빅 데 이 터 혁 신 공 유 대 학

파이썬으로 배우는 데이터 구조

한동대학교 전산전자공학부

김영섭 교수

교육부 NRF 한국연구재단

Data Structures in Python
Chapter 3 - 3

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

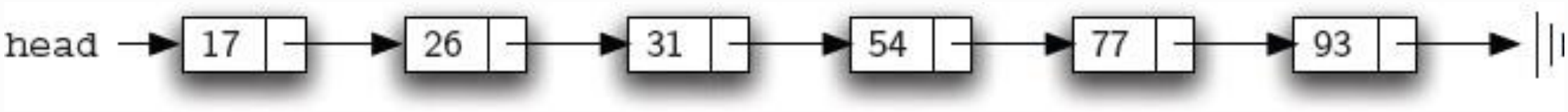
Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204 Handong Global University

Agenda

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

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.

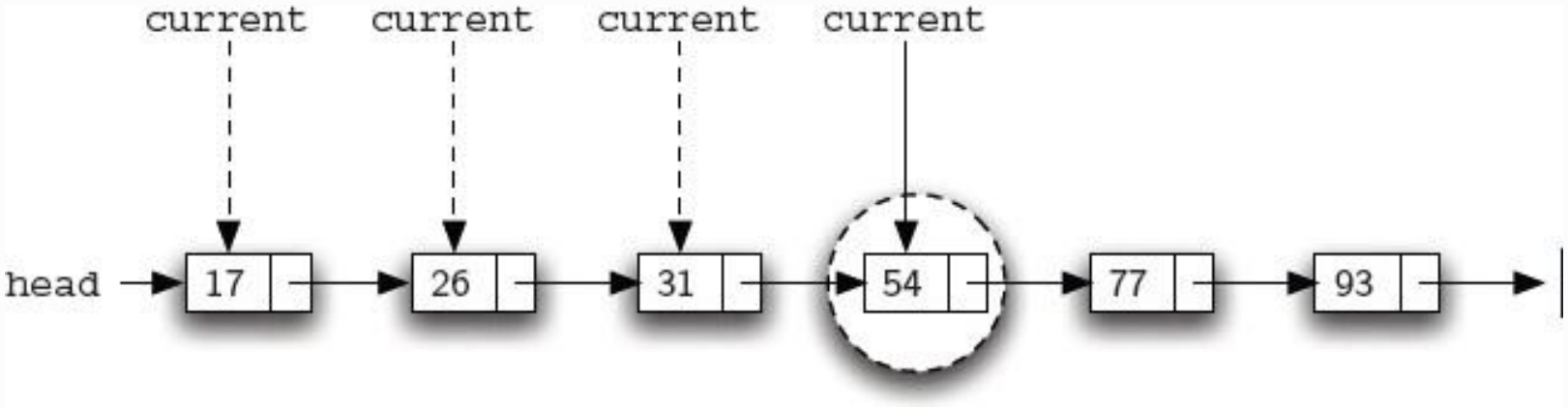
abstract methods

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)
```

