Lecture 12

By Urooj Ainuddin

CS-218 Data Structures and Algorithms

In the last lecture…

We learned how to insert before or after a node containing a certain value in a SLL.

We learned how to delete a node containing a certain value in a SLL.

We built singly linked lists.

We inserted a tail node in a SLL.

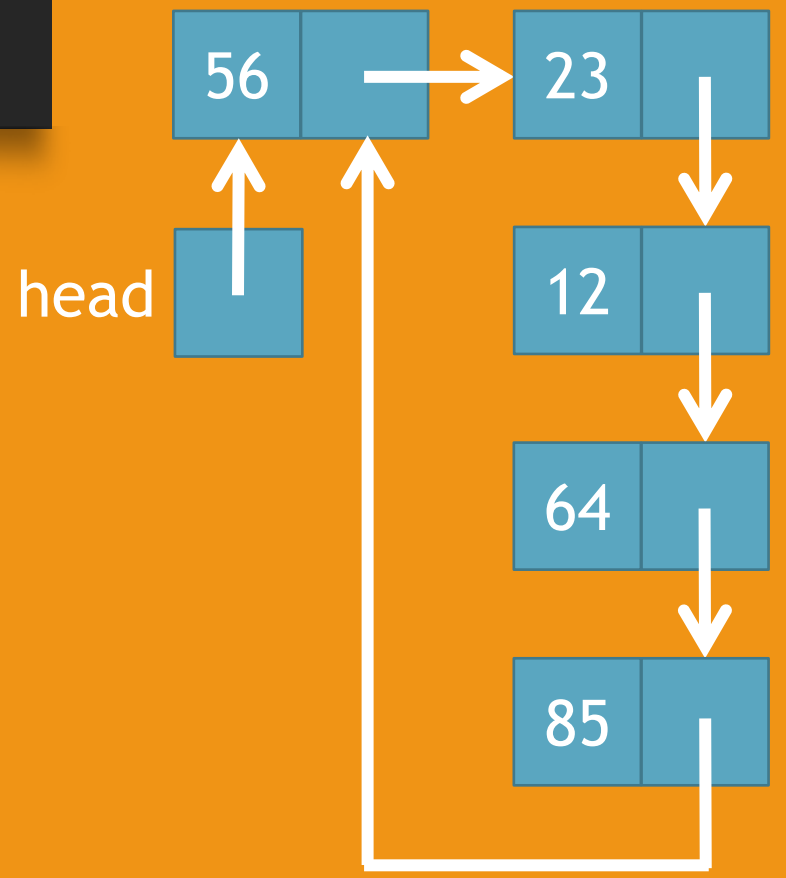
We conducted complexity analysis for the running times of our functions.

Advanced Linked Structures

Book 1 Chapter 9

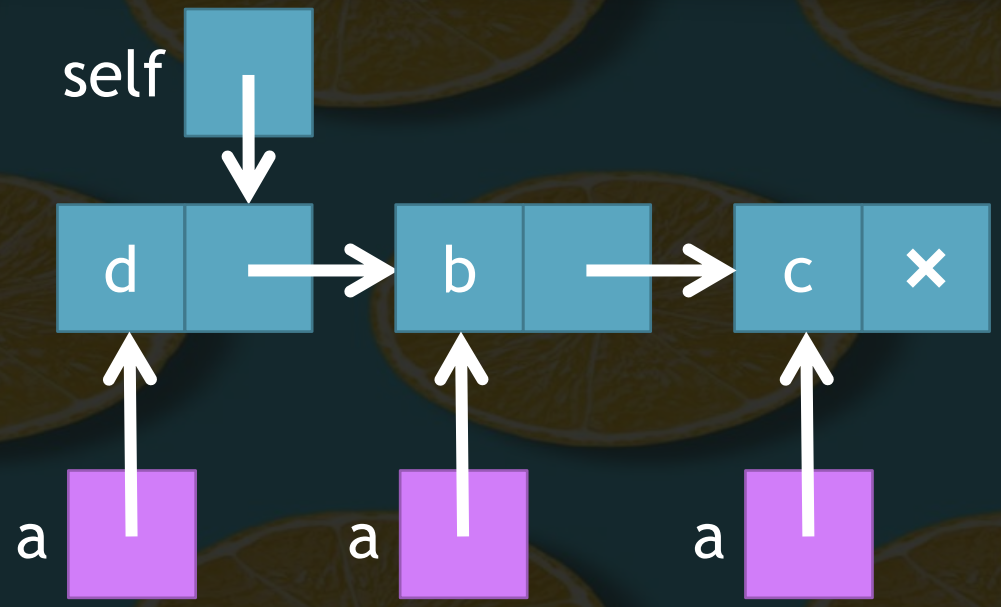
The circular singly linked list (CSLL)

- The circular singly linked list is a structure composed of **nodes**.
- The pointer in the last node points to the first node instead of being None.
- The entire data structure can be accessed via a **head** pointer, which can be pointing to any node in the CSLL.
- If we start traversal in a SLL using a pointer x, we can visit all nodes after x, but we cannot visit the nodes before x.
- If we start traversal in a CSLL using a pointer x, we can visit all nodes in the linked list.



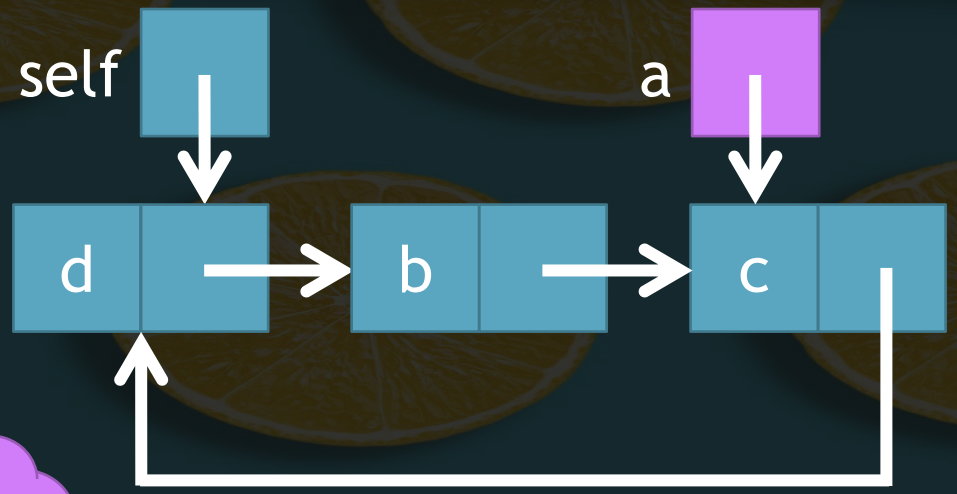
The singlylinkedlist.py file - circularize

```
def circularize(self):
        a=self
        while a.next is not None:
            a=a.next
        a.next=self
```



The singlylinkedlist.py file - circularize

```
def circularize(self):
        a=self
        while a.next is not None:
            a=a.next
        a.next=self
```



O(n)

The singlylinkedlist.py file - linearize

```
def linearize(self):
        a = self
        while a.next is not self:
            a = a.next
        a.next = None
```

Draw step by step pictures depicting how this function would execute

$O(n)$

The singlylinkedlist.py file - traverse_circular

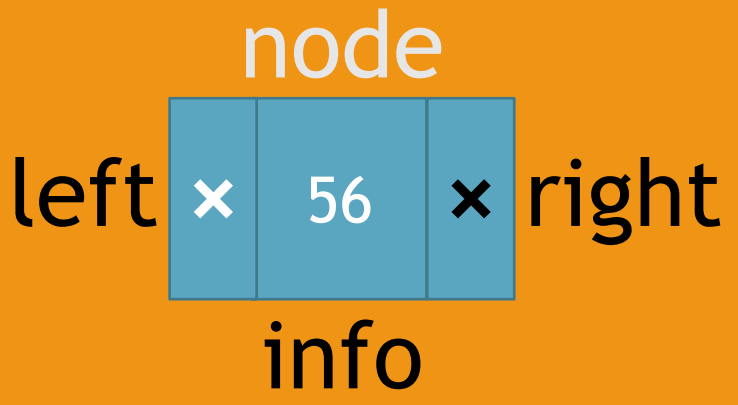
```
def traverse_circular(self):
        a = self
        print("Traversing the list...")
        while a.next is not self:
            print(a.data, end=" ")
            a = a.next
        print(a.data, end=" ")
        print()
```

Draw step by step pictures depicting how this function would execute

$O(n)$

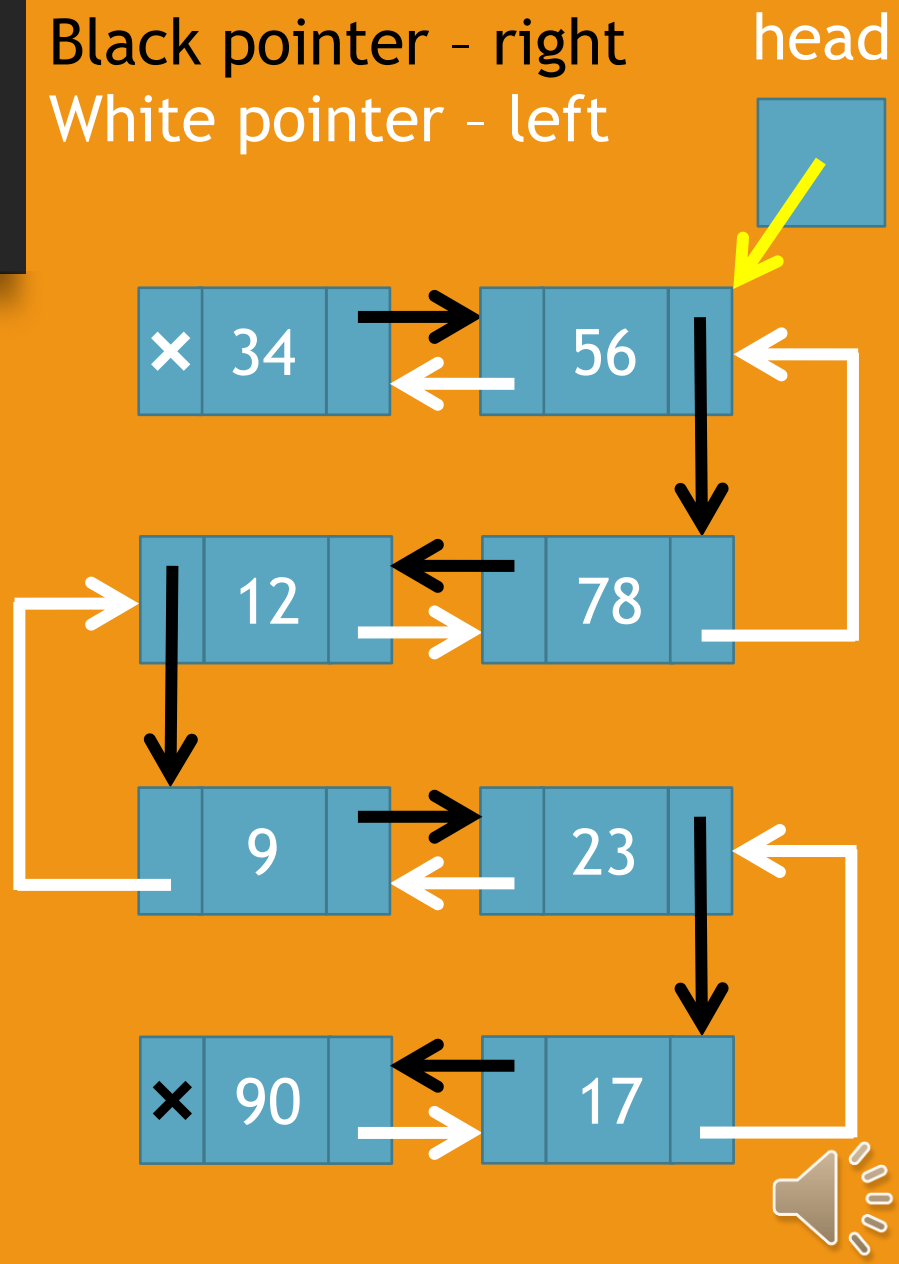
The doubly linked list (DLL)

- The doubly linked list is a structure composed of **nodes**.
- Each node has an **info** field and two pointer fields called **right** and **left**.
- The info field carries the data item that needs to be stored in the data structure.
- The right field contains a pointer to the node on the right of the current node.
- The left field contains a pointer to the node on the left of the current node.



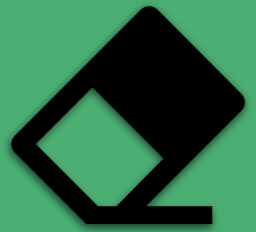
The doubly linked list (DLL)

- The doubly linked list is a structure composed of **nodes**.
- Each node has an **info** field and a **next** field.
- The info field carries the data item that needs to be stored in the data structure.
- The right field contains a pointer to the node on the right of the current node.
- The left field contains a pointer to the node on the left of the current node.
- The entire data structure can be accessed via a **head** pointer, which can be pointing to any node in the DLL.



Advantages of DLL over SLL

A DLL accommodates traversal of the linked list in both directions.

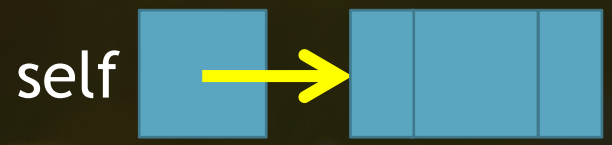


For deletion, the pointer to the node before the node to be deleted is not required in DLL.

Any node x can be deleted using a pointer that points to x itself.

