MUTABLE TREES & MUTABLE FUNCTIONS

CS 61A GROUP MENTORING

July 23, 2018

1 Mutable Trees

For the following problems, use this definition for the Tree class:

```
class Tree:
    def __init__(self, label, branches=[]):
        self.label = label
        self.branches = list(branches)

    def is_leaf(self):
        return not self.branches

    def __repr__(self):
        if self.branches:
            branch_str = ', ' + repr(self.branches)
        else:
            branch_str = ''
        return 'Tree({0}{1})'.format(self.label, branch_str)

    def __str__(self):
        def print_tree(t, indent=0):
            tree_str = '  ' * indent + str(t.label) + '\n'
            for b in t.branches:
                tree_str += print_tree(b, indent + 1)
            return tree_str
        return print_tree(self).rstrip()
```

1. Implement `height` which takes in a tree and returns the height of that tree. Recall that the height of a tree is defined as the depth of the lowest leaf, where the depth of the top node of the tree is 0.

```
def tree_sum(t):
    """
    >>> t = Tree(1, [Tree(4), Tree(2, [Tree(3)])])
    >>> height(t)
    2
    """
```

Solution:

```
    if t.is_leaf():
        return 0
    return 1 + max([height(b) for b in t.branches])
```

2. Given a tree `t`, mutate the tree so that each leaf's label becomes the sum of the labels of all nodes in the path from the leaf node to the root node.

```
def replace_leaves_sum(t):
    """
    >>> t = Tree(1, [Tree(3, [Tree(2), Tree(8)]), Tree(5)])
    >>> replace_leaves_sum(t)
    >>> t
    Tree(1, [Tree(3, [Tree(6), Tree(12)]), Tree(6)])
    """
    def helper(______________ , _________________):

        if t.is_leaf():

            _____________________________________

        else:

            for b in t.branches:

                _________________________________

    _____________________________________________
```

Solution:

```
def replace_leaves_sum(t):
    def helper(t, total):
        if t.is_leaf():
            t.label = total + t.label
        else:
            for b in t.branches:
                helper(b, total + t.label)
    helper(t, 0)
```

3. Define `delete_path_duplicates`, which takes in `t`, a tree with non-negative labels. If there are any duplicate labels on any path from root to leaf, the function should mutate the label of the occurances deeper in the tree (i.e. closer to the root) to be the value `-1`.

```
def delete_path_duplicates(t):
    """
    >>> t = Tree(1, [Tree(2, [Tree(1), Tree(1)])])
    >>> delete_path_duplicates(t)
    >>> t
    Tree(1, [Tree(2, [Tree(-1), Tree(-1)])])
    >>> t2 = Tree(1, [Tree(2), Tree(2, [Tree(2, [Tree(1)])])])
    >>> delete_path_duplicates(t2)
    >>> t2
    Tree(1, [Tree(2), Tree(2, [Tree(-1, [Tree(-1)])])])
    """
```

Solution:

```
def delete_path_duplicates(t):
    def helper(t, seen_so_far):
        if t.label in seen_so_far:
            t.label = -1
        else:
            seen_so_far = seen_so_far + [t.label]
        for b in t.branches:
            helper(b, seen_so_far)
    helper(t, [])
```

2 Mutable Functions

4. **Nonlocal Kale**

Draw the environment diagram for the following code.

```
eggplant = 8
def vegetable(kale):
    def eggplant(spinach):
        nonlocal eggplant
        nonlocal kale
        kale = 9
        eggplant = spinach
        return eggplant + kale
    eggplant(kale)
    return eggplant

spinach = vegetable(10)
```

Solution: https://goo.gl/2bmMk9

5. **Pingpong again...**

Implement a function `make_pingpong_tracker` that returns the next value in the pingpong sequence each time it is called. You may use assignment statements.

```
def has_seven(k): # Use this function for your answer below
    if k % 10 == 7:
        return True
    elif k < 10:
        return False
    else:
        return has_seven(k // 10)
```

Solution:

```
def make_pingpong_tracker():
    index, current, add = 1, 0, True
    def pingpong_tracker():
        nonlocal index, current, add
        if add:
            current = current + 1
        else:
            current = current - 1
        if has_seven(index) or index % 7 == 0:
            add = not add
        index += 1
        return current
    return pingpong_tracker
```

6. Write a function `make_digit_getter` that, given a positive integer `n`, returns a new function that returns the digits in the integer one by one, starting from the rightmost digit. Once all digits have been removed, subsequant calls to the function should return the sum of all the digits in the original integer.

```
def make_digit_getter(n):
    """ Returns a function that returns the next digit in n
    each time it is called, and the total value of all the integers
    once all the digits have been returned.
    >>> year = 8102
    >>> get_year_digit = make_digit_generator(year)
    >>> for _ in range(4):
    ...     print(get_year_digit())
    2
    0
    1
    8
    >>> get_year_digit()
    11
    """
    ____________________________________

    def get_next():

        ________________________________

        if ______________________________:

              ______________________________

        ________________________________

        ________________________________

        return __________________________

    return ________________________________
```

Solution:

```
def make_digit_getter(n):
    total = 0
    def get_next():
        nonlocal n, total
        if n == 0:
             return total
        val = n % 10
        n = n // 10
        total += val
        return val
    return get_next
```

7. Define `make-increasing-checker`, which takes in no arguments and returns a function which takes in a positive integer. Each time this function is called, if its argument is strictly larger than every integer passed in previously, it should return `#t`, and `#f` otherwise.

```
scm> (define increasing (make-increasing-checker))
scm> (increasing 1)
#t
scm> (increasing 2)
#t
scm> (increasing 0)
#f
scm> (increasing 2)
#f
scm> (increasing 3)
#t
```

```
(define (make-increasing-checker n)




)
```

Solution:

```
(define (make-increasing-checker)
    (define largest 0)
    (lambda (x)
        (if (> x largest)
            (begin
                (set! largest x)
                #t)
            #f)))
```