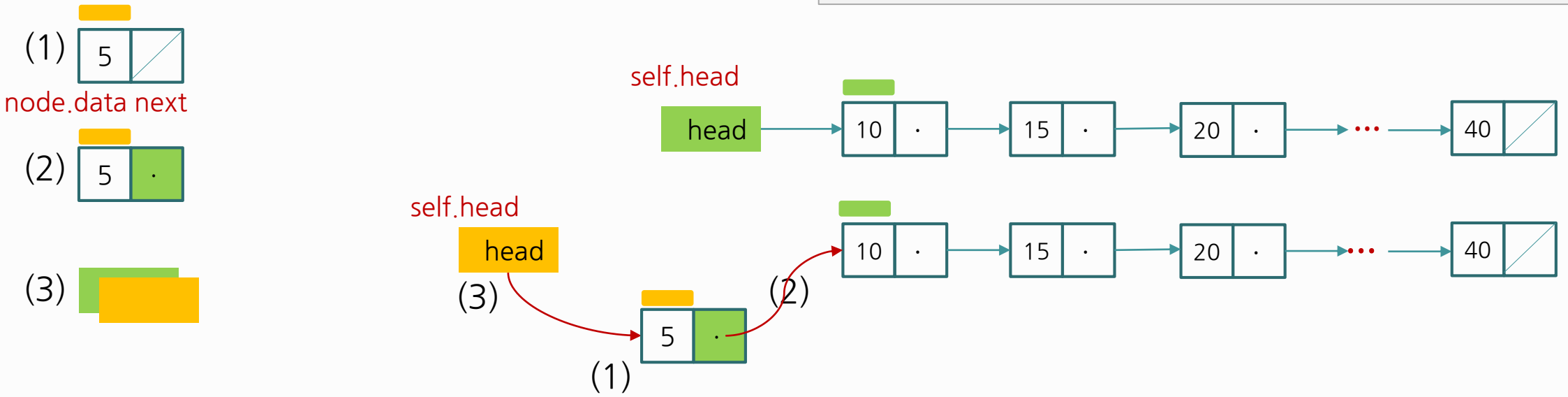


The ListSorted Class - push()

- Insert at the beginning of a linked list

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

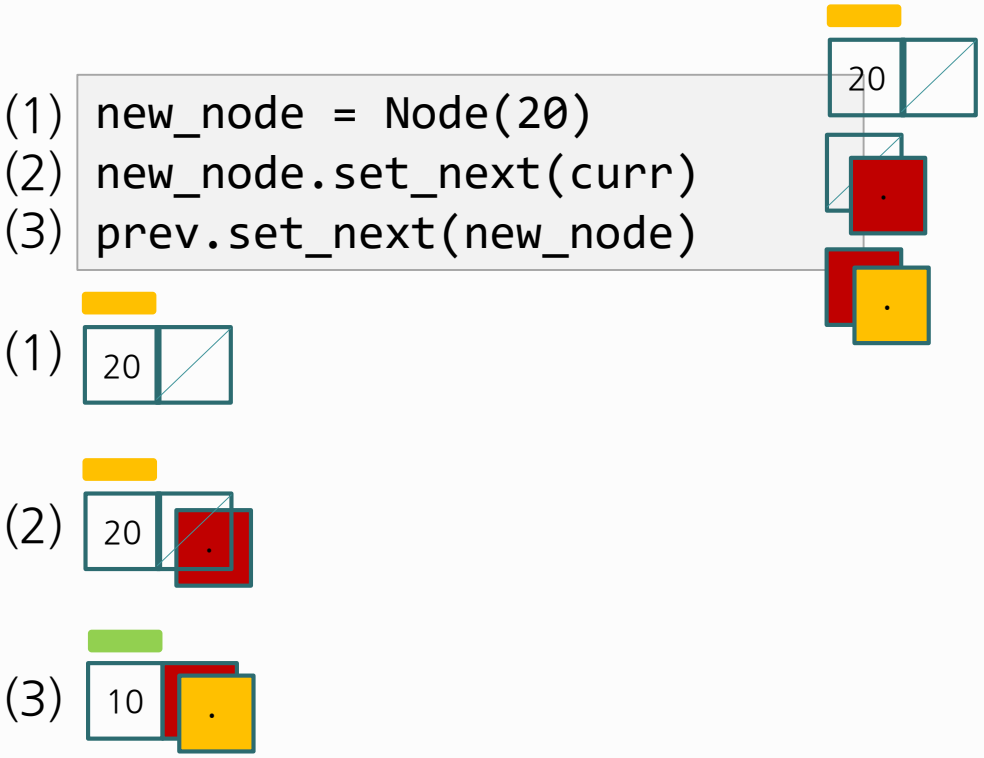
```
curr = self.head
prev = None
stop = False
while curr != None and not stop:
    if curr.get_data() > data:
        stop = True
    else:
        prev = curr
        curr = curr.get_next()
```

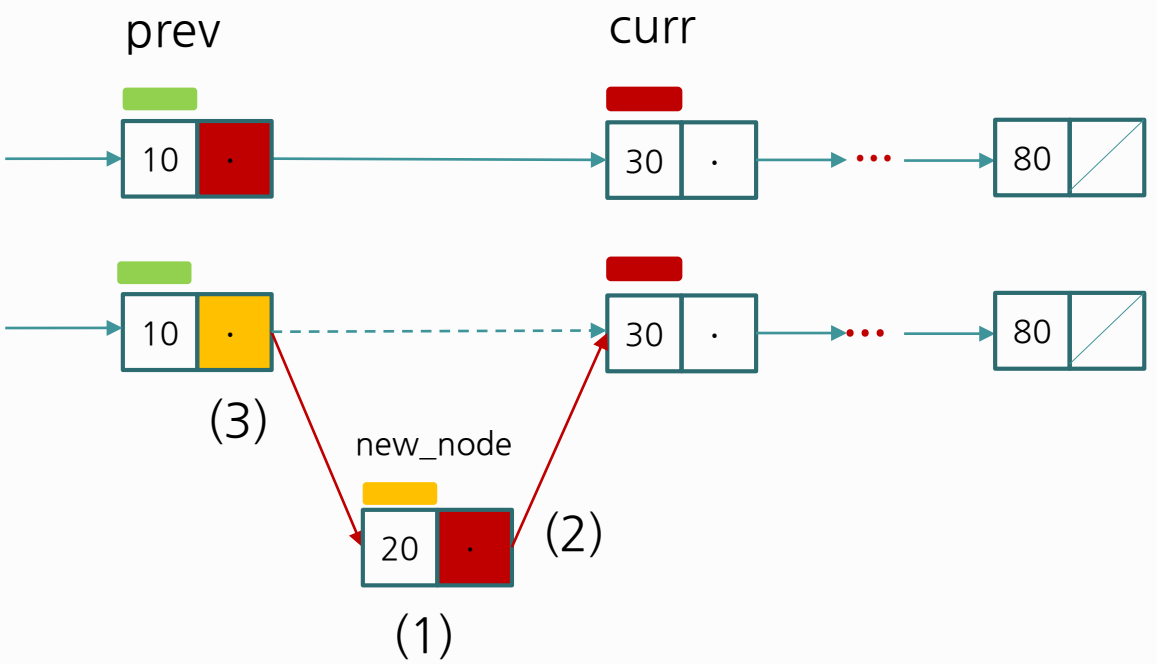


The ListSorted Class - push()

- 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)
```

