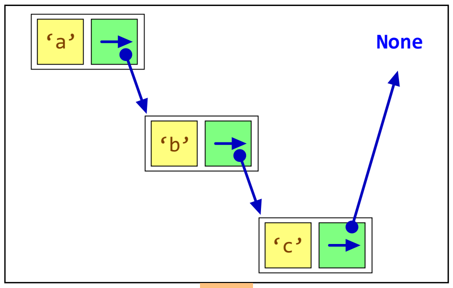
Reverse single linked list

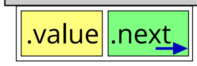
by yuri@silkmind.com

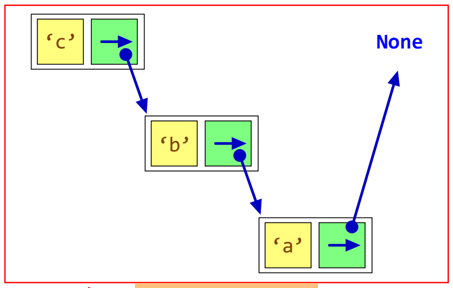
```python
class Node:
    def __init__(self, value, next_ = None):
        self.value = value
        self.next = next_


head = Node('a', Node('b', Node('c'))) # a -> b -> c

# O(1) - extra memory
# O(n) - time

def reverse(node):
    # return c -> b -> a
    pass
```

Task



Node class





Task

ALARM! !THINK FIRST! DON'T READ THE NEXT PART

Solution: go through the **linked list** and change next pointers to the previous element and at the end - return the last element.

solution (debug version)

```python
def prt(node):
    i = 0
    n = node
    while True and i<10:
        print(n.value)
        i += 1
        if n.next == None:
            return
        else:
            n = n.next

def reverse(node):
    first = node
    prev  = node
    cur   = node

    i = 0
    if cur == None:
        return node

    if cur.next != None:
        cur  = cur.next

    while cur.next != None:
        print(i, cur.value, cur.next, prev.value, prev.next)
        cur.next, prev, cur = prev, cur, cur.next
        i += 1

    last = cur
    first.next, last.next = None, prev

    return last
```

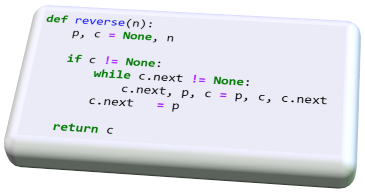
>> in

```python
head = Node('a', Node('b', Node('c', Node('d')))) # a -> b -> c

prt(head)
print("#: reverse")
r = reverse(head)
prt(r)
print("#: print original head")
prt(head)
```

<< out

```
a
b
c
d
#: reverse
0 b <__main__.Node object at 0x0000014D90E09948>
  a <__main__.Node object at 0x0000014D90E099C8>
1 c <__main__.Node object at 0x0000014D90E09E08>
  b <__main__.Node object at 0x0000014D90E09A88>
d
c
b
a
#: print original head
a
```

short version:



```python
def reverse(n):
    p, c = None, n

    if c != None:
        while c.next != None:
            c.next, p, c = p, c, c.next
        c.next    = p

    return c
```

Reverse single linked list

by yuri@silkmind.com

solution (debug version)

```
def prt(node):
    i = 0
    n = node
    while True and i<10:
        print(n.value)
        i += 1
        if n.next == None:
            return
        else:
            n = n.next

def reverse(node):
    first = node
    prev  = node
    cur   = node

    i = 0
    if cur == None:
        return node

    if cur.next != None:
        cur  = cur.next

    while cur.next != None:
        print(i, cur.value, cur.next, prev.value, prev.next)
        cur.next, prev, cur = prev, cur, cur.next
        i += 1

    last = cur
    first.next, last.next = None, prev

    return last
```

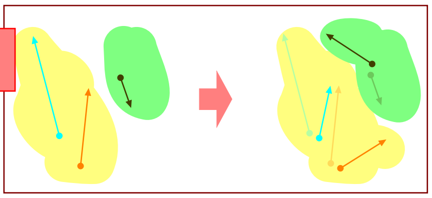
init phase:
init first element,
previous and current

