Lab 9: Midterm Review **lab09.zip (lab09.zip)**

Due by 11:59pm on Tuesday, October 27.

Starter Files

Download lab09.zip (lab09.zip). Inside the archive, you will find starter files for the questions in this lab, along with a copy of the Ok (ok) autograder.

Submission

In order to facilitate midterm studying, solutions to this lab were released with the lab. We encourage you to try out the problems and struggle for a while before looking at the solutions!
Note: You do not need to run `python ok --submit` **to receive credit for this assignment.**

Required Questions

Q1: All Questions Are Optional

Updated: The following questions in this assignment are not graded, but they are highly recommended to help you prepare for the upcoming midterm. You will receive credit for this lab even if you not complete these questions.

This question has no Ok tests.

Suggested Questions

Recursion and Tree Recursion

Q2: Subsequences

A subsequence of a sequence `S` is a sequence of elements from `S`, in the same order they appear in `S`, but possibly with elements missing. Thus, the lists `[]`, `[1, 3]`, `[2]`, and `[1, 2, 3]` are some (but not all) of the subsequences of `[1, 2, 3]`. Write a function that takes a list and returns a list of lists, for which each individual list is a subsequence of the original input.

In order to accomplish this, you might first want to write a function `insert_into_all` that takes an item and a list of lists, adds the item to the beginning of nested list, and returns the resulting list.

```
def insert_into_all(item, nested_list):
    """Assuming that nested_list is a list of lists, return a new list
    consisting of all the lists in nested_list, but with item added to
    the front of each.

    >>> nl = [[], [1, 2], [3]]
    >>> insert_into_all(0, nl)
    [[0], [0, 1, 2], [0, 3]]
    """
    return _______________________________

def subseqs(s):
    """Assuming that S is a list, return a nested list of all subsequences
    of S (a list of lists). The subsequences can appear in any order.

    >>> seqs = subseqs([1, 2, 3])
    >>> sorted(seqs)
    [[], [1], [1, 2], [1, 2, 3], [1, 3], [2], [2, 3], [3]]
    >>> subseqs([])
    [[]]
    """
    if _________________:
        _________________
    else:
        _________________
        _________________
```

Use Ok to test your code:

```
python3 ok -q subseqs
```

Q3: Increasing Subsequences

Just like the last question, we want to write a function that takes a list and returns a list of lists, where each individual list is a subsequence of the original input.

This time we have another condition: we only want the subsequences for which consecutive elements are *nondecreasing*. For example, `[1, 3, 2]` is a subsequence of `[1, 3, 2, 4]`, but since 2 < 3, this subsequence would *not* be included in our result.

Fill in the blanks to complete the implementation of the `inc_subseqs` function. You may assume that the input list contains no negative elements.

You may use the provided helper function `insert_into_all`, which takes in an `item` and a list of lists and inserts the `item` to the front of each list.

```
def inc_subseqs(s):
    """Assuming that S is a list, return a nested list of all subsequences
    of S (a list of lists) for which the elements of the subsequence
    are strictly nondecreasing. The subsequences can appear in any order.

    >>> seqs = inc_subseqs([1, 3, 2])
    >>> sorted(seqs)
    [[], [1], [1, 2], [1, 3], [2], [3]]
    >>> inc_subseqs([])
    [[]]
    >>> seqs2 = inc_subseqs([1, 1, 2])
    >>> sorted(seqs2)
    [[], [1], [1], [1, 1], [1, 1, 2], [1, 2], [1, 2], [2]]
    """
    def subseq_helper(s, prev):
        if not s:
            return ______________________
        elif s[0] < prev:
            return ______________________
        else:
            a = _________________________
            b = _________________________
            return insert_into_all(________, _______________) + _________________
    return subseq_helper(____, ____)
```

Use Ok to test your code:

```
python3 ok -q inc_subseqs
```

Q4: Number of Trees

A **full binary tree** is a tree where each node has either 2 branches or 0 branches, but never 1 branch.

How many possible full binary tree structures exist that have exactly n leaves?

