MIDTERM EXAMINATION 2

COMPUTER SCIENCE 61A

July 25, 2012

Instructions: Read Me!

- You will first be given 15 minutes to work on the group portion of the midterm first, followed by 105 minutes to work on the individual portion of the midterm. **Do not start unless told to do so by the teaching staff.**
- This exam is closed book. Electronic devices (except dedicated timekeepers) must be turned off. You can use one double-sided 8.5" $\times$ 11" sheet of handwritten notes.
- Please write neatly and legibly, because *if we can't read it, we can't grade it*. If you are not sure of your answer, you may wish to provide a brief explanation.
- Finally, please take a deep breath and calm down before starting the exam. This exam is not worth having a heart attack for. We hope you do a CS61Awesome job!

0 A Question of Identity (1 point)

Fill in the table below. Also, **write your login at the top of each page of the midterm, once you are told to start the midterm**.

Name	
Login (`cs61a-`	
Section TA	
All of the work on this exam is my own. (Please sign.)	

Question	0	1	2	3	4	5	Group	**Total**
Score	/1	/14	/4	/5	/13	/5	/8	/50

1 Can You Give Me a Few Pointers? (14 points)

We execute the following statements at the Python interpreter:

```
>>> def bar(x):
...     return [x, 2, 3, 4]
...
>>> carly = [-1, 0, 1]
>>> foo = bar(carly)
>>> baz = {'foo': foo, 'carly': [-1, 0, 1]}
```

Write what each of the following expressions would evaluate to in an interactive Python session. If evaluating the expression would cause an error, write "ERROR" and give the reason for the error.

(a) `foo[0] is [-1, 0, 1]`

(b) `foo[0] is carly`

(c) `tuple(map(bar, (foo, [5, 6])))`

(d) `baz['carly'][1] + (4 * baz['foo'][0][0])`

You may use the space below for scratch work. No work done here will be graded.

(e) We execute the following snippet of code at the interpreter:

```
>>> c = [10, 20, 30]
>>> lst = [5, 6, c]
>>> lst[0] = lst
```

(i) In the space below, draw the box-and-pointer diagram that results from the above interaction.

(ii) What is the value of `lst[0][1]`?

(iii) Suppose the following line is now executed at the interpreter:

```
>>> c = [5, 10, 15]
```

What would the following expressions evaluate to? If evaluating the expression would cause an error, write "ERROR" and give the reason for the error.

```
>>> lst[2]
```

```
>>> lst[2] is c
```

2 You From Around Here? (4 points)

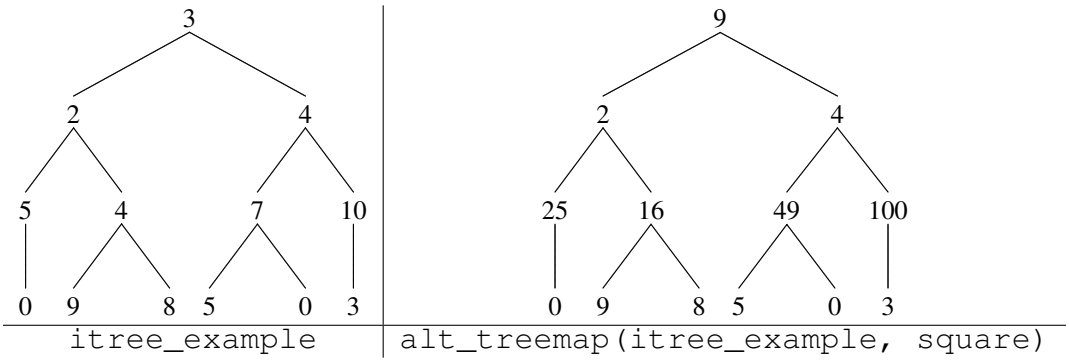
Define a function `make_confused_fn` that, given a function `fn` that takes one argument, returns a new function of one argument that behaves just as `fn` would, but only every three invocations. Otherwise, it returns the string `'wat'`. The first call to the confused function should call the original function. **You should use the given skeleton code, and you may not define any other new functions or any other new variables.** We have provided doctests that demonstrate the intended usage.

```
def make_confused_fn(fn):
    """ Given a one-argument function fn, returns a new
    function that behaves as fn would every three
    invocations, and the string 'wat' otherwise.

    >>> square = lambda x: x * x
    >>> confused_square = make_confused_fn(square)
    >>> confused_square(3)
    9
    >>> confused_square(8)
    'wat'
    >>> confused_square(42)
    'wat'
    >>> confused_square(6)
    36
    >>> confused_square(3)
    'wat'
    """
    count = 0
    def confused_fn(x):









    return confused_fn
```

3 This Tree Seems Uneven (5 points)

Write the function `alt_treemap` that, given a function and an ITree, applies the function to all of the data at every other level of the ITree, starting at the root node.

