Fall 2015, Midterm 2, #3a

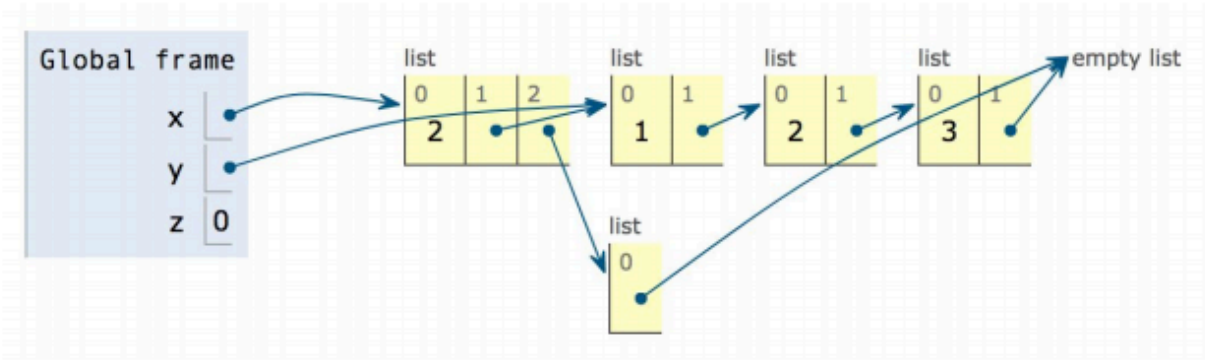
3. (24 points) Return of the Digits

(a) (4 pt) Implement `complete`, which takes a `Tree` instance `t` and two positive integers `d` and `k`. It returns whether `t` is *d-k-complete*. A tree is *d-k-complete* if every node at a depth less than `d` has exactly `k` branches and every node at depth `d` is a leaf. *Notes:* The depth of a node is the number of steps from the root; the root node has depth 0. The built-in `all` function takes a sequence and returns whether all elements are true values: `all([1, 2])` is `True` but `all([0, 1])` is `False`. `Tree` appears on the Midterm 2 Study Guide.

```
def complete(t, d, k):
    """Return whether t is d-k-complete.

    >>> complete(Tree(1), 0, 5)
    True
    >>> u = Tree(1, [Tree(1), Tree(1), Tree(1)])
    >>> [ complete(u, 1, 3)  ,  complete(u, 1, 2)  ,  complete(u, 2, 3) ]
    [True, False, False]
    >>> complete(Tree(1, [u, u, u]), 2, 3)
    True
    """
    if not t.branches:

        return d == 0

    bs = [complete(b, d-1, k) for b in t.branches]

    return len(t.branches) == k and all(bs)
```

Spring 2018, Exam-Prep 03, #1

1. Translating a List Diagram to Code



Fill in the following blanks so that after all lines have been executed, the environment looks as in the diagram above. You **may not use numerals or mathematical operators** in your solution.

Solution:

```
x, y, z = 1, 2, 3
y = [x, [y, [z, []]]]
x = [y[1][0], y, [y[1][1][1]]]
z = len([])
```

Spring 2015, Midterm 2, #3c

(c) (4 pt) Implement `closest`, which takes a `Tree` of numbers `t` and returns the smallest absolute difference anywhere in the tree between an entry and the sum of the entries of its branches. The `Tree` class appears on the midterm 2 study guide. The built-in `min` function takes a sequence and returns its minimum value. *Reminder*: A branch of a branch of a tree `t` is *not* considered to be a branch of `t`.

```
def closest(t):
    """Return the smallest difference between an entry and the sum of the
    root entries of its branches.

    >>> t = Tree(8, [Tree(4), Tree(3)])
    >>> closest(t) # |8 - (4 + 3)| = 1
    1
    >>> closest(Tree(5, [t])) # Same minimum as t
    1
    >>> closest(Tree(10, [Tree(2), t])) # |10 - (2 + 8)| = 0
    0
    >>> closest(Tree(3)) # |3 - 0| = 3
    3
    >>> closest(Tree(8, [Tree(3, [Tree(1, [Tree(5)])])])) # 3 - 1 = 2
    2
    >>> sum([])
    0
    """

    diff = abs(t.entry - sum([b.entry for b in t.branches]))

    return min([diff] + [closest(b) for b in t.branches])
```

Custom Question

```
def is_path(t, path):
    if label(t) != path[0]:
        return False
    if len(path) == 1:
        return True
    return any([is_path(b, path[1:]) for b in branches(t)])
```

Spring 2015, Midterm 2, #4b

(b) (6 pt) Implement `decrypt`, which takes 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 []


    messages = []


    if s in d:


        messages.append(d[s])


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


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


        if first in d:


            for rest in decrypt(suffix, d):


                messages.append(d[first] + ' ' + rest)


    return messages
```