LINKED LISTS

COMPUTER SCIENCE MENTORS 61A

February 22 to February 26, 2016

1. Given the following code, what will Python output for the following prompts?

```
class Baller:
        all_players = []
        def __init__(self, name, has_ball = False):
                self.name = name
                self.has_ball = has_ball
                Baller.all_players.append(self)

        def pass_ball(self, other_player):
                if self.has_ball:
                self.has_ball = False
                other_player.has_ball = True
                return True
        else:
                return False

class BallHog(Baller):
        def pass_ball(self, other_player):
                return False
>>> tiffany = Baller('Tiffany', True)
>>> garrett = BallHog('Garrett')
>>> len(Baller.all_players)

>>> Baller.name

>>> len(garrett.all_players)
```

```
>>> tiffany.pass_ball()

>>> tiffany.pass_ball(garrett)

>>> tiffany.pass_ball(garrett)

>>> BallHog.pass_ball(garrett, tiffany)

>>> garrett.pass_ball(tiffany)

>>> garrett.pass_ball(garrett, tiffany)
```

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(2, Link(3, Link(4)))) # Original is unchanged
    """
```

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))
    """
```

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)))
    """
```

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__()
-4
```

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.

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)))
    """
```

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
    """
```