

# Lab 9: Midterm Review

---

Adapted from cs61a of UC Berkeley.

## Starter Files

---

Get your starter file by cloning the repository: <https://github.com/JacyCui/sicp-lab09.git>

```
1 | git clone https://github.com/JacyCui/sicp-lab09.git
```

`lab09.zip` is the starter file you need, you might need to unzip the file to get the skeleton code.

```
1 | unzip lab09.zip
```

`README.md` is the handout for this homework. `solution` is a probable solution of the lab. However, I might not give my solution exactly when the lab is posted. You need to finish the task on your own first. If any problem occurs, please make use of the comment section.

## Required Questions

---

### Q1: All Questions Are Optional

The following questions in this assignment are not compulsory, but they are highly recommended to help you review what you have learned in the previous lectures.

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.

---

```

1  def insert_into_all(item, nested_list):
2      """Assuming that nested_list is a list of lists, return a new list
3      consisting of all the lists in nested_list, but with item added to
4      the front of each.
5
6      >>> nl = [[], [1, 2], [3]]
7      >>> insert_into_all(0, nl)
8      [[0], [0, 1, 2], [0, 3]]
9      """
10     return _____
11
12  def subseqs(s):
13      """Assuming that S is a list, return a nested list of all subsequences
14      of S (a list of lists). The subsequences can appear in any order.
15
16      >>> seqs = subseqs([1, 2, 3])
17      >>> sorted(seqs)
18      [[], [1], [1, 2], [1, 2, 3], [1, 3], [2], [2, 3], [3]]
19      >>> subseqs([])
20      [[]]
21      """
22     if _____:
23         _____
24     else:
25         _____
26         _____

```

Use Ok to test your code:

```
1 python3 ok -q subseqs --local
```

### 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.

```
1 def inc_subseqs(s):
```

```

2      """Assuming that S is a list, return a nested list of all subsequences
3      of S (a list of lists) for which the elements of the subsequence
4      are strictly nondecreasing. The subsequences can appear in any order.
5
6      >>> seqs = inc_subseqs([1, 3, 2])
7      >>> sorted(seqs)
8      [[], [1], [1, 2], [1, 3], [2], [3]]
9      >>> inc_subseqs([])
10     []
11     >>> seqs2 = inc_subseqs([1, 1, 2])
12     >>> sorted(seqs2)
13     [[], [1], [1], [1, 1], [1, 1, 2], [1, 2], [1, 2], [2]]
14     """
15     def subseq_helper(s, prev):
16         if not s:
17             return _____
18         elif s[0] < prev:
19             return _____
20         else:
21             a = _____
22             b = _____
23             return insert_into_all(_____, _____) + _____
24     return subseq_helper(_____, _____)

```

Use Ok to test your code:

```

1  python3 ok -q inc_subseqs --local

```

## 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?

```

1  def num_trees(n):
2      """How many full binary trees have exactly n leaves? E.g.,
3
4      1   2       3       3   ...
5      *   *       *       *
6      / \   / \   / \   / \
7      *   *   *   *   *   *
8          / \       / \
9          *   *       *   *
10
11     >>> num_trees(1)
12     1
13     >>> num_trees(2)

```

```

14     1
15     >>> num_trees(3)
16     2
17     >>> num_trees(8)
18     429
19
20     """
21     if _____:
22         return _____
23     return _____

```

Use Ok to test your code:

```

1 | python3 ok -q num_trees --local

```

## 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`.

```

1  def make_generators_generator(g):
2      """Generates all the "sub"-generators of the generator returned by
3      the generator function g.
4
5      >>> def every_m_ints_to(n, m):
6          ...     i = 0
7          ...     while (i <= n):
8              ...         yield i
9              ...         i += m
10         ...
11     >>> def every_3_ints_to_10():
12         ...     for item in every_m_ints_to(10, 3):
13             ...         yield item
14         ...
15     >>> for gen in make_generators_generator(every_3_ints_to_10):
16         ...     print("Next Generator:")
17         ...     for item in gen:
18             ...         print(item)
19         ...
20     Next Generator:
21     0
22     Next Generator:

```

```

23     0
24     3
25     Next Generator:
26     0
27     3
28     6
29     Next Generator:
30     0
31     3
32     6
33     9
34     """
35     def gen(i):
36         for _____ in _____:
37             if _____:
38                 _____
39                 _____
40                 _____
41         _____
42         for _____ in _____:
43             _____
44             _____

```

Use Ok to test your code:

```

1 | python3 ok -q make_generators_generator --local

```

## 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 position 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`.

```

1 | class Button:
2 |     """
3 |     Represents a single button
4 |     """
5 |     def __init__(self, pos, key):
6 |         """
7 |         Creates a button
8 |         """
9 |         self.pos = pos
10 |        self.key = key