case for empty list [when node == None]
=> reverse(None) return None

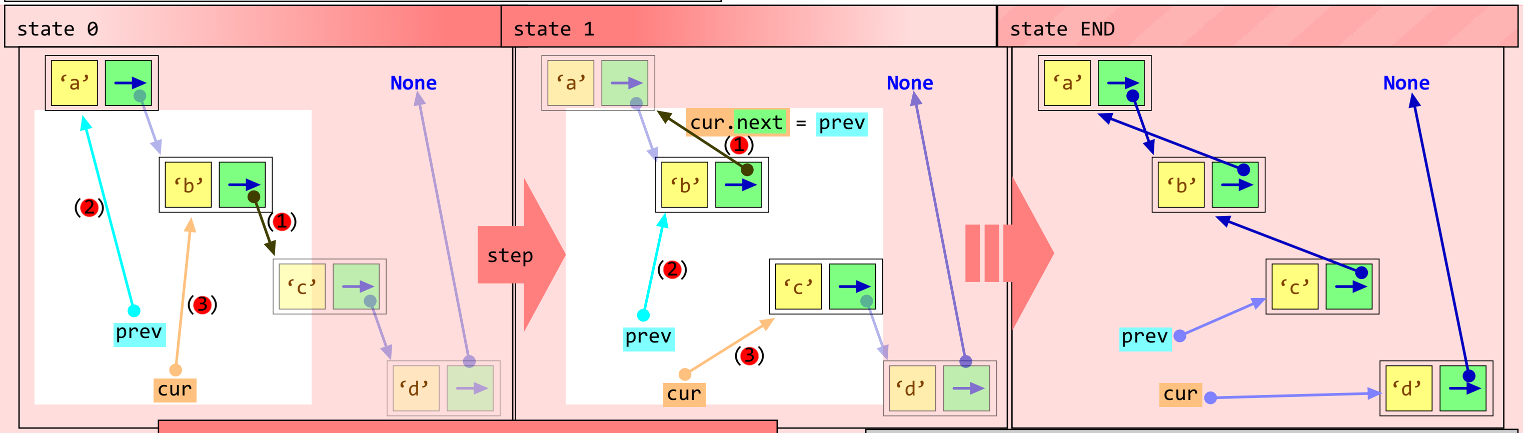
we want to have previous and current element which point on the right position. prev -> node, cur -> node.next
so at the beggining we try to make one step on the next element
prev = node; cur = cur.next (second element)

algorithm step

completion step



context (2),(3) changes (1)



algorithm step

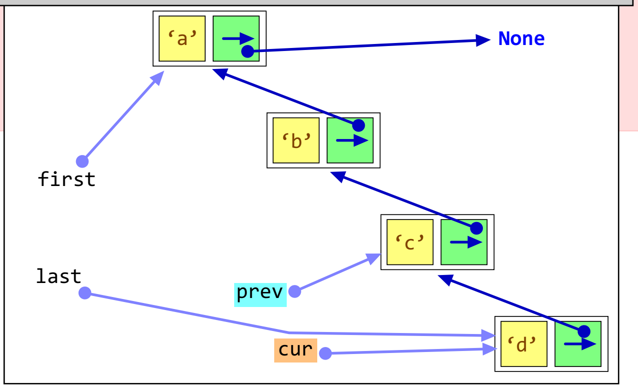
```
cur.next = prev
    prev = cur
     cur = cur.next
```

```
while cur.next != None:
  cur.next, prev, cur = prev, cur, cur.next
    (1)      (2)   (3)
```

completion step

```
last = cur
first.next, last.next = None, prev

return last
```

result of **completion step**

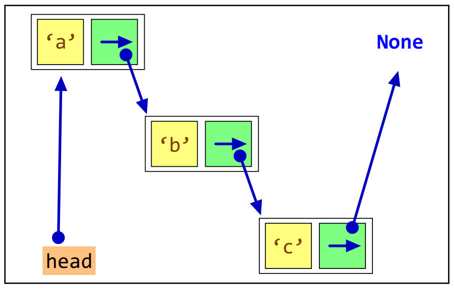


Reverse single linked list

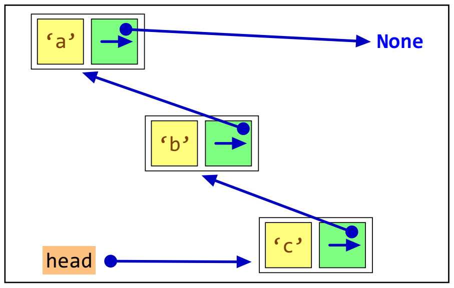
by yuri@silkmind.com

final solution

```
class Node:
    def __init__(self, value, next_ = None):
        self.value = value
        self.next = next_


head = Node('a', Node('b', Node('c'))) # a -> b -> c

# O(1) - extra memory
# O(n) - time

def reverse(node):
    prev, cur = None, node

    if cur == None:
        return node

    while cur.next != None:
        cur.next, prev, cur = prev, cur, cur.next
    cur.next = prev

    return cur

head = reverse(head)
```



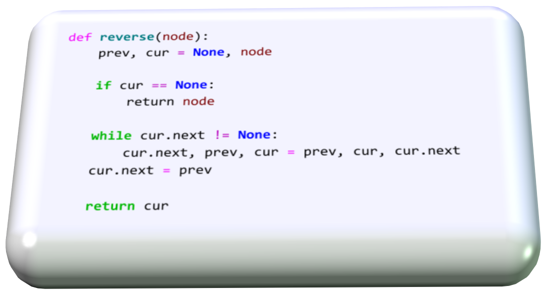
before reverse()



after reverse()

final solution version in AML language

```
s: prev
   cur


def backward(s):
  # in st(s):
    s.cur.next = s.prev


def step_forward(s) aka st:
  if s.cur.next != None
    s.prev, s.cur = s.cur, s.cur.next
    return True
  else:
    return False


rev   def reverse(node)
        s: .prev = None
           .cur  = node

        while ⇄ on s          while st(s):
                                backward(s)

        s.cur.next = s.prev

        return s.cur
```



```
def reverse(node):
    prev, cur = None, node

    if cur == None:
        return node

    while cur.next != None:
        cur.next, prev, cur = prev, cur, cur.next
    cur.next = prev

    return cur
```

Final solution complexity = O(n),
all solutions use O(1) extra memory.