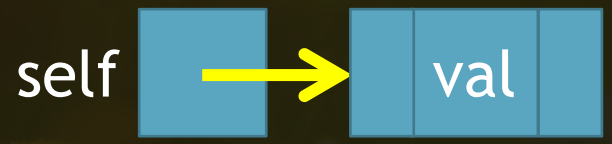
The doublylinkedlist.py file - the class DLNode

```
class DLNode:
    def __init__(self,val):
        self.data = val
        self.right = None
        self.left = None
```



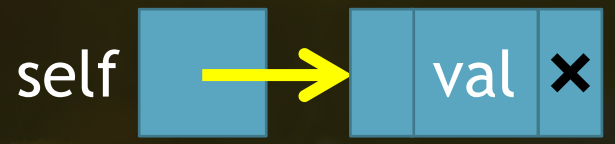
The doublylinkedlist.py file - the class DLNode

```
class DLNode:
    def __init__(self,val):
        self.data = val
        self.right = None
        self.left = None
```



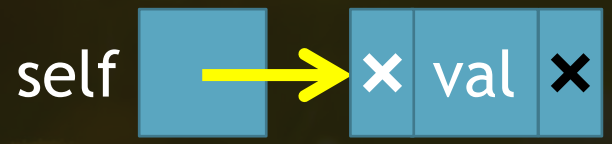
The doublylinkedlist.py file - the class DLNode

```
class DLNode:
    def __init__(self,val):
        self.data = val
        self.right = None
        self.left = None
```



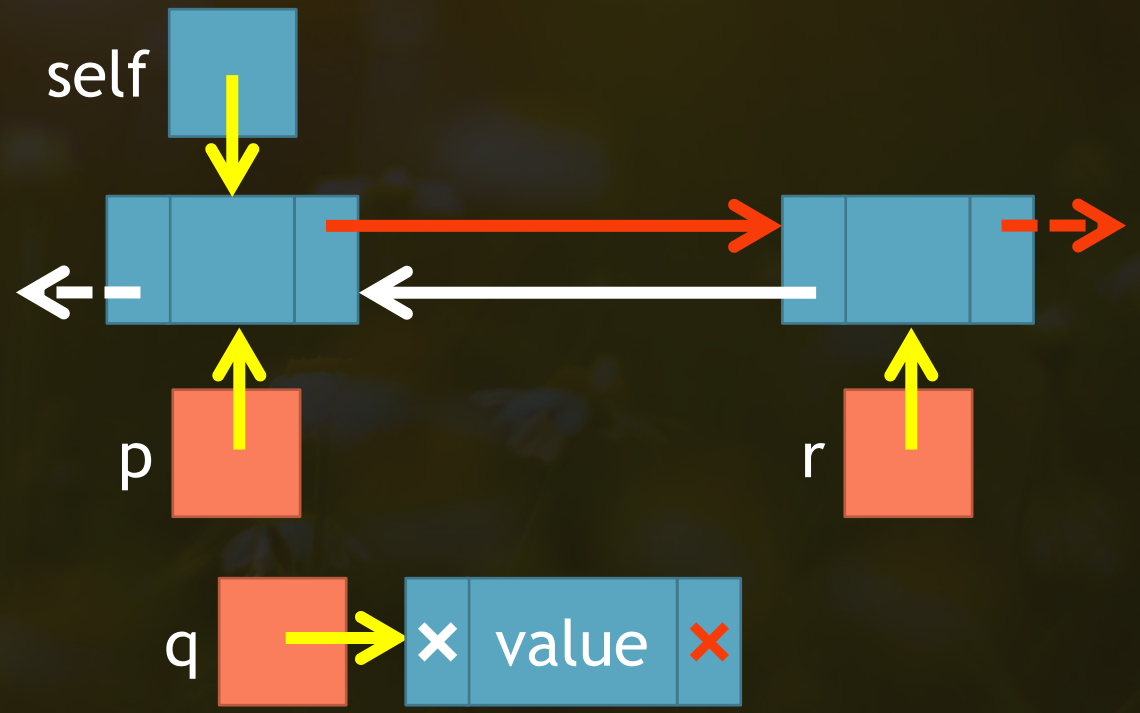
The doublylinkedlist.py file - the class DLNode

```
class DLNode:
    def __init__(self,val):
        self.data = val
        self.right = None
        self.left = None
```



The doublylinkedlist.py file - insertright

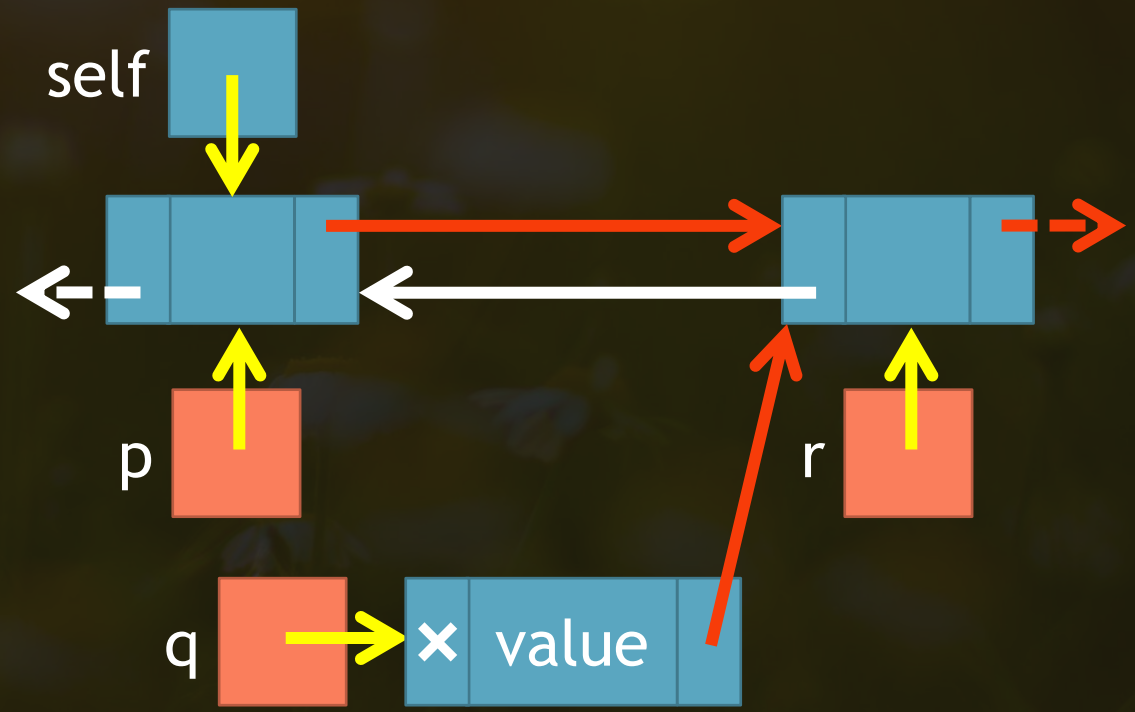
```
def insertright(self,value):
        #p  q   r
        p = self
        q = DLNode(value)
        r = p.right
        q.right = r
        q.left = p
        p.right = q
        if r is not None:
            r.left=q
```



This function inserts a node containing value on the right of the node pointed to by self. We assume the node pointed to by self is not the last node of the DLL.

The doublylinkedlist.py file - insertright

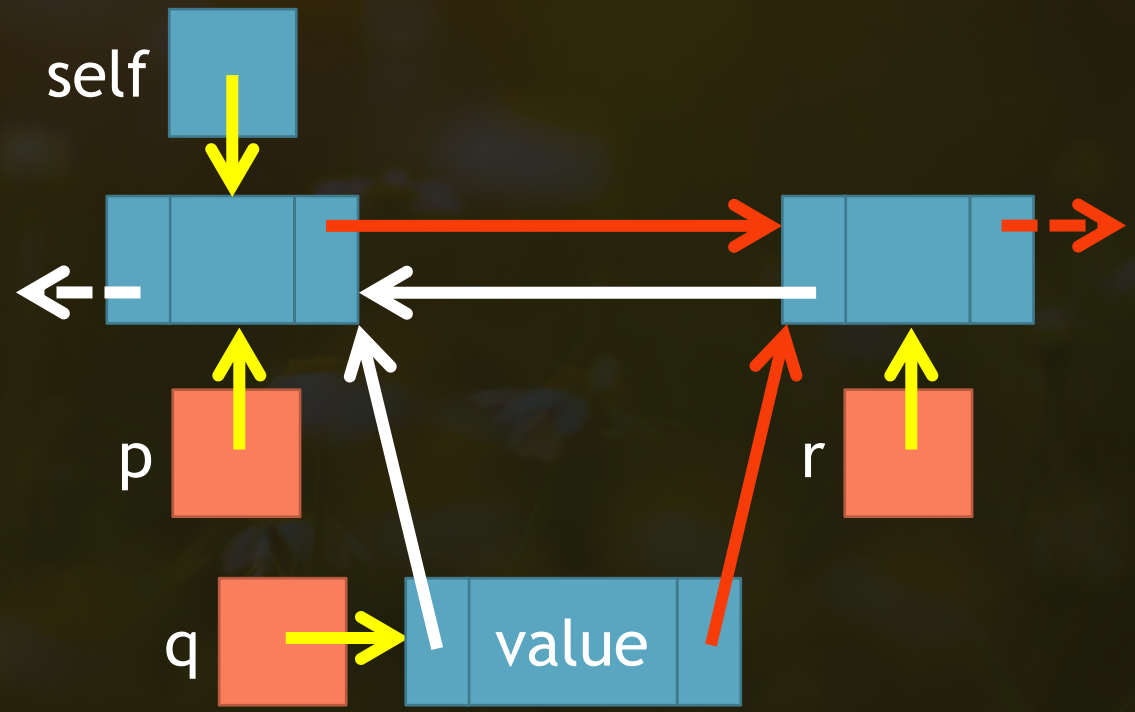
```
def insertright(self,value):
        #p  q   r
        p = self
        q = DLNode(value)
        r = p.right
        q.right = r
        q.left = p
        p.right = q
        if r is not None:
            r.left=q
```



This function inserts a node containing value on the right of the node pointed to by self. We assume the node pointed to by self is not the last node of the DLL.

The doublylinkedlist.py file - insertright

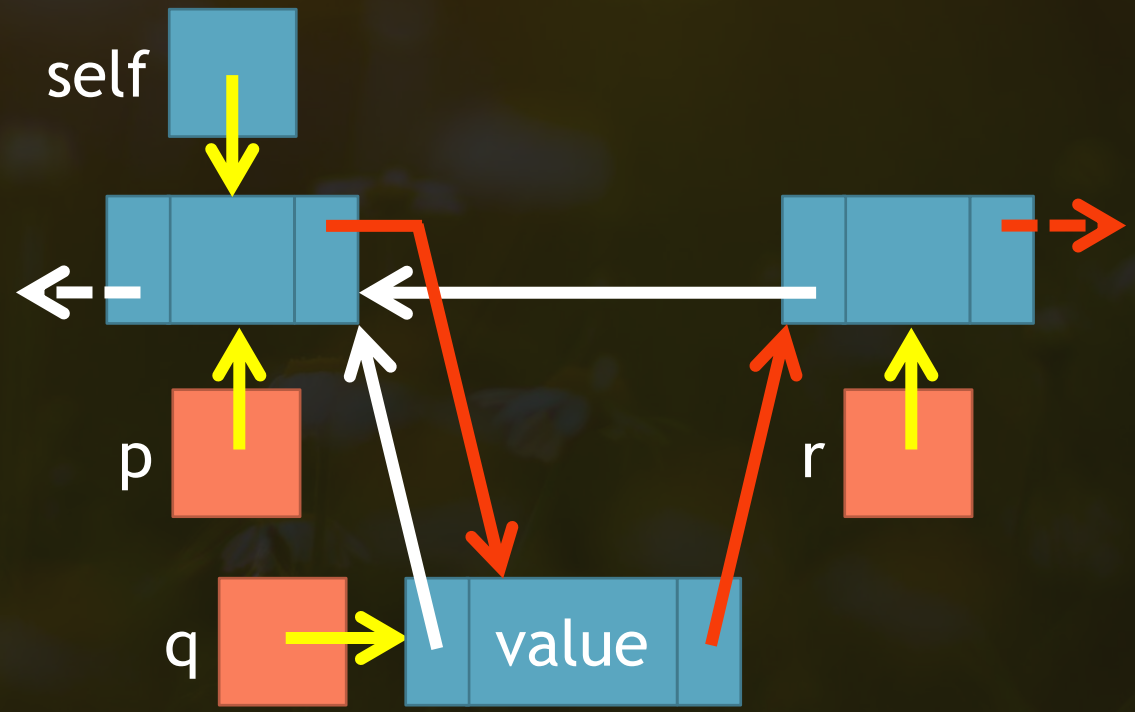
```
def insertright(self,value):
        #p  q   r
        p = self
        q = DLNode(value)
        r = p.right
        q.right = r
        q.left = p
        p.right = q
        if r is not None:
            r.left=q
```



This function inserts a node containing value on the right of the node pointed to by self. We assume the node pointed to by self is not the last node of the DLL.

