

파이썬으로 배우는 데이터 구조



한동대학교
전산전자공학부
김영섭 교수



학습 목표

ListUnsorted 클래스의 push(), pop(), find()
등 을 구현할 수 있다

Data Structures in Python

Chapter 3 - 3

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

Agenda

- Linked List - Review
- ListUnsorted Implementation
 - ADT
 - Constructor
 - Traversal
 - push(), pop(), find()

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 ListUnsorted Class - ADT

- `ListUnsorted(): __init__()`
 - Creates a new list that is empty and returns an empty list.
- `push(data)`
 - Pushes a new node to the list at the front.
- `pop(data)`
 - Removes the node with data from the list.
- `find(data)`
 - Searches for the data in the list and returns a Boolean value.

The ListUnsorted Class - ADT

- The linked list is built from a collection of nodes, each linked to the next by explicit references.
 - It must maintain a reference to the first node (head).
 - It is commonly known as a **linked list**.
- Examples:
 - An Empty List:

```
mylist = ListUnsorted()
```



The ListUnsorted Class - Constructor

- To use the abstract methods, import The constructor contains
 - A **head** reference variable
 - References the list's first node
 - Always exists even when the list is empty

```
from abc import ABC, abstractmethod
```

```
class LinkedList(ABC):  
    def __init__(self):  
        self.head = None  
    ...
```

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

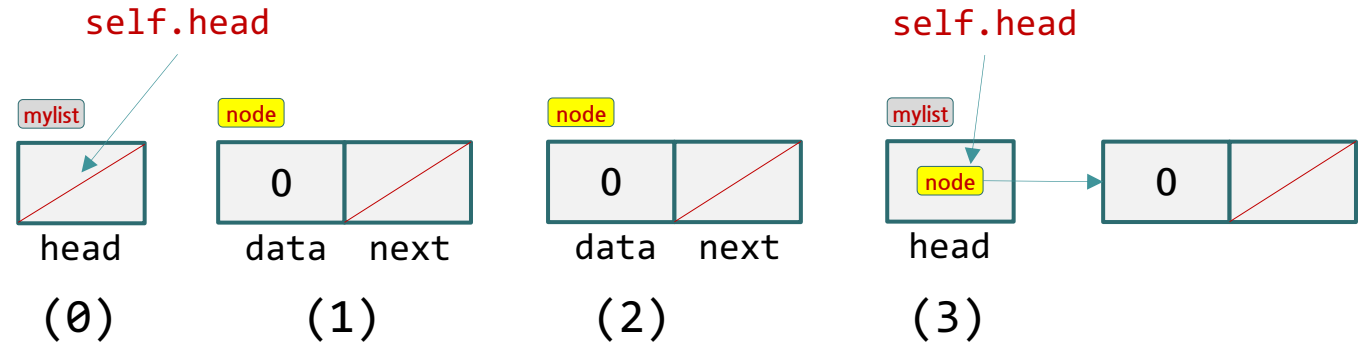
```
if __name__ == '__main__':  
    mylist = ListUnsorted()
```