```

```

11         self.times_pressed = 0
12
13 class Keyboard:
14     """A Keyboard takes in an arbitrary amount of buttons, and has a
15     dictionary of positions as keys, and values as Buttons.
16
17     >>> b1 = Button(0, "H")
18     >>> b2 = Button(1, "I")
19     >>> k = Keyboard(b1, b2)
20     >>> k.buttons[0].key
21     'H'
22     >>> k.press(1)
23     'I'
24     >>> k.press(2) #No button at this position
25     ''
26     >>> k.typing([0, 1])
27     'HI'
28     >>> k.typing([1, 0])
29     'IH'
30     >>> b1.times_pressed
31     2
32     >>> b2.times_pressed
33     3
34     """
35
36     def __init__(self, *args):
37         _____
38         for _____ in _____:
39             _____
40
41     def press(self, info):
42         """Takes in a position of the button pressed, and
43         returns that button's output"""
44         if _____:
45             _____
46             _____
47             _____
48         _____
49
50     def typing(self, typing_input):
51         """Takes in a list of positions of buttons pressed, and
52         returns the total output"""
53         _____
54         for _____ in _____:
55             _____
56         _____

```

Use Ok to test your code:

## 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.

```

1  def make_advanced_counter_maker():
2      """Makes a function that makes counters that understands the
3      messages "count", "global-count", "reset", and "global-reset".
4      See the examples below:
5
6      >>> make_counter = make_advanced_counter_maker()
7      >>> tom_counter = make_counter()
8      >>> tom_counter('count')
9      1
10     >>> tom_counter('count')
11     2
12     >>> tom_counter('global-count')
13     1
14     >>> jon_counter = make_counter()
15     >>> jon_counter('global-count')
16     2
17     >>> jon_counter('count')
18     1
19     >>> jon_counter('reset')
20     >>> jon_counter('count')
21     1
22     >>> tom_counter('count')
23     3
24     >>> jon_counter('global-count')
25     3
26     >>> jon_counter('global-reset')
27     >>> tom_counter('global-count')
28     1
29     """
30
31     def _____(_____):
32         _____
33         def _____(_____):
34             _____
35             """*** YOUR CODE HERE ***"""
36             # as many lines as you want

```

37

38

Use Ok to test your code:

```
1 python3 ok -q make_advanced_counter_maker --local
```

## 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.

```
1 >>> a = [1, 2, 3, 4, 5, 6]
2 >>> b = a
3 >>> a[2:5] = [10, 11, 12, 13]
4 >>> a
5 [1, 2, 10, 11, 12, 13, 6]
6 >>> b
7 [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]`.

```
1 def trade(first, second):
2     """Exchange the smallest prefixes of first and second that have equal sum.
3
4     >>> a = [1, 1, 3, 2, 1, 1, 4]
5     >>> b = [4, 3, 2, 7]
6     >>> trade(a, b) # Trades 1+1+3+2=7 for 4+3=7
7     'Deal!'
8     >>> a
9     [4, 3, 1, 1, 4]
10    >>> b
```



```

11     [1, 1, 3, 2, 2, 7]
12     >>> c = [3, 3, 2, 4, 1]
13     >>> trade(b, c)
14     'No deal!'
15     >>> b
16     [1, 1, 3, 2, 2, 7]
17     >>> c
18     [3, 3, 2, 4, 1]
19     >>> trade(a, c)
20     'Deal!'
21     >>> a
22     [3, 3, 2, 1, 4]
23     >>> b
24     [1, 1, 3, 2, 2, 7]
25     >>> c
26     [4, 3, 1, 4, 1]
27     """
28     m, n = 1, 1
29
30     equal_prefix = lambda: _____
31     while _____:
32         if _____:
33             m += 1
34         else:
35             n += 1
36
37     if equal_prefix():
38         first[:m], second[:n] = second[:n], first[:m]
39         return 'Deal!'
40     else:
41         return 'No deal!'

```

Use Ok to test your code:

```

1 | python3 ok -q trade --local

```

## 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.

```

1 | def card(n):
2 |     """Return the playing card numeral as a string for a positive n <= 13."""
3 |     assert type(n) == int and n > 0 and n <= 13, "Bad card n"
4 |     specials = {1: 'A', 11: 'J', 12: 'Q', 13: 'K'}
5 |     return specials.get(n, str(n))
6 |

```