The doublylinkedlist.py file - insertright

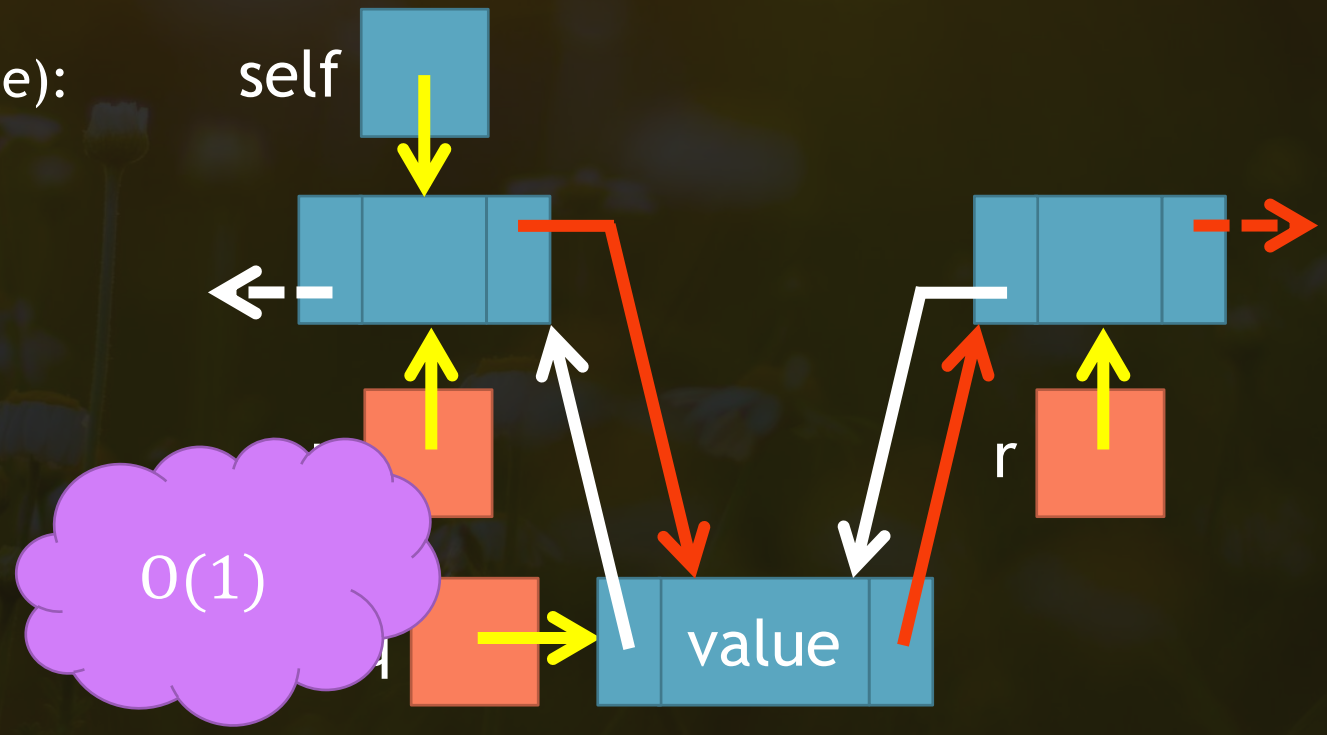
```
def insertright(self,value):
        #p  q   r
        p = self
        q = DLNode(value)
        r = p.right
        q.right = r
        q.left = p
        p.right = q
        if r is not None:
            r.left=q
```



This function inserts a node containing value on the right of the node pointed to by self. We assume the node pointed to by self is not the last node of the DLL.

The doublylinkedlist.py file - insertright

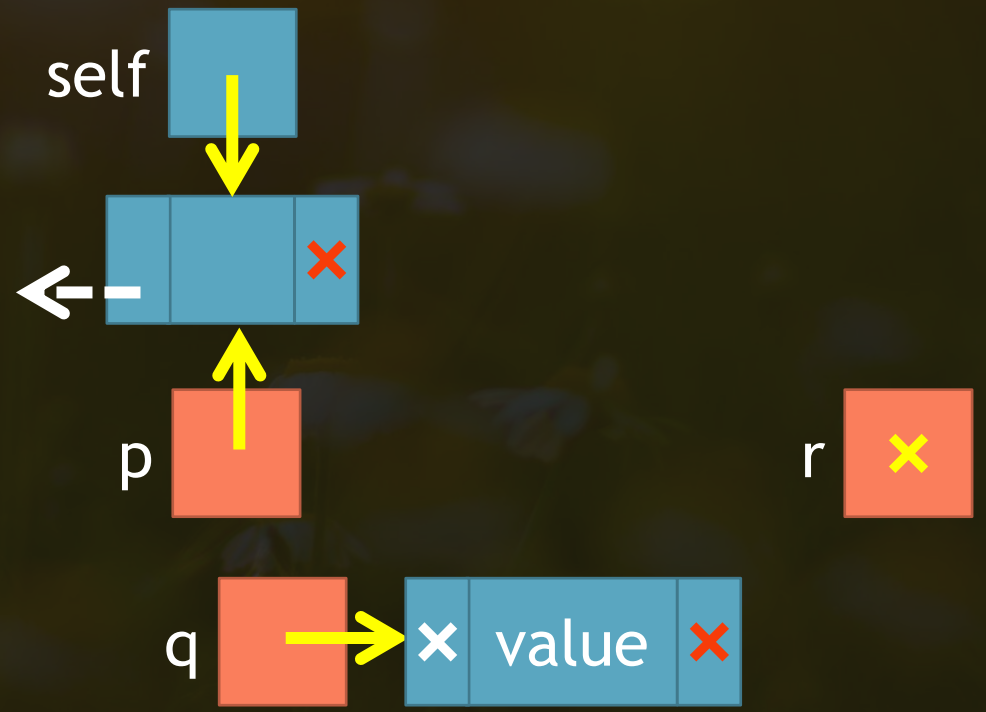
```
def insertright(self,value):
        #p  q   r
        p = self
        q = DLNode(value)
        r = p.right
        q.right = r
        q.left = p
        p.right = q
        if r is not None:
            r.left=q
```



This function inserts a node containing value on the right of the node pointed to by self. We first assume the node pointed to by self is not the last node of the DLL.

The doublylinkedlist.py file - insertright

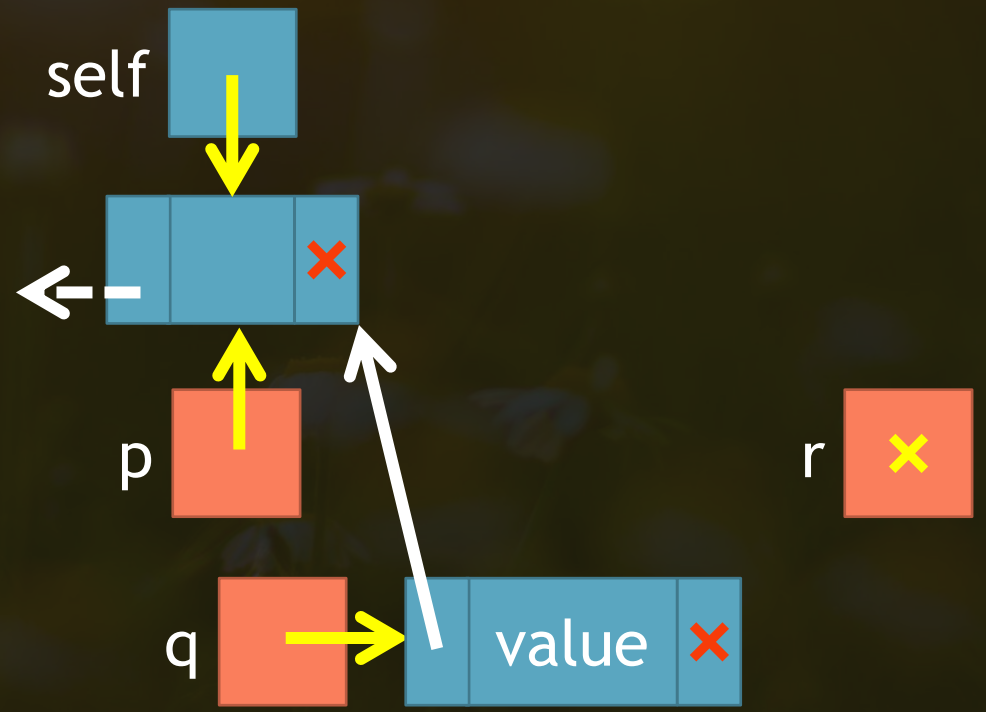
```
def insertright(self,value):
        #p  q   r
        p = self
        q = DLNode(value)
        r = p.right
        q.right = r
        q.left = p
        p.right = q
        if r is not None:
            r.left=q
```



This function inserts a node containing value on the right of the node pointed to by self. We assume the node pointed to by self is the last node of the DLL.

The doublylinkedlist.py file - insertright

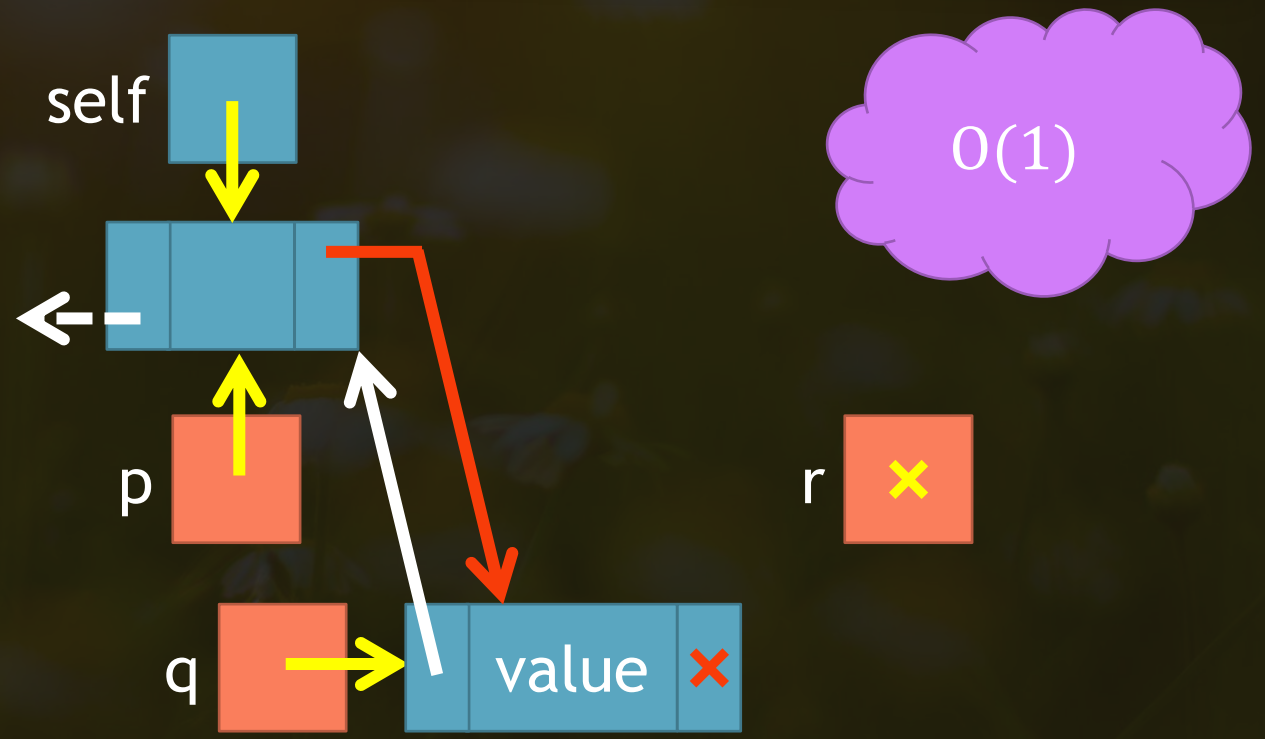
```
def insertright(self,value):
        #p  q   r
        p = self
        q = DLNode(value)
        r = p.right
        q.right = r
        q.left = p
        p.right = q
        if r is not None:
            r.left=q
```



This function inserts a node containing value on the right of the node pointed to by self. We assume the node pointed to by self is the last node of the DLL.

The doublylinkedlist.py file - insertright

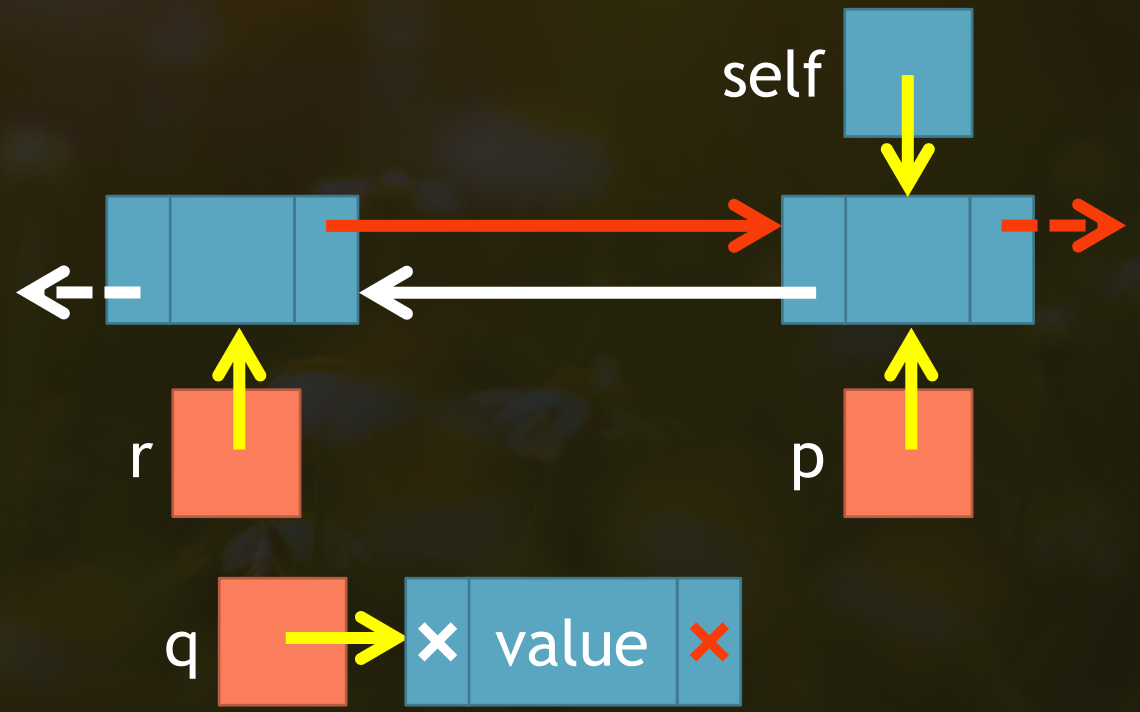
```
def insertright(self,value):
        #p  q   r
        p = self
        q = DLNode(value)
        r = p.right
        q.right = r
        q.left = p
        p.right = q
        if r is not None:
            r.left=q
```



This function inserts a node containing value on the right of the node pointed to by self. We assume the node pointed to by self is the last node of the DLL.

