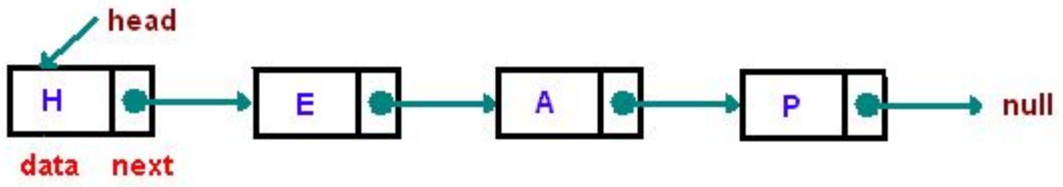
Lecture 15

Linked Lists. Part 2

Last time:



```
class LinkNode:

    def __init__(self, value, nxt = None):

        assert isinstance(nxt, LinkNode) or nxt is None

        self.value = value
        self.next = nxt
```

Let’s practice with simple functions

# Assume there are elements in the list. In the lab you should cover the case where list might be empty.

```
class LinkNode:
    def __init__(self,value,nxt = None):
       assert ...
        self.value = value
        self.next = nxt
```

```
def print_list(lst):

    while lst != None:
        print(lst.value, end = " ")
        lst = lst.next

    print()
```

```
nodes = LinkNode(1, LinkNode(2, None))
print_list(nodes)
```

Replace Last element

Write a function that replaces the last element of the list with a given element:

```
def replace_last (lst, elem):
“”””
>>> node2 = LinkNode(1, LinkNode(2))
>>> replace_last(node2, 3)
>>> print_list (node2)
1 3

>>> replace_last(LinkNode(None, None), 3)
“”””
```

```
def replace_last (lst, elem):
    # check if empty:
    if lst.next == None and lst.value == None:
        return lst
    if lst.next == None and lst.value != None:
        lst.value = elem

    while lst.next != None:
        lst = lst.next

    lst.value = elem
```

Replace Last element

```
>>> nodes = LinkNode(1, LinkNode(2, LinkNode(3)))
>>> replace_last(nodes, 17)
>>> print_list(nodes)
```

```
A:  1 2 3
B:  1 2 17
C:  1 17 3
D:  17 2 3
E:  Error
```

```python
def replace_last (lst, elem):
    # check if empty:
    if lst.next == None and lst.value == None:
        return lst
    if lst.next == None and lst.value != None:
        lst.value = elem

    while lst != None:
        lst = lst.next

    lst.value = elem
```

Practice Question

-
  ```
  def extend_link(s, t):
      """Return a list with the elements of s followed by those of t.
      >>> s = LinkNode(1, LinkNode(2, LinkNode(3)))
      >>> t = LinkNode(4, LinkNode(5, LinkNode(6)))
      >>> print_list(extend_link(s, t))
      1 2 3 4 5 6
  ```

Practice Question. Solution

-
  ```
  def extend_link(s, t):
    """Return a Link with the
  elements of s followed by
  those of t.
    >>> s = LinkNode(1,
  LinkNode(2, LinkNode(3)))
    >>> t = LinkNode(4,
  LinkNode(5, LinkNode(6)))
    >>> print_list(extend_link(s,
  t))
    1 2 3 4 5 6
  ```

```
def extend_list (s, t):
    # check if empty:
    if s.next == None and s.value == None:
        return t
    else:
        start = s  # otherwise s is destroyed
        while s.next != None:
            s = s.next

        s.next = t
        return start
```

Practice Question. (ENV Diagram)

```
>>> s = LinkNode(1, LinkNode(2, LinkNode(3)))
>>> t = LinkNode(4, LinkNode(5, LinkNode(6)))
>>> extend_link(s, t)
>>> print_list(s)
```

```
A: 1 2 3
B: 4 5 6
C: 1 2 3 4 5 6
D: 3 4 5 6
E: None
```

```
def extend_list (s, t):
    # check if empty:
    if s.next == None and s.value == None:
        return t
    else:
        start = s  # otherwise s is destroyed
        while s.next != None:
            s = s.next

        s.next = t
        return start
```

