(self): # create an empty list
        self.head = None
        self.size = 0
    def insertAtHead(self, data):
        new_node = Node(data, None)
        new_node.setNext(self.head)
        self.head = new_node
        self.size = self.size + 1
    # and more method definitions for linked list operations . . .

def main(): # use the LinkedList class
    my_list = LinkedList()
    my_list.insertAtHead(25)
```