The doublylinkedlist.py file - insertleft

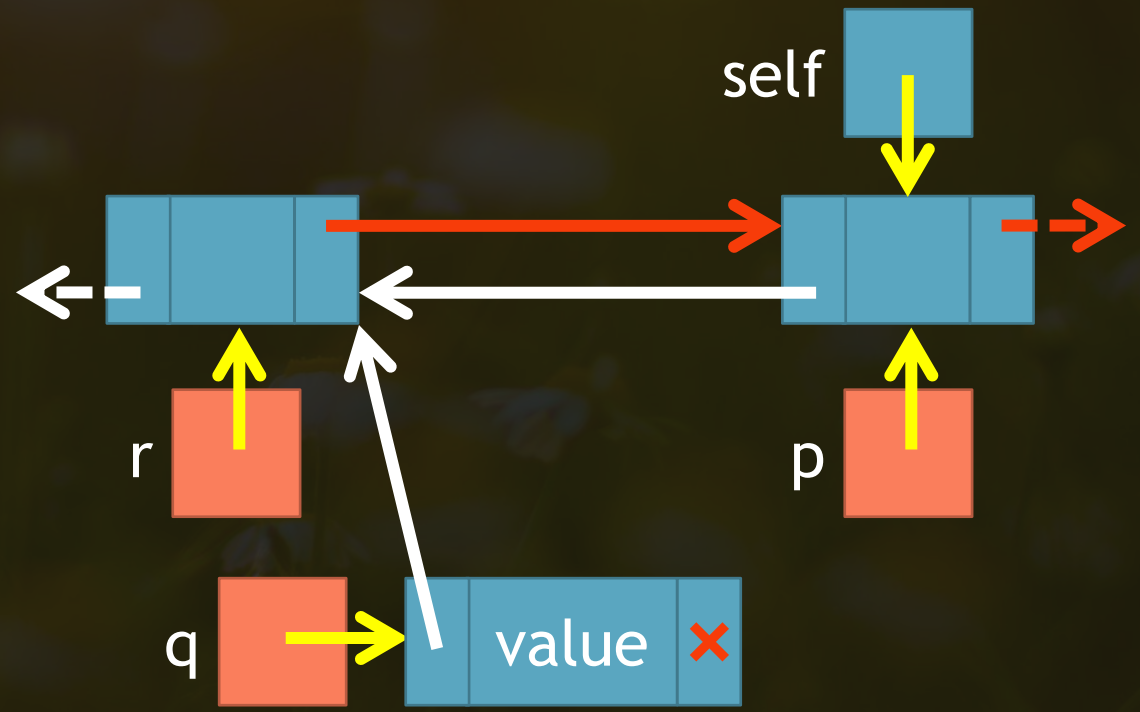
```
def insertleft(self,value):
        #r  q   p
        p=self
        q = DLNode(value)
        r = p.left
        q.left = r
        q.right = p
        p.left = q
        if r is not None:
            r.right=q
```



This function inserts a node containing value on the left of the node pointed to by self. We assume the node pointed to by self is not the first node of the DLL.

The doublylinkedlist.py file - insertleft

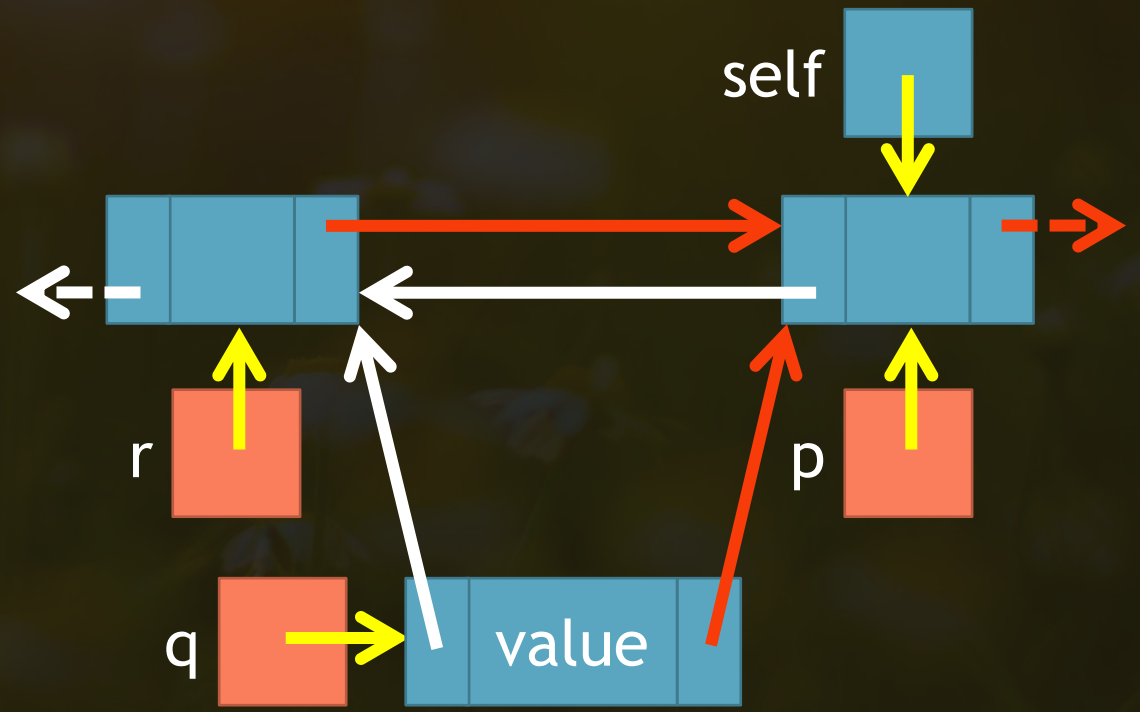
```
def insertleft(self,value):
        #r  q   p
        p=self
        q = DLNode(value)
        r = p.left
        q.left = r
        q.right = p
        p.left = q
        if r is not None:
            r.right=q
```



This function inserts a node containing value on the left of the node pointed to by self. We assume the node pointed to by self is not the first node of the DLL.

The doublylinkedlist.py file - insertleft

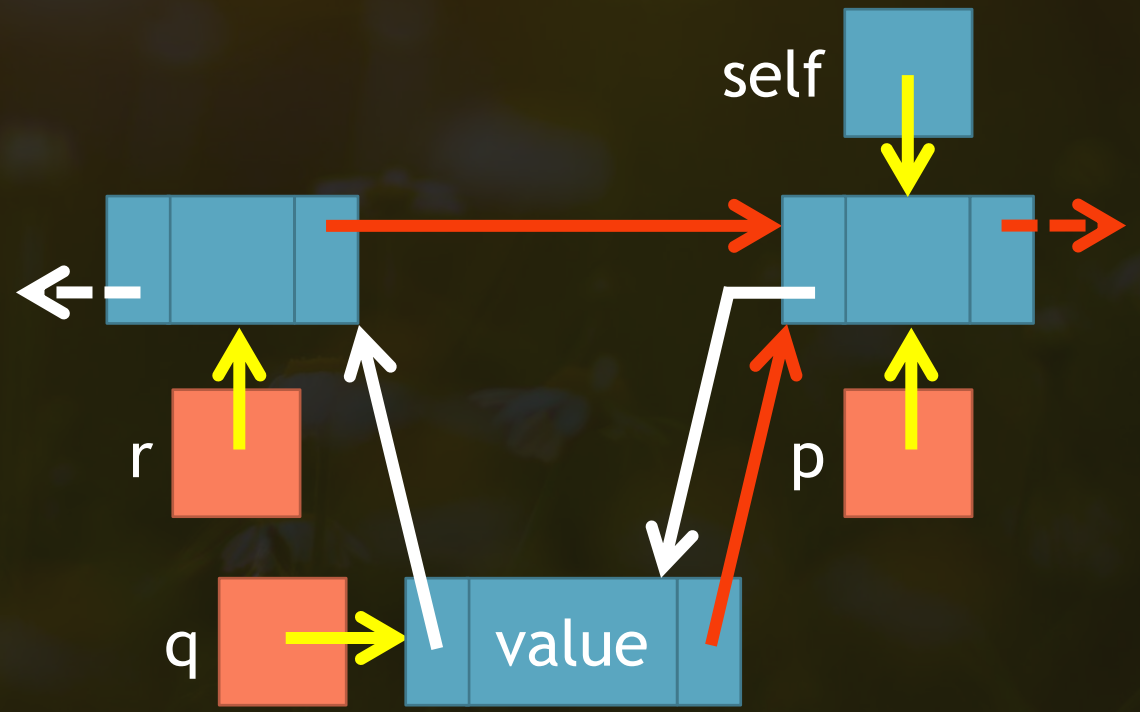
```
def insertleft(self,value):
        #r  q   p
        p=self
        q = DLNode(value)
        r = p.left
        q.left = r
        q.right = p
        p.left = q
        if r is not None:
            r.right=q
```



This function inserts a node containing value on the left of the node pointed to by self. We assume the node pointed to by self is not the first node of the DLL.

The doublylinkedlist.py file - insertleft

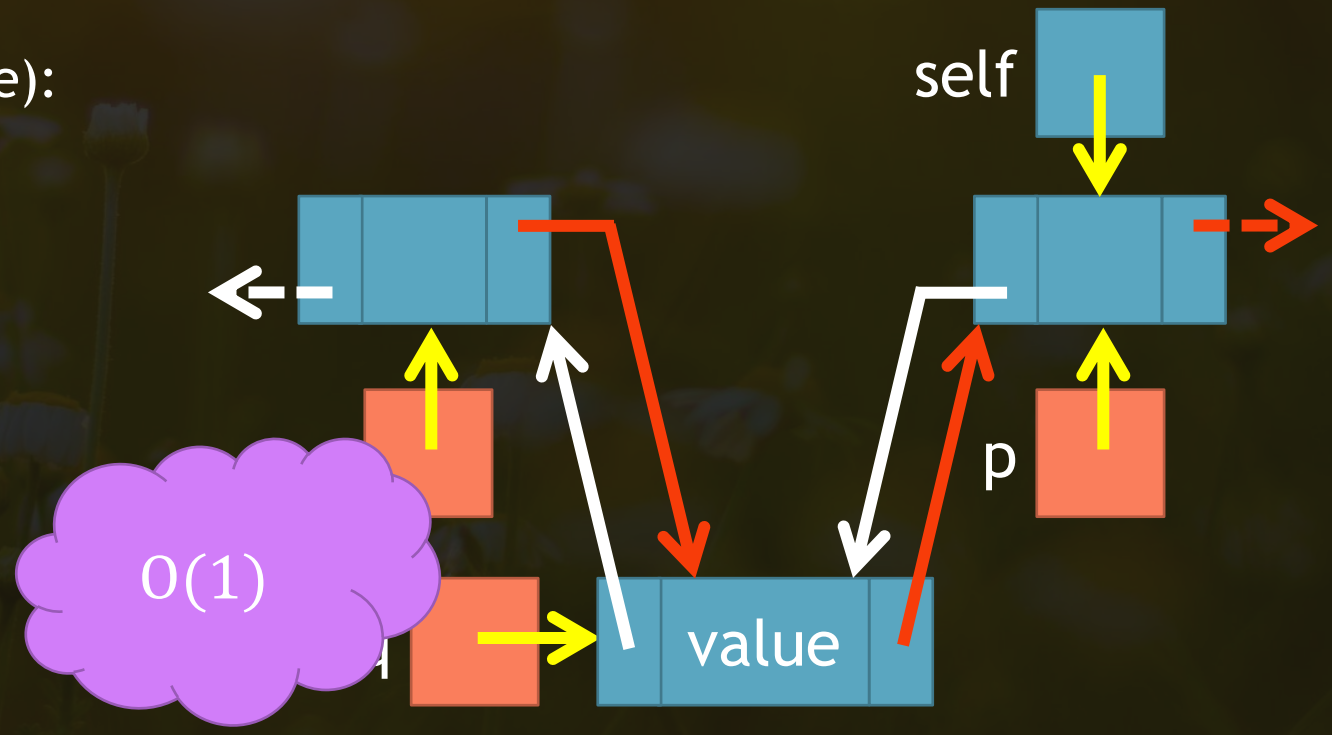
```
def insertleft(self,value):
        #r  q   p
        p=self
        q = DLNode(value)
        r = p.left
        q.left = r
        q.right = p
        p.left = q
        if r is not None:
            r.right=q
```



This function inserts a node containing value on the left of the node pointed to by self. We assume the node pointed to by self is not the first node of the DLL.

