

Data Structures in Python

Chapter 3

- Linked List
- OOP Inheritance
- ListUnsorted Class
- ListSorted Class
- Iterator
- Doubly Linked List - Structures
- Doubly Linked List - Operations
- **Doubly Linked List - DequeCircular**

Agenda

- Define DequeCircular (a deque) using DoublyLinked
 - Singly Linked List vs. Doubly Linked List
 - Exercises in DoublyLinked
 - Override `__str__()` and `__iter__()`
 - Define `reverse()`
 - Enhance constructor `__init__()`
 - Define DequeCircular Class Using DoublyLinked
 - Deque in Python - Review
 - Superclass vs subclass
 - DequeCircular Class - ADT
 - Constructor: `__init__(items=[], maxlen=None)`
 - Methods: `rotate(n = 1)`

Singly Linked List vs Doubly Linked List

- Summary:

Singly Linked List	Doubly Linked List
Each node consists of a data value and a pointer to the next node.	Each node consists of a data value, a pointer to the next node, and a pointer to the previous node.
Traversal can occur in one way only (forward direction).	Traversal can occur in both ways.
It requires less space.	It requires more space because of an extra pointer.
It can implement the stack and queue.	It has multiple usages. It can be implemented on the stack, queue, deque, heap, and binary tree.
The time complexity of most operations is $O(n)$.	The time complexity of most operations is $O(1)$ except insertion of which is $O(n)$.

Overriding Methods: `__str__()`

- **Exercise 1:** `__str__()` returns a human readable string for the list. The Node class simply returns a string for **data** field, excluding both **prev** and **next** fields.

```
class Node:
    ...
    def __str__(self):
        return str(self.__data)
```

```
class DoublyLinked:
    ...
    def __str__(self):
        lstr = "["
        curr = self.begin()
        while curr != self.end():
            # your code here
            curr = curr.get_next()
        return lstr + "]"
```

[9, 6, 3]

Overriding Methods: `__iter__()`

- **Exercise 2:** `__iter__()` returns an iterable object that contains the current object to iterate.
 - Create a list iterator class as an inner class of `DoublyLinked` class.
 - Define `__iter__()` in `DoublyLinked` class such that it instantiates the list iterator class object and returns it.

```
class DoublyLinked:
    ...
    def __iter__(self):
        return self.Listerator(self)
```

```
class DoublyLinked:
    ...
    class ListIterator:
        def __init__(self, urlist):
            self.curr = urlist.begin()

        def __next__(self):
            if self.curr.next is None:
                raise StopIteration
            else:
                # your code here
                return data
```


Rewrite Method: `__str__()`

- **Exercise 3:** `__str__()` returns a human readable string for the list. Rewrite the `__str__()` code to take an advantage of the iterable object of **DoublyLinked**.
 - Make the object iterable of DoublyLinked first. Then work on this problem.

```
class Node:  
    ...  
    def __str__(self):  
        return str(self.__data)
```

```
class DoublyLinked:  
    ...  
    def __str__(self):  
        lstr = '['  
        curr = self.begin()  
        while curr != self.end():  
            # your code here  
            curr = curr.get_next()  
        return lstr + ']'
```

[9, 6, 3]



```
class DoublyLinked:  
    ...  
    def __str__(self):  
        lstr = # your code here  
        return '[' + lstr + ']'
```

[9, 6, 3]

Enhance constructor: `__init__()`

- **Exercise 4:** It would be exceptionally handy if the constructor accepts the following formats of initialization.

```
print(DoublyLinked())  
print(DoublyLinked('a'))  
print(DoublyLinked('abc'))  
print(DoublyLinked(['world', 'is', 'small']))  
print(DoublyLinked(range(10)))  
print(DoublyLinked(random.sample(range(10), 10)))
```

```
[]  
[a]  
[a, b, c]  
[world, is, small]  
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]  
[7, 1, 0, 8, 6, 4, 9, 3, 5, 2]
```

Enhance constructor: `__init__()` solution

- **Exercise 4:** It would be exceptionally handy if the constructor accepts the following formats of initialization.

```
print(DoublyLinked())
print(DoublyLinked('a'))
print(DoublyLinked('abc'))
print(DoublyLinked(['world', 'is', 'small']))
print(DoublyLinked(range(10)))
print(DoublyLinked(random.sample(range(10), 10)))
```