The ListUnsorted Class

- Example: a linked list of integers

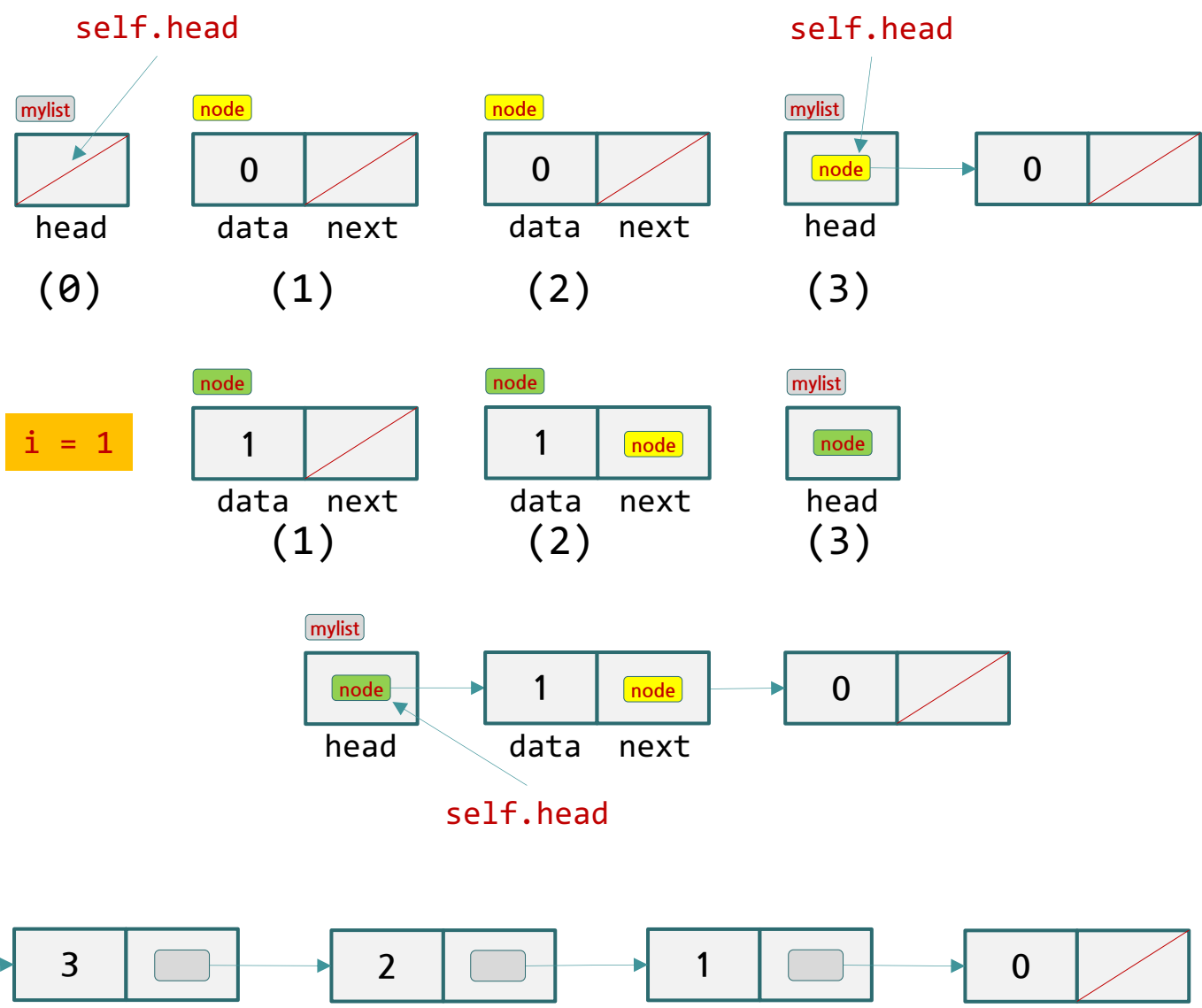
```
class ListUnsorted(LinkedList):  
    ...  
  
    def push(self, data):  
        node = Node(data)  
        node.set_next(self.head)  
        self.head = node  
    ...  
  
(0) if __name__ == '__main__':  
    myList = ListUnsorted()  
    for i in range(6):  
        myList.push(i)
```