For example, given the ITree on the left (`itree_example`) and the `square` function, the output of the function should be the ITree on the right:



You may not convert the ITree to any other different data structure (such as tuples, lists, or other sequences) in your solution. You may find it useful to write and use helper functions.

```
def alt_treemap(itree, fn):
```

4 Classic Inheritance (13 points)

(a) Suppose we want to represent a book and a bookstore as objects. We will have one class for books and another class for bookstores. For each of the following, in the middle column, write the class that it should be associated with: either the BOOK class or the BOOKSTORE class. Also, in the rightmost column, use any of the following keywords to best describe what each item would be with regard to the class:

INSTANCE SUBCLASS INSTANCE VARIABLE CLASS VARIABLE

Each keyword may be used multiple times or not at all. However, each row should only have one entry per column.

	Associated Class (BOOK or BOOKSTORE)	**Best Descriptive Keyword**
Harry Potter		
Novel		
Title		
ASUC Bookstore		
Inventory of books		

(b) Suppose the following class definition was provided to the Python interpreter:

```
class Domo:
    spawns = 0
    def __init__(self, name):
        Domo.spawns += 1
        self.my_name = name
        self.num_eaten = 0
    def eat(self, food):
        print(self.my_name + " enjoyed eating this "  + food)
        self.num_eaten += 1
    def name(self):
        return self.my_name
```

What would the Python interpreter print during the following interaction? If evaluating the expression would cause an error, write "ERROR" and give the reason for the error.

```
>>> my_little_domo = Domo('Alonzo')
>>> my_little_domo.eat('kitten')




>>> self.num_eaten




>>> for name in ('Dom', 'Dimitri', 'Dominick'):
...     Domo(name)
...
>>> Domo.spawns




>>> my_little_domo.my_name = 'Al'
>>> Domo.name(my_little_domo)
```

(c) We want to create a child class, `PickyDomo`, which inherits from `Domo`. **Fill in the bodies for the following methods, using the principle of inheritance wherever applicable.** We have provided doctests that demonstrate the intended functionality and usage.

```
class PickyDomo(Domo):
    def __init__(self, name, fav_food):
        """ A PickyDomo is a type of Domo who will not eat the
        same food more than once, unless that food is its fav_food.
        """
```

```
    def eat(self, food):
        """ PickyDomo will not eat the same food twice, ever.
        A PickyDomo will, however, eat its fav_food any number of times.

        >>> domo_sally = PickyDomo('Sally', 'apple')
        >>> domo_sally.eat('apple')
        Sally enjoyed eating this apple
        >>> domo_sally.eat('pear')
        Sally enjoyed eating this pear
        >>> domo_sally.eat('apple')
        Sally enjoyed eating this apple
        >>> domo_sally.eat('pear')
        Sally does not want to eat this pear
        >>> domo_sally.eat('banana')
        Sally enjoyed eating this banana
        >>> domo_sally.num_eaten
        4
        """
```

5 Yummy In My Summy (5 points)

Write a function `sum_two` that takes an RList of numbers and combines each successive pair of numbers. It does this by mutating the RList so that every other element is no longer part of the RList, and the value of each removed element is summed into the element immediately preceding it. You may assume that the input RList is of even length.

You may not convert the RList to any other data structure, such as tuples, lists or other sequences, and you may not create any new RLists. We have provided doctests that demonstrate the intended functionality and usage.

```
def sum_two(rlist):
    """ Given an even length RList, combines successive
    pairs of elements, mutating the original RList.

    >>> my_rlist = RList.populate(5, 10, 1000, 200)
    >>> str(my_rlist)
    '<5, 10, 1000, 200>'
    >>> sum_two(my_rlist)
    >>> str(my_rlist)
    '<15, 1200>'
    """
```

You may use the space below for scratch work. No work done here will be graded.