For those interested in combinatorics, this problem does have a closed form solution (https://web.archive.org/web/20201214030224/http://en.wikipedia.org/wiki/Catalan_number)):

```
def num_trees(n):
    """How many full binary trees have exactly n leaves? E.g.,

    1   2        3       3    ...
    *   *        *       *
       / \      / \     / \
      *   *    *   *   *   *
               / \         / \
              *   *       *   *

    >>> num_trees(1)
    1
    >>> num_trees(2)
    1
    >>> num_trees(3)
    2
    >>> num_trees(8)
    429

    """
    if _____________________:
        return _________________
    return _________________
```

Use Ok to test your code:

```
python3 ok -q num_trees
```

Generators

Q5: Generators generator

Write the generator function `make_generators_generator`, which takes a zero-argument generator function `g` and returns a generator that yields generators. For each element `e` yielded by the generator object returned by calling `g`, a new generator object is yielded that will generate entries 1 through `e` yielded by the generator returned by `g`.

```
def make_generators_generator(g):
    """Generates all the "sub"-generators of the generator returned by
    the generator function g.

    >>> def every_m_ints_to(n, m):
    ...     i = 0
    ...     while (i <= n):
    ...         yield i
    ...         i += m
    ...
    >>> def every_3_ints_to_10():
    ...     for item in every_m_ints_to(10, 3):
    ...         yield item
    ...
    >>> for gen in make_generators_generator(every_3_ints_to_10):
    ...     print("Next Generator:")
    ...     for item in gen:
    ...         print(item)
    ...
    Next Generator:
    0
    Next Generator:
    0
    3
    Next Generator:
    0
    3
    6
    Next Generator:
    0
    3
    6
    9
    """
    def gen(i):
        for __________ in __________:
            if __________________________:
                __________________________
            ________________________
            ________________________
    ___________________________
    for _________ in ___________________:
        _______________________________
        _______________________________
```

Use Ok to test your code:

```
python3 ok -q make_generators_generator
```

Objects

Q6: Keyboard

We'd like to create a `Keyboard` class that takes in an arbitrary number of `Button`s and stores these `Button`s in a dictionary. The keys in the dictionary will be ints that represent the postition on the `Keyboard`, and the values will be the respective `Button`. Fill out the methods in the `Keyboard` class according to each description, using the doctests as a reference for the behavior of a `Keyboard`.

```
class Button:
    """
    Represents a single button
    """
    def __init__(self, pos, key):
        """
        Creates a button
        """
        self.pos = pos
        self.key = key
        self.times_pressed = 0

class Keyboard:
    """A Keyboard takes in an arbitrary amount of buttons, and has a
    dictionary of positions as keys, and values as Buttons.

    >>> b1 = Button(0, "H")
    >>> b2 = Button(1, "I")
    >>> k = Keyboard(b1, b2)
    >>> k.buttons[0].key
    'H'
    >>> k.press(1)
    'I'
    >>> k.press(2) #No button at this position
    ''
    >>> k.typing([0, 1])
    'HI'
    >>> k.typing([1, 0])
    'IH'
    >>> b1.times_pressed
    2
    >>> b2.times_pressed
    3
    """

    def __init__(self, *args):
        ________________
        for _________ in _________________:
            ________________

    def press(self, info):
        """Takes in a position of the button pressed, and
        returns that button's output"""
        if ______________________:
            ________________
            ________________
            ________________
        ________________

    def typing(self, typing_input):
        """Takes in a list of positions of buttons pressed, and
        returns the total output"""
        ________________
        for ________ in ______________________:
            ________________
        ________________
```

Use Ok to test your code:

```
python3 ok -q Keyboard
```

Nonlocal

Q7: Advanced Counter

Complete the definition of `make_advanced_counter_maker`, which creates a function that creates counters. These counters can not only update their personal count, but also a shared count for all counters. They can also reset either count.

```
def make_advanced_counter_maker():
    """Makes a function that makes counters that understands the
    messages "count", "global-count", "reset", and "global-reset".
    See the examples below:

    >>> make_counter = make_advanced_counter_maker()
    >>> tom_counter = make_counter()
    >>> tom_counter('count')
    1
    >>> tom_counter('count')
    2
    >>> tom_counter('global-count')
    1
    >>> jon_counter = make_counter()
    >>> jon_counter('global-count')
    2
    >>> jon_counter('count')
    1
    >>> jon_counter('reset')
    >>> jon_counter('count')
    1
    >>> tom_counter('count')
    3
    >>> jon_counter('global-count')
    3
    >>> jon_counter('global-reset')
    >>> tom_counter('global-count')
    1
    """
    _________________
    def _____________(__________):
        _________________
        def _____________(__________):
            _________________
            "*** YOUR CODE HERE ***"
            # as many lines as you want
        _________________
    _________________
```

Use Ok to test your code:

```
python3 ok -q make_advanced_counter_maker
```

Mutable Lists

Q8: Trade

In the integer market, each participant has a list of positive integers to trade. When two participants meet, they trade the smallest non-empty prefix of their list of integers. A prefix is a slice that starts at index 0.

Write a function `trade` that exchanges the first `m` elements of list `first` with the first `n` elements of list `second`, such that the sums of those elements are equal, and the sum is as small as possible. If no such prefix exists, return the string `'No deal!'` and do not change either list. Otherwise change both lists and return `'Deal!'`. A partial implementation is provided.