Practice Question

```
s = LinkNode(3, LinkNode(4, LinkNode(5)))
square = lambda x: x * x


def map_link(s, f):
    """Apply f to each element of s.

    >>> map_link(s, square)
    LinkNode(9, LinkNode(16, LinkNode(25)))
    """
```

Practice Question. Solution

```
s = LinkNode(3, LinkNode(4,
LinkNode(5)))
square = lambda x: x * x


def map_link(s, f):
    """Apply f to each element of s.

    >>> map_link(s, square)
    LinkNode(9, LinkNode(16,
LinkNode(25)))
    """
```

```
def map_link(s, f):

    if s.next == None and s.value == None:
        return s
    else:
        while s != None:
            s.value = f(s.value)
            s = s.next
```

Practice Question.

```
s = LinkNode(3, LinkNode(4, LinkNode(5)))
square = lambda x: x * x


def map_link(s, f):
    """Apply f to each element of s.

    >>> map_link(s, square)
    LinkNode(9, LinkNode(16, LinkNode(25)))
    """
```

```
def map_link(s, f):

    if s.next == None and s.value == None:
        return s
    else:
        while s.next != None:
            s.value = f(s.value)
            s = s.next
```

Output?

```
A:  9   16   25
B:  9   16    5
C:  3    4    5
D:  9    4    5
E:  Error
```

Practice Question

```
def remove_second(t):
    """
    >>> s = LinkNode(1, LinkNode(2, LinkNode(3, LinkNode(4))))
    >>> remove_second(s)
    >>> print_list(s)
    1 3 4
    """
```

Practice Question + Env. diagram

```
def remove_second(t):
    """
    >>> s = LinkNode(1,
LinkNode(2, LinkNode(3,
LinkNode(4))))
    >>> remove_second(s)
    >>> print_list(s)
    1 3 4
    """
```

```
def remove_second(s):
    if s.next == None and s.value == None:
        return s
    if s.next == None and s.value != None:
        return s



    before = s
    after  = s.next.next

    before.next = after
```

Practice Question. Do not have to use before

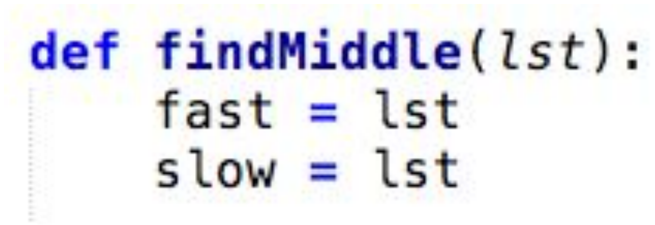
```
def remove_second(t):
    """
    >>> s = LinkNode(1,
LinkNode(2, LinkNode(3,
LinkNode(4))))
    >>> remove_second(s)
    >>> print_list(s)
    1 3 4
    """
```

```
def remove_second(s):
    if s.next == None and s.value == None:
        return s
    if s.next == None and s.value != None:
        return s

    after  = s.next.next
    s.next = after
```

Practice Question: find middle

Write a function that finds a middle of a linked list. You can't use len method.

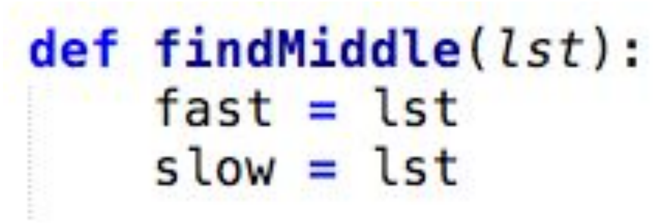


```
def findMiddle(lst):
    fast = lst
    slow = lst
```

```
>>> lst = LinkNode(3,  LinkNode (4, LinkNode (5, LinkNode (6, LinkNode(7)))))
>>> mid = findMiddle(lst)
>>> 5
```

Practice Question: find middle

Write a function that finds a middle of a linked list. You can't use len method.



```
def findMiddle(lst):
    fast = lst
    slow = lst
```

```
>>> lst = LinkNode(3,  LinkNode (4,
LinkNode (5, LinkNode (6,
LinkNode(7)))))
>>> mid = findMiddle(lst)
>>> 5
```

```
def find_middle(lst):
    fast = lst
    slow = lst

    while fast != None and fast.next != None:
        fast = fast.next.next
        slow = slow.next

    return slow.value
```