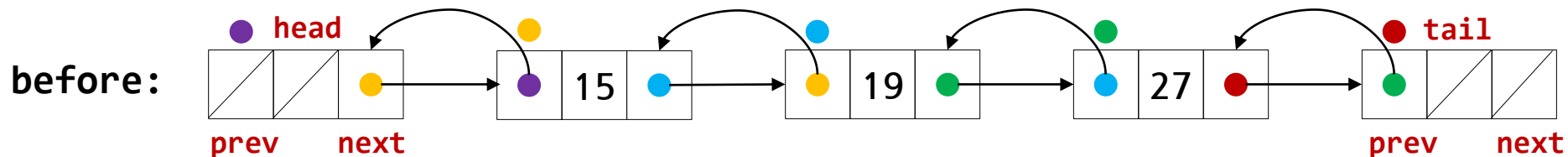
```
[]
[a]
[a, b, c]
[world, is, small]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[7, 1, 0, 8, 6, 4, 9, 3, 5, 2]
```

```
from collections.abc import Iterable
...
def __init__(self, items = []):
    self.__head = self.Node()
    self.__tail = self.Node()
    self.__head.next = self.__tail
    self.__tail.prev = self.__head

    if not isinstance(items, Iterable):
        items = [items]
    for item in items:
        self.insert(item, self.end()) # add it to the end like list
                                     # use self.end() instead of self.tail
```

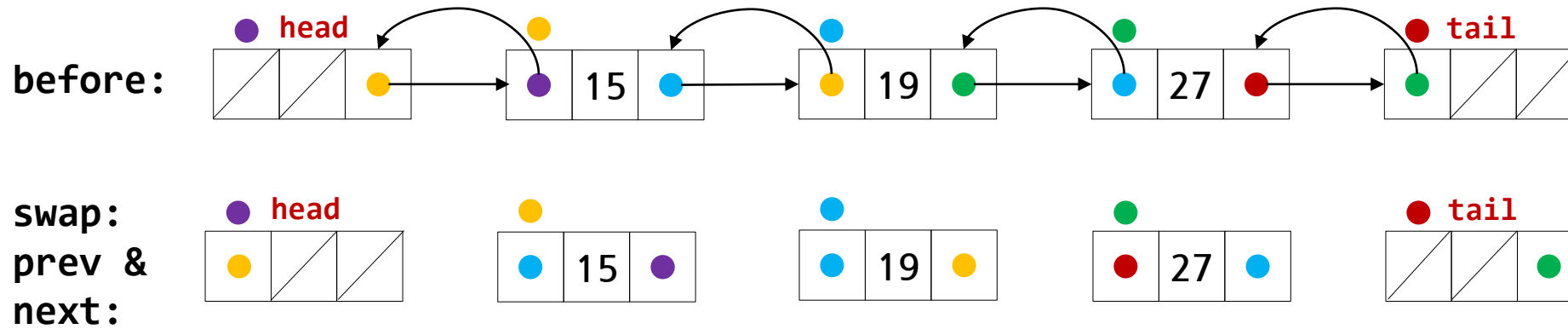

Exercise: reverse()

- It reverses the order of nodes in the list in-place and then return None.
 - No change is made if the list is empty.
 - The entire operation does not involve the construction or destruction of any node.
 - The nodes are not moved, but links are moved within the list.
 - Its time complexity is $O(n)$.
 - Algorithm:
 - Step 1: swap **prev** and **next** in every node including two sentinel nodes.
 - Step 2: swap **head** and **tail** node.



