FINAL REVIEW

1 What Would Python Print? Iterators

1.
```python
class SkipIterator:
    def __init__(self, rng, n):
        self.obj = rng
        self.skip = n

    def __iter__(self):
        return self

    def __next__(self):
        result = self.obj.curr
        self.obj.curr += self.skip
        return result

class SkippedNaturals:
    def __init__(self):
        self.curr = 0
        self.skip = 1

    def __iter__(self):
        return SkipIterator(self, self.skip)
```

Expression	Interactive Output

```p = SkippedNaturals()```
```twos = iter(p)```
```p.skip = p.skip + 1```
```threes = iter(p)```
```next(twos)```	

```next(twos)```	
```next(threes)```	
```next(threes)```	

## 2 Environment Diagrams

2. Draw the environment diagram for the following code snippet:

```
def one(two):
    three = two
    def four(five):
        nonlocal three
        if len(three) < 1:
            three.append(five)
            five = lambda x: four(x)
        else:
            five = seven + 7
        return five
    two = two + [1]
    seven = 8
    return four(three)

eight = one([])
print(eight(9))
```

3   Recursive Data Structures

3. DoubleTree hired you to architect one of their hotel expansions! As you might expect, their floor plan can be modeled as a tree and the expansion plan requires doubling each node (the patented double tree floor plan). Here's what some sample expansions look like:

Before   **After**
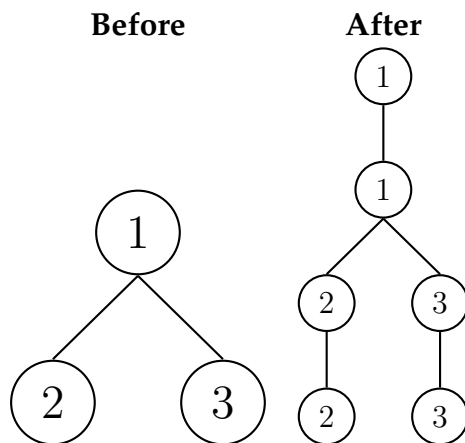
Fill in the implementation for `double_tree`.

```
def double_tree(t):
    """
    Given a tree, return a new tree where entries appear
    twice.
    >>> double_tree(Tree(1))
    Tree(1, [Tree(1)])
    >>> double_tree(Tree(1, [Tree(2), Tree(3)]))
    Tree(1, [Tree(1, [Tree(2, [Tree(2)]),
                      Tree(3, [Tree(3)])
                     ])
            ])
    """
```

4. Fill in the implementation of `double_link`.

```
def double_link(lst):
    """
    Using mutation, replaces the second in each pair of items
    with the first. The first of each pair stays as is.
    >>> double_link(Link(1, Link(2, Link(3, Link(4)))))
    Link(1, Link(1, Link(3, Link(3))))
    >>> double_link(
            Link('c', Link('s', Link(6, Link(1, Link('a')))))
        )
    Link('c', Link('c', Link(6, Link(6, Link('a')))))
    """
    if _____:
        return _____
    _____
    _____
    return _____
```

5. Fill in the implementation of `shuffle`.

```
def shuffle(lst):
    """
    Swaps each pair of items in a linked list.
    >>> shuffle(Link(1, Link(2, Link(3, Link(4)))))
    Link(2, Link(1, Link(4, Link(3))))
    >>> shuffle(
            Link('s', Link('c', Link(1, Link(6, Link('a')))))
        )
    Link('c', Link('s', Link(6, Link(1, Link('a')))))
    """
    if _____:
        return _____
    new_head = lst.rest
    lst.rest = _____
    _____
    return _____
```

4 Scheme

6. Write a Scheme function insert that creates a new list that would result from inserting an item into an existing list at the given index. Assume that the given index is between 0 and the length of the original list, inclusive.
```scheme
(define (insert lst item index)



)
```
Extra: Write this as a tail recursive function. Assume append is tail recursive.

5 Interpreters

7. Circle the number of calls to `scheme_eval` and `scheme_apply` for the code below.
```scheme
(define (square x) (* x x))
(+ (square 3) (- 3 2))
```

Calls to `scheme_eval` (circle one)	2 5 14 24
Calls to `scheme_apply` (circle one)	1 2 3 4

6 Recursive Select in SQL

8. Create a `mod_seven` table that has two columns, a number from 0 to 100 and then its value mod 7.
Hint: You can create a table first with all of the initial data you will build from, and then build the `mod_seven` table.

7 Iterators, Generators, and Streams

9. Write a generator that will take in two iterators and will compare the first element of each iterator and yield the smaller of the two values.

```
def interleave(iter1, iter2):
    """
    >>> gen = interleave(iter([1, 3, 5, 7, 9]),
                         iter([2, 4, 6, 8, 10]))
    >>> for elem in gen:
    ...     print(elem)
    ...
    1
    2
    3
    4
    5
    6
    7
    8
    9
    """
```

10. **Stream Supreme**

(a) You and your friends are preparing for the 61A final by streaming lectures. You get tired and want to take a rest from studying but first you realize you are hungry so you check the refrigerator. You notice that **every other food** (starting with the first food in the list) in there is stale! Write a function that takes in a `Linked List` of foods, called `foods` and outputs a stream that contains all your stale food. **Bonus:** Count all the puns in this question!

```
def stale_foods(foods):
```

(b) Can you magically find a way to have infinite food? Find a way to cycle through the foods so that when the last food is exhausted, the stream loops back to the first food.

```
def food_stream(foods):
```