빅 데 이 터 혁 신 공 유 대 학

파이썬으로 배우는 데이터 구조

한동대학교 전산전자공학부

김영섭 교수

교육부   NRF 한국연구재단   COSS   HANDONG GLOBAL UNIVERSITY

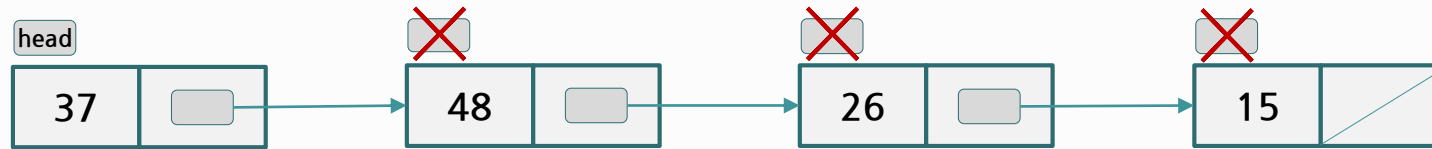# Data Structures in Python
Chapter 3 - 3

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

Agenda

- Linked List – Review
- ListUnsorted vs. ListSorted
- Superclass and Subclasses
  - Inheritance
  - Abstract Base Classes

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

3

Linked List - Review

- Step 1: Code a singly linked list. The first node is called `'head'`.


```python
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None




if __name__ == '__main__':

```
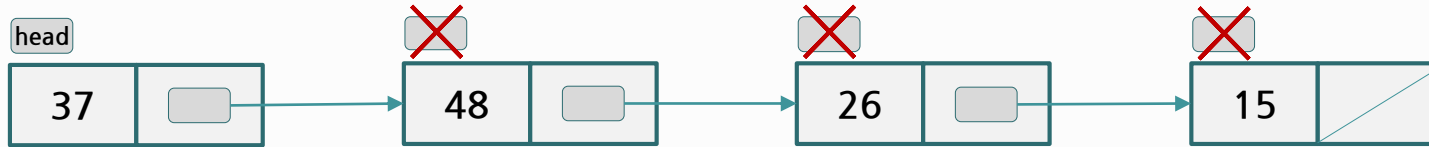
Linked List - Review

- Step 1: Code a singly linked list. The first node is called `'head'`.
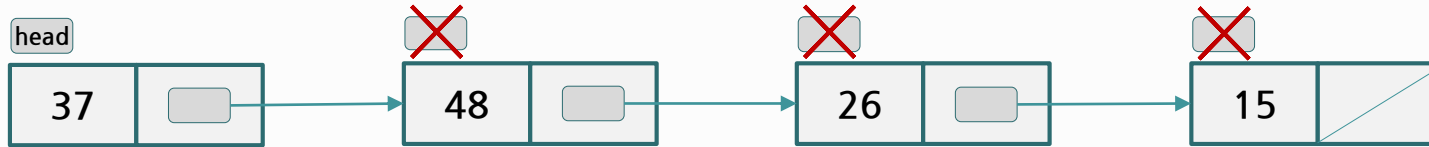


```python
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None




if __name__ == '__main__':
    head = Node(37)
    head.next = Node(48)
    head.next.next = Node(26)
    head.next.next.next = Node(15)
```

Linked List - Review

- Step 2: Traverse the whole list if the first node reference or the head is known.


```python
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

    def get_data(self):
        return self.data

    def get_next(self):
        return self.next

if __name__ == '__main__':
    head = Node(37)
    head.next = Node(48)
    head.next.next = Node(26)
    head.next.next.next = Node(15)
```

```python
def print_chain(head):
```

```
37 48 26 15
```

Linked List - Review

- Step 2: **Traverse the whole list** if the first node reference or the **head** is known.


```python
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

    def get_data(self):
        return self.data

    def get_next(self):
        return self.next

if __name__ == '__main__':
    head = Node(37)
    head.next = Node(48)
    head.next.next = Node(26)
    head.next.next.next = Node(15)
```
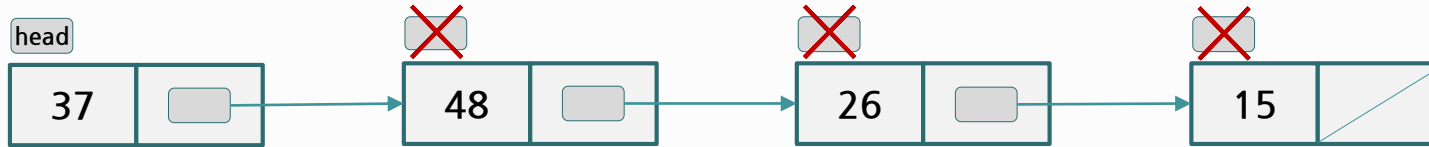
```python
def print_chain(head):
    while not head == None:
        print(head.get_data(), end = " ")
        head = head.get_next()
```

```
37 48 26 15
```

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

7

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), push_back(data)`
  - Pushes a new node with data to the list.
- `pop(data)`
  - Removes the node with data from the list.
- `find(data)`
  - Searches for the data in the list and **returns** a Boolean value.

