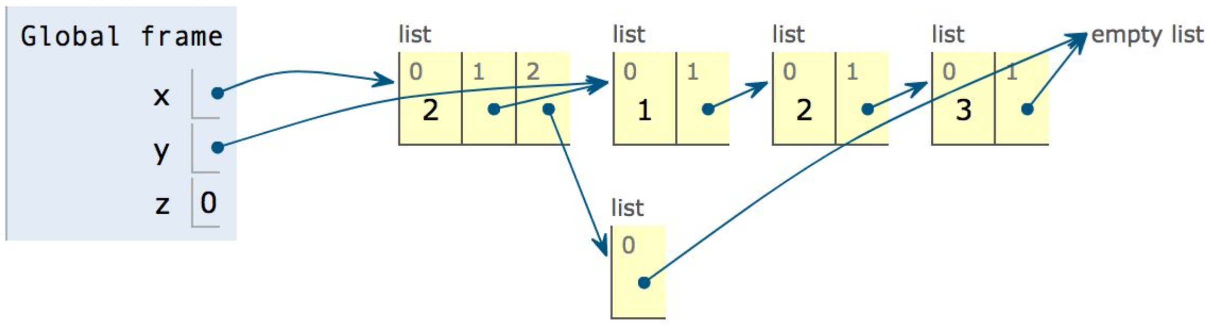
Exam Prep Section 3 Sols. - CS61A Spring 2018

Worksheet 3: Lists, Trees, and Tree Recursion

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.

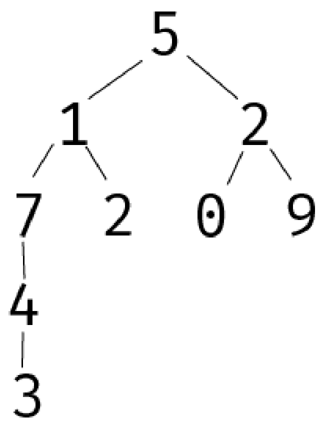
Solution:

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

(PythonTutor for solution)

2. Tree Recursion with Trees

A number x is a sum of a tree t if and only if there is a path from t's root to one of t's leaves whose labels sum to x. For example, the sums of the tree below are 20 (5+1+7+4+3), 8 (5+1+2), 7 (5+2+0), and 16 (5+2+9).



Fill in the following blanks so that the function behaves as indicated by its docstring. You may assume you have access to the tree, branches, label, and is_leaf functions.

Solution:
```
def sum_range(t):
      """Returns the range of the sums of t — that is, the difference
            between the largest and the smallest sums of t."""
      def helper(t):
            if is_leaf(t):
                  return [label(t), label(t)]
            else:
                  a = min([helper(b)[1] for b in branches(t)])
                  b = max([helper(b)[0] for b in branches(t)])
                  x = label(t)
                  return [b + x, a + x]
      x, y = helper(t)
      return x - y
```

3. This One Goes To Eleven (Fa14 Midterm 2 Q3b)

(4 pt) Fill in the blanks of the implementation of **no_eleven** below, a function that returns a list of all distinct length-**n** lists of ones and sixes in which 1 and 1 do not appear consecutively.

```
def no_eleven(n):
    """Return a list of lists of 1's and 6's that do not contain 1 after 1.

    >>> no_eleven(2)
    [[6, 6], [6, 1], [1, 6]]
    >>> no_eleven(3)
    [[6, 6, 6], [6, 6, 1], [6, 1, 6], [1, 6, 6], [1, 6, 1]]
    >>> no_eleven(4)[:4] # first half
    [[6, 6, 6, 6], [6, 6, 6, 1], [6, 6, 1, 6], [6, 1, 6, 6]]
    >>> no_eleven(4)[4:] # second half
    [[6, 1, 6, 1], [1, 6, 6, 6], [1, 6, 6, 1], [1, 6, 1, 6]]
    """

    if n == 0:

        return [[]]

    elif n == 1:

        return [[6], [1]]

    else:

        a, b = no_eleven(n-1), no_eleven(n-2)

        return [[6] + s for s in a] + [[1, 6] + s for s in b]
```

4. Expression Trees (Fa14 Final Q3a)

Your partner has created an interpreter for a language that can add or multiply positive integers. Expressions are represented as instances of the `Tree` class and must have one of the following three forms:

- **(Primitive)** A positive integer `entry` and no branches, representing an integer
- **(Combination)** The `entry` `'+'`, representing the sum of the values of its branches
- **(Combination)** The `entry` `'*'`, representing the product of the values of its branches

The `Tree` class is on the Midterm 2 Study Guide. The sum of no values is 0. The product of no values is 1.

a) (6 pt) Unfortunately, multiplication in Python is broken on your computer. Implement `eval_with_add`, which evaluates an expression without using multiplication. You may fill the blanks with names or call expressions, but the only way you are allowed to combine two numbers is using addition.

```
def eval_with_add(t):
    """Evaluate an expression tree of * and + using only addition.

    >>> plus = Tree('+', [Tree(2), Tree(3)])
    >>> eval_with_add(plus)
    5
    >>> times = Tree('*', [Tree(2), Tree(3)])
    >>> eval_with_add(times)
    6
    >>> deep = Tree('*', [Tree(2), plus, times])
    >>> eval_with_add(deep)
    60
    >>> eval_with_add(Tree('*'))
    1
    """
    if label(t) == '+':
        return sum([eval_with_add(b) for b in branches(t)])
    elif label(t) == '*':
        total = 1
        for b in branches(t):
            term, total = total, 0
            for _ in range(eval_with_add(b)):
                total = total + term
        return total
    else:
        return label(t)
```