The ListUnsorted Class

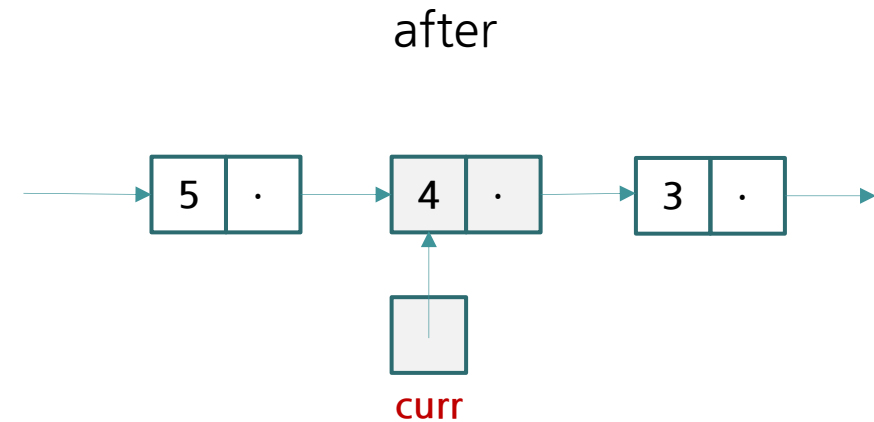
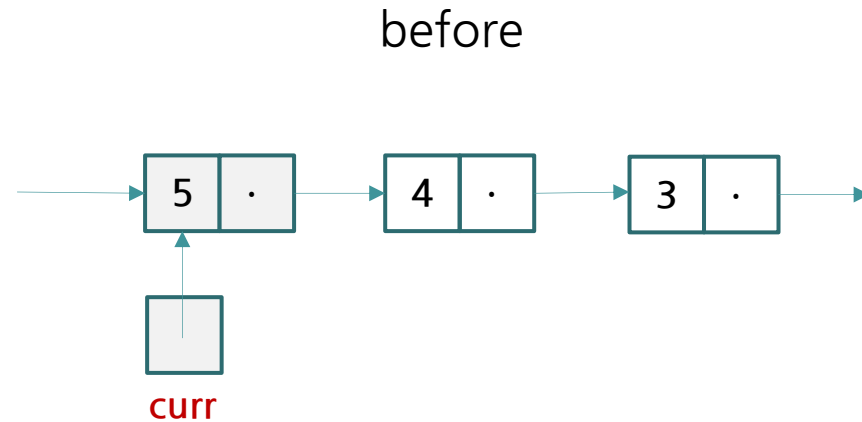
- Example: a linked list of integers

```
class ListUnsorted(LinkedList):  
    ...  
    def push(self, data):  
        node = Node(data)  
        node.set_next(self.head)  
        self.head = node  
    ...  
  
if __name__ == '__main__':  
    mylist = ListUnsorted()  
    for i in range(6):  
        mylist.push(i)
```



The ListUnsorted Class - Traversals

- To traverse a linked list, set a pointer to be the same address as **head**, process the data in the node, move the pointer to the **next** node, and so on.



The ListUnsorted Class - Traversals

- Loop stops when the next pointer is **None**

- Use a reference variable: **curr**
 - References the current node
 - Initially references the first node (head)

```
curr = self.head
```

