FINAL REVIEW

COMPUTER SCIENCE MENTORS

November 30, 2020 - December 3, 2020

1 Practice Final

For extra practice, CSM has put together some past exam problems together in the final exam format. You can access this here: https://exam.cs61a.org/cs61a-csm-practice. Note that these are mostly problems from finals in past semesters, but converted to examtool format with short answer or multiple choice inputs.

The password to access the exam is `ewXuqUSZLL0Bah0MIFfQ3K7SSWad3bokPfG_ePgTNOE`

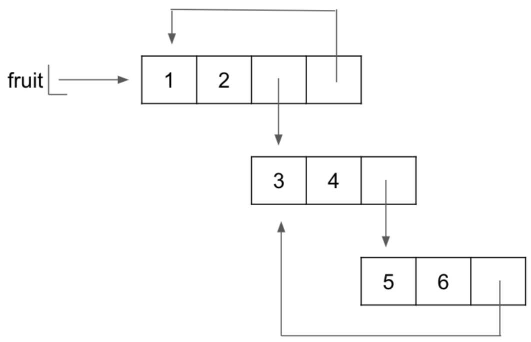
2 Environment Diagrams

1. Draw the environment diagram that results from running the following code.

```
def f(f):
    def h(x, y):
        z = 4
        return lambda z:  (x + y) * z

    def g(y):
        nonlocal g, h
        g = lambda y: y[:4]
        h = lambda x, y: lambda f: f(x + y)
        return y[3] + y[5:8]

    return h(g("sarcasm!"), g("why?"))

f = f("61a")(2)
```

2. Fill in each blank in the code example below so that its environment diagram is the following. You do not need to use all the blanks.

```
fruit = [1, 2, [3, 4]]
fruit.___________________________
fruit[3][2].____________________
fruit[2][2].____________________
fruit[3][3][2][2][2][1] = ____
```



3 OOP

1. The `DLList` class is a spin off of the normal `Link` class we learned in class; each `DLList` link has a `prev` attribute that keeps track of the previous link and a **`next`** attribute that keeps track of the next link. Fill in the following methods for the `DLList` class.

(a)
```
class DLList:
    """
    >>> lst = DLList(6, DLList(1))
    >>> lst.value
    6
    >>> lst.next.value
    1
    >>> lst.prev.value
    AttributeError: 'NoneType' object has no attribute 'value
       '
    """
    empty = None
    def __init__(self, value, next=empty, prev=empty):

        ________________________________

        ________________________________

        ________________________________
```

(b)
```
    def add_last(self, value):
        """
        >>> lst = DLList(6)
        >>> lst.add_last(1)
        >>> lst.value
        6
        >>> lst.next.value
        1
        >>> lst.next.prev.value
        6
        """
        pointer = self
        while ________________________________:

            ________________________________
    ______________ = DLList(___________________________)
```

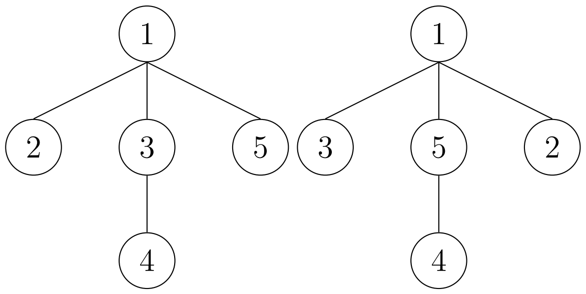
(c)
```
def add_first(self, value):
    """
    >>> lst = DLList('A')
    >>> lst.add_first(1)
    >>> lst.value
    1
    >>> lst.next.value
    'A'
    >>> lst.next.prev.value
    1
    >>> lst.add_first(6)
    >>> lst.value
    6
    >>> lst.next.next.prev.value
    1
    """
    old_first = DLList(____________________________)

    ______________ = ______________________________

    ______________ = ______________________________

    if ______________________________:

      _______________________________________________
```

If you're looking for some more practice on OOP problems. Check out this worksheet!