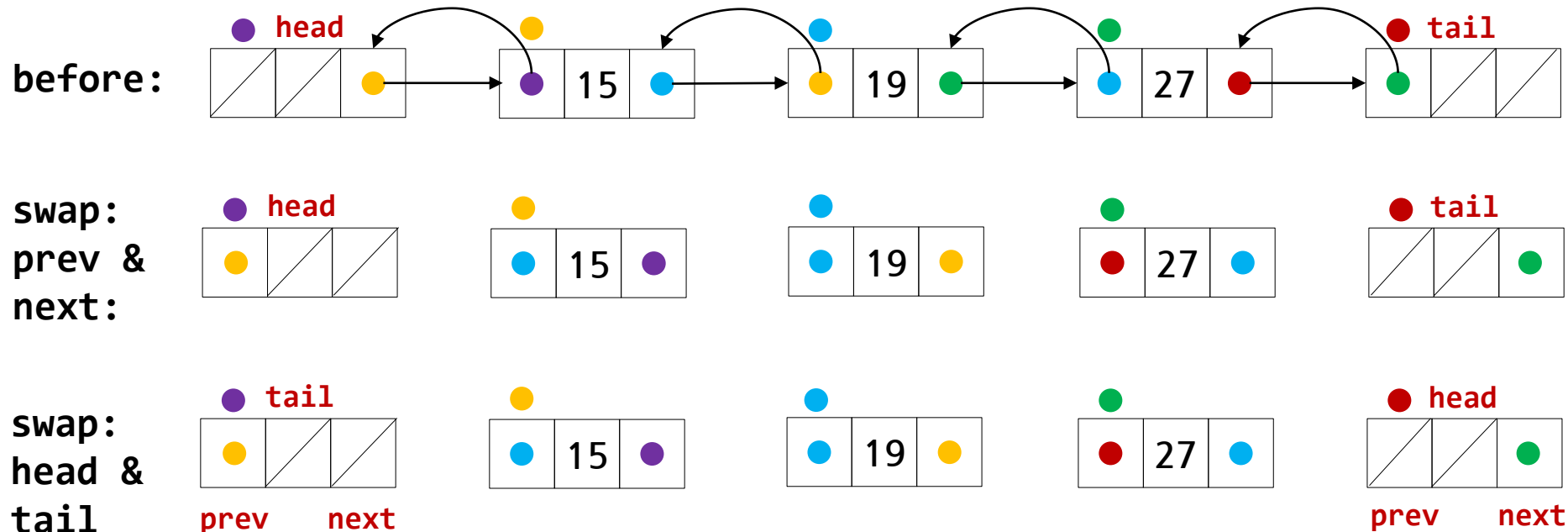
Exercise: reverse()

- Algorithm:
 - Step 1: swap **prev** and **next** in every node including two sentinel nodes.
 - Step 2: swap **head** and **tail** node.



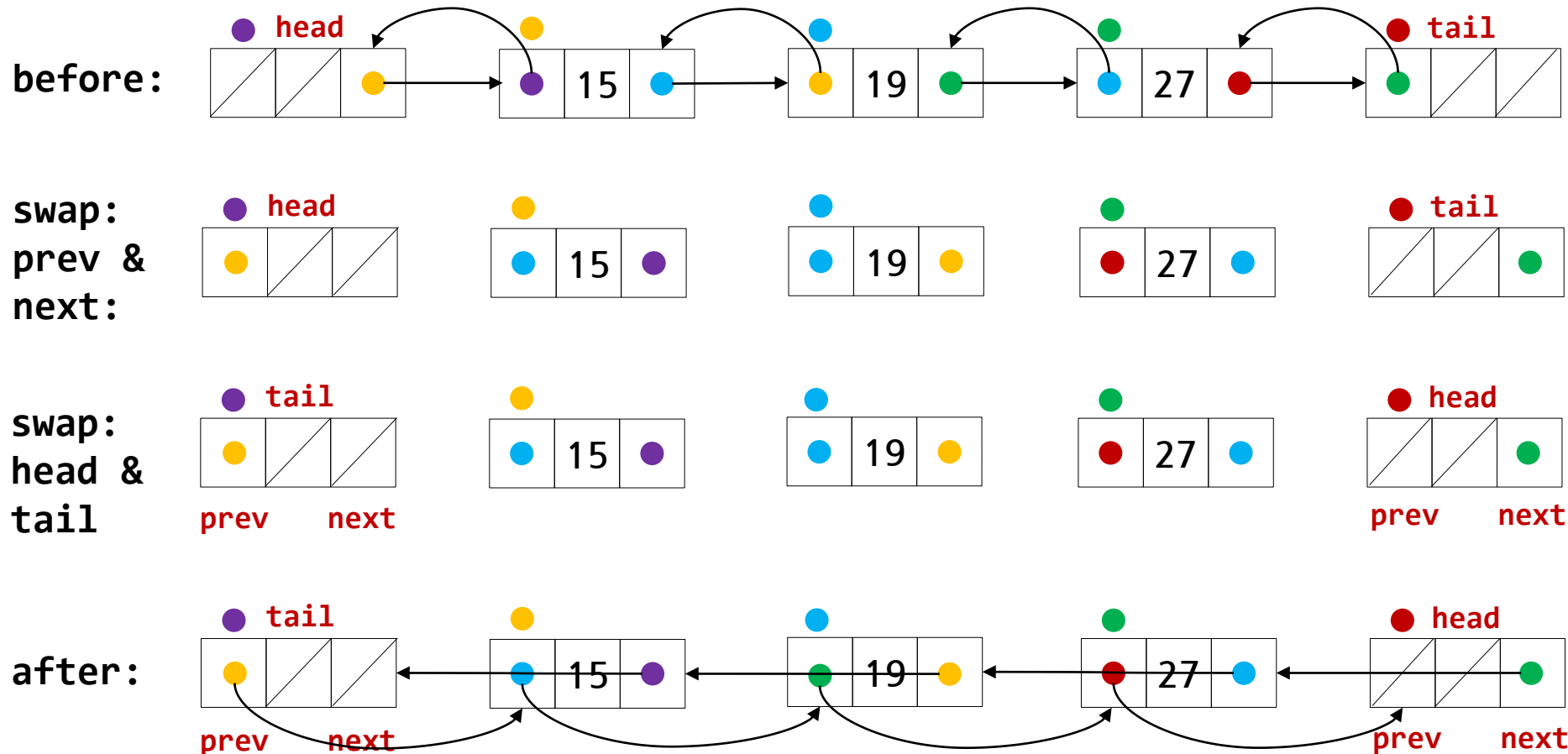
Exercise: reverse()