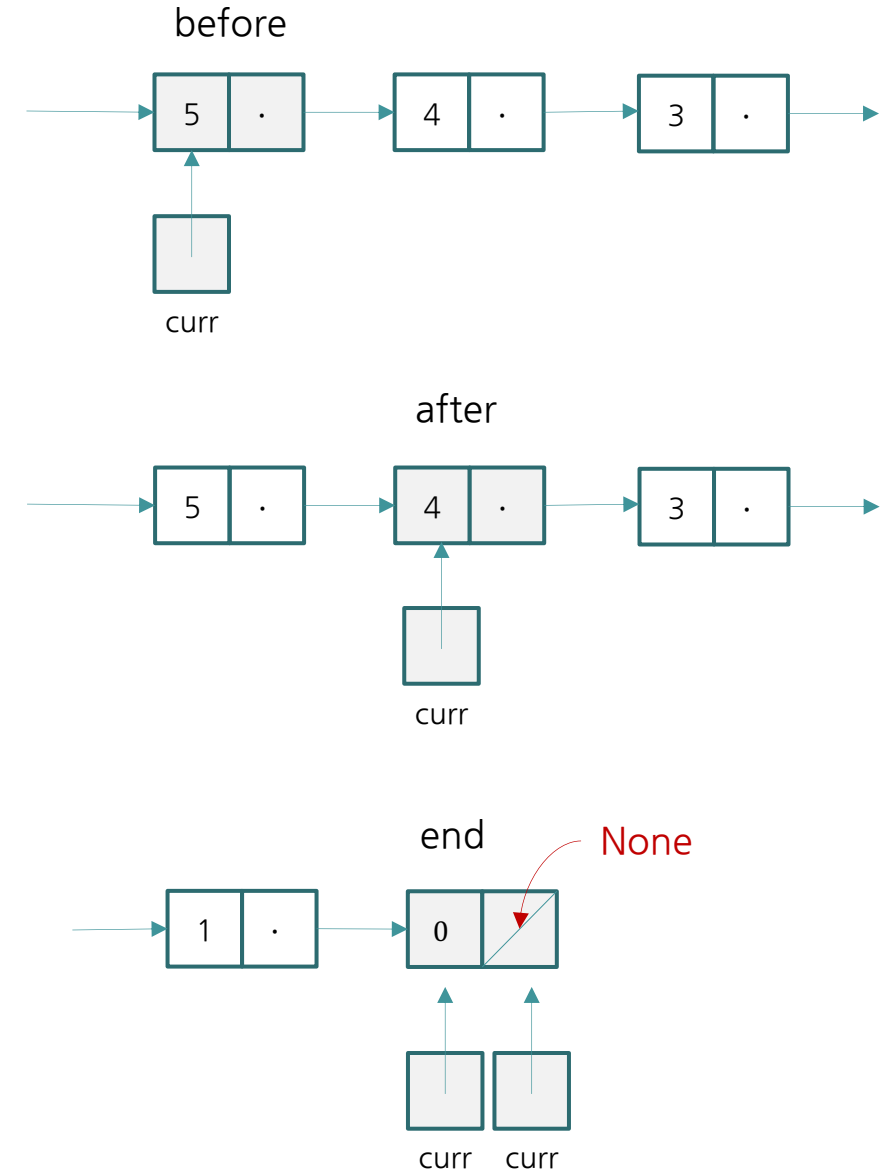
- To advance the current position to the next node

```
curr = curr.get_next()
```

- Loop

```
curr = self.head
while not curr == None:
    ...
    curr = curr.get_next()
```

after while loop, **curr == None**



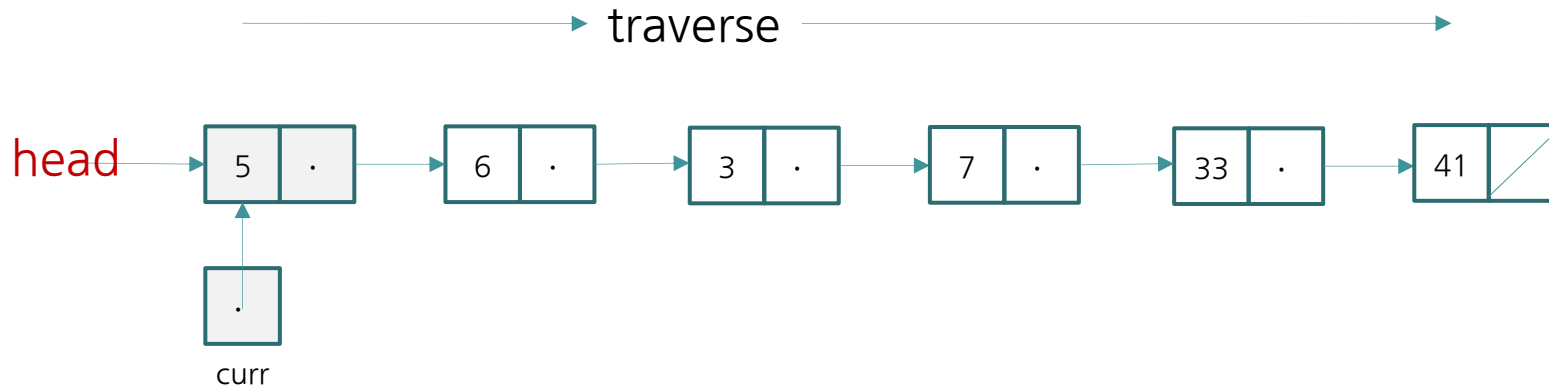
The ListUnsorted Class - Displaying the Contents

- Traversing the Linked List from the **head** to the **tail**
 - Use a reference variable: **curr**
 - This functionality may be implemented in **LinkedList.__str__()**

```
curr = self.head
while curr != None:
    print(curr.get_data(), end=' ')
    curr = curr.get_next()
```

