EXAM PREPARATION SECTION 4

LIST MUTATION, DICTIONARIES, AND MORE TREES

February 27 to March 1, 2018

1 Lists

1. **Lots of Lists**
   Draw the environment diagram that results from executing the following code.

```
a = [1, 2, 3, 4, 5]
a.pop(3)
b = a[:]
a[1] = b
b[0] = a[:]
b.pop()
b.remove(2)
c = [].append(b[1])
a.insert(b.pop(1), a[-3:4:3])
b.extend(b)
if b == b[:] and b[1][1][0] is b[0][1][1]:
        a, b, c = [c] + a[-4:4:2]
```

2 Trees

1. **Tree Recursion with Trees**
   Fill in the function below so that it conforms to its docstring.

```
def about_equal(t1, t2):
    """Returns whether two trees are 'about equal.'
    Two trees are about equal if and only if they contain
    the same labels the same number of times.

    >> x = tree(1, [tree(2), tree(2), tree(3)])
    >> y = tree(3, [tree(2), tree(1), tree(2)])
    >> about_equal(x, y)
    True
    >> z = tree(3, [tree(2), tree(1), tree(2), tree(3)])
    >> about_equal(x, z)
    False
    """
    def label_counts(t):

        if is_leaf(t):

            return {label(t):1}

        else:

            counts = {}

            for b in branches(t) + ~~[t]~~ [tree(label(t))]:

                for label, count in ~~counts~~ label_counts(b).items():

                    if label not in counts:

                        counts[label] = 0

                    counts[label] += ~~label_counts(b)[label]~~ count

            return counts

    return label_counts(t1) == label_counts(t2)
```

3 Dictionaries

1. **What color is it? (Sp15 Midterm 2 Q4b)** Implement `decrypt`, which takes in a string `s` and a dictionary `d` that contains words as values and their secret codes as keys. It returns a list of all possible ways in which `s` can be decoded by splitting it into secret codes and separating the corresponding words by spaces.

```
def decrypt(s, d):
    """List all possible decoded strings of s.
    >>> codes = {
    ...     'alan': 'spooky',
    ...     'al': 'drink',
    ...     'antu': 'your',
    ...     'turing': 'ghosts',
    ...     'tur': 'scary',
    ...     'ing': 'skeletons',
    ...     'ring': 'ovaltine'
    ... }
    >>> decrypt('alanturing', codes)
    ['drink your ovaltine', 'spooky ghosts', 'spooky scary
       skeletons']
    """
    if s == '':

        return []

    ms = []

    if d.get(s,0):

        ms.append(d[s])

    for k in range(1,len(s)):

        first, suffix = s[:k], s[k:]

        if d.get(first,0) and decrypt(suffix,d):

            for rest in decrypt(suffix,d):

                ms.append(d[first] + rest)
    return ms
```

2. **Consistency is Key** Fill in the function below so that it conforms to its docstring.

```
def ensure_consistency(fn):
    """Returns a function that calls fn on its argument, returns fn's
       return value, and returns None if fn's return value is different
       from any of its previous return values for those same argument.
       Also returns None if more than 20 calls are made.

    >>> def consistent(x):
    >>>     return x
    >>>
    >>> lst = [1, 2, 3]
    >>> def inconsistent(x):
    >>>    return x + lst.pop()
    >>>
    >>> a = ensure_consistency(consistent)
    >>> a(5)
    5
    >>> a(5)
    5
    >>> a(6)
    6
    >>> a(6)
    6
    >>> b = ensure_consistency(inconsistent)
    >>> b(5)
    8
    >>> b(5)
    None
    >>> b(6)
    7
    """
     n = ____________
     z = ____________
     def helper(x):
          __________________
          __________________
          if ____________:
                  return ____________
          val = fn(x)
          if ____________:
                  z[x] = [val]
          if ____________:
                  return ____________
          else:
                  z[x] = ____________
                  return ____________
     return helper
```