Data Structures in Python Chapter 3

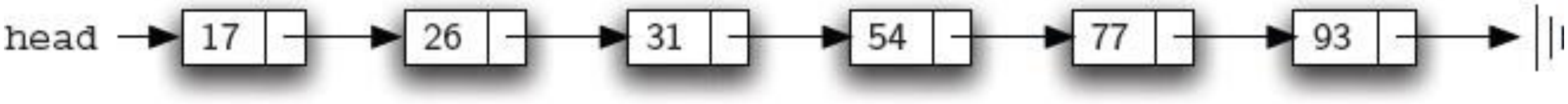
- Linked List
- OOP Inheritance
- ListUnsorted Class
- **ListSorted Class**
- Iterator
- Doubley Linked List

Agenda

- The ListSorted Class
  - Linked List - Review
  - Implementation
    - push(), pop(), find()
  - Time Complexity

The ListSorted Class

- Sorted linked-list example:



Linked List ADT

- `LinkedList()`
  - Creates a new list that is empty and returns an empty list.
- `is_empty()`
  - Tests to see whether the list is empty and **returns** a Boolean value.
- `size()` and `__len__()`
  - Returns the number of nodes in the list.
- `__str__()`
  - Returns contents of the list in human readable format.
- `push(data)`
  - Pushes a new node with the data to the list.
- `pop(data)`
  - Removes the node from the list.
- `find(data)`
  - Searches for the data in the list and **returns** a Boolean value.

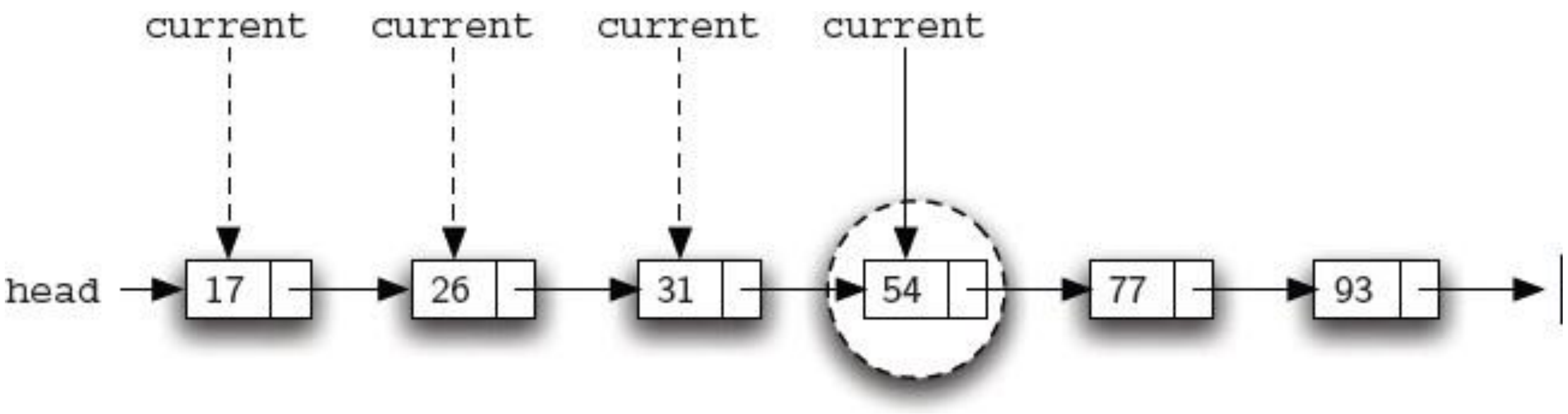
abstract methods

The ListSorted Class - push()

- push(data) the new node with data in sorted list.
- Determine the point of insertion.
  - Starting point:
    - `curr = self.head`
    - `prev = None`
    - `stop = False`

```
curr = self.head
prev = None
stop = False
while curr != None and not stop:
    if curr.get_data() > data:
        stop = True
    else:
        prev = curr
        curr = curr.get_next()
```

```
mylist.push(49)
```