5 6 3 7 33 41

after while curr == None



The ListUnsorted Class - is_empty() & size()

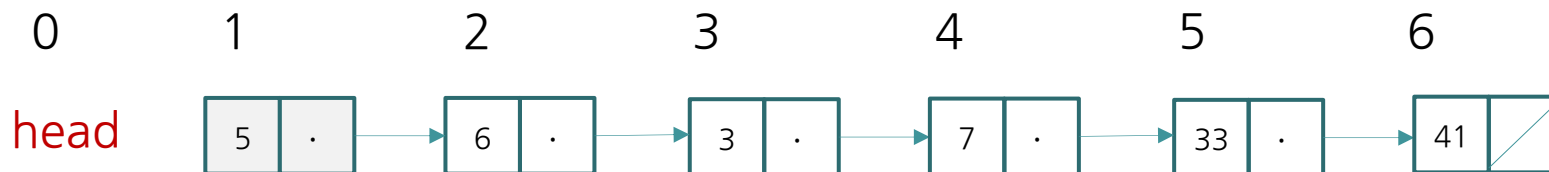
These methods may be implemented in `LinkedList` class.

- `is_empty()` - Tests to see whether the list is empty.

```
return self.head == None
```

- `size()`, `__len__()`
 - Returns the number of nodes in the list.
 - Traverses the list and counts the number of nodes.

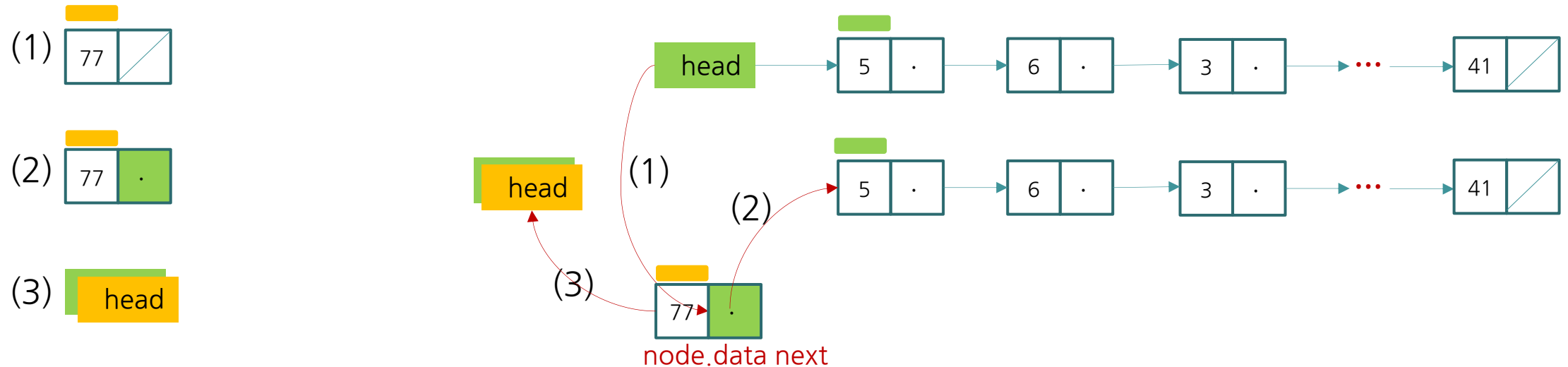
```
curr = self.head
count = 0
while curr != None:
    count = count + 1
    curr = curr.get_next()
```



The ListUnsorted Class - push()

- Push(data) inserts a node with data **at the beginning** of a linked list
 - Create a new Node and store the new data into it.
 - Connect the new node to the linked list by changing references.
 - Change the **next** reference of the new node to refer to the old first node of the list.
 - Modify the **head** of the list to refer to the new node.