ListUnsorted vs. ListSorted

- In general, a linked list is formed as each node with data is provided.
- Let us suppose that the user wants to create two different kinds of linked lists, either sorted or unsorted.
  - Then, we must design two different classes: ListUnsorted and ListSorted
  - If we create two different classes as requested, many parts of the code between two classes will be identical and violating the DRY coding principle.

Unsorted: 54 → 27 → 93 → 17 → 77 → 31

Sorted: 17 → 27 → 31 → 54 → 77 → 93
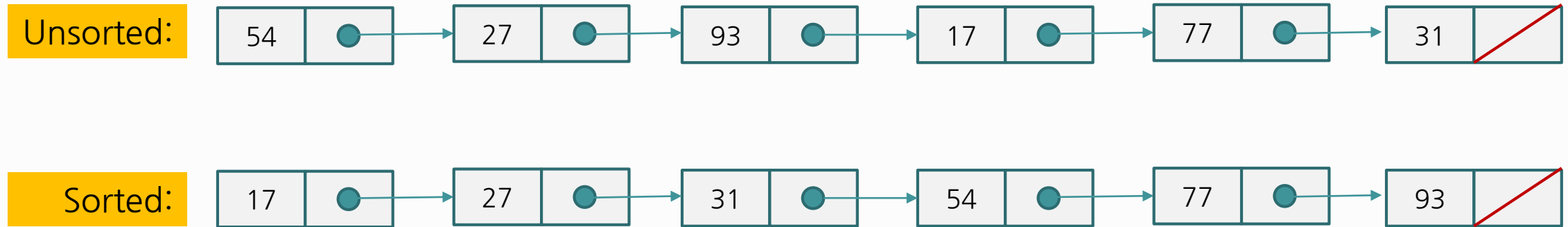
Superclass and Two Subclasses

- Goals: Two Different Kinds of Linked Lists
  - Create a superclass: `LinkedList`
  - Create two subclasses: `ListUnsorted` and `ListSorted`
  - Use the inheritance of Object-Oriented Programming.

Base Class	Superclass	Parent Class
↑	↑	↑
Derived Class	Subclass	Child Class

```
          LinkedList
              ↑
      ┌───────┴───────┐
ListUnsorted      ListSorted
```

LinkedList ADT
LinkedList()
is_empty()
size()
push(data), push_back(data)
pop(data)
find(data)
__len__()
__str__()

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

10

Superclass and Two Subclasses

- Determine the methods not to implement at the level of LinkedList
  - Those methods become **abstract methods** in LinkedList.
  - The **subclasses** of LinkedList must implement those methods, respectively.