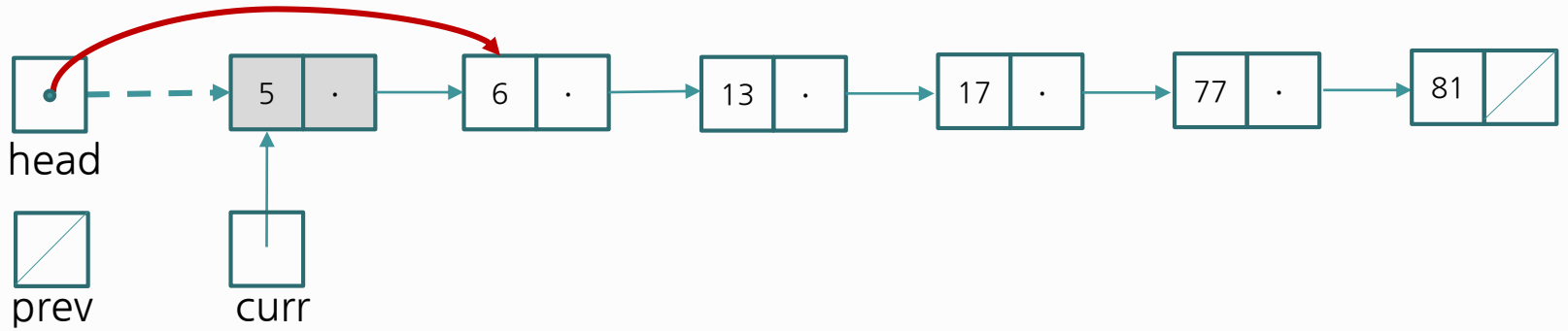




The ListSorted Class - pop()

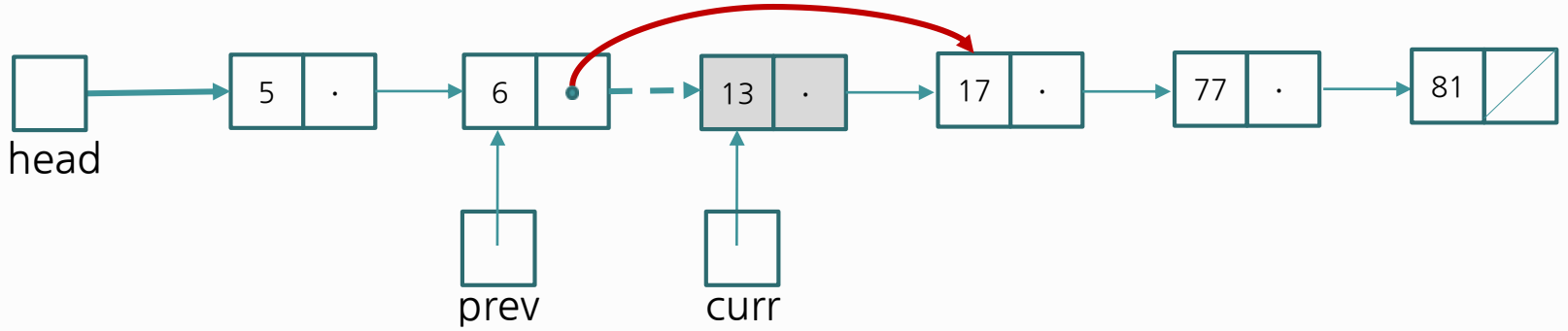
- pop(data) removes a node with data from the list.
  - **What is different from pop() of ListUnsorted class**?
- Examples:
  - Delete the first node.