The doublylinkedlist.py file - insertleft

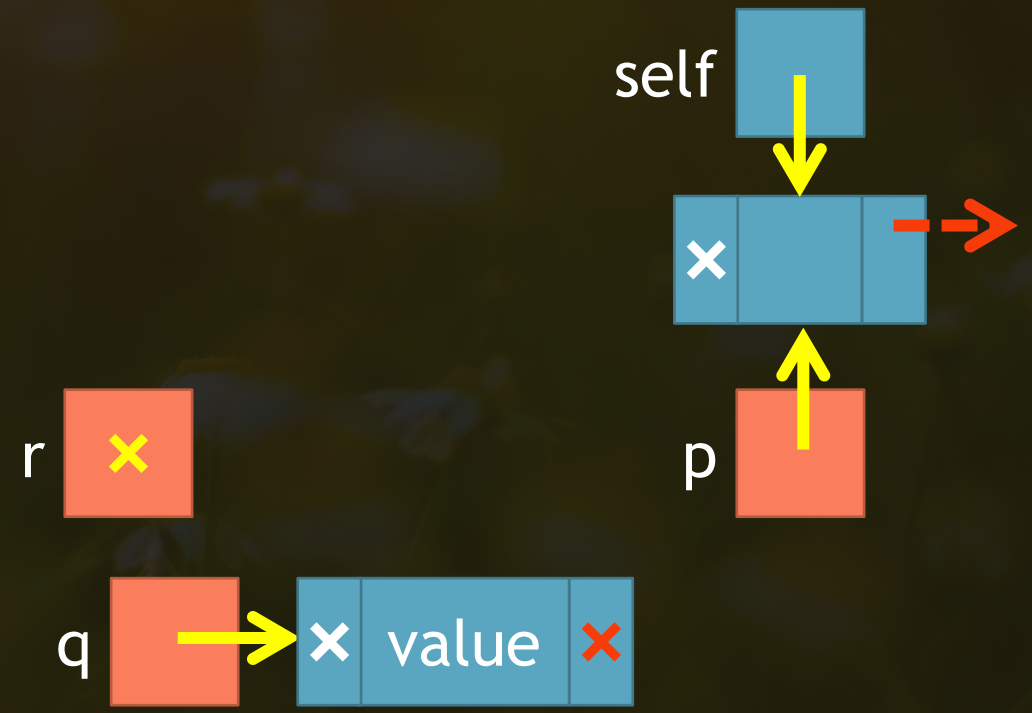
```
def insertleft(self,value):
        #r  q   p
        p=self
        q = DLNode(value)
        r = p.left
        q.left = r
        q.right = p
        p.left = q
        if r is not None:
            r.right=q
```



This function inserts a node containing value on the left of the node pointed to by self. We assume the node pointed to by self is not the first node of the DLL.

The doublylinkedlist.py file - insertleft

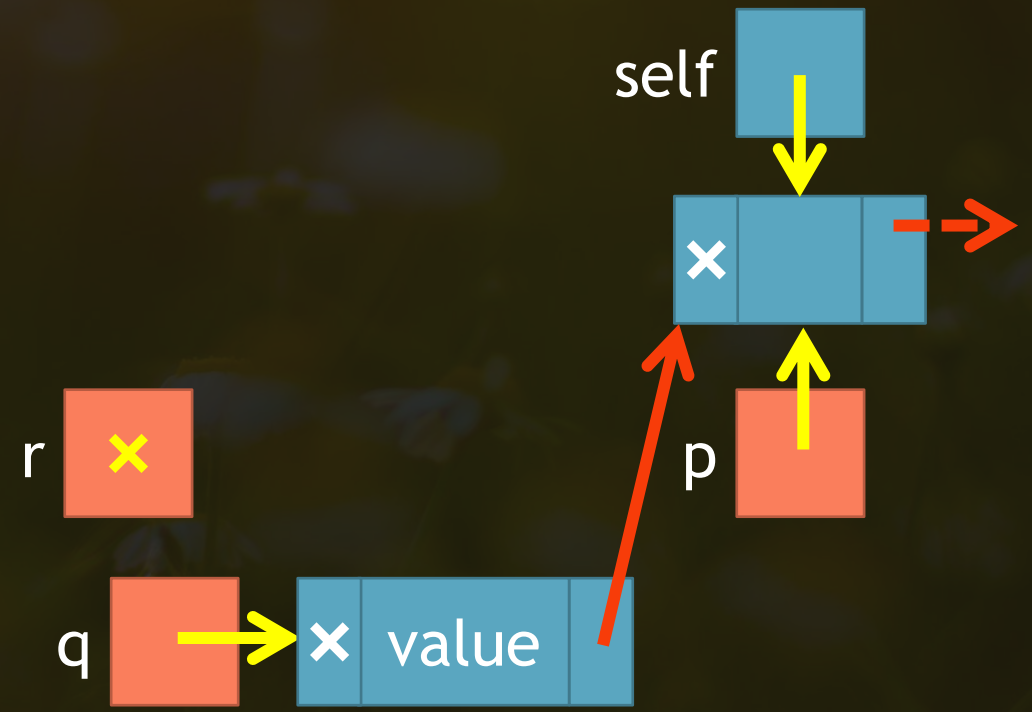
```
def insertleft(self,value):
        #r  q   p
        p=self
        q = DLNode(value)
        r = p.left
        q.left = r
        q.right = p
        p.left = q
        if r is not None:
            r.right=q
```



This function inserts a node containing value on the left of the node pointed to by self. We assume the node pointed to by self is the first node of the DLL.

The doublylinkedlist.py file - insertleft

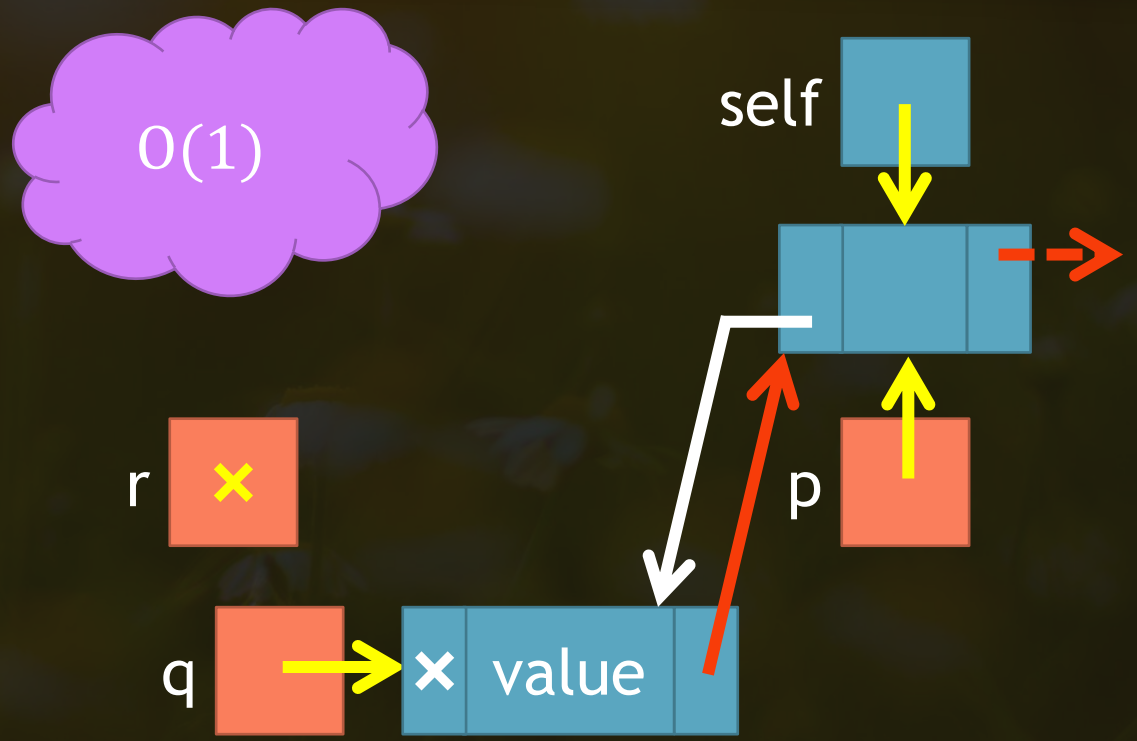
```
def insertleft(self,value):
        #r  q   p
        p=self
        q = DLNode(value)
        r = p.left
        q.left = r
        q.right = p
        p.left = q
        if r is not None:
            r.right=q
```



This function inserts a node containing value on the left of the node pointed to by self. We assume the node pointed to by self is the first node of the DLL.

The doublylinkedlist.py file - insertleft

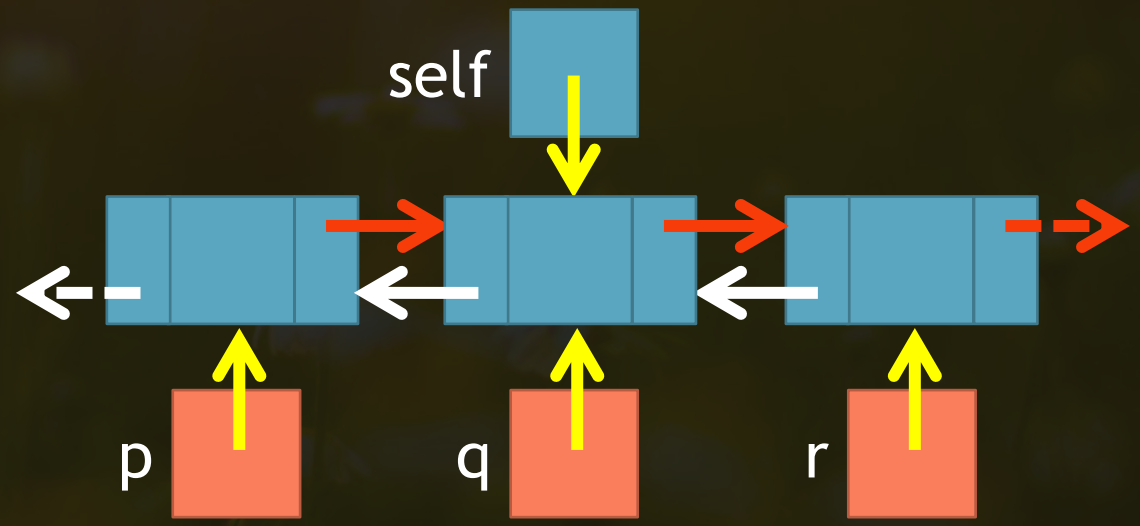
```
def insertleft(self,value):
        #r  q   p
        p=self
        q = DLNode(value)
        r = p.left
        q.left = r
        q.right = p
        p.left = q
        if r is not None:
            r.right=q
```



This function inserts a node containing value on the left of the node pointed to by self. We assume the node pointed to by self is the first node of the DLL.

The doublylinkedlist.py file - delete

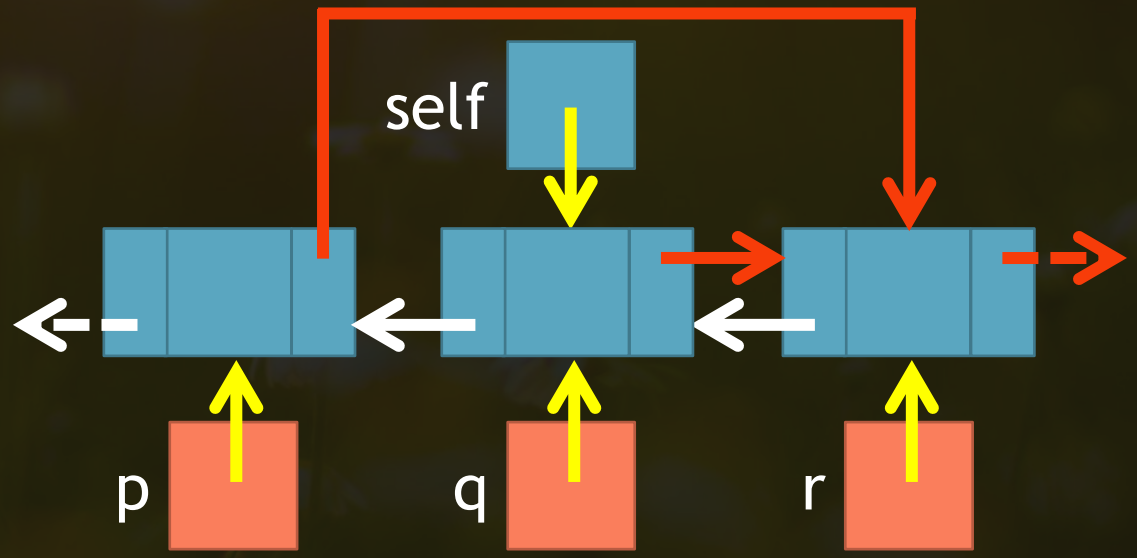
```
def delete(self):
    #p  q   r
    p = self.left
    q = self
    r = self.right
    if p is not None:
        p.right = r
    if r is not None:
        r.left = p
    if p is None:
        return r
    return p
```



This function deletes a node pointed to by self. We assume the node pointed to by self is not the first or the last node of the DLL.