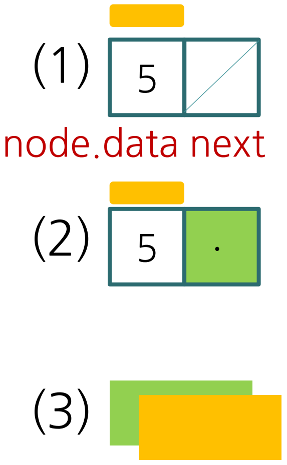


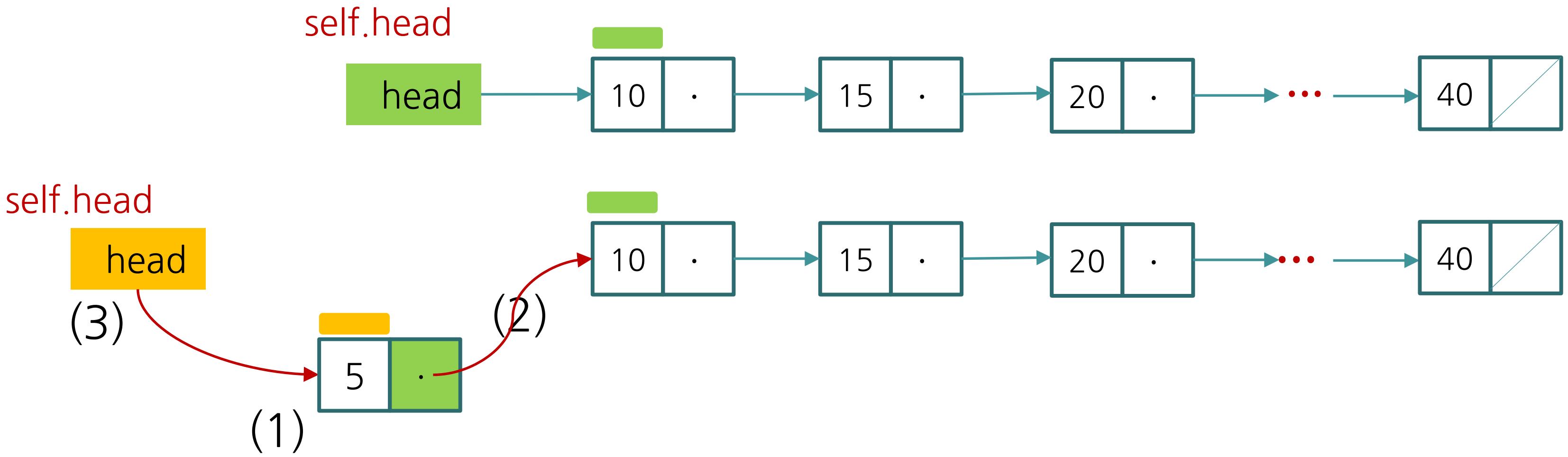
The ListSorted Class - push()

- Insert at the beginning of a linked list

```
curr = self.head
prev = None
stop = False
while curr != None and not stop:
    if curr.get_data() > data:
        stop = True
    else:
        prev = curr
        curr = curr.get_next()
```

```
(1) node = Node(5)
(2) node.set_next(self.head)
(3) self.head = node
```