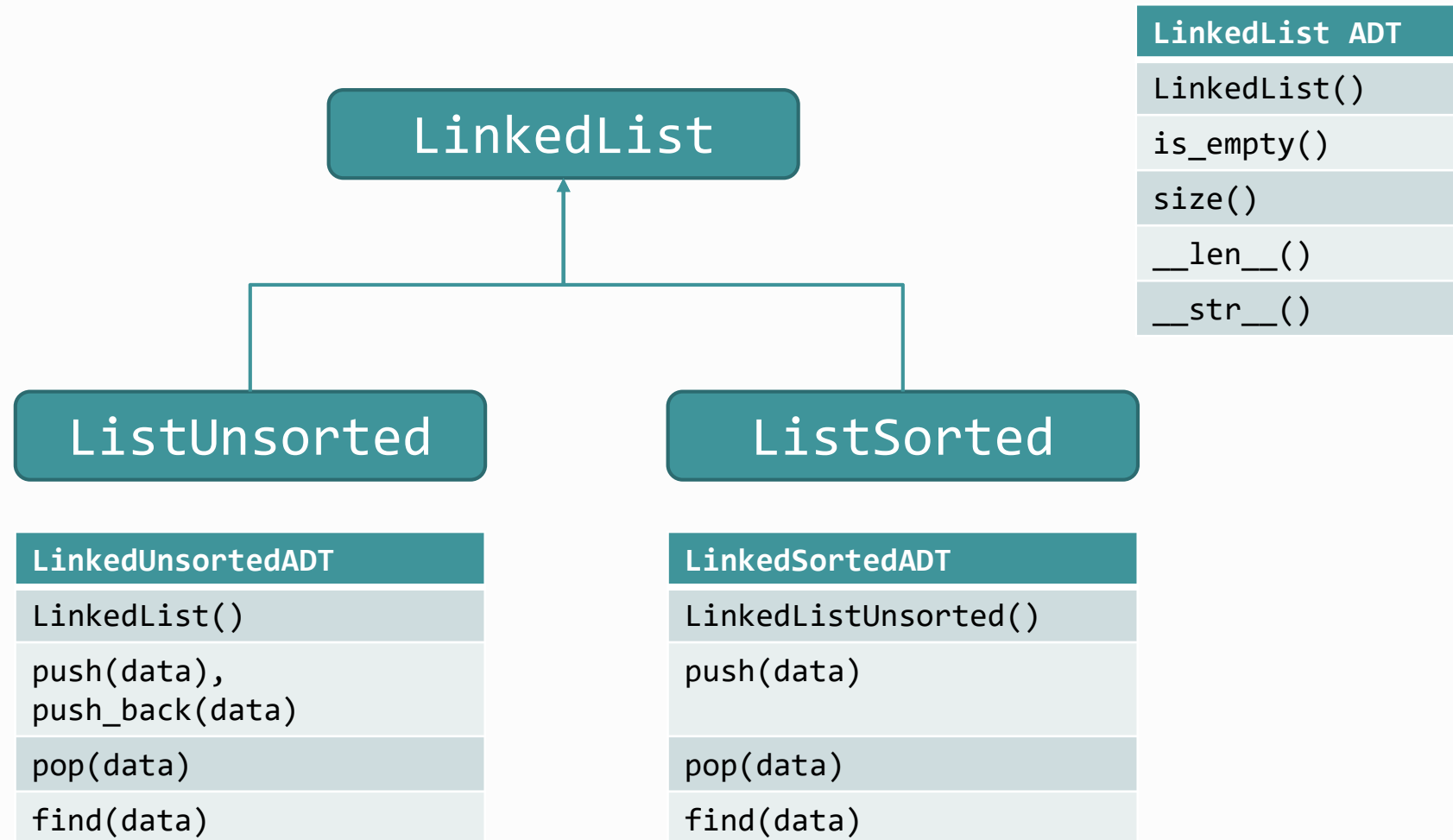| LinkedList ADT	LinkedList ADT	ListUnsorted ADT	ListSoted ADT
LinkedList()	LinkedList()	ListUnsorted()	ListSorted()
is_empty()			
size()			
push(data)			
push_back(data)		push_back(data)	
pop(data)			
find(data)			
__len__()			
__str__()			

Superclass and Two Subclasses

- Determine the methods not to implement at the level of LinkedList
  - Those methods become **abstract methods** in LinkedList.
  - The **subclasses** of LinkedList must implement those methods, respectively.

LinkedList ADT	LinkedList ADT	ListUnsorted ADT	ListSoted ADT
LinkedList()	LinkedList()	ListUnsorted()	ListSorted()
is_empty()	is_empty()		
size()	size()		
push(data)		push(data)	push(data)
push_back(data)		push_back(data)	n/a
pop(data)		pop(data)	pop(data)
find(data)		find(data)	find(data)
__len__()	__len__()		
__str__()	__str__()		

Superclass and Two Subclasses

- Identify some necessary methods that the subclasses must implement to make LinkedList to work properly.

LinkedList ADT
LinkedList()
is_empty()
size()
__len__()
__str__()

```
LinkedList
```

```
ListUnsorted          ListSorted
```

LinkedUnsortedADT
LinkedList()
push(data), push_back(data)
pop(data)
find(data)