The doublylinkedlist.py file - delete

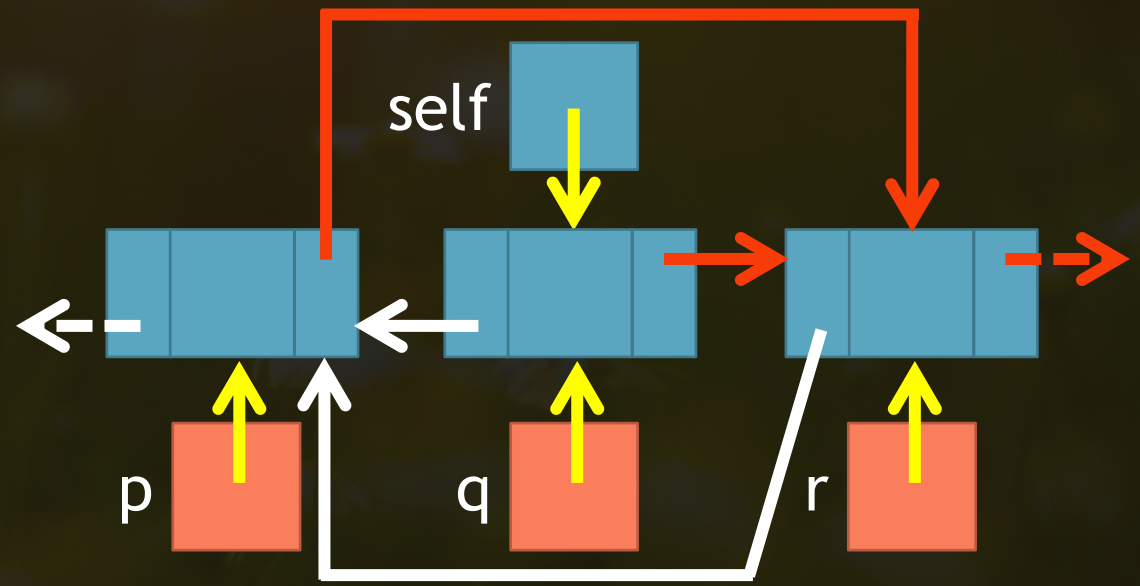
```
def delete(self):
    #p  q   r
    p = self.left
    q = self
    r = self.right
    if p is not None:
        p.right = r
    if r is not None:
        r.left = p
    if p is None:
        return r
    return p
```



This function deletes a node pointed to by self. We assume the node pointed to by self is not the first or the last node of the DLL.

The doublylinkedlist.py file - delete

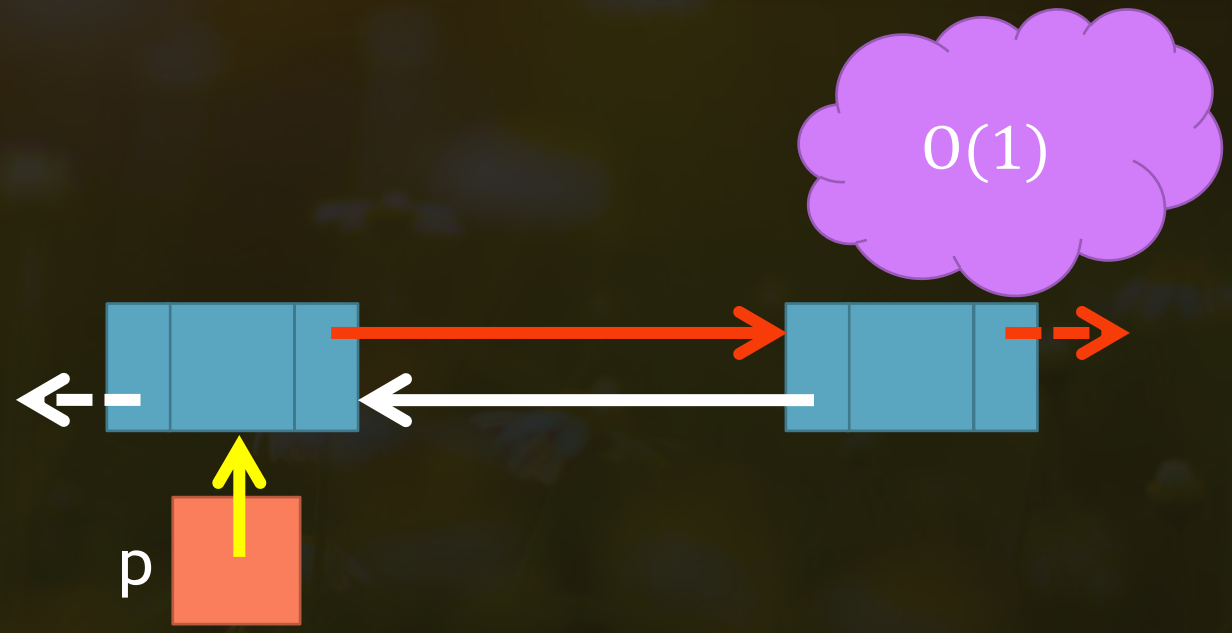
```
def delete(self):
    #p  q   r
    p = self.left
    q = self
    r = self.right
    if p is not None:
        p.right = r
    if r is not None:
        r.left = p
    if p is None:
        return r
    return p
```



This function deletes a node pointed to by self. We assume the node pointed to by self is not the first or the last node of the DLL.

The doublylinkedlist.py file - delete

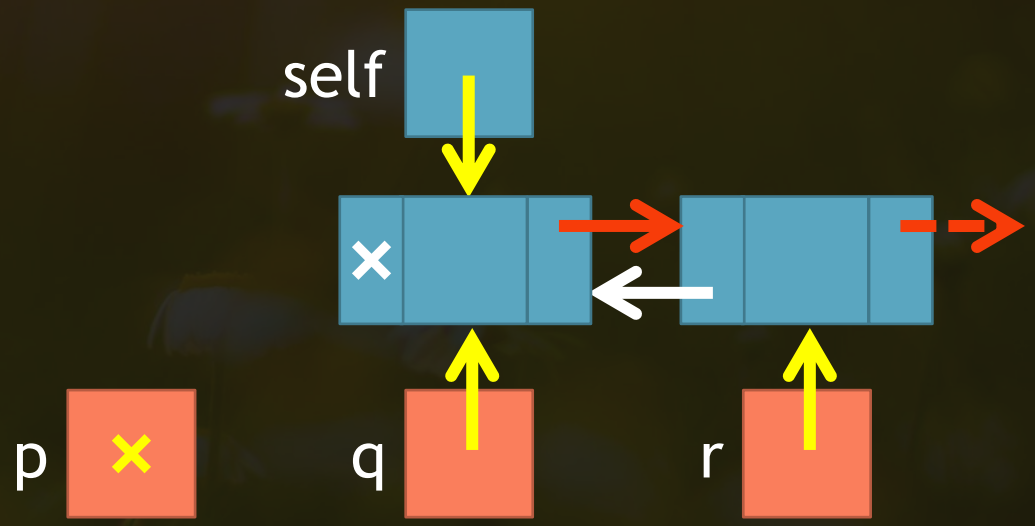
```
def delete(self):
    #p  q   r
    p = self.left
    q = self
    r = self.right
    if p is not None:
        p.right = r
    if r is not None:
        r.left = p
    if p is None:
        return r
    return p
```



This function deletes a node pointed to by self. We assume the node pointed to by self is not the first or the last node of the DLL.

The doublylinkedlist.py file - delete

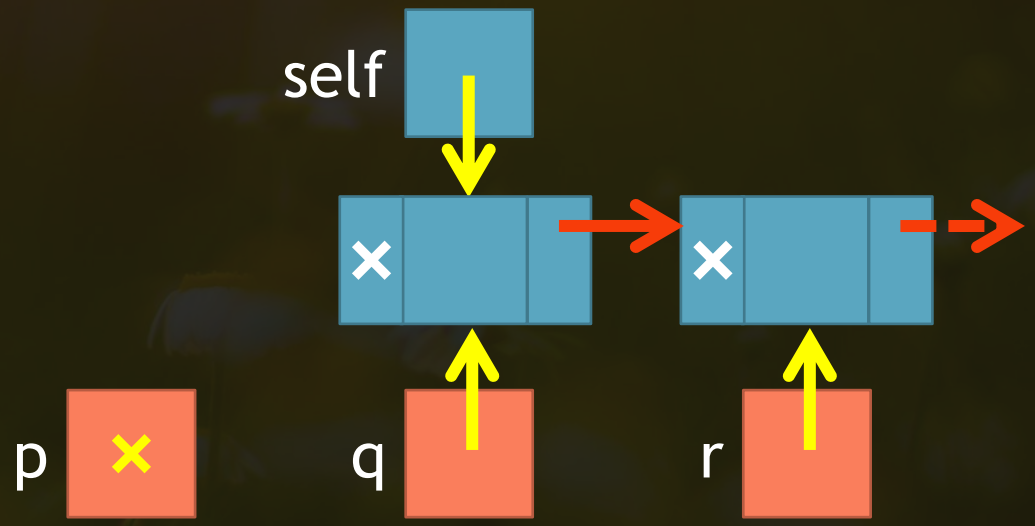
```
def delete(self):
    #p  q   r
    p = self.left
    q = self
    r = self.right
    if p is not None:
        p.right = r
    if r is not None:
        r.left = p
    if p is None:
        return r
    return p
```



This function deletes a node pointed to by self. We assume the node pointed to by self is the first node of the DLL.

The doublylinkedlist.py file - delete

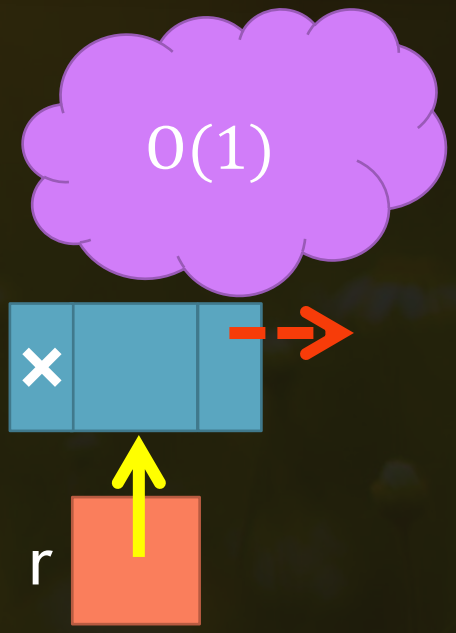
```
def delete(self):
    #p  q   r
    p = self.left
    q = self
    r = self.right
    if p is not None:
        p.right = r
    if r is not None:
        r.left = p
    if p is None:
        return r
    return p
```



This function deletes a node pointed to by self. We assume the node pointed to by self is the first node of the DLL.

The doublylinkedlist.py file - delete

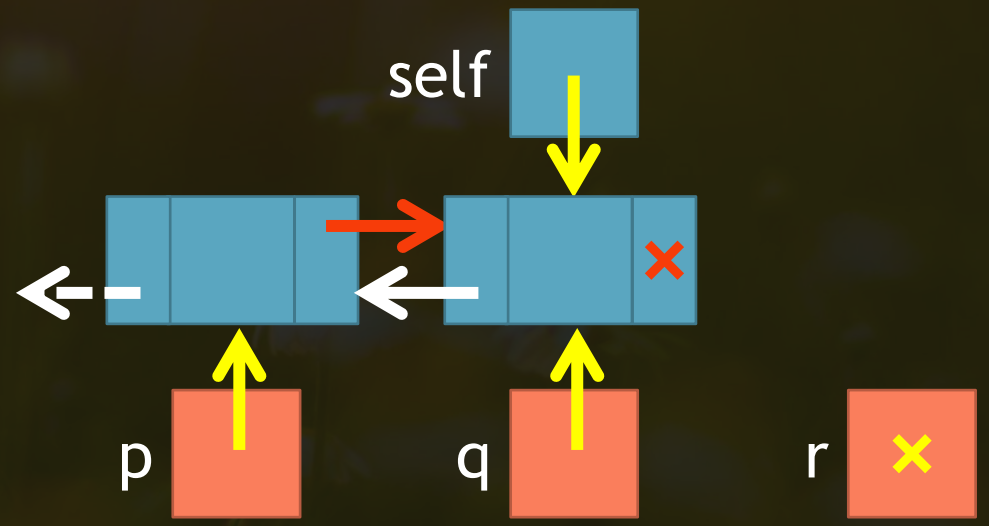
```
def delete(self):
        #p  q   r
        p = self.left
        q = self
        r = self.right
        if p is not None:
            p.right = r
        if r is not None:
            r.left = p
        if p is None:
            return r
        return p
```



This function deletes a node pointed to by self. We assume the node pointed to by self is the first node of the DLL.

The doublylinkedlist.py file - delete

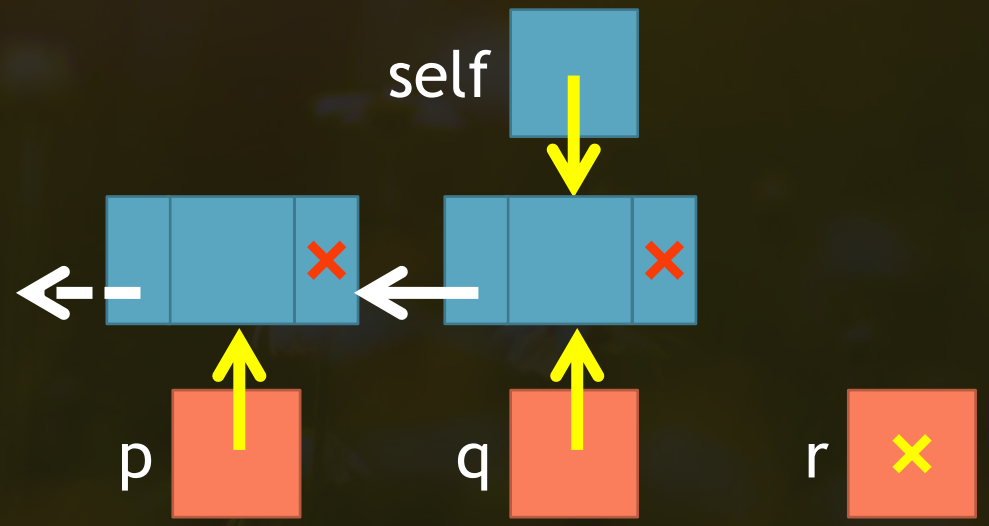
```
def delete(self):
    #p  q   r
    p = self.left
    q = self
    r = self.right
    if p is not None:
        p.right = r
    if r is not None:
        r.left = p
    if p is None:
        return r
    return p
```



This function deletes a node pointed to by self. We assume the node pointed to by self is the last node of the DLL.

