CS 61A Structure and Interpretation of Computer Programs

Spring 2013

Midterm 2 Solutions

INSTRUCTIONS

- You have 2 hours to complete the exam.
- The exam is closed book, closed notes, closed computer, closed calculator, except one hand-written 8.5" × 11" crib sheet of your own creation and the official 61A midterm 2 study guide attached to the back of this exam.
- Mark your answers ON THE EXAM ITSELF. If you are not sure of your answer you may wish to provide a *brief* explanation.

Last name	
First name	
SID	
Login	
TA & section time	
Name of the person to your left	
Name of the person to your right	
All the work on this exam is my own. **(please sign)**	

For staff use only

Q. 1	Q. 2	Q. 3	Q. 4	Q. 5	Total
/15	/12	/6	/6	/11	/50

1. (15 points) You Will Be Baked. And Then There Will Be Cake.

(a) Assume that you have started Python 3 and executed the following statements:

```
the_cake = [1, 2, [3], 4, 5]
a_lie = the_cake[1:4]
the_cake = the_cake[1:4]
great = a_lie
delicious = the_cake
moist = great[:-1]
```

For each of the following expressions, write the value to which it evaluates. If the value is a method value, write Method. If it is a function value, write Function. If evaluation causes an error, write Error. If evaluation would run forever, write Forever. Otherwise, write the resulting value as the interactive interpreter would display it.

Expression	**Evaluates to**
`the_cake`	[2, [3], 4]
`the_cake is a_lie`	False
`the_cake == great`	True
`the_cake is delicious`	True
`the_cake == moist + 4`	Error
`the_cake.append`	Method
`the_cake.append == a_lie.append`	False
`the_cake[1] is a_lie[1]`	True

(b) The following is the recursive list abstract data type from lecture:

```
empty_rlist = None

def rlist(first, rest):
    """Creates an rlist from the element first and the rlist rest."""
    return (first, rest)

def first(s):
    """Returns the first element of the rlist s"""
    return s[0]

def rest(s):
    """Returns the rest (itself an rlist) of s."""
    return s[1]

def len_rlist(s):
    """Returns the length of the rlist s."""
    if s == empty_rlist:
        return 0
    return 1 + len_rlist(rest(s))

def getitem_rlist(s, i):
    """Returns the element at index i in rlist s."""
    if i == 0:
        return first(s)
    return getitem_rlist(rest(s), i - 1)
```

For each of the following pieces of code, circle **Y** if the code contains at least one data abstraction violation, and **N** if the code contains no data abstraction violations. **Do not guess; leave the answer blank if you do not know it.** We will award one point for each correct answer, no points for an incorrect answer, and 0.5 points for each answer left blank.

[✓ Y] N `rlist(4, rlist(5, None))`

[✓ Y] N `rlist(1, (2, (3, empty_rlist)))`

Y [✓ N] `rlist(rlist(1, empty_rlist), rlist(2, empty_rlist))`

[✓ Y] N `first(rest( (1, (2, (3, empty_rlist))) ))`

Y [✓ N]
```
x = rlist(5, rlist( (4, 3, 2), rlist(1, empty_rlist)))
first(rest(x))[1]
```