```

7  def shuffle(cards):
8      """Return a shuffled list that interleaves the two halves of cards.
9
10     >>> shuffle(range(6))
11     [0, 3, 1, 4, 2, 5]
12     >>> suits = ['♥', '♦', '♠', '♣']
13     >>> cards = [card(n) + suit for n in range(1,14) for suit in suits]
14     >>> cards[:12]
15     ['A♥', 'A♦', 'A♠', 'A♣', '2♥', '2♦', '2♠', '2♣', '3♥', '3♦', '3♠', '3♣']
16     >>> cards[26:30]
17     ['7♠', '7♣', '8♥', '8♦']
18     >>> shuffle(cards)[:12]
19     ['A♥', '7♠', 'A♦', '7♣', 'A♠', '8♥', 'A♣', '8♦', '2♥', '8♠', '2♦', '8♣']
20     >>> shuffle(shuffle(cards))[:12]
21     ['A♥', '4♦', '7♠', '10♠', 'A♦', '4♠', '7♣', 'J♥', 'A♠', '4♣', '8♥', 'J♦']
22     >>> cards[:12] # Should not be changed
23     ['A♥', 'A♦', 'A♠', 'A♣', '2♥', '2♦', '2♠', '2♣', '3♥', '3♦', '3♠', '3♣']
24     """
25     assert len(cards) % 2 == 0, 'len(cards) must be even'
26     half = _____
27     shuffled = []
28     for i in _____:
29         _____
30         _____
31     return shuffled

```

Use Ok to test your code:

```
1 | python3 ok -q shuffle --local
```

## 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:

```
1 | raise IndexError
```

```

1  def insert(link, value, index):
2      """Insert a value into a Link at the given index.
3
4      >>> link = Link(1, Link(2, Link(3)))

```

```

5     >>> print(link)
6     <1 2 3>
7     >>> insert(link, 9001, 0)
8     >>> print(link)
9     <9001 1 2 3>
10    >>> insert(link, 100, 2)
11    >>> print(link)
12    <9001 1 100 2 3>
13    >>> insert(link, 4, 5)
14    IndexError
15    """
16    if _____:
17        _____
18        _____
19        _____
20    elif _____:
21        _____
22    else:
23        _____

```

Use Ok to test your code:

```
1 python3 ok -q insert --local
```

## 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.

```

1  def deep_len(lnk):
2      """ Returns the deep length of a possibly deep linked list.
3
4      >>> deep_len(Link(1, Link(2, Link(3))))
5      3
6      >>> deep_len(Link(Link(1, Link(2)), Link(3, Link(4))))
7      4
8      >>> levels = Link(Link(Link(1, Link(2)), \
9                          Link(3)), Link(Link(4), Link(5)))
10     >>> print(levels)
11     <<<1 2> 3> <4> 5>
12     >>> deep_len(levels)
13     5

```

```

14     """
15     if _____:
16         return 0
17     elif _____:
18         return 1
19     else:
20         return _____

```

Use Ok to test your code:

```
1 python3 ok -q deep_len --local
```

## 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 `+`.

```

1 >>> str(4)
2 '4'
3 >>> 'si' + 'cp ' + str(2022)
4 'sicp 2022'

```

```

1 def make_to_string(front, mid, back, empty_repr):
2     """ Returns a function that turns linked lists to strings.
3
4     >>> kevins_to_string = make_to_string("[", "|-|-->", "", "[]")
5     >>> jerrys_to_string = make_to_string("(", " . ", ")", "()")
6     >>> lst = Link(1, Link(2, Link(3, Link(4))))
7     >>> kevins_to_string(lst)
8     '[1|-|-->[2|-|-->[3|-|-->[4|-|-->[]]'
9     >>> kevins_to_string(Link.empty)
10    '[]'
11    >>> jerrys_to_string(lst)
12    '(1 . (2 . (3 . (4 . ())))'
13    >>> jerrys_to_string(Link.empty)
14    '()'
15    """
16    def printer(lnk):
17        if _____:
18            return _____
19        else:
20            return _____

```

```
21 | return printer
```

Use Ok to test your code:

```
1 | python3 ok -q make_to_string --local
```

## 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.

```
1 | def prune_small(t, n):
2 |     """Prune the tree mutatively, keeping only the n branches
3 |     of each node with the smallest label.
4 |
5 |     >>> t1 = Tree(6)
6 |     >>> prune_small(t1, 2)
7 |     >>> t1
8 |     Tree(6)
9 |     >>> t2 = Tree(6, [Tree(3), Tree(4)])
10 |    >>> prune_small(t2, 1)
11 |    >>> t2
12 |    Tree(6, [Tree(3)])
13 |    >>> t3 = Tree(6, [Tree(1), Tree(3, [Tree(1), Tree(2), Tree(3)]), Tree(5,
14 |    [Tree(3), Tree(4)])])
15 |    >>> prune_small(t3, 2)
16 |    >>> t3
17 |    Tree(6, [Tree(1), Tree(3, [Tree(1), Tree(2)])])
18 |    """
19 |    while _____:
20 |        largest = max(_____, key=_____)
21 |    for __ in _____:
22 |        _____
```

Use Ok to test your code:

```
1 | python3 ok -q prune_small --local
```

Congratulations! You've finished all problems of the lab. Feel free to run doctest to verify your answer again.

```
1 | python3 -m doctest lab09.py
```

Note: You should delete the testcase that will cause an exception in insert function before running doctest and then undo your deletion. Because doctest doesn't support test for exceptions.