Hint: You can mutate a slice of a list using *slice assignment*. To do so, specify a slice of the list `[i:j]` on the left-hand side of an assignment statement and another list on the right-hand side of the assignment statement. The operation will replace the entire given slice of the list from `i` inclusive to `j` exclusive with the elements from the given list. The slice and the given list need not be the same length.

```
>>> a = [1, 2, 3, 4, 5, 6]
>>> b = a
>>> a[2:5] = [10, 11, 12, 13]
>>> a
[1, 2, 10, 11, 12, 13, 6]
>>> b
[1, 2, 10, 11, 12, 13, 6]
```

Additionally, recall that the starting and ending indices for a slice can be left out and Python will use a default value. `lst[i:]` is the same as `lst[i:len(lst)]`, and `lst[:j]` is the same as `lst[0:j]`.

```
def trade(first, second):
    """Exchange the smallest prefixes of first and second that have equal sum.

    >>> a = [1, 1, 3, 2, 1, 1, 4]
    >>> b = [4, 3, 2, 7]
    >>> trade(a, b) # Trades 1+1+3+2=7 for 4+3=7
    'Deal!'
    >>> a
    [4, 3, 1, 1, 4]
    >>> b
    [1, 1, 3, 2, 2, 7]
    >>> c = [3, 3, 2, 4, 1]
    >>> trade(b, c)
    'No deal!'
    >>> b
    [1, 1, 3, 2, 2, 7]
    >>> c
    [3, 3, 2, 4, 1]
    >>> trade(a, c)
    'Deal!'
    >>> a
    [3, 3, 2, 1, 4]
    >>> b
    [1, 1, 3, 2, 2, 7]
    >>> c
    [4, 3, 1, 4, 1]
    """
    m, n = 1, 1

    equal_prefix = lambda: ________________________
    while ___________________________________:
        if ____________________:
            m += 1
        else:
            n += 1

    if equal_prefix():
        first[:m], second[:n] = second[:n], first[:m]
        return 'Deal!'
    else:
        return 'No deal!'
```

Use Ok to test your code:

```
python3 ok -q trade
```

Q9: Shuffle

Define a function `shuffle` that takes a sequence with an even number of elements (cards) and creates a new list that interleaves the elements of the first half with the elements of the second half.

```
def card(n):
    """Return the playing card numeral as a string for a positive n <= 13."""
    assert type(n) == int and n > 0 and n <= 13, "Bad card n"
    specials = {1: 'A', 11: 'J', 12: 'Q', 13: 'K'}
    return specials.get(n, str(n))

def shuffle(cards):
    """Return a shuffled list that interleaves the two halves of cards.

    >>> shuffle(range(6))
    [0, 3, 1, 4, 2, 5]
    >>> suits = ['♡', '♢', '♤', '♧']
    >>> cards = [card(n) + suit for n in range(1,14) for suit in suits]
    >>> cards[:12]
    ['A♡', 'A♢', 'A♤', 'A♧', '2♡', '2♢', '2♤', '2♧', '3♡', '3♢', '3♤', '3♧']
    >>> cards[26:30]
    ['7♤', '7♧', '8♡', '8♢']
    >>> shuffle(cards)[:12]
    ['A♡', '7♤', 'A♢', '7♧', 'A♤', '8♡', 'A♧', '8♢', '2♡', '8♤', '2♢', '8♧']
    >>> shuffle(shuffle(cards))[:12]
    ['A♡', '4♢', '7♤', '10♧', 'A♢', '4♤', '7♧', 'J♡', 'A♤', '4♧', '8♡', 'J♢']
    >>> cards[:12]  # Should not be changed
    ['A♡', 'A♢', 'A♤', 'A♧', '2♡', '2♢', '2♤', '2♧', '3♡', '3♢', '3♤', '3♧']
    """
    assert len(cards) % 2 == 0, 'len(cards) must be even'
    half = ________________
    shuffled = []
    for i in _______________:
        ________________
        ________________
    return shuffled
```

Use Ok to test your code:

```
python3 ok -q shuffle
```

Linked Lists

Q10: Insert

Implement a function `insert` that takes a `Link`, a `value`, and an `index`, and inserts the `value` into the `Link` at the given `index`. You can assume the linked list already has at least one element. Do not return anything -- `insert` should mutate the linked list.