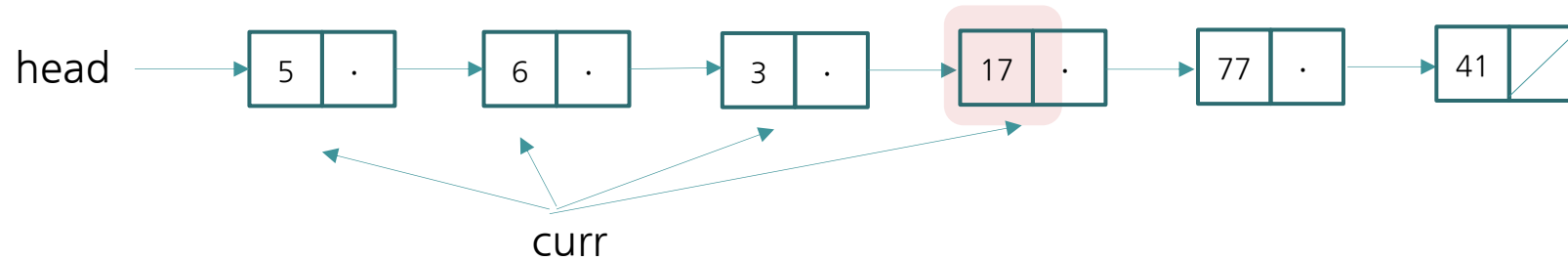
```
(1) node = Node(77)
(2) node.set_next(self.head)
(3) self.head = node
```



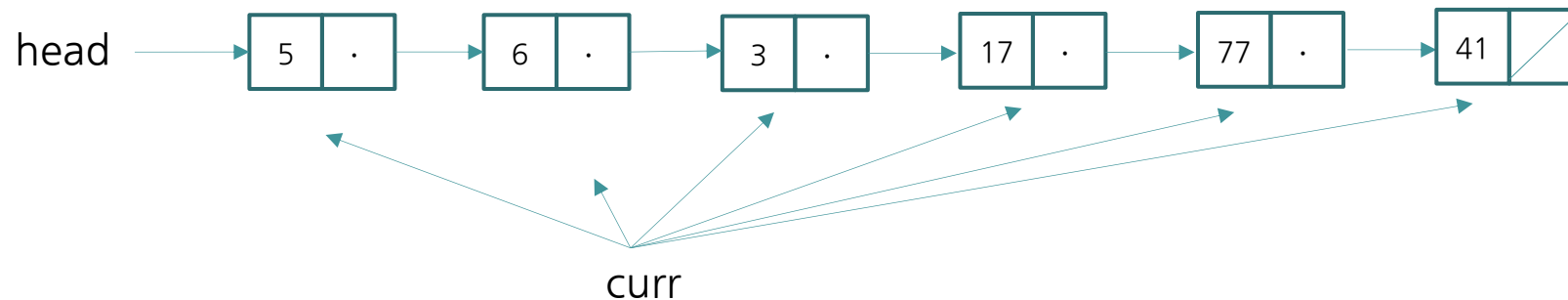
The ListUnsorted Class - find()

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

```
print(mylist.find(17))
```



```
print(mylist.find(19))
```



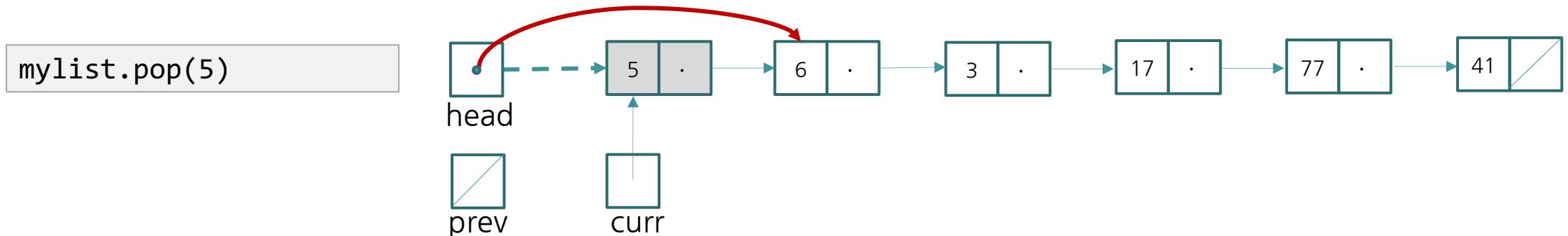
The ListUnsorted Class - find()

- ▶ To find a node in a linked list:
 - Set a pointer to be the same address as head.
 - Process the data in the node, move the pointer to the next node, and so on.
 - Loop stops either
 - The data is found.
 - The next pointer is None.