- Algorithm:
 - Step 1: swap **prev** and **next** in every node including two sentinel nodes.
 - Step 2: swap **head** and **tail** node.



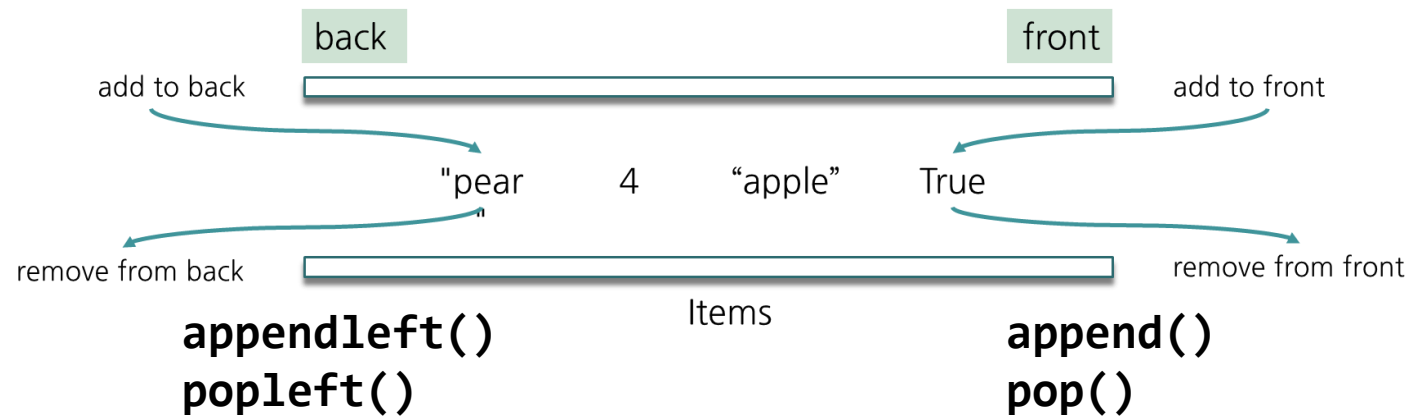
Exercise: reverse()

- Algorithm:
 - Step 1: swap **prev** and **next** in every node including two sentinel nodes.
 - Step 2: swap **head** and **tail** node.