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



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

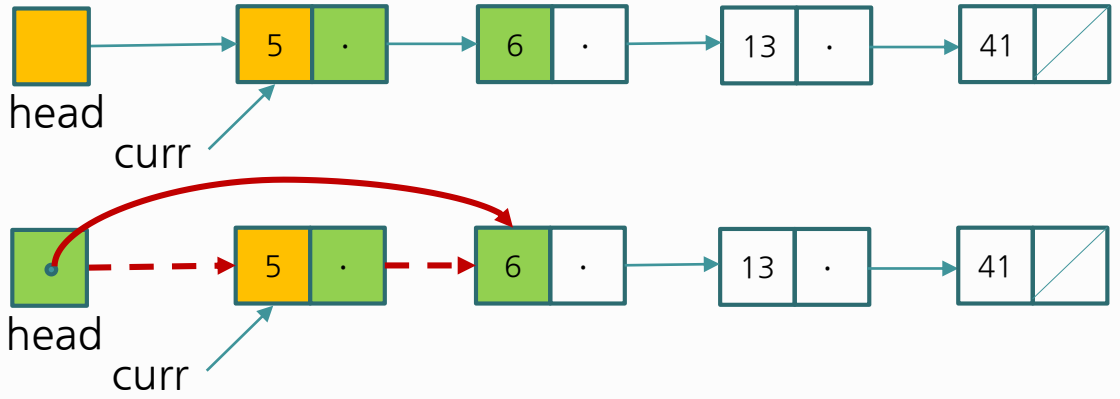
```
mylist.pop(13)
```



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()
```



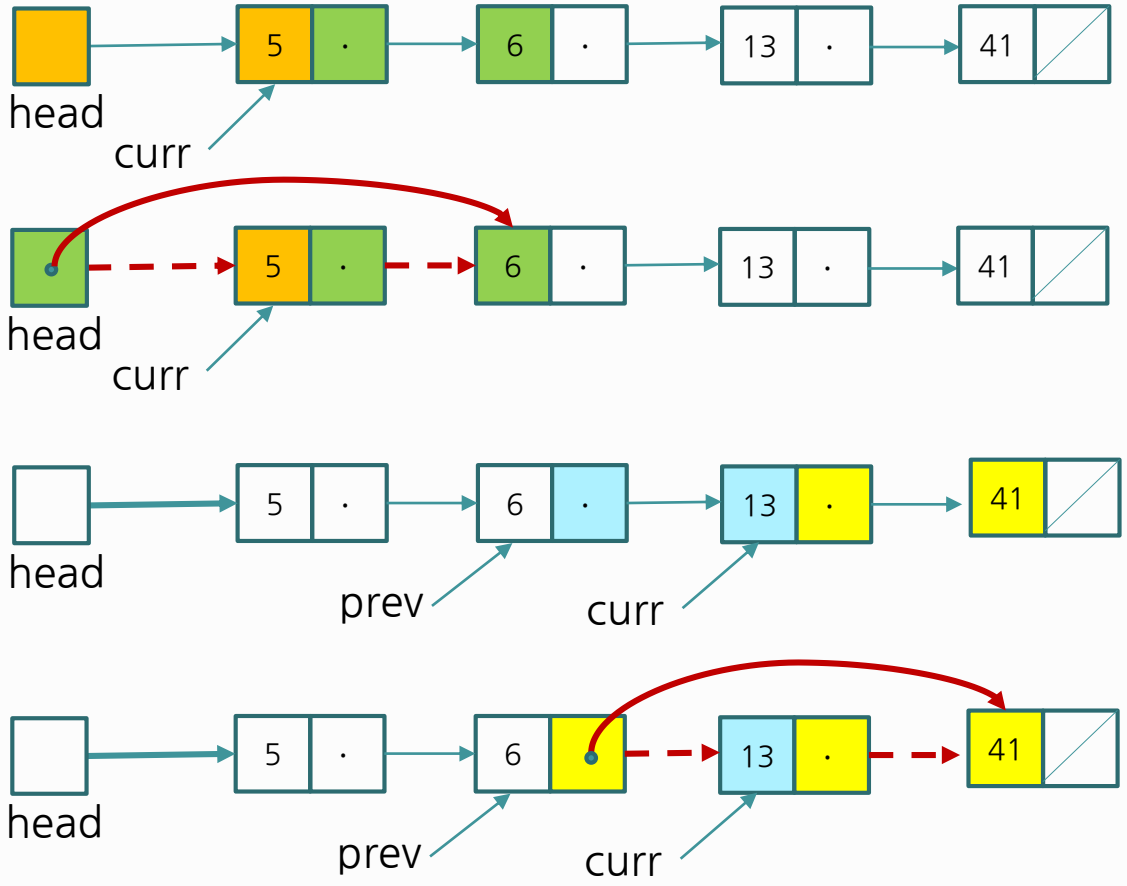
The ListSorted Class - pop()

- 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())
```