The doublylinkedlist.py file - delete

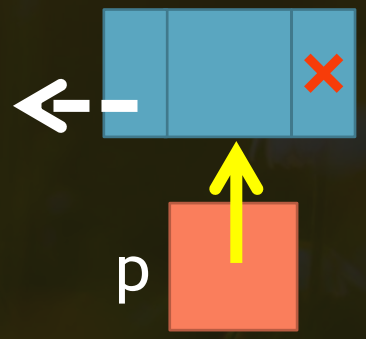
```
def delete(self):
    #p  q   r
    p = self.left
    q = self
    r = self.right
    if p is not None:
        p.right = r
    if r is not None:
        r.left = p
    if p is None:
        return r
    return p
```



This function deletes a node pointed to by self. We assume the node pointed to by self is the last node of the DLL.

The doublylinkedlist.py file - delete

```
def delete(self):
        #p  q   r
        p = self.left
        q = self
        r = self.right
        if p is not None:
            p.right = r
        if r is not None:
            r.left = p
        if p is None:
            return r
        return p
```

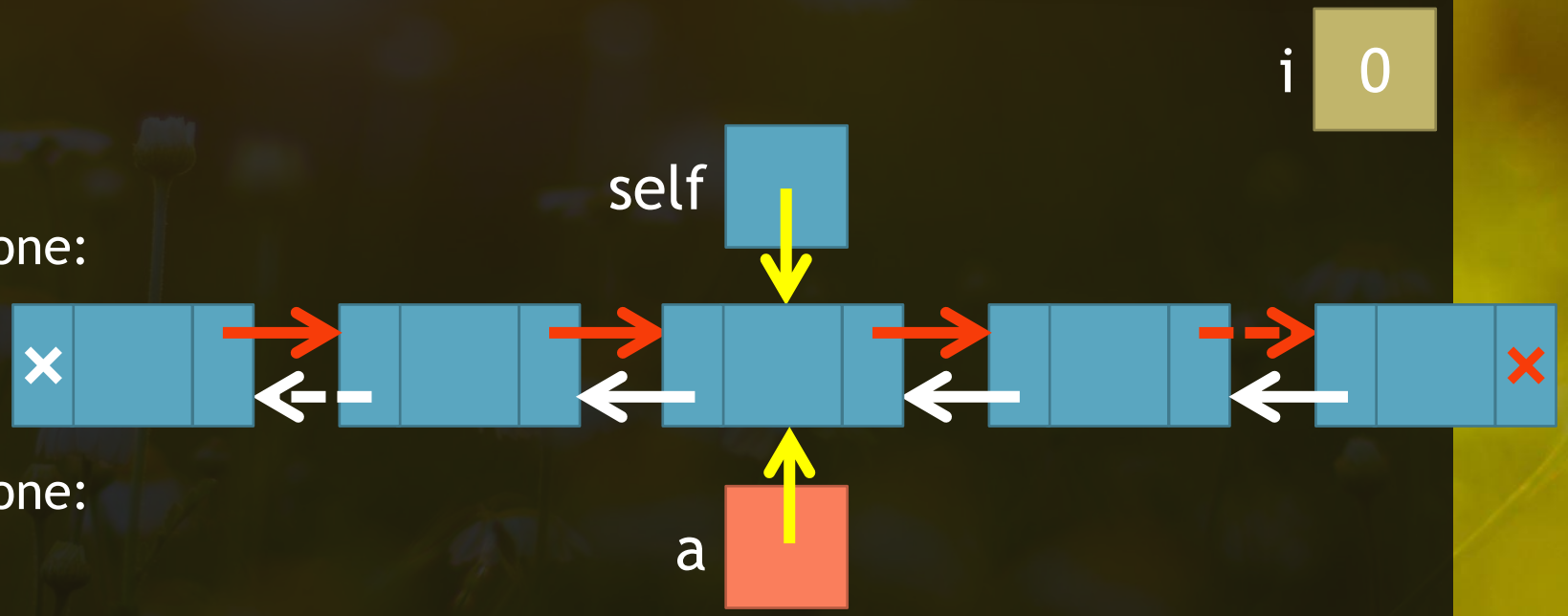


O(1)

This function deletes a node pointed to by self. We assume the node pointed to by self is the last node of the DLL.

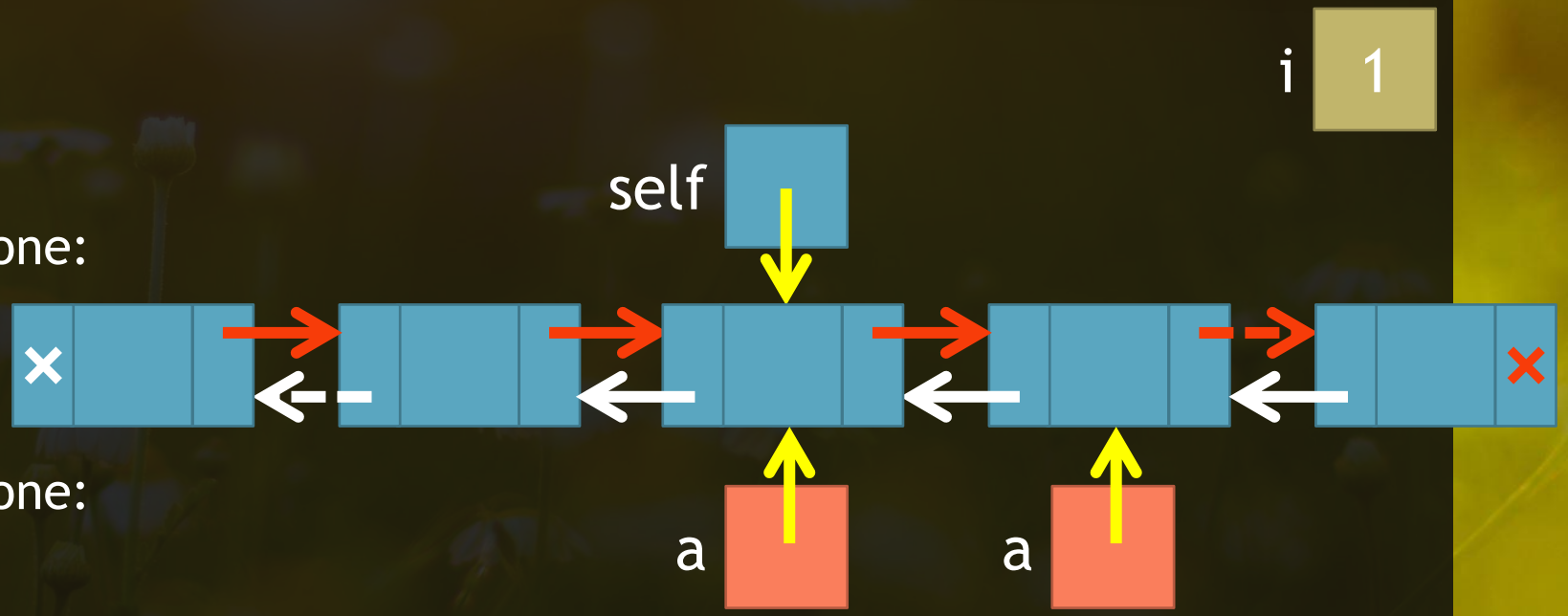
The doublylinkedlist.py file - len

```
def __len__(self):
    a = self
    i = 0
    while a is not None:
        i += 1
        a = a.right
    a = self.left
    while a is not None:
        i += 1
        a = a.left
    return i
```



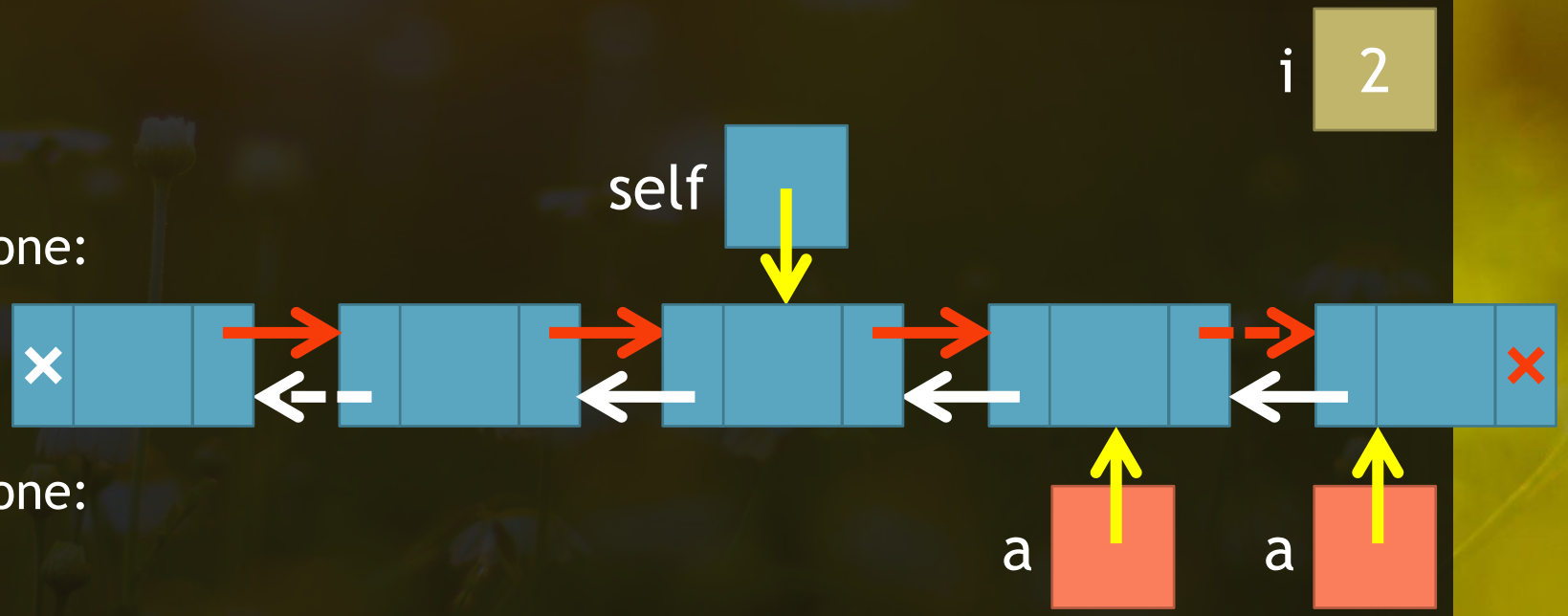
The doublylinkedlist.py file - len

```
def __len__(self):
    a = self
    i = 0
    while a is not None:
        i += 1
        a = a.right
    a = self.left
    while a is not None:
        i += 1
        a = a.left
    return i
```



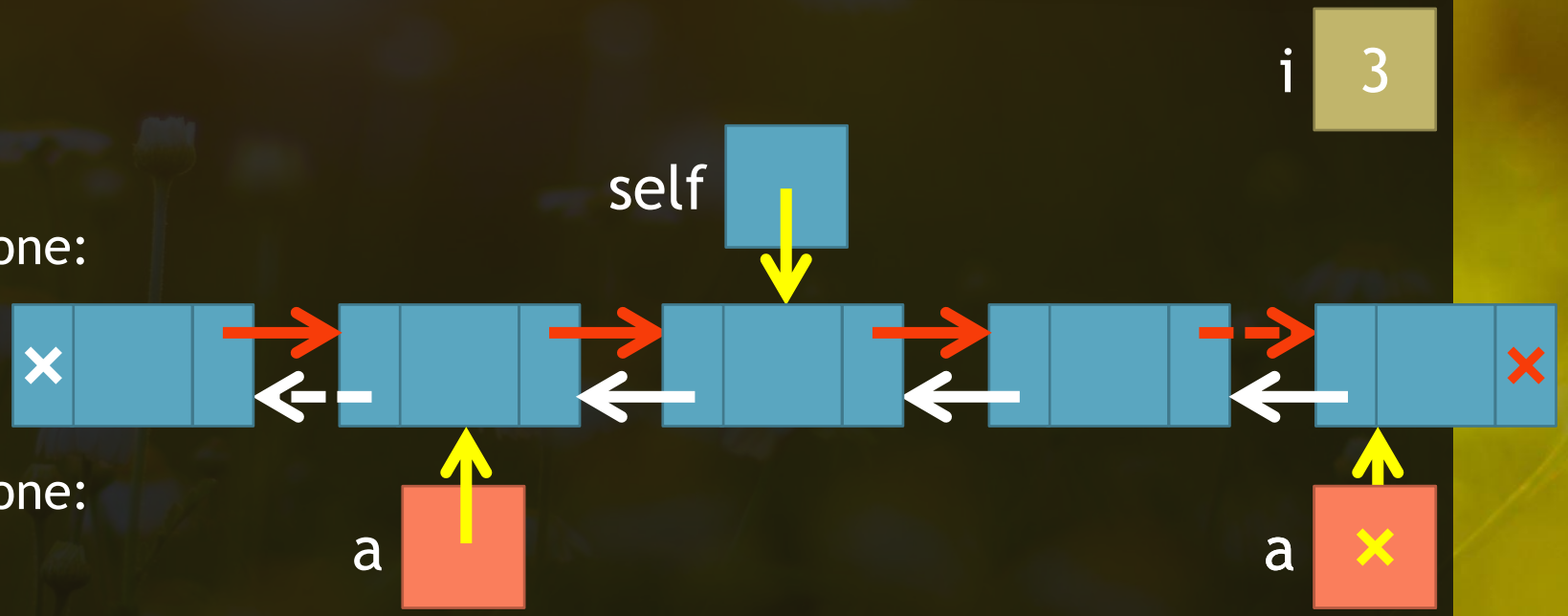
The doublylinkedlist.py file - len

```
def __len__(self):
    a = self
    i = 0
    while a is not None:
        i += 1
        a = a.right
    a = self.left
    while a is not None:
        i += 1
        a = a.left
    return i
```