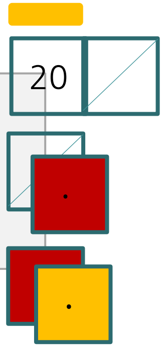


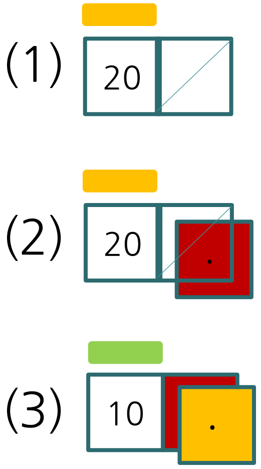


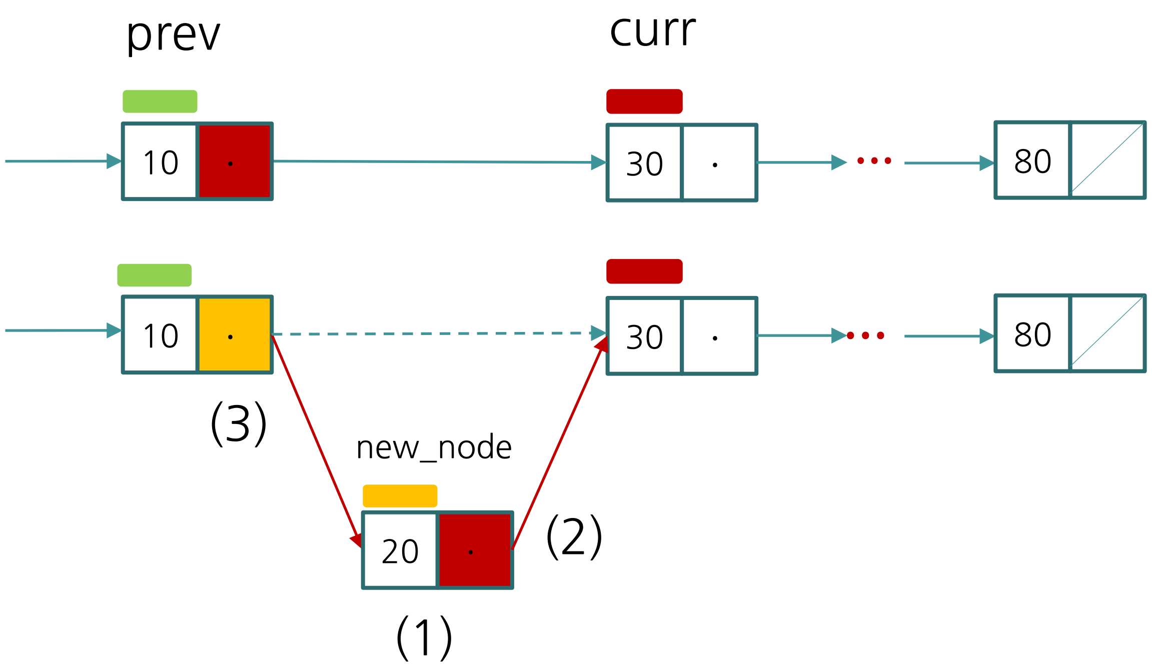
The ListSorted Class - push()

- push(data) inserts at the middle of a sorted linked list.
  - Change the next reference of the new node to refer to the current node of the list.
  - Modify the next reference of the previous node to refer to the new node.

```
(1) new_node = Node(20)
(2) new_node.set_next(curr)
(3) prev.set_next(new_node)
```





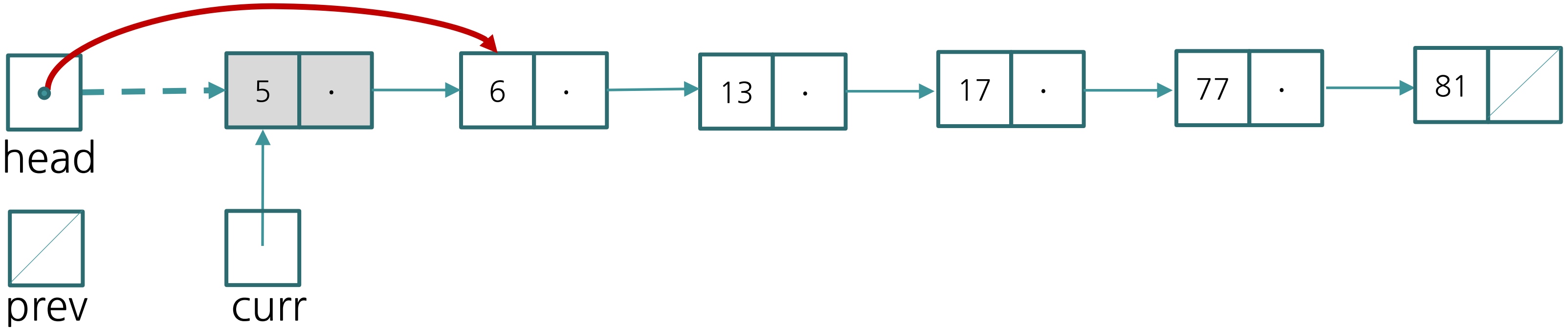


The ListSorted Class - pop()

- pop(data) removes a node with data from the list.
  - **What is different from pop() of ListUnsorted class**?
- Examples:
  - Delete the first node.

