1. Write a function that takes in two values \( x \) and \( e_l \), and a list, and adds as many \( e_l \)'s to the end of the list as there are \( x \)'s.

```python
def add_this_many(x, el, lst):
    ""
    Adds \( e_l \) to the end of \( lst \) the number of times \( x \) occurs in \( lst \).
    ""
    >>> lst = [1, 2, 4, 2, 1]
    >>> add_this_many(1, 5, lst)
    >>> lst
    [1, 2, 4, 2, 1, 5, 5]
    >>> add_this_many(2, 2, lst)
    >>> lst
    [1, 2, 4, 2, 1, 5, 5, 2, 2]
```

2. Given a deep dictionary $d$, replace all occurrences of $x$ as a value (not a key) with $y$.

Hint: You will need to combine iteration and recursion.

```python
def replace_all_deep(d, x, y):
    """
    >>> d = {1: {2: 3, 3: 4}, 2: {4: 4, 5: 3}}
    >>> replace_all_deep(d, 3, 1)
    >>> d
    {1: {2: 1, 3: 4}, 2: {4: 4, 5: 1}}
    """
```

2. Object-Oriented Programming

1. Assume these commands are entered in order. What would Python output?

```python
>>> class Foo:
...     def __init__(self, a):
...         self.a = a
...     def garply(self):
...         return self.baz(self.a)

>>> class Bar(Foo):
...     a = 1
...     def baz(self, val):
...         return val

>>> f = Foo(4)
>>> b = Bar(3)
>>> f.a

>>> b.a

>>> f.garply()
```

CS 61A Summer 2016: Brian Hou and Marvin Zhang, with
Daniel Haas, Eric Bo, Jack Thakar, Jerome Baek, Katya Stukalova, Marsalis Gibson, Mitas Ray, Neil Agarwal,
Samantha Wong, Tammy Nguyen, Vivian Fang, and Zhen Qin
>>> b.garply()

>>> b.a = 9
>>> b.garply()

>>> f.baz = lambda val: val * val
>>> f.garply()

3  Mutable Linked Lists and Trees

3.1  Linked Lists

Here is the implementation of the linked list class:

class Link:
    empty = ()
    def __init__(self, first, rest=empty):
        assert rest is Link.empty or isinstance(rest, Link)
        self.first = first
        self.rest = rest
    def __getitem__(self, i):
        if i == 0:
            return self.first
        return self.rest[i-1]
    def __len__(self):
        return 1 + len(self.rest)
    def __repr__(self):
        if self.rest is Link.empty:
            return 'Link({})'.format(self.first)
        else:
            return 'Link({}, {})'.format(self.first, repr(self.rest))
1. Write a recursive function `flip_two` that takes as input a linked list `lnk` and mutates `lnk` so that every pair is flipped.

```python
def flip_two(lnk):
    ""
    >>> one_lnk = Link(1)
    >>> flip_two(one_lnk)
    >>> one_lnk
    Link(1)
    >>> lnk = Link(1, Link(2, Link(3, Link(4, Link(5)))))
    >>> flip_two(lnk)
    >>> lnk
    Link(2, Link(1, Link(4, Link(3, Link(5)))))
    ""
```

3.2 Trees

class Tree:
    def __init__(self, entry, children=[]):
        for c in children:
            assert isinstance(c, Tree)
        self.entry = entry
        self.children = children

    def is_leaf(self):
        return not self.children
1. Assuming that every entry in \( t \) is a number, let’s define \( \text{average}(t) \), which returns the average of all the entries in \( t \).

   ```python
def average(t):
    """
    Returns the average value of all the entries in \( t \).
    >>> t0 = Tree(0, [Tree(1), Tree(2, [Tree(3)])])
    >>> average(t0)
    1.5
    >>> t1 = Tree(8, [t0, Tree(4)])
    >>> average(t1)
    3.0
    """
   ```

2. Write a program `flatten` that given a Tree \( t \), will return a linked list of the elements of \( t \), ordered by level. Entries on the same level should be ordered from left to right. For example, the following tree will return the linked list \(<1 2 3 4 5 6 7>\).

   ```python
def flatten(t):
   ```
4 Scheme

1. Write a Scheme function that, when given an element, a list, and an index, inserts the element into the list at that index.
   (define (insert element lst index)
       ...
   )

2. Define deep-apply, which takes a nested list and applies a given procedure to every element. deep-apply should return a nested list with the same structure as the input list, but with each element replaced by the result of applying the given procedure to that element. Use the built-in list? procedure to detect whether a value is a list. The procedure map has been defined for you.
   (define (map fn lst)
       (if (null? lst)
           nil
           (cons (fn (car lst)) (map fn (cdr lst))))
   )
   (define (deep-apply fn nested-list)
       ...
   )

scm> (deep-apply (lambda (x) (* x x)) '(1 2 3))
   (1 4 9)
scm> (deep-apply (lambda (x) (* x x)) '(1 ((4) 5) 9))
   (1 ((16) 25) 81)
scm> (deep-apply (lambda (x) (* x x)) 2)
   4
4.1 Streams

1. What would Scheme display?
   
   ```scheme
   scm> (define (has-even? s)
   (cond ((null? s) False)
         ((even? (car s)) True)
         (else (has-even? (cdr-stream s)))))
   has-even?
   scm> (define ones (cons-stream 1 ones))
   scm> (define twos (cons-stream 2 twos))
   scm> ones
   scm> (cdr ones)
   scm> (cdr-stream ones)
   scm> (has-even? ones)
   scm> (has-even? twos)
   ```

5. Logic

1. Write facts for `match`, a relation between two lists if and only if the two lists are identical.

   ```scheme
   > (query (match (i am so cool) (i am . ?you)))
   Success!
   you: (so cool)
   ```
6 Generators

1. Write a generator function that returns all subsets of the positive integers from 1 to \( n \).
   Each call to this generator’s \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_
   \_