Lab 6: Midterm Review **lab06.zip (lab06.zip)**

Due by 11:59pm on Thursday, July 15.

Starter Files

Download lab06.zip (lab06.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

The 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 do 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 subset of elements from `S`, in the same order they appear in `S`. Consider the list `[1, 2, 3]`. Here are a few of it's subsequences `[]`, `[1, 3]`, `[2]`, and `[1, 2, 3]`.

Write a function that takes in a list and returns all possible subsequences of that list. The subsequences should be returned as a list of lists, where each nested 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 each nested list, and returns the resulting list.

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

    >>> nl = [[], [1, 2], [3]]
    >>> insert_into_all(0, nl)
    [[0], [0, 1, 2], [0, 3]]
    """
    "*** YOUR CODE HERE ***"

def subseqs(s):
    """Return a nested list (a list of lists) of all subsequences of S.
    The subsequences can appear in any order. You can assume S is a list.

    >>> 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: Non-Decreasing 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 non_decrease_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 = non_decrease_subseqs([1, 3, 2])
    >>> sorted(seqs)
    [[], [1], [1, 2], [1, 3], [2], [3]]
    >>> non_decrease_subseqs([])
    [[]]
    >>> seqs2 = non_decrease_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 non_decrease_subseqs
```

Mutable Lists

Q4: 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
```

Q5: 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
```

Trees

Q6: Same shape

Define a function `same_shape` that, given two trees, `t1` and `t2`, returns `True` if the two trees have the same shape (but not necessarily the same data in each node) and `False` otherwise.

```
def same_shape(t1, t2):
    """Return True if t1 is indentical in shape to t2.

    >>> test_tree1 = tree(1, [tree(2), tree(3)])
    >>> test_tree2 = tree(4, [tree(5), tree(6)])
    >>> test_tree3 = tree(1,
    ...                   [tree(2,
    ...                         [tree(3)])])
    >>> test_tree4 = tree(4,
    ...                   [tree(5,
    ...                         [tree(6)])])
    >>> same_shape(test_tree1, test_tree2)
    True
    >>> same_shape(test_tree3, test_tree4)
    True
    >>> same_shape(test_tree2, test_tree4)
    False
    """
    "*** YOUR CODE HERE ***"
```

Q7: Add trees

Define the function `add_trees`, which takes in two trees and returns a new tree where each corresponding node from the first tree is added with the node from the second tree. If a node at any particular position is present in one tree but not the other, it should be present in the new tree as well. At each level of the tree, nodes correspond to each other starting from the leftmost node.

> *Hint*: You may want to use the built-in zip function to iterate over multiple sequences at once.
>
> *Note*: If you feel that this one's a lot harder than the previous tree problems, that's totally fine! This is a pretty difficult problem, but you can do it! Talk about it with other students, and come back to it if you need to.

```
def add_trees(t1, t2):
    """
    >>> numbers = tree(1,
    ...                [tree(2,
    ...                      [tree(3),
    ...                       tree(4)]),
    ...                 tree(5,
    ...                      [tree(6,
    ...                            [tree(7)]),
    ...                       tree(8)])])
    >>> print_tree(add_trees(numbers, numbers))
    2
      4
        6
        8
      10
        12
          14
        16
    >>> print_tree(add_trees(tree(2), tree(3, [tree(4), tree(5)])))
    5
      4
      5
    >>> print_tree(add_trees(tree(2, [tree(3)]), tree(2, [tree(3), tree(4)])))
    4
      6
      4
    >>> print_tree(add_trees(tree(2, [tree(3, [tree(4), tree(5)])]), \
    tree(2, [tree(3, [tree(4)]), tree(5)])))
    4
      6
        8
        5
      5
    """
    "*** YOUR CODE HERE ***"
```

Use Ok to test your code:

```
python3 ok -q add_trees
```