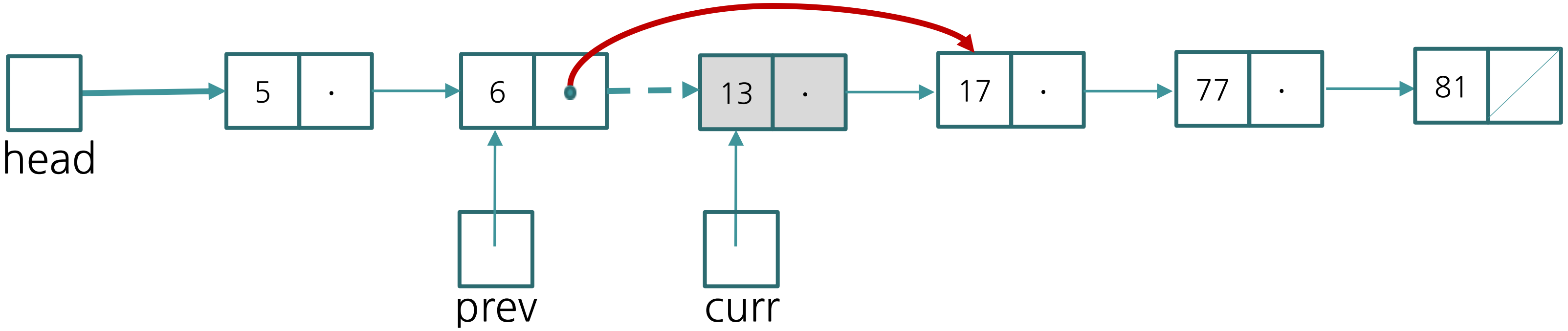
```
mylist.pop(5)
```

```
mylist.pop_front()
```



  - Delete a node in the middle of the list with **prev** and **curr** references.

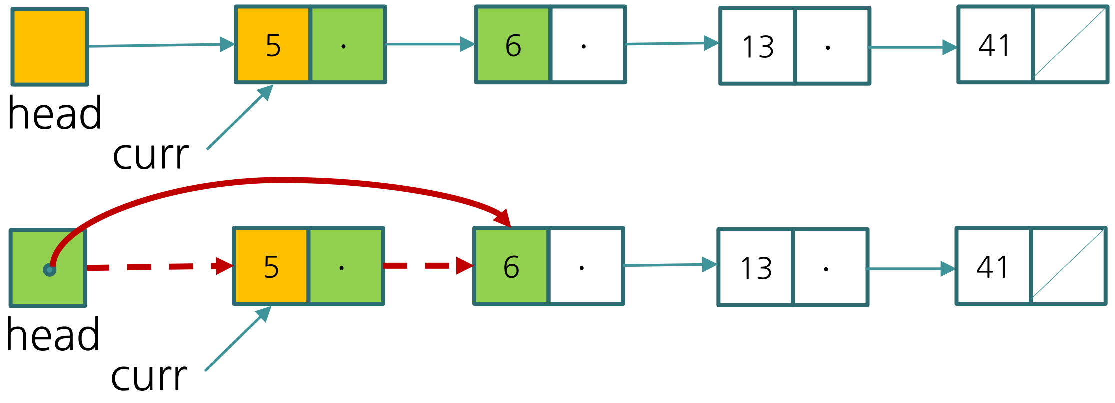
```
mylist.pop(13)
```



The ListSorted Class - pop()

- To delete a node from a linked list
  - Locate the node that you want to delete (curr)
  - **Disconnect** this node from the linked list by changing references.
- Two situations:
  - (1) To delete the first node,
    - Modify head to refer to the node after the current node