Y [✓ N] `rlist(empty_rlist, empty_rlist)`

[✓ Y] N `len(rlist(3, rlist(4, empty_rlist)))`

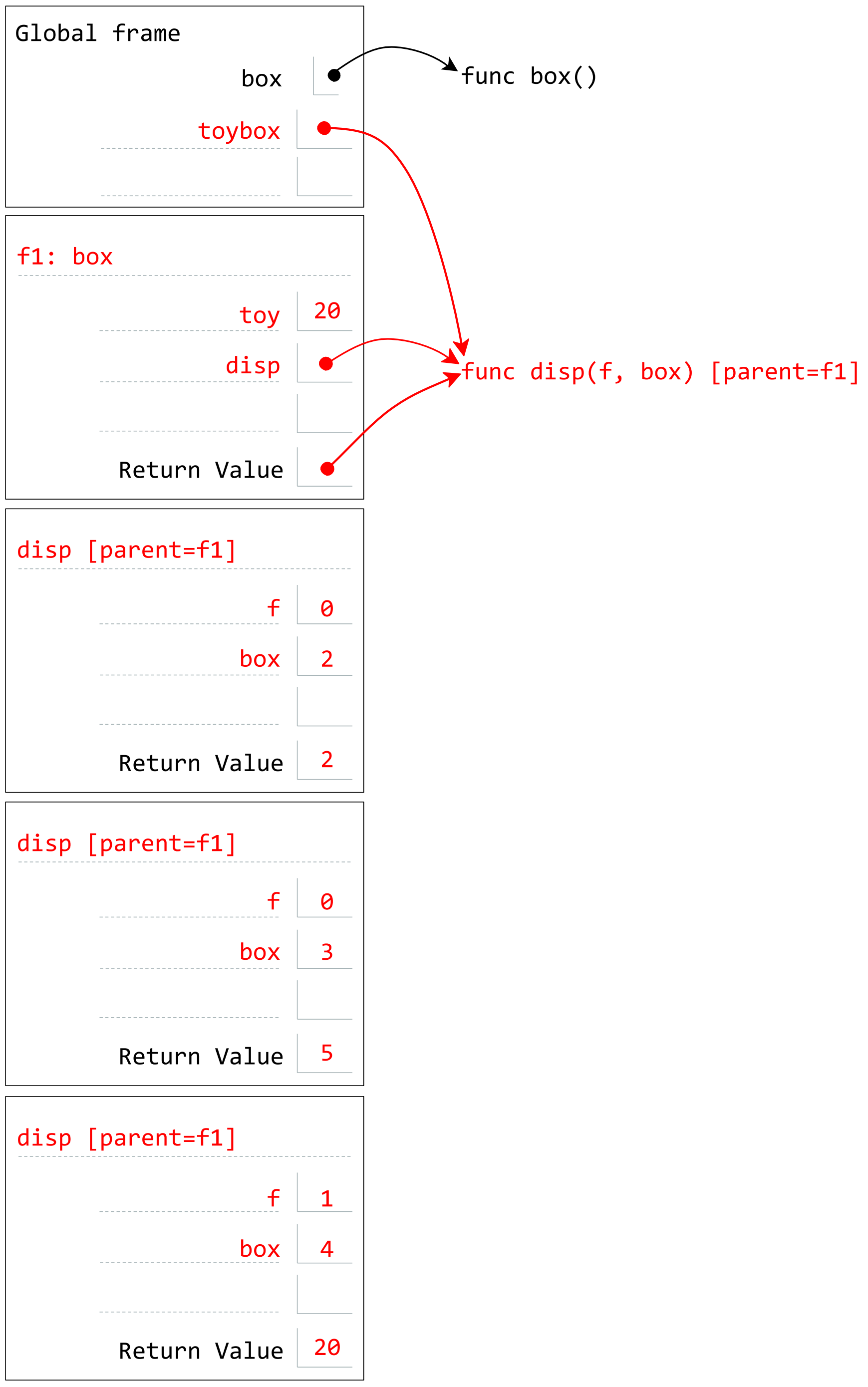
2. (12 points) Environmental Disaster

(a) (6 pt) Fill in the environment diagram that results from executing the code below until the entire program is finished, an error occurs, or all frames are filled. You need only show the final state of each frame. *You may not need to use all of the spaces or frames.*

A complete answer will:

- Add all missing names, labels, and parent annotations to all local frames.
- Add all missing values created during execution.
- Show the return value for each local frame.

```
def box():
    toy = 0
    def disp(f, box):
        nonlocal toy
        if f == 0:
            toy += box
        elif f == 1:
            toy *= box
        return toy
    return disp
toybox = box()
toybox(0, 2)
toybox(0, 3)
toybox(1, 4)
```



(b) **(6 pt)** Fill in the environment diagram that results from executing the code below until the entire program is finished, an error occurs, or all frames are filled. You need only show the final state of each frame. *You may not need to use all of the spaces or frames.*

