FINAL REVIEW

CS 61A GROUP MENTORING

April 23 to April 25, 2018

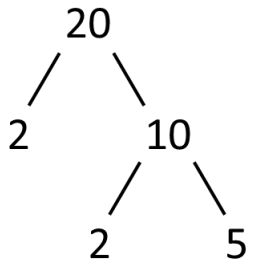
1 Environment Diagrams, Nonlocal, and Mutation

1. Draw the environment diagram that results from running the following code. If the code errors, draw the environment diagram up to the point that the error occurs.

```
earth = [0]
earth.append([earth])

def wind(fire, groove):
    fire[1][0][0] = groove
    def fire():
        nonlocal fire
        fire = lambda fantasy: earth.pop(1).extend(fantasy)
        return fire(groove)
    return fire()

sep = earth[1]
wind(earth, [earth[0]] + [earth.append(0)])
```

2 Trees

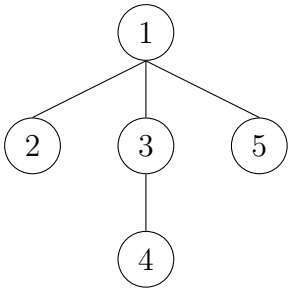
1. Define the function `factor_tree` which takes in a positive integer `n` and returns a factor tree for `n`. In a factor tree, multiplying the leaves together is the prime factorization of the root, `n`. See below for an example of a factor tree for `n = 20`.



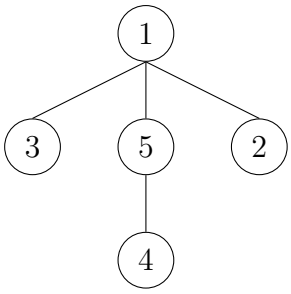

```
def factor_tree(n):
    """
    >>> factor_tree(20)
    Tree(20, [Tree(2), Tree(10, [Tree(2), Tree(5)])])
    >>> factor_tree(1)
    Tree(1)

    for i in _____________________:

        if _____________________________:

            return Tree(_____, ____________________________________)

    ___________________________________
```

2. Implement `rotate`, which takes in a tree and rotates the labels at each level of the tree by one to the left destructively. This rotation should be modular (That is, the leftmost label at a level will become the rightmost label after running rotate). You do NOT need to rotate across different branches.

   For example, given the following tree,`t`



   calling rotate on `t` should mutate it to give us



   Fill in your implementation on the next page.

```
def rotate(t):
    """
    >>> t1 = Tree(1, [Tree(2), Tree(3, [Tree(4)]), Tree(5)])
    >>> rotate(t1)
    >>> t1
    Tree(1, [Tree(3), Tree(5, [Tree(4)]), Tree(2)])
    >>> t2 = Tree(1, [Tree(2, [Tree(3), Tree(4)]),
                      Tree(5, [Tree(6)])])
    >>> rotate(t2)
    >>> t2
    Tree(1, [Tree(5, [Tree(4), Tree(3)]),
                    Tree(2, [Tree(6)])])
    """

    branch_labels = ____________________________________

    n = len(t.branches)

    for ____________________________________________________:

        branch = ____________________________________________

        ____________________________________________________

        ____________________________________________________
```

3 Scheme

1. Implement `slice`, which takes in a a list `lst`, a starting index `i`, and an ending index `j`, and returns a new list containing the elements of `lst` from index `i` to `j - 1`.

```
;Doctests
scm> (slice '(0 1 2 3 4) 1 3)
(1 2)
scm> (slice '(0 1 2 3 4) 3 5)
(3 4)
scm> (slice '(0 1 2 3 4) 3 1)
()

(define (slice lst i j)




)
```

2. Now implement slice with the same specifications, but make you implementation tail recurisve.
   You may wish to use the built-in append function, which takes in two lists and returns a new list containing the elements of the two lists concatenated together.

```
(define (slice lst i j)




)
```

3. Fill in the implementation of `shuffle`, which takes in a Scheme list and modifies the list such that each pair of elements in the list is swapped. It should additionally return the new list.

```
;Doctests
scm> (shuffle '(1 2 3 4)
(2 1 4 3)
scm> shuffle('(s c 1 6 a))
(c s 6 1 a)

(define (shuffle lst)
   (if _____________________________________________

       _____________________________________________

       (begin

           (define front _____________________________ )

           (set-cdr! lst _____________________________ )

           _____________________________________________

           front)))
```

4 Iterators, Generators, and Streams

1. What Would Python Display?

```
class SkipMachine:
    skip = 1
    def __init__(self, n=2):
        self.skip = n + SkipMachine.skip

    def generate(self):
        current = SkipMachine.skip
        while True:
            yield current
            current += self.skip
            SkipMachine.skip += 1

p = SkipMachine()
twos = p.generate()
SkipMachine.skip += 1
twos2 = p.generate()
threes = SkipMachine(3).generate()
```

(a) `next(twos)`

(b) `next(threes)`

(c) `next(twos)`

(d) `next(twos)`

(e) `next(threes)`

(f) `next(twos2)`

2. (a) You and your CS 61A friends are cons. You cdr'd just studied for the final, but instead you scheme to drive away across a stream in a car during dead week. Of course, you would like a variety of food to eat on your road trip.

   Write an infinite stream that takes in a list of foods and loops back to the first food in the list when the list is exhausted.

```
;Doctests
scm> (define fruit (food-stream '(apple banana orange)))
fruit
scm> (car fruit)
apple
scm> (car (cdr-stream fruit))
banana
scm> (car (cdr-stream (cdr-stream (cdr-stream fruit))))
apple

(define (food-stream foods)
```

   (b) We discover that some of our food is stale! Every other food that we go through is stale, so put it into a new stale food stream. Assume `is-stale` starts off as `#f`.

```
;Doctests
scm> (define cookies (stale-stream '(oatmeal chocolate
   sugar oreo)))
cookies
scm> (car cookies)
chocolate
scm> (car (cdr-stream cookies))
oreo


(define (stale-stream foods is-stale)
```