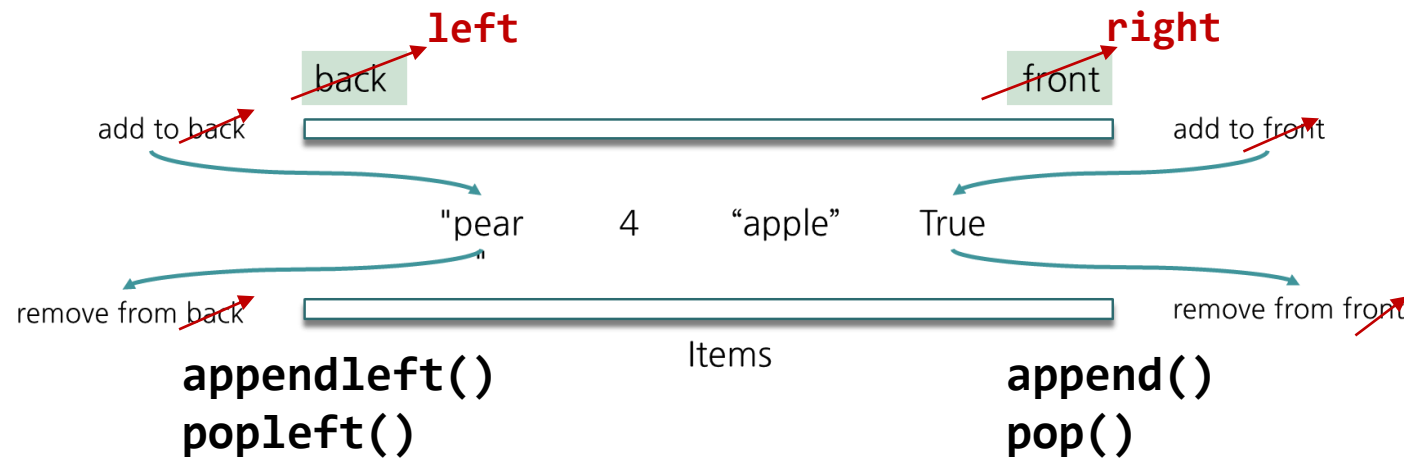
DequeCircular Class Using DoublyLinked

- Define the circular double-ended queue (deque) or DequeCircular using DoublyLinked class.
 - Define **DequeCircular** to be a subclass of **DoublyLinked** class.



DequeCircular Class Using DoublyLinked

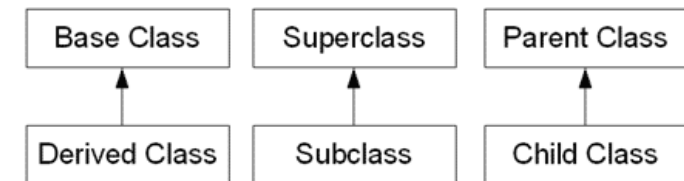
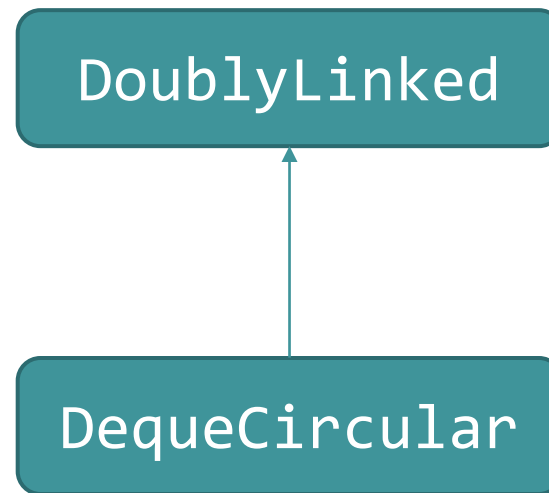
- Define the circular double-ended queue (deque) or `DequeCircular` using `DoublyLinked` class.
 - Define **`DequeCircular`** to be a subclass of **`DoublyLinked`** class.



- Let us not to use the subjective terms such as "**front**" or "**back**". Instead, let us use "**left**" or "**right**" of the deque.

DequeCircular Class Using DoublyLinked

- Define **DequeCircular** to be a subclass of **DoublyLinked** class.
 - Take an advantage of Object-oriented programming in this implementation.
 - For example, **reverse()** or **is_empty()** in the superclass **DoublyLinked** must operate the same in **DequeCircular**, the subclass. Therefore, those methods are inherited in the subclass. However, **rotate(n)** must be implemented solely in the subclass since it is not defined in the superclass.



