LINKED LISTS

COMPUTER SCIENCE MENTORS 61A

February 22 to February 26, 2016

For each of the following problems, assume linked lists are defined as follows:

```
class Link:
```

```
empty = ()

def __init__(self, first, rest=empty):
    assert rest is Link.empty or isinstance(rest, Link)
    self.first = first
    self.rest = rest
```

To check if a \mathtt{Link} is empty, compare it against the class attribute \mathtt{Link} . empty:

```
if link is Link.empty:
    print('This linked list is empty!')
```

1 What Would Python Print?

1. What will Python output? Draw box-and-pointer diagrams to help determine this.

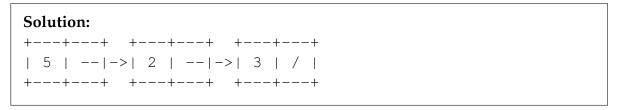
>>>
$$a = Link(1, Link(2, Link(3)))$$

```
Solution:
+---+---+ +---+---+
| 1 | --|->| 2 | --|->| 3 | / |
+---+---+ +---+---+
```

>>> a.first

Solution:

>>> a.first = 5



>>> a.first

Solution: 5

>>> a.rest.first

```
Solution: 2
```

>>> a.rest.rest.rest.rest.first

Solution: Error: tuple object has no attribute rest (Link.empty has no rest)

>>> a.rest.rest.rest = a

>>> a.rest.rest.rest.first

```
Solution: 2
```

2 Code Writing Questions

2. Write a function skip, which takes in a Link and returns a new Link.

```
def skip(lst):
    """
    >>> a = Link(1, Link(2, Link(3, Link(4))))
    >>> a
    Link(1, Link(2, Link(3, Link(4))))
    >>> b = skip(a)
    >>> b
    Link(1, Link(3))
    >>> a
    Link(1, Link(3, Link(4)))) # Original is unchanged
    """
```

```
Solution:
    if lst is Link.empty:
        return Link.empty
    if lst.rest is Link.empty:
        return lst
    return Link(lst.first, skip(lst.rest.rest))
```

3. Now write function skip by mutating the original list, instead of returning a new list. Do NOT call the Link constructor.

```
def skip(lst):
    """
    >>> a = Link(1, Link(2, Link(3, Link(4))))
    >>> b = skip(a)
    >>> b
    Link(1, Link(3))
    >>> a
    Link(1, Link(3))
    """
```

```
Solution:
def skip(lst): # Recursively
   if lst is Link.empty or lst.rest is Link.empty:
       return lst
   lst.rest = skip(lst.rest.rest)
       return lst

def skip(lst): # Iteratively
   if lst is Link.empty:
       return Link.empty
   original = lst
   while lst.rest is not Link.empty:
       lst.rest = lst.rest.rest
       lst = lst.rest
   return original
```

4. Write a function reverse, which takes in a Link and returns a new Link that has the order of the contents reversed.

Hint: You may want to use a helper function if you're solving this recursively.

```
def reverse(lst):
    """
    >>> a = Link(1, Link(2, Link(3)))
    >>> b = reverse(a)
    >>> b
    Link(3, Link(2, Link(1)))
    >>> a
    Link(1, Link(2, Link(3)))
    """
```

```
Solution: There are quite a few different methods. We have listed some here –
can you think of any others?
# Recursive w/ Helper
def reverse(lst):
    def helper(so_far, rest):
        if rest is Link.empty:
             return so_far
        else:
             return helper(Link(rest.first, so_far), rest.
    return helper(Link.empty, lst)
# Recursive w/o Helper
def reverse(lst):
    if lst is Link.empty or lst.rest is Link.empty:
        return 1st.
    secondElement = lst.rest
    lst.rest = Link.empty
    reversedRest = reverse(secondElement)
    secondElement.rest = lst
    return reversedRest
# Iterative
def reverse(lst):
    rev = Link.empty
    while 1st is not Link.empty:
        rev = Link(lst.first, rev)
        lst = lst.rest
    return rev
```

5. (Optional) Implement negation so that linked lists have the following behaviour:

```
>>> a = Link(1, Link(2, Link(3)))
>>> -a # This should output a new Linked List
Link(3, Link(2, Link(1)))
```

You may use your work from question 4.

Hint:

```
>>> a = 4
>>> -a
-4
>>> a.__neg__()
```

```
Solution:
```

```
# In class Link
def __neg__(self):
    return reverse(self)
```

6. **(Optional)** Now write reverse by modifying the existing Links. Assume reverse returns the head of the new list (so the last Link object of the previous list).

First, draw out the box and pointer for the following:

```
>>> a = Link(1, Link(2))
>>> a.rest.rest = a
>>> a.rest = Link.empty
```

Observe how the pointers change, as well as the order in which they are modified.

Solution:

Now, generalize this to reverse an entire linked list.

```
def reverse(lst):
    """
    >>> a = Link(1, Link(2, Link(3)))
    >>> b = reverse(a)
    >>> b
    Link(3, Link(2, Link(1)))
    >>> a
    Link(3, Link(2, Link(1)))
    """
```

```
Solution: Here are two possible solutions.
def reverse(lst):
    if lst == Link.empty or lst.rest == Link.empty:
        return lst
    else:
        new_start = reverse(lst.rest)
        lst.rest.rest = lst
        lst.rest = Link.empty
        return new_start

def reverse(lst):
    if lst.rest is not Link.empty:
        second, last = lst.rest, lst
        lst = reverse(second)
        second.rest, last.rest = last, Link.empty
    return lst
```

7. (Optional) Write has_cycle which takes in a Link True if and only if there is a cycle in the Link.

```
def has_cycle(s):
    """
    >>> has_cycle(Link.empty)
    False
    >>> a = Link(1, Link(2, Link(3)))
    >>> has_cycle(a)
    False
    >>> a.rest.rest.rest = a
    >>> has_cycle(a)
    True
    """
```

```
Solution:
    if s is Link.empty:
        return False
        slow, fast = s, s.rest
    while fast is not Link.empty:
        if fast.rest is Link.empty:
            return False
        elif fast is slow or fast.rest is slow:
            return True
        slow, fast = slow.rest, fast.rest.rest
        return False
```