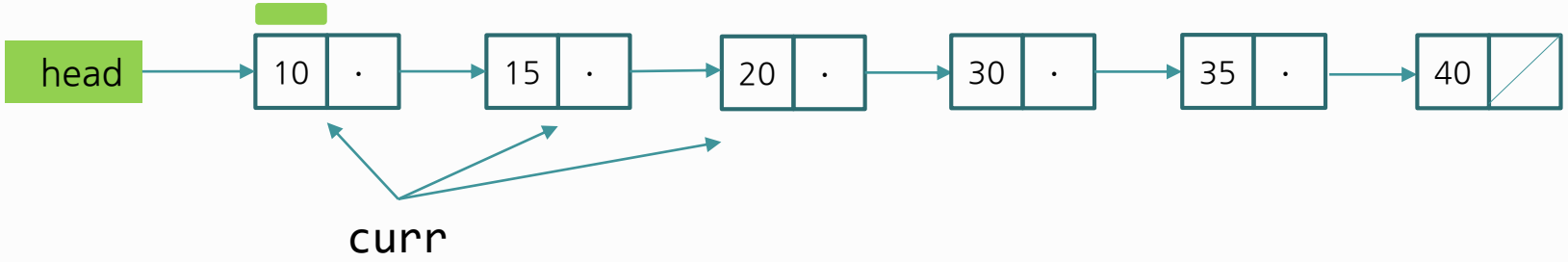


The ListSorted Class - find()

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

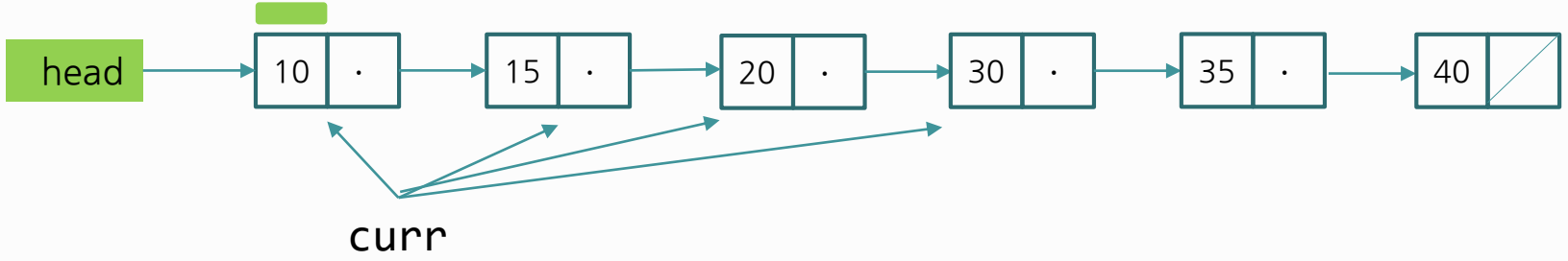
```
print(mylist.find(20))
```

```
True
```