> **Note**: If the index is out of bounds, you can raise an `IndexError` with:
>
> ```
> raise IndexError
> ```

```
def insert(link, value, index):
    """Insert a value into a Link at the given index.

    >>> link = Link(1, Link(2, Link(3)))
    >>> print(link)
    <1 2 3>
    >>> insert(link, 9001, 0)
    >>> print(link)
    <9001 1 2 3>
    >>> insert(link, 100, 2)
    >>> print(link)
    <9001 1 100 2 3>
    >>> insert(link, 4, 5)
    IndexError
    """
    if ____________________:
        ____________________
        ____________________
        ____________________
    elif ____________________:
        ____________________
    else:
        ____________________
```

Use Ok to test your code:

```
python3 ok -q insert
```

Q11: Deep Linked List Length

A linked list that contains one or more linked lists as elements is called a *deep* linked list. Write a function `deep_len` that takes in a (possibly deep) linked list and returns the *deep length* of that linked list. The deep length of a linked list is the total number of non-link elements in the list, as well as the total number of elements contained in all contained lists. See the function's doctests for examples of the deep length of linked lists.

> **Hint:** Use `isinstance` to check if something is an instance of an object.

```
def deep_len(lnk):
    """ Returns the deep length of a possibly deep linked list.

    >>> deep_len(Link(1, Link(2, Link(3))))
    3
    >>> deep_len(Link(Link(1, Link(2)), Link(3, Link(4))))
    4
    >>> levels = Link(Link(Link(1, Link(2)), \
            Link(3)), Link(Link(4), Link(5)))
    >>> print(levels)
    <<<1 2> 3> <4> 5>
    >>> deep_len(levels)
    5
    """
    if _______________:
        return 0
    elif _______________:
        return 1
    else:
        return ____________________________
```

Use Ok to test your code:

```
python3 ok -q deep_len
```

Q12: Linked Lists as Strings

Kevin and Jerry like different ways of displaying the linked list structure in Python. While Kevin likes box and pointer diagrams, Jerry prefers a more futuristic way. Write a function `make_to_string` that returns a function that converts the linked list to a string in their preferred style.

Hint: You can convert numbers to strings using the `str` function, and you can combine strings together using `+`.

```
>>> str(4)
'4'
>>> 'cs ' + str(61) + 'a'
'cs 61a'
```

```
def make_to_string(front, mid, back, empty_repr):
    """ Returns a function that turns linked lists to strings.

    >>> kevins_to_string = make_to_string("[", "|-]-->", "", "[]")
    >>> jerrys_to_string = make_to_string("(", " . ", ")", "()")
    >>> lst = Link(1, Link(2, Link(3, Link(4))))
    >>> kevins_to_string(lst)
    '[1|-]-->[2|-]-->[3|-]-->[4|-]-->[]'
    >>> kevins_to_string(Link.empty)
    '[]'
    >>> jerrys_to_string(lst)
    '(1 . (2 . (3 . (4 . ()))))'
    >>> jerrys_to_string(Link.empty)
    '()'
    """
    def printer(lnk):
        if _____________:
            return ____________________________
        else:
            return ____________________________
    return printer
```

Use Ok to test your code:

```
python3 ok -q make_to_string
```

Trees

Q13: Prune Small

Complete the function `prune_small` that takes in a `Tree t` and a number `n` and prunes `t` mutatively. If `t` or any of its branches has more than `n` branches, the `n` branches with the smallest labels should be kept and any other branches should be *pruned*, or removed, from the tree.

```
def prune_small(t, n):
    """Prune the tree mutatively, keeping only the n branches
    of each node with the smallest label.

    >>> t1 = Tree(6)
    >>> prune_small(t1, 2)
    >>> t1
    Tree(6)
    >>> t2 = Tree(6, [Tree(3), Tree(4)])
    >>> prune_small(t2, 1)
    >>> t2
    Tree(6, [Tree(3)])
    >>> t3 = Tree(6, [Tree(1), Tree(3, [Tree(1), Tree(2), Tree(3)]), Tree(5, [Tree(3), Tree(4
    >>> prune_small(t3, 2)
    >>> t3
    Tree(6, [Tree(1), Tree(3, [Tree(1), Tree(2)])])
    """
    while ______________________________:
        largest = max(_________________, key=_____________________)
        ____________________________
    for __ in _____________:
        ____________________
```

Use Ok to test your code:

```
python3 ok -q prune_small
```

CS 61A (/web/20201214030224/https://cs61a.org/)

Resources (/web/20201214030224/https://cs61a.org/)

Policies (/web/20201214030224/https://cs61a.org/articles/about.html)

Weekly Schedule (/web/20201214030224/https://cs61a.org/weekly.html)

Office Hours (/web/20201214030224/https://cs61a.org/articles/office-hours.html)

Studying Guide (/web/20201214030224/https://cs61a.org/articles/studying.html)

Debugging Guide (/web/20201214030224/https://cs61a.org/articles/debugging.html)

Composition Guide

Assignments (/web/20201214030224/https://cs61a.org/articles/about.html#a

Exams (/web/20201214030224/https://cs61a.org/articles/about.html#e

Grading