DequeCircular Class - ADT

The **DequeCircular** class may support following operations:

- **DequeCircular(items=[], maxlen=None)**: Constructor, set a **maxlen** of the deque to be.
- **is_empty()**: checks whether the deque is empty or not.
- **is_full()**: checks whether the deque is full or not.
- **size()**: returns the number of items in the deque.
- **clear()**: removes all items from the deque leaving it with the size 0.

Yet to be coded:

- **appendleft()**: adds an item at the left side of the deque.
- **append()**: adds an item at the right side of the deque.
- **popleft()**: removes an item from the left of the deque, returns the item popped.
- **pop()**: deletes an item from the last (rightmost) of deque, returns the item popped.
- **getleft()**: gets the leftmost item from the deque. If the deque is empty, return **None**.
- **getright()**: gets the rightmost item from the deque. If the deque is empty, return **None**.
- **rotate(n=1)**: rotates the deque n steps to the right. If n is negative, rotate to the left.
- **maxlen**: Maximum size of a deque or **None** if unbounded.

DequeCircular Class - `__init__()`

- The constructor `__init__()` is provided as shown below.
- Optionally, it takes an iterable or a single object to populate the deque initially since `DoublyLinked` already support this capability.
- For example,
 - `DequeCircular([range(5)])` → `[0, 1, 2, 3, 4]`
 - `DequeCircular("abc")` → `['a', 'b', 'c']`
 - `DequeCircular(['a', 'b', 'c'])` → `['a', 'b', 'c']`
 - `DequeCircular(11)` → `[11]`
 - `DequeCircular()` → `[]`

```
def __init__(self, items = [], maxlen = None):  
    super().__init__(items)  
    self.maxlen = maxlen
```