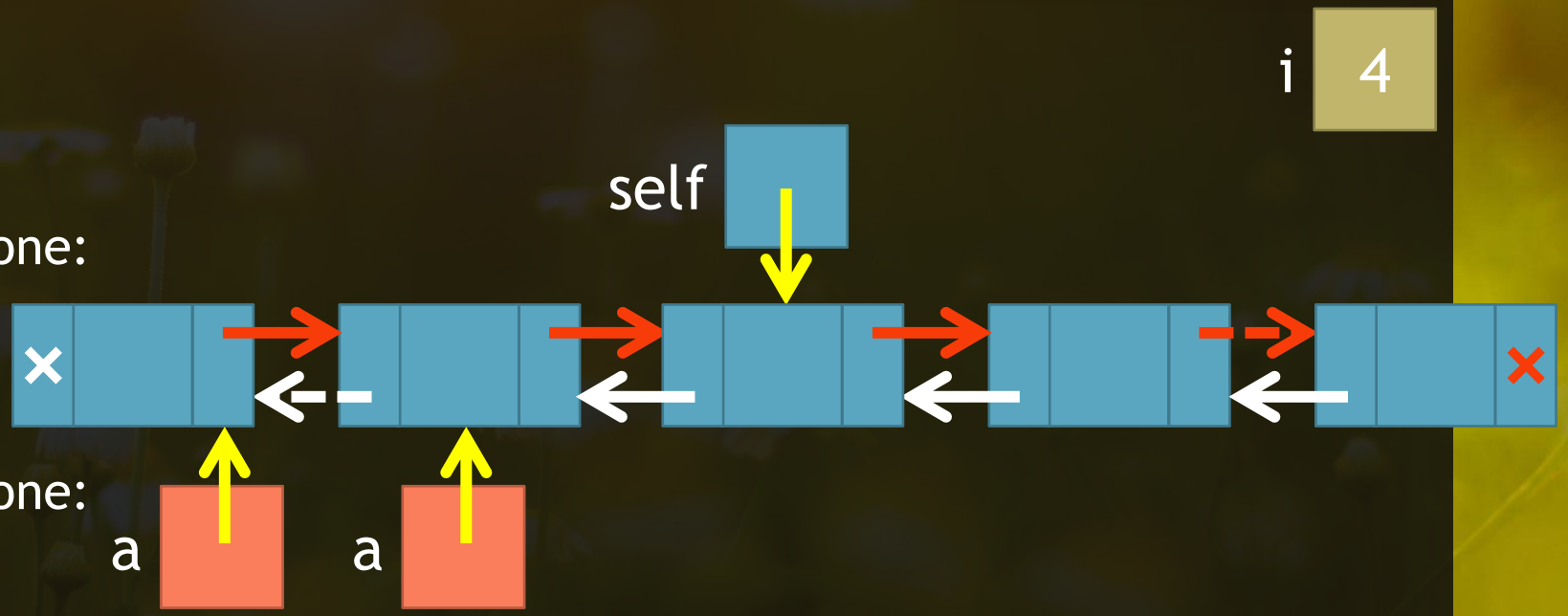
The doublylinkedlist.py file - len

```
def __len__(self):
    a = self
    i = 0
    while a is not None:
        i += 1
        a = a.right
    a = self.left
    while a is not None:
        i += 1
        a = a.left
    return i
```



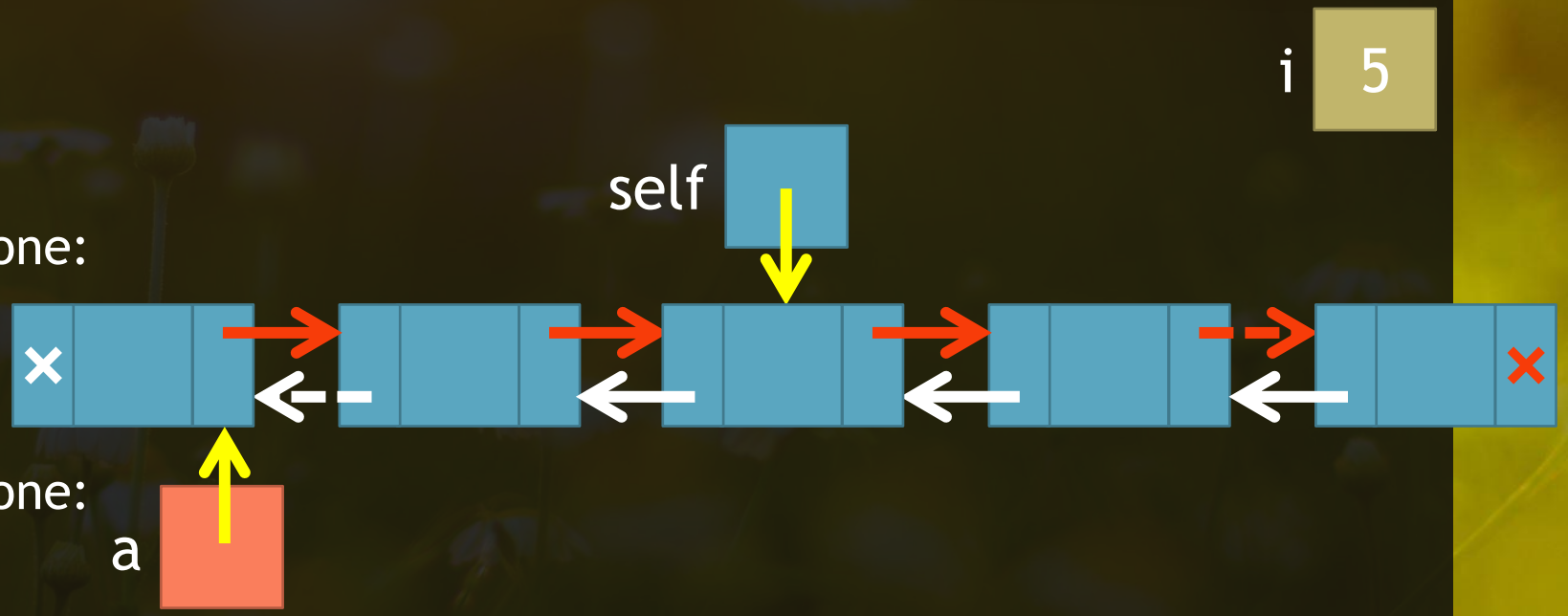
The doublylinkedlist.py file - len

```
def __len__(self):
    a = self
    i = 0
    while a is not None:
        i += 1
        a = a.right
    a = self.left
    while a is not None:
        i += 1
        a = a.left
    return i
```



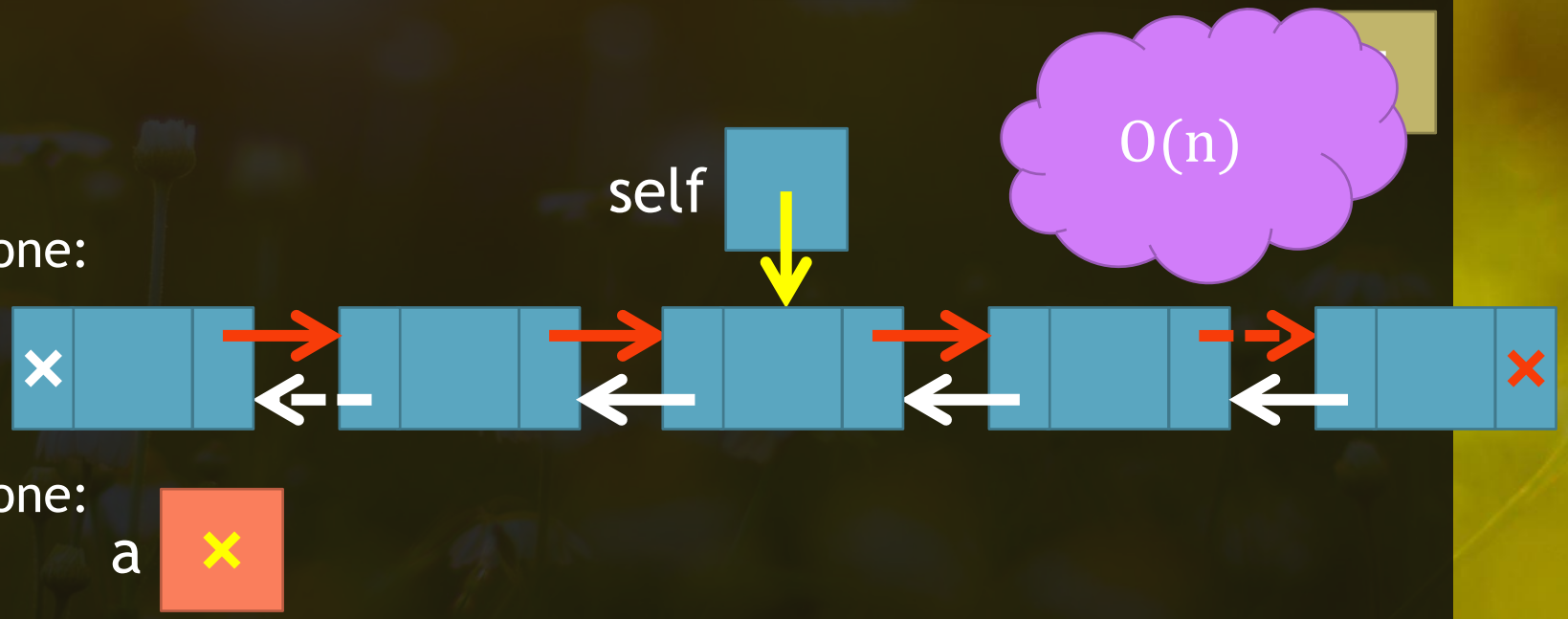
The doublylinkedlist.py file - len

```
def __len__(self):
    a = self
    i = 0
    while a is not None:
        i += 1
        a = a.right
    a = self.left
    while a is not None:
        i += 1
        a = a.left
    return i
```



The doublylinkedlist.py file - len

```
def __len__(self):
    a = self
    i = 0
    while a is not None:
        i += 1
        a = a.right
    a = self.left
    while a is not None:
        i += 1
        a = a.left
    return i
```



The doublylinkedlist.py file - traverse

```
def traverse(self):
        a = self
        #go all the way back to the left
        while a.left is not None:
            a = a.left
        #now go all the way to the right
        print("Traversing...")
        while a is not None:
            print(a.data,end=" ")
            a = a.right
        print()
```

O(n)

Draw step by step pictures depicting how this function would execute

The doublylinkedlist.py file - search

```
def search(self,target):
    b=self
    #check the nodes on the right
    while b is not None and b.data!=target:
        b=b.right
    if  b is not None:
        return b
    # check the nodes on the left
    b=self.left
    while b is not None and b.data!=target:
        b=b.left
    return b
```

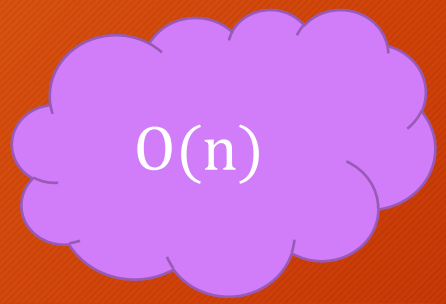
O(n)

Draw step by step pictures depicting how this function would execute

Building a doubly linked list towards the right

```
from doublylinkedlist import DLNode

def buildlistright(val):
    assert len(val)>0,"no elements"
    a=DLNode(val[0])
    for i in range(1,len(val),1):
        a.insertright(val[i])
        a=a.right
    return a
```



Draw step by step pictures depicting how this function would execute

- val is a Python list containing the data we want to store in a doubly linked list.
- val must have one or more elements.
- We create a node and put val[0] in it. A pointer a points to this node.
- A for loop runs from 1 to n – 1.
- Inside the loop in iteration i, we insert a node containing val[i] to the right of a.
- Then we advance a to the right.
- When all elements of val have been stored, we come out of the loop and return a, which now points to the extreme right node.

Homework

We want to build a doubly linked list towards the left. The function is called buildlistleft(val). Write the body of this function. Return a pointer to the list. Analyze the function for time complexity.

So what did we learn today?

- We were introduced to circular singly linked lists.
- We implemented functions for CSLLs.
- We were introduced to doubly linked lists.
- We implemented a class to create and manage a doubly linked list.
- We saw pictorial depictions of code fragments.
- We learned how to build a DLL.
- We found time complexities of all codes.

Things to do

Read the book!

Note your questions and put them up in the relevant online session.

Email suggestions on content or quality of this lecture at uroojain@neduet.edu.pk