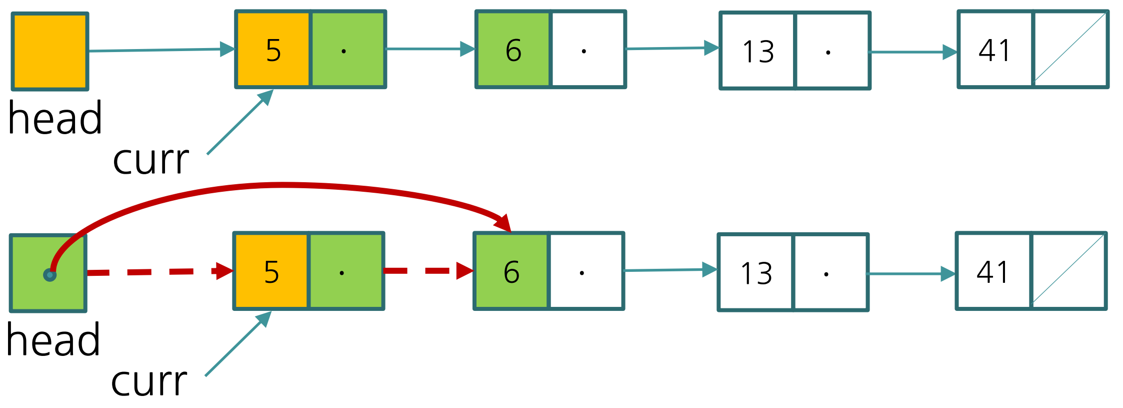
```
self.head = curr.get_next()
```



The ListSorted Class - pop()

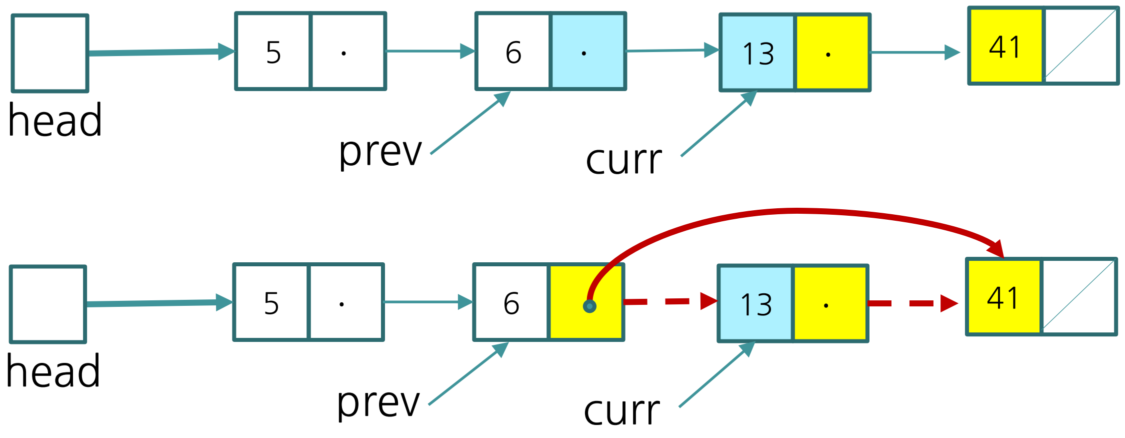
- To delete a node from a linked list
  - Locate the node that you want to delete (**curr**)
  - **Disconnect** this node from the linked list by changing references.
- Two situations:
  - (1) To delete the **first** node,
    - Modify head to refer to the node after the current node

```
self.head = curr.get_next()
```



  - (2) To delete a node in the **middle,**
  - Set next of the **prev** node to refer to the node **after the current node.**

```
prev.set_next(curr.get_next())
```

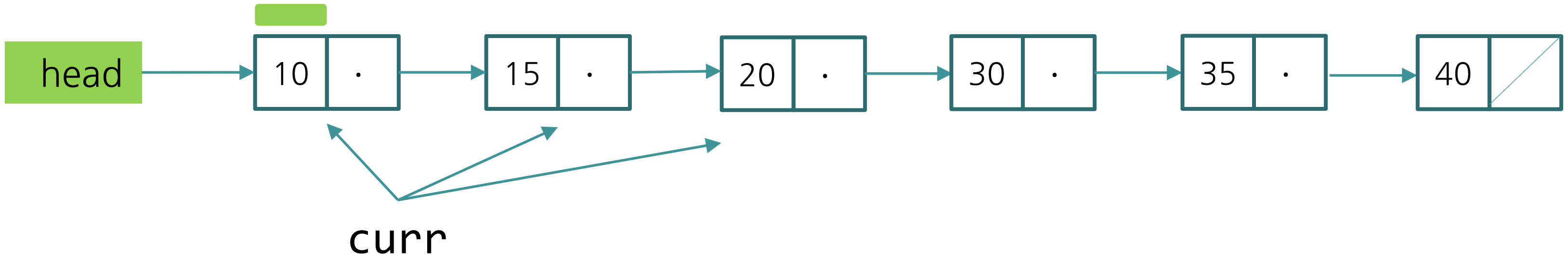


The ListSorted Class - find()