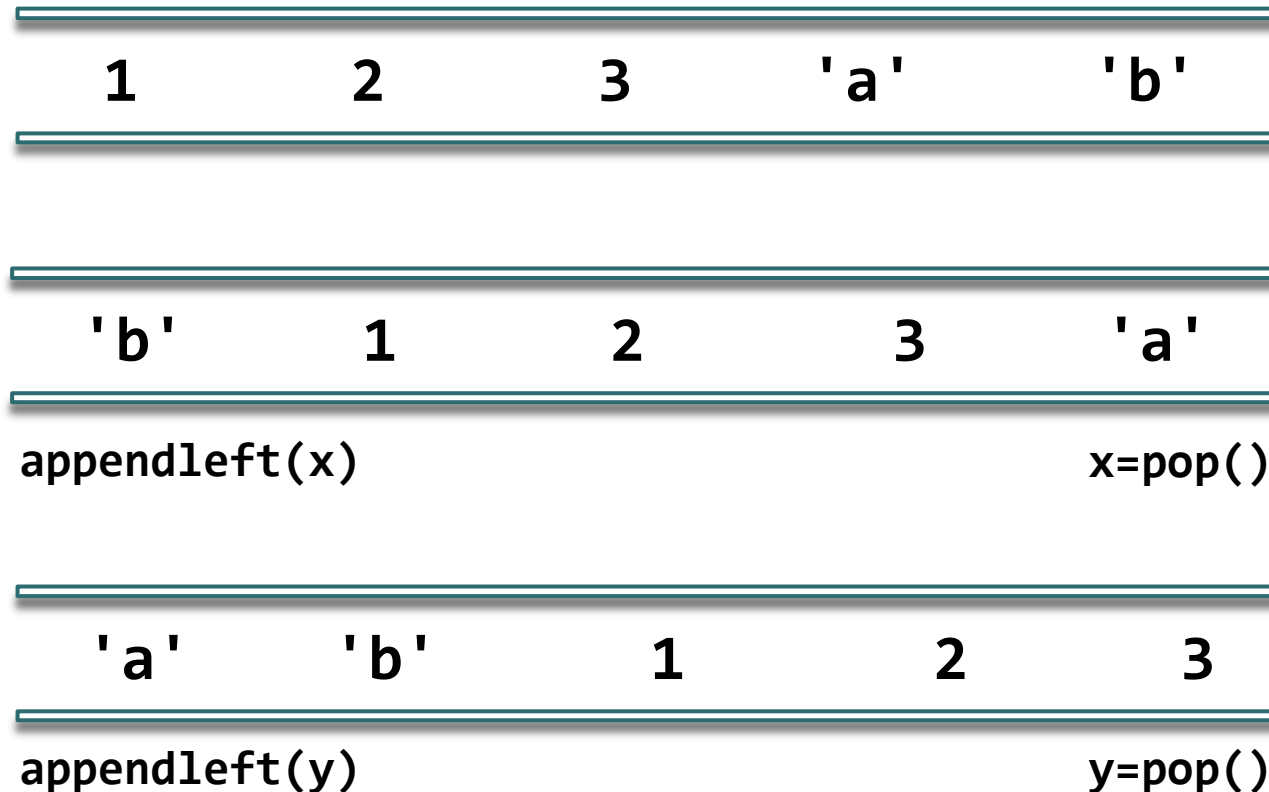
DequeCircular Class - `__init__()`

- The constructor `__init__()` is provided as shown below.
- Optionally, it takes **maxlen**, the maximum size of a deque or **None** if unbounded.
 - In this implementation, remove an item from the other side of the deque to make a room if the deque is full while queuing is requested.
 - For example, **pop()** an item (from the right) if the deque is full and **appendleft()** requested.
- The **maxlen** is **None** by default. It means that the deque is never full.

```
def __init__(self, items = [], maxlen = None):  
    super().__init__(items)  
    self.maxlen = maxlen
```

DequeCircular Class - rotate(n = 1)

- It rotates the deque n steps to the right. If n is negative, rotate to the left.
 - For example, **rotate(2)** rotates by executing **pop()** first and **appendleft()** twice. The output of **pop()** goes into **appendleft()** as shown below.



DequeCircular Class - rotate(n = 1)

- It rotates the deque n steps to the right. If n is negative, rotate to the left.
 - For example, **rotate(-2)** rotates by executing **pop()** first and **appendleft()** twice. The output of **pop()** goes into **appendleft()** as shown below.

1	2	3	'a'	'b'
---	---	---	-----	-----

2	3	'a'	'b'	1
---	---	-----	-----	---

x = popleft(x)

append(x)

3	'a'	'b'	1	2
---	-----	-----	---	---

x = popleft(y)

append(y)

DequeCircular Class - Sample Run and Expected Output

```
if __name__ == '__main__':
    cDeque = DequeCircular(items='abc', maxlen=5) # same as ['a', 'b', 'c']
    print(cDeque)                                # [a, b, c]
    cDeque.append('1')                            #
    cDeque.append('2')                            #
    cDeque.append('3')                            # since maxlen = 5
    print(cDeque)                                # [b, c, 1, 2, 3]
    print(cDeque.size())                          # 5
    print(cDeque.appendleft('a'))                  # True
    print(cDeque.getright())                       # a
    print(cDeque)                                # [a, b, c, 1, 2]
    print(cDeque.is_full())                       # True
    cDeque.rotate(2)
    print('rotate( 2):', cDeque)                   # [1, 2, a, b, c]
    cDeque.rotate(-3)
    print('rotate(-3):', cDeque)                  # [b, c, 1, 2, a]
    print(cDeque.pop())                           # a
    print(cDeque.appendleft('a'))                  # True
    print(cDeque)                                # [a, b, c, 1, 2]
    print(cDeque.getleft())                       # a
    cDeque.reverse()
    print(cDeque)                                # [2, 1, c, b, a]
```

Summary

- Define DequeCircular (a deque) using DoublyLinked
 - Singly Linked List vs. Doubly Linked List
 - Exercises in DoublyLinked
 - Iterable and its usage
 - Enhance constructor `__init__()`
 - Define DequeCircular Class
 - Using DoublyLinked
 - maxlen
 - rotate(n = 1)

Data Structures in Python

Chapter 3

- Linked List
- OOP Inheritance
- ListUnsorted Class
- ListSorted Class
- Iterator
- Doubly Linked List - Structures
- Doubly Linked List - Operations
- **Doubly Linked List - DequeCircular**