Midterm 1 Review

Question 1

Draw box & pointer diagram for:

```
a = [1,2,3,4,5]
b = [a.append([a.pop(0)]) for _ in range(2*len(a))]
c = [[a[i]] + [b[i]] for i in range(len(a))]
```

Recall:

`lst.pop(i)` removes the `i`th index from the list and returns it

```
>>> lst = [6, 4, 8]
>>> lst.pop(1)
4
>>> lst
[6, 8]
```

Question 2

Finish implementing this `Link` class:

```
class Link:
  empty = ()

  def __init__(self, first, rest=Link.empty):


  def copy(self):
    """
    Returns a copy of the linked list
    """



  def __len__(self):
    """
    >>> lst = Link(1, Link(2, Link(3)))
    >>> len(lst)
    3
    """



  def __add__(self, other):
    """
    >>> lst1 = Link(1, Link(2, Link(3)))
    >>> lst2 = Link(4, Link(5, Link(6)))
    >>> lst1 + lst2
    Link(1, Link(2, Link(3, Link(4, Link(5, Link(6))))))
    """
```

Question 3

```
def knapsack(items, weight):
  """
  You must return the maximum weight can be put
  into the knapsack with the requirements that
  - the total weight is less weight
  - each item is used once
  - list of items is a list of number corresponding to weights
  - should be done recursively

  >>> knapsack([5,3,7], 10)
  10     # 3 + 7
  >>> knapsack([7,2,5,9], 13)
  12     # 7 + 5
  """
```

Question 4

```
class Shmoe:
  w = {}
  def __init__(self, x, y):
    if x in Shmoe.w:
      Schmoe.w[x].append(y)
      self.a = x * y
      self.s = Joe(self.a, 0)
      print(Shmoe.w)
    else:
      Schmoe.w[x] = [y]
      print('Wowie')

  def moe(self, h):

    def f(x):
      nonlocal h
      h = h*self.a
      return Joe(h, h)
    return f

class Joe(Shmoe):
  def __len__(self):
    return len(Shmoe.w[self.a])
```

What would python display? (for objects put: or if displayed, as well as for simplicity you can write `'hi'*3` instead of `'hihihi'`)

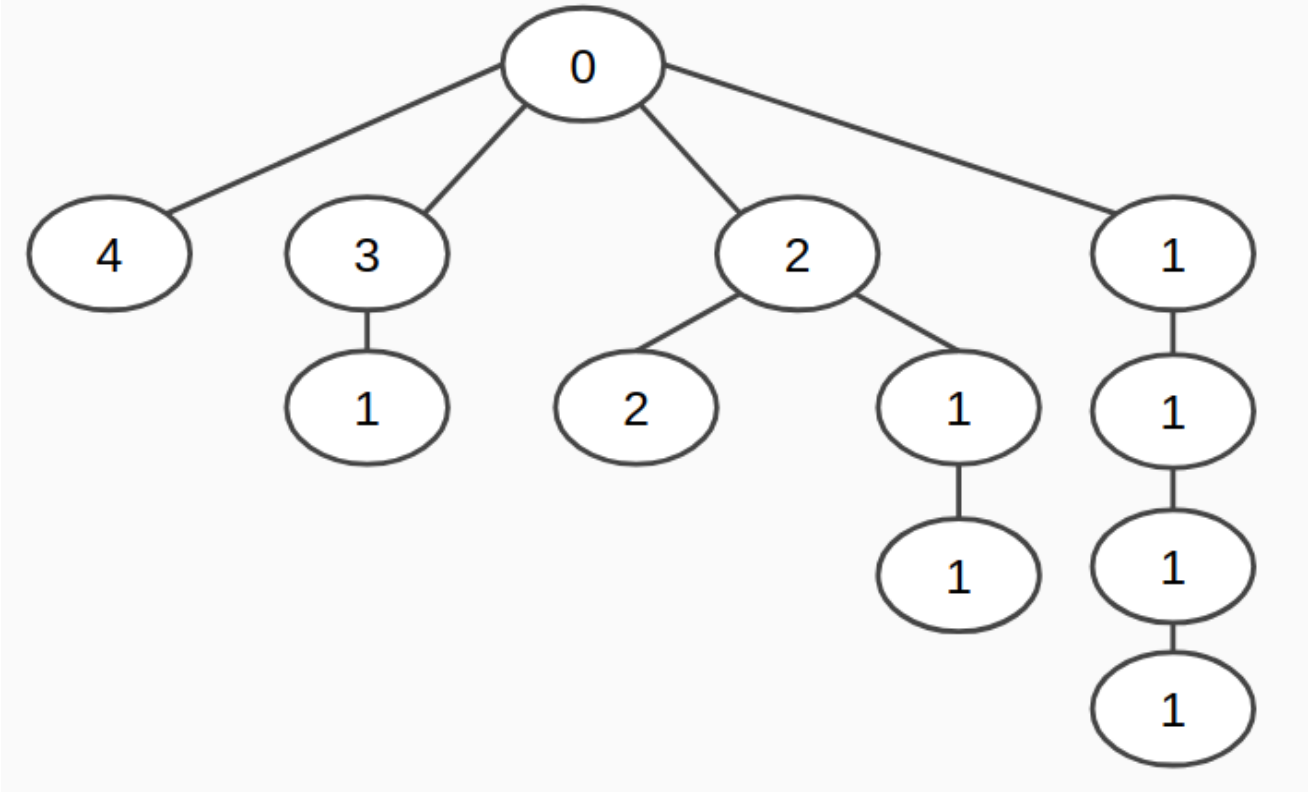
```
>>> a = Shmoe('hi', 3)
Wowie
>>> b = Shmoe('hi', 2)


>>> d = Shmoe(9, 9)


>>> c = b.moe(3)
>>> c(5)


>>> e = Joe(1, 3)


>>> f = e.moe(3)
>>> f(3)


>>> g = Shmoe(1, 3)


>> g.moe(3)(3)
```

Question 5 (Difficult)

Create a function that constructs a tree contains all of the possible partitions starting at n.

In other words, any path from root to leaf must sum to n

Example: `tree_partitions(4)`




```
def tree_partitions(n):
    """Returns a tree with each of the top level branches corresponding to a
    partition of n.
    >>> tree_partitions(1)
    Tree(0, [Tree(1)])
    >>> tree_partitions(2)
    Tree(0, [Tree(1, [Tree(1)]), Tree(2)])
    >>> tree_partitions(3)
    Tree(0, [Tree(1, [Tree(1, [Tree(1)])]), Tree(2, [Tree(1)]), Tree(3)])
    """
    assert n > 0
    def helper(n, m):
        if ___________:
            return _____________
        b = []
        for k in _________________:
          b += [helper(____,____)]
        return ____________________________________
    return Tree(0, [_____________________________ ])
```