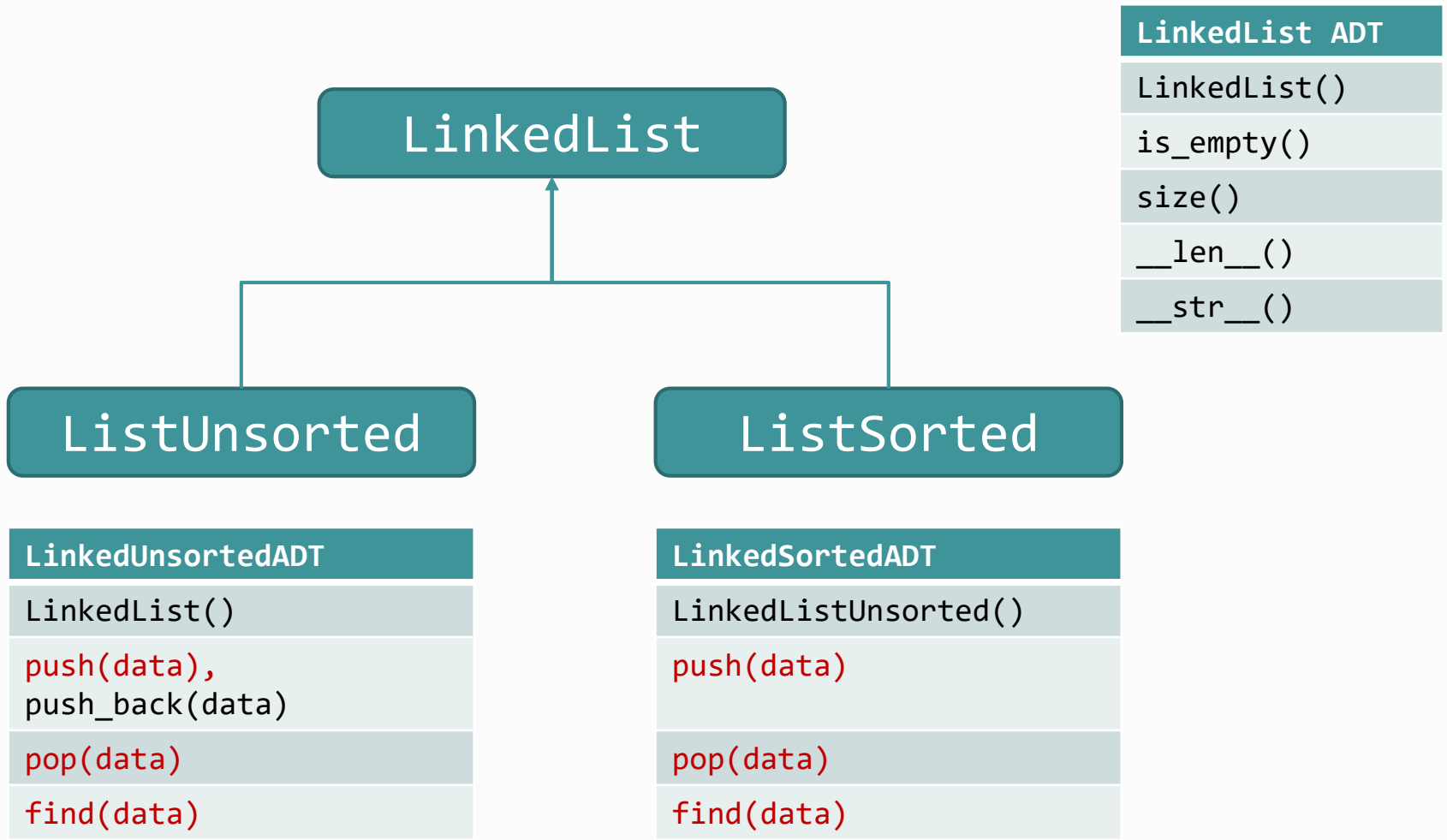| LinkedSortedADT |
| --- |
| LinkedListUnsorted() |
| push(data) |
| pop(data) |
| find(data) |

Superclass and Two Subclasses

- Identify some necessary methods that the subclasses must implement to make LinkedList to work properly.

LinkedList ADT
LinkedList()
is_empty()
size()
__len__()
__str__()

```
              LinkedList
```

LinkedUnsortedADT
LinkedList()
push(data), push_back(data)
pop(data)
find(data)

ListUnsorted

LinkedSortedADT
LinkedListUnsorted()
push(data)
pop(data)
find(data)

ListSorted

Superclass and Two Subclasses

- As a superclass of the view, it would be better if there is a way to **force the subclasses to implement** those necessary methods.

Superclass and Two Subclasses

- As a superclass of the view, it would be better if there is a way to **force the subclasses to implement** those necessary methods.

- In Python, **abstract base classes (i.e., superclass)** provide a blueprint for **concrete classes (i.e., subclass)**.

- The superclass do not contain implementation.
  - Instead, It provides an interface and make sure that derived concrete classes (subclasses) are properly implemented.

- **Abstract base classes cannot be instantiated**.
  - Instead, they are inherited and extended by the concrete subclasses.