4 Trees

1. 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 tree `t` on the left, `rotate(t)` should mutate `t` to give us the right.



```
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 ______________________________________:

        ______________________________________________

        ______________________________________________

        ______________________________________________
```

5 Generators

1. (a) Implement `n_apply`, which takes in 3 inputs `f`, `n`, `x`, and outputs the result of applying `f`, a function, `n` times to `x`. For example, for `n = 3`, output the result of `f(f(f(x)))`.

```
def n_apply(f, n, x):
  """
  >>> n_apply(lambda x: x + 1, 3, 2)
  5
  """

  for ____________________________:

    x = ____________________________

  return ___________________________
```

(b) Now implement `list_gen`, which takes in some list of integers `lst` and a `function f`. For the element at index `i` of `lst`, `list_gen` should apply `f` to the element `i` times and yield this value `lst[i]` times. You may use `n_apply` from the previous part.

```
def list_gen(lst, f):
  """
  >>> a = list_gen([1, 2, 3], lambda x: x + 1)
  >>> list(a)
  [1, 3, 3, 5, 5, 5]
  """

  for ____________________________:

    yield from [______________________________________]
```

2. Complete the implementation of `iter_link`, which takes in a linked list and returns a generator which will iterate over the values of the linked list in order. Your function should support deep linked lists.

```
def iter_link(lnk):
    """
    Yield the values of a linked list in order; your function
       should support deep linked lists.
    >>> lst1 = Link(1, Link(2, Link(3, Link(4))))
    >>> list(iter_link(lst1))
    [1, 2, 3, 4]
    >>> lst2 = Link(1, Link(Link(2, Link(3)), Link(4, Link(5))))
    >>> print(lst2)
    <1 <2 3> 4 5>
    >>> iter_lst2 = iter_link(lst2)
    >>> next(iter_lst2)
    1
    >>> next(iter_lst2)
    2
    >>> next(iter_lst2)
    3
    >>> next(iter_lst2)
    4
    """
    if lnk is not Link.empty:

        if type(_____________________) is Link:

            _____________________________________
        else:

            _____________________________________

        _________________________________________
```

6 Scheme

1. Suppose Isabelle bought turnips from the Stalk Market and has stored them in random amounts among an ordered sequence of boxes. By the magic of time travel, Isabelle's friend Tom Nook can fast-forward one week into the future and determine exactly how many of Isabelle's turnips will rot over the week and have to be discarded.

   Assuming that boxes of turnips will rot in order, i.e. all of box 1's turnips will rot before any of box 2's turnips, help Isabelle determine which turnips will still be fresh by week's end. Specifically, fill in `decay`, which takes in a list of positive integers `boxes`, which represents how many turnips are in each box, and a positive integer `rotten` representing the number of turnips that will rot, and returns a list of non-negative integers that represents how many fresh turnips will remain in each box.

```
; doctests
scm> (define a '(1 6 3 4))
a
scm> (decay a 1)
(0 6 3 4)
scm> (decay a 5)
(0 2 3 4)
scm> (decay a 9)
(0 0 1 4)

(define (decay boxes rotten)








)
```

7 SQL

Examine the table, `mentors`, depicted below.

Name	**Food**	**Color**	**Editor**	**Language**
Catherine	Thai	Purple	Notepad++	Java
Jamie	Pie	Green	Sublime	Java
Alina	Sushi	Orange	Emacs	Ruby
Kenny	Tacos	Blue	Vim	Python
Ethan	Ramen	Green	Vim	Python

1. Write a query that has the same data, but alphabetizes the rows by name. (Hint: Use `order by`.)

```
Alina|Sushi|Orange|Emacs|Ruby
Catherine|Thai|Purple|Notepad++|Java
Ethan|Ramen|Green|Vim|Python
Jamie|Pie|Green|Sublime|Java
Kenny|Tacos|Blue|Vim|Python
```

2. Write a query that lists the food and the color of every person whose favorite language is *not* Python.

```
Thai|Purple
Pie|Green
Sushi|Orange
```

3. Write a query that lists all the pairs of mentors who like the same language. (How can we make sure to remove duplicates?)

```
Catherine|Jamie
Ethan|Kenny
```