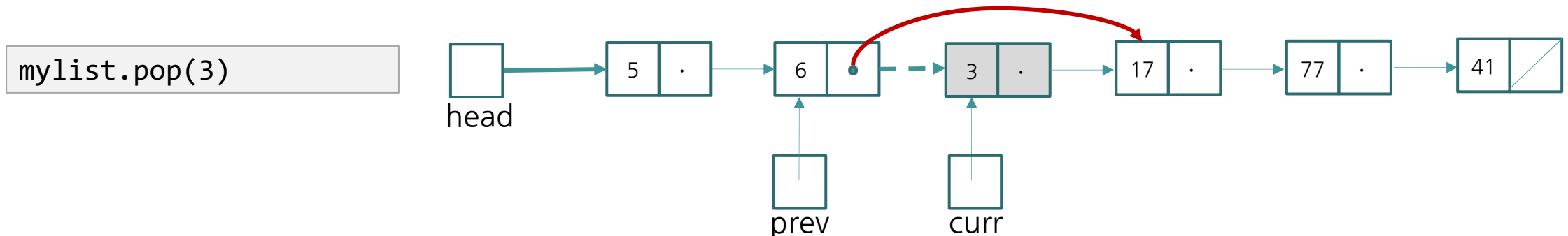
```
def find(self, data):  
    curr = self.head  
    while curr != None:  
        if curr.get_data() == data:  
            return True  
        curr = curr.get_next()  
    return False
```

The ListUnsorted Class - pop()

- ▶ `pop(data)` removes a node with data from the list.
 - ▶ Assume the node to pop is present in the list.
- ▶ Examples:
 - ▶ Delete the first node.



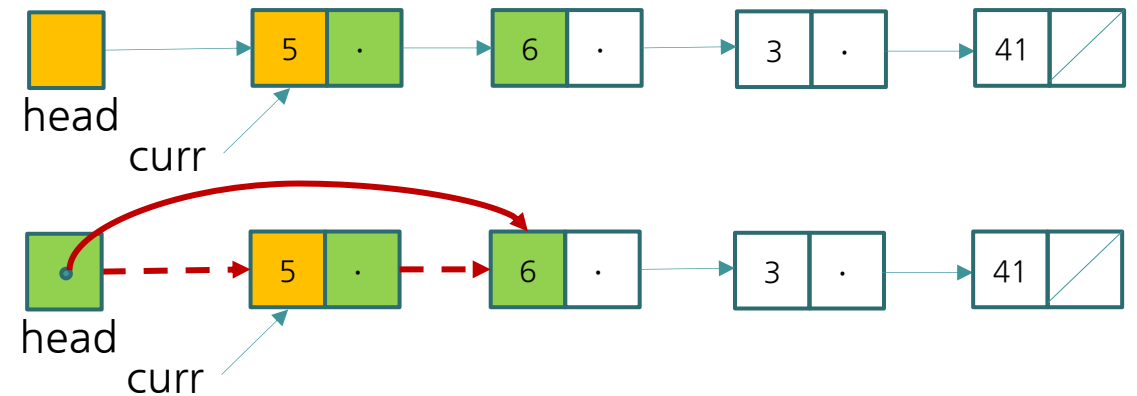
- ▶ Delete a node in the middle of the list with **prev** and **curr** references.