- Subclasses derived from a specific abstract base class **must implement the methods** and properties provided in that abstract base class. Otherwise, an error is raised during the object instantiation.

Superclass and Two Subclasses

▪ Use '**abc**' module in the to define abstract base class.
  ▪ Define abstract methods in LinkedList class and it becomes an abstract base class.
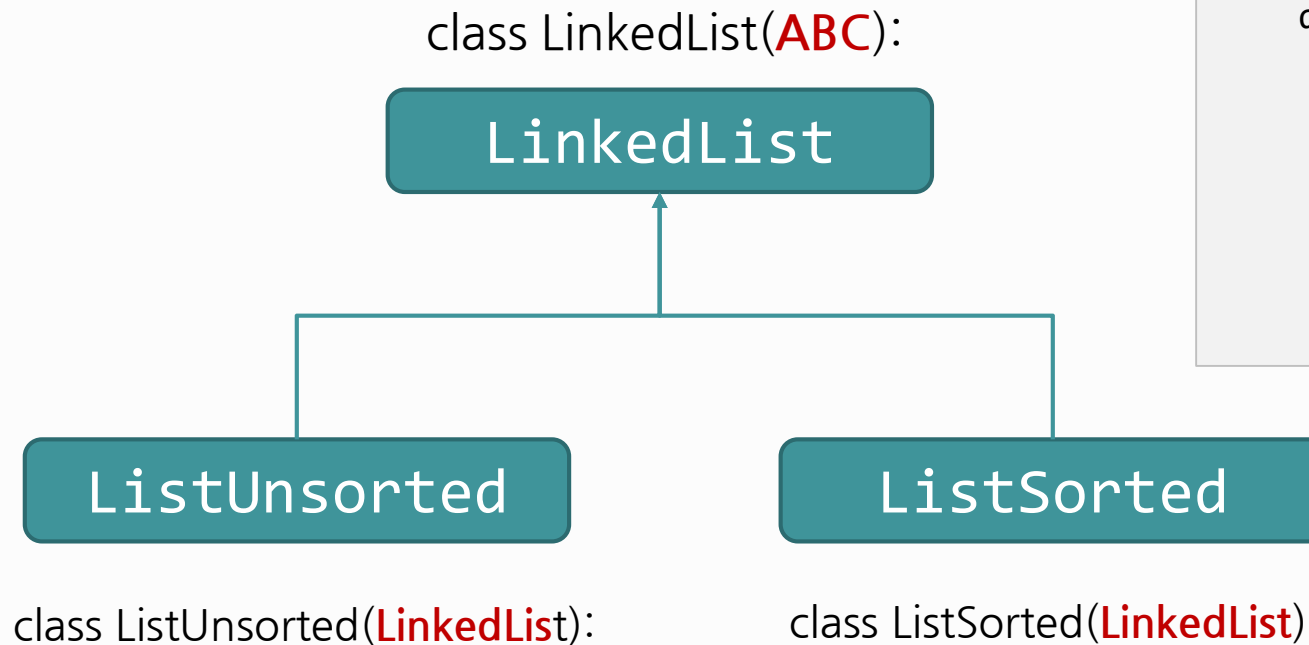
```
from abc import ABC

class LinkedList(ABC):

    ...
    @abstractmethod
    def push(self, data):
        pass
    @abstractmethod
    def pop(self, data):
        pass
    @abstractmethod
    def find(self, data):
        pass
```

class LinkedList(**ABC**):

```
LinkedList
```

```
ListUnsorted          ListSorted
```

class ListUnsorted(**LinkedLis**t):   class ListSorted(**LinkedList**):

Superclass and Two Subclasses

- The 'abc' module in the Python library provides the infrastructure for defining custom abstract base classes. To use abstract base classes, `from abc import ABC`.

```python
from abc import ABC, abstractmethod

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

    ...


    @abstractmethod
    def push(self, data):
        pass


    @abstractmethod
    def pop(self, data):
        pass


    @abstractmethod
    def find(self, data):
        pass
```

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

    ...


    def push(self, data):
        pass


    def pop(self, data):
        pass


    def find(self, data):
        pass
```

```python
class ListSorted(LinkedList):
    def __init__(self):
        LinkedList.__init__(self)

    ...
    ...
```

Summary

- We may use **OOP inheritance** to prevent from duplicating the code and to maximize the reusability of the code.

- The '**abc**' module in the Python library provides the infrastructure for defining custom abstract base classes.

Data Structures in Python
Chapter 3 - 3

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