CS 61A
Summer 2017

Mutable Linked Lists & Trees

Mentoring 10: July 24, 2017

1 Linked Lists

```
class Link:
    empty = ()

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

1.1 What does `wild` do? Try drawing a box-and-pointer diagram!

```
def wild(lnk):
    if lnk is Link.empty or lnk.rest is Link.empty:
        return lnk
    breath = wild(lnk.rest)
    lnk.rest.rest = lnk
    lnk.rest = Link.empty
    return breath
```

```
>>> triforce = Link(3, Link(1, Link(4)))
>>> wild(triforce)
```

1.2 Fill in the implementation of `double_link`.

```
def double_link(lst):
    """Using mutation, replaces the second in each pair of items
    with the first. The first of each pair stays as is.

    >>> double_link(Link(1, Link(2, Link(3, Link(4)))))
    Link(1, Link(1, Link(3, Link(3))))
    >>> double_link(Link('c', Link('s', Link(6, Link(1, Link('a'))))))
    Link('c', Link('c', Link(6, Link(6, Link('a')))))
    """

    if ____________________________________________________________________:

        return ____________________________________________________________

    ____________________________________________________________________

    ____________________________________________________________________

    return ____________________________________________________________
```

1.3 Fill in the implementation of `shuffle`.

```
def shuffle(lst):
    """Swaps each pair of items in a linked list.

    >>> shuffle(Link(1, Link(2, Link(3, Link(4)))))
    Link(2, Link(1, Link(4, Link(3))))
    >>> shuffle(Link('s', Link('c', Link(1, Link(6, Link('a'))))))
    Link('c', Link('s', Link(6, Link(1, Link('a')))))
    """

    if ____________________________________________________________________:

        return ____________________________________________________________

    front = lst.rest

    lst.rest = ________________________________________________________

    ____________________________________________________________________

    return ____________________________________________________________
```

2 Trees

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

    def is_leaf(self):
        return not self.branches
```

2.1 Implement `tree_sum` which takes in a Tree object and replaces the root value with the sum of all the values in the tree. `tree_sum` should also return the new root value.

```
def tree_sum(t):
    """
    >>> t = Tree(1, [Tree(2, [Tree(3)]), Tree(4)])
    >>> tree_sum(t)
    10
    >>> t.root
    10
    >>> t.branches[0].root
    5
    >>> t.branches[1].root
    4
    """
```

Binary Search Trees

```
class BST:
    empty = ()

    def __init__(self, root, left=empty, right=empty):
        self.root = root
        self.left = left
        self.right = right
```

2.2 Implement `is_symmetric`, which returns whether a BST is a mirror of itself, or symmetric across its center. Both the values and the shape of the BST needs to match for a BST to be symmetric.

```
def is_symmetric(t):
    """
    >>> t = BST(1, BST(2), BST(2))
    >>> is_symmetric(t)
    True
    >>> is_symmetric(t.left)
    True
    >>> t = BST(1, BST(2, BST(4, BST(5), BST(5)),
    ...                   BST(1)),
    ...            BST(3, BST(4)))
    >>> is_symmetric(t)
    False
    >>> is_symmetric(t.left)
    False
    >>> is_symmetric(t.left.left)
    True
    """
```