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

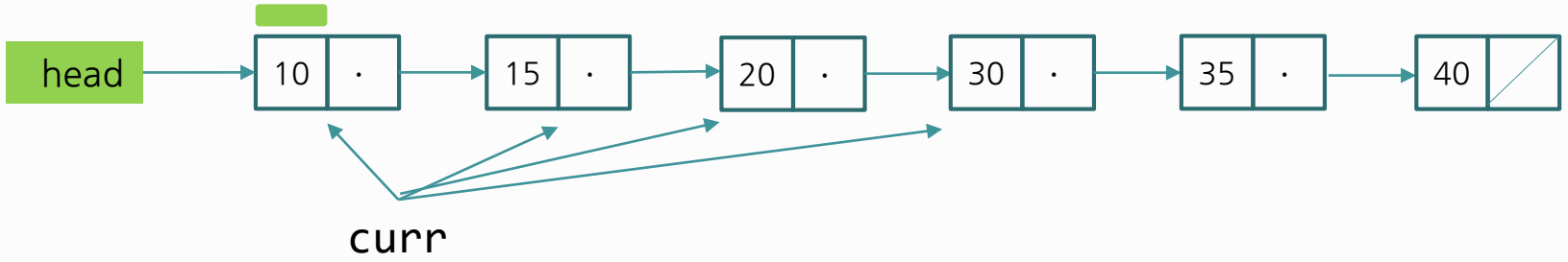
```
False
```



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
```



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)*

Iterator

- Enhancements of LinkedList Classes (LinkedList, ListUnSorted, ListSorted)
  - Adding Count
  - Adding Iterator

The ListUnsorted Class - adding count

- We can add a **count** variable to count the number of nodes in the list.

```
class ListUnsorted(LinkedList):
    def __init__(self):
        ...

    def push(self, data):
        new_node = Node(data)
        ...
        self.count += 1

    def pop(self, data):
        current = self.head
        ...
        self.count -= 1

    def size(self):
        return self.count

    def is_empty(self):
        return self.count == 0
```

Time complexity: O(1)

The ListUnsorted Class - Time Complexity

- Summary

	Python List		ListUnsorted
`if len(mylist) == 0: …`	*O(1)*	__len()__	*O(1)*
`len`	*O(1)*	size	*O(1) with count variable* *O(n) without count variable*
`push()` `insert(i, data)`	*O(1)* *O(n)*	push	*O(1) (beginning of the linked list)*
`pop  del`	*O(n)* *O(n)*	pop	*O(n)*
`in`	*O(n)*	find	*O(n)*

Iterators

- **Traversals** are very common operations, especially on containers.
- Python's for loop allows programmer to traverse items in strings, lists, tuples, and dictionaries:

- Lists

```
for item in [1, 2, 3, 4]:
    print(item)
```

- Tuples

```
for item in (1, 2, 3, 4):
    print(item)
```

- Dictionaries:

```
for key in {'a': 1, 'b':2, 'c':3}:
    print(key)
```

- Strings:

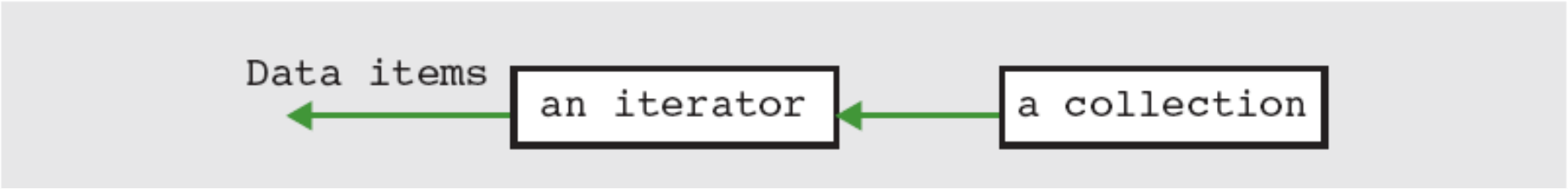
```
for ch in 'hello':
    print(ch)
```

iterable object

```
for <eachItem> in <collection>:
    <do something with eachItem>
```

Iterators

- Python compiler translates for loop to code that uses a special type of object called an **iterator**.



- An iterator guarantees that each element is visited exactly **once**.
  - It is useful to be able to traverse an ListUnsorted or an ListSorted, i.e., visit each element exactly once.
- To explicitly create an iterator, use the built-in **iter** function:

```
it = iter([1, 2, 3])
print(next(it))
print(next(it))
```

```
1
2
```

```
>>> it = iter([1, 2, 3])
>>> print(next(it))
1
>>> print(next(it))
2
```

Iterators

- You can create your own iterators if you write a function to generate the next item.
- You need to add:
  - Constructor
  - The `__iter__()` method, which must return the iterator object.
  - The `__next__()` method, which returns the next element from a sequence.

- For example:

```
obj = MyIterObj(5, 10)
for num in obj:
    print(num, end=" ")
```

```
5 6 7 8 9 10
```

Iterators

- Define the `MyIterObj` class which is **iterable**:

```
class MyIterObj:
    def __init__(self, low, high):
        self.curr = low
        self.high = high

    def __iter__(self):
        return self

    def __next__(self):
        if self.curr > self.high:
            raise StopIteration
        else:
            self.curr += 1
            return self.curr - 1
```

Iterators - Linked List Traversals

- Now, we would like to traverse an `ListUnsorted` or an `ListSorted` using a for-loop, i.e., visit each element exactly once.

```
for num in mylist:
    print(num, end=" ")
```

- However, we will get the following error:

```
for num in mylist:
    print(num, end=" ")
```

```
for num in mylist:
TypeError: 'ListUnsorted' object is not iterable
```

- Solution:
  - Create an iterator class for the linked list
  - Add the `__iter__()` method to returns an instance of the `LinkedListIterator` class

Iterators - The LinkedListIterator

- Define `LinkedListIterator` class that defines an iterator object of the LinkedList.
  - The object stores the head of the list.
  - It implements `__next__()` method that returns data of the current node and advances to the next node.
  - It maintains the reference of the current node.

```
#%%writefile linkedlistIterator.py
class LinkedListIterator:
    def __init__(self, head):
        self.head = head
        self.curr = head
    def __next__(self):
        if self.curr != None:
            data = self.curr.get_data()
            self.curr = self.curr.get_next()
            return data
        else:
            raise StopIteration
```

Iterators - The LinkedListIterator

- Define `__iter__()` method that returns an iterator object of the LinkedList.
  - The iterator has the head of LinkedList and knows how to traverse the list.

```
from linkedlistIterator import LinkedListIterator

class ListUnsorted(LinkedList):
    ...
    def  __iter__(self):
        return LinkedListIterator(self.head)
    ...

class ListSorted(LinkedList):
    ...

    def  __iter__(self):
        return LinkedListIterator(self.head)
    ...
```

```
#%%writefile linkedlistIterator.py
class LinkedListIterator:
    def __init__(self, head):
        self.head = head
        self.curr = head
    def __next__(self):
        if self.curr != None:
            data = self.curr.get_data()
            self.curr = self.curr.get_next()
            return data
        else:
            raise StopIteration
```

Iterators - The LinkedListIterator

- Adding LinkedListIterator in ListUnsorted/ListSorted classes as needed:

```
...

class ListUnsorted(LinkedList):
    ...
    def  __iter__(self):
        return LinkedListIterator(self.head)
    ...

class LinkedListIterator(LinkedList):
    def __init__(self, head):
        self.head = head
        self.curr = head
    def __next__(self):
        if self.curr != None:
            data = self.curr.get_data()
            self.curr = self.curr.get_next()
            return data
        else:
            raise StopIteration
```

Iterators - The LinkedListIterator

- Example:

```
if __name__ == '__main__':
    mylist = ListUnsorted()
    num_list = [24, 65, 12]
    for num in num_list:
        mylist.push(num)

    for num in mylist:
        print(num, end=" ")
```

```
12 65 24
```

Exercise - get_sum() function

- Write a function that returns the sum of the list data.

```
def get_sum(node):
    sum = 0

    # your code here

    return sum
```

```
if __name__ == '__main__':
    mylist = ListSorted()
    num_list = [1, 3, 5]

    for num in num_list:
        mylist.push(num)   # pushing numbers to the linked list

    print(mylist)
    print('sum =', get_sum(mylist.head))
```

```
[5, 3, 1]
sum = 9
```

Summary

- Different implementations may have different time and space complexity.
- The linked-list can be sorted.
- Adding a simple count let `size()` operate in O(1) instead of O(n).
- Adding `__iter__()` function let the user traverse the list using for-loop.

Data Structures in Python
Chapter 3 - 3

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

Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204 Handong Global University