The ListUnsorted 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()
```



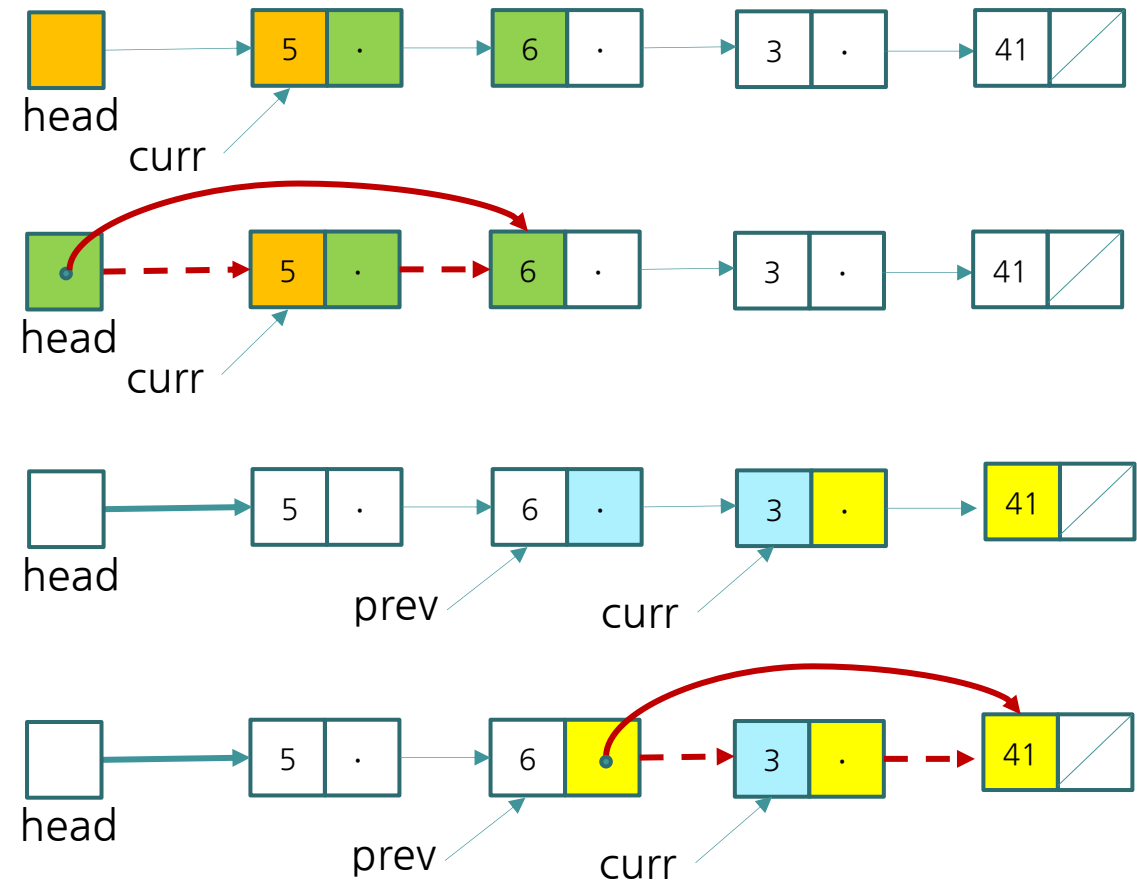
The ListUnsorted Class - pop()

- ▶ 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 ListUnsorted Class - Example

- Example:

```
def TestListUnsorted():  
    mylist = ListUnsorted()  
    number_list = [31, 77, 17, 93, 26, 54]  
    for num in number_list:  
        mylist.push(num)  
  
    print (mylist.size())  
    print (mylist.find(17))  
    print (mylist.find(1))  
    mylist.pop(31)  
    mylist.pop(54)  
    print (mylist.size())
```



6
True
False
4

Summary

- Any element in a list can be accessed; however, you must traverse a linked list to access a particular node.
- A node can be inserted into and deleted from a reference based linked list without shifting nodes.

학습 정리

- 1) ListUnsorted 클래스에서 push()는 보통 reference를 덮어쓰는 방식이다
- 2) ListUnsorted 클래스에서 pop()을 구현할 때, 이전 노드의 reference가 필요할 수도 있으니 미리 저장해야 한다
- 3) LinkedList 에서는 노드를 추가하거나 삭제해도 기존의 노드들을 이동할 필요가 없다

파이썬으로 배우는 데이터 구조

수고했습니다
곧 다음 시간에
다시 뵙겠습니다