- find(data) searches for the node with data in the list.
  - Returns a Boolean
  - Examples:

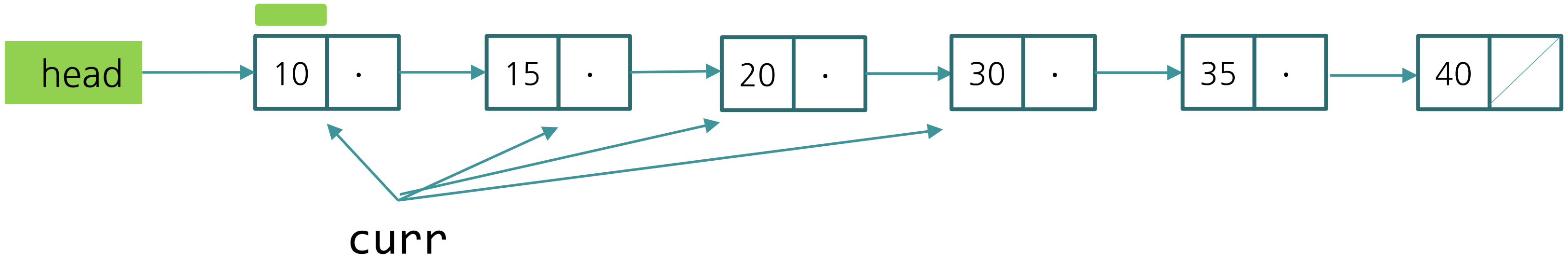
```
print(mylist.find(20))
```

```
True
```



```
print(mylist.find(25))
```

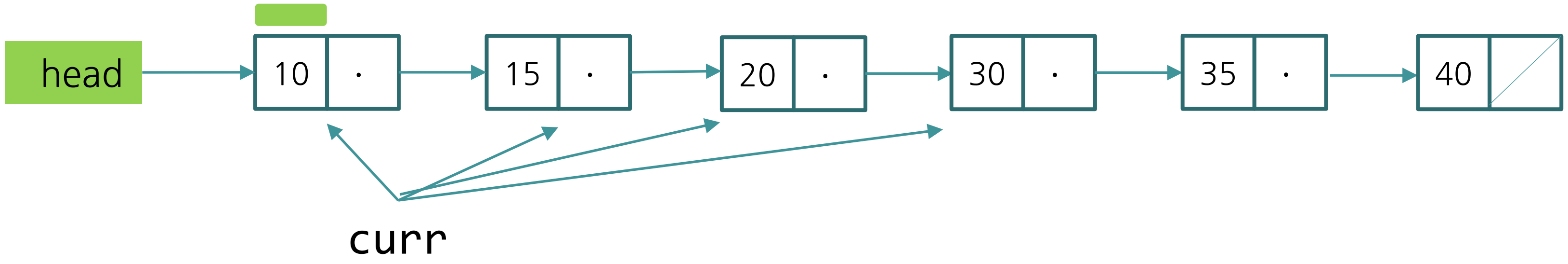
```
False
```



The ListSorted Class - find()

- find(data) searches for the node with data in the list.
  - Set a pointer to be the same address as head, process the data in the node, (search) move the pointer to the next node, and so on.
  - Loop stops either
    - Found the item
    - The next pointer is None
    - The value in the node is greater than the item that we are searching

```
curr = self.head
while curr != None:
    if curr.get_data() == data:
        return True
    elif curr.get_data() > data:
        return False
    curr = curr.get_next()
return False
```



head

10 · 15 · 20 · 30 · 35 · 40

curr

The ListSorted Class - Time Complexity

- Summary:

	ListUnsorted	ListSorted
is_empty	$O(1)$	$O(1)$
size	$O(n)$	$O(n)$
push	$O(1)$	$O(n)$
pop	$O(n)$	$O(n)$
find	$O(n)$	$O(n)$

Summary

- Different implementations may have different time and space complexity.
- The linked-list can be sorted.