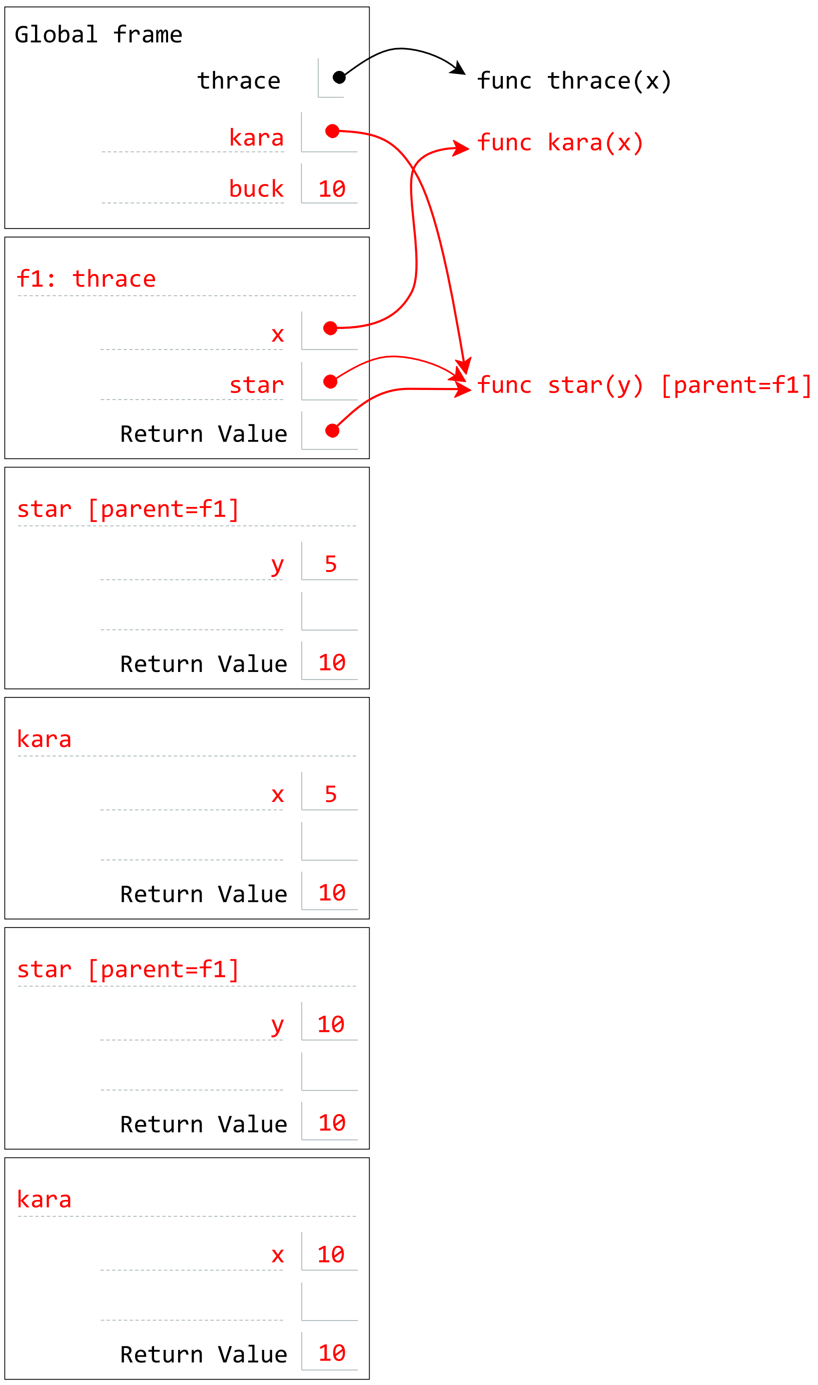
A complete answer will:

- Add all missing names, labels, and parent annotations to all local frames.
- Add all missing values created during execution.
- Show the return value for each local frame.

```
def thrace(x):
    def star(y):
        print(x, y)
        return x(y)
    return star

def kara(x):
    if x > 7:
        return x
    return kara(x * 2)

kara = thrace(kara)
buck = kara(5)
```



3. (6 points) Cue the Queue that Starts with a Q

For each of the following, cross out any incorrect or unnecessary lines in the following code so that the doctests pass for both classes. **Do not cross out class declarations, doctests, or docstrings.** You can cross out anything else, including method declarations, and your final code should make as much use of inheritance as possible. **Make sure to cross out the entire line for anything you wish to remove.**

Note: The `pop` method of a `list` removes the item at the given position and returns it.

(a)
```
class Queue(object):
    """Creates a Queue, which is like a list that supports 2
    operations: enqueue (adding an item to the back of the queue) and
    dequeue (removing an item from the front of the queue).

    >>> q = Queue()
    >>> q.enqueue(5)
    >>> q.enqueue(3)
    >>> q.enqueue(2)
    >>> q.dequeue()
    5
    """

    def __init__(self):
        self.items = []

    def enqueue(self, item):
        self.items.append(item)

    def dequeue(self):
        return self.items.pop(0)
```

(b)
```
class PriorityQueue(Queue):
    """A PriorityQueue is like a sorted list that supports two
    operations: enqueue (adding an item to the PriorityQueue) and
    dequeue (removing the smallest item from the PriorityQueue).

    >>> p = PriorityQueue(2)
    >>> p.enqueue(5)
    >>> p.enqueue(3)
    >>> p.enqueue(2)
    >>> p.dequeue()
    2
    """

    def enqueue(self, item):
        Queue.enqueue(self, item)
        self.items.sort()
```

4. (6 points) Prime RBIs

The Cal Mathletic Club is a group of math enthusiasts who compete in mathematical competitions. Since a sharp mind requires a sharp body, they also field an intramural baseball team. Unfortunately, this team has not been very good in recent years. In fact, they only had 2 runs batted in (RBIs) in all of 2010! The next two years were nearly as dreadful, with 3 RBIs in 2011, and 5 RBIs in 2012.

Being mathletes, they notice that their RBI totals have been consecutive prime numbers in each of the last three years. Being mathletes, they decide they should continue this trend, slowly improving their play each year by batting in the next prime number of runs.

Help the mathletes to determine their long-term goals by writing a higher-order function `make_prime_generator` that returns a function to generate primes. The latter function should return 2 the first time it is called, 3 the next time, then 5, 7, 11, and so on, returning the next prime number each time it is called.

(For the non-mathletic, a prime number can be defined as an integer greater than 1 that is not divisible by any other integer greater than 1. Thus, a prime number p's only positive divisors are 1 and p.)

```
def make_prime_generator():
    """Return a function that computes the next prime number each time it
    is called.

    >>> gen = make_prime_generator()
    >>> gen(), gen(), gen()
    (2, 3, 5)
    >>> [gen() for _ in range(10)]
    [7, 11, 13, 17, 19, 23, 29, 31, 37, 41]
    """
    last_prime = 1
    def generate():
        nonlocal last_prime
        found_prime = False
        while not found_prime:
            last_prime += 1
            found_prime = True
            for d in range(2, last_prime):
                if last_prime % d == 0:
                    found_prime = False
        return last_prime
    return generate
```

Here is a recursive solution:

```
def make_prime_generator():
    last_prime = 1
    def generate():
        nonlocal last_prime
        last_prime += 1
        for d in range(2,last_prime):
            if last_prime % d == 0:
                return generate()
        return last_prime
    return generate
```

5. (11 points) Mutation: It is the Key to Our Evolution

The following is an object-oriented recursive list implementation:

```
class Rlist(object):
    """A recursive list consisting of a first element and the rest."""
    class EmptyList(object):
        def __len__(self):
            return 0
    empty = EmptyList()

    def __repr__(self):
        f = repr(self.first)
        if self.rest is Rlist.empty:
            return 'Rlist({0})'.format(f)
        else:
            return 'Rlist({0}, {1})'.format(f, repr(self.rest))

    def __init__(self, first, rest=empty):
        self.first = first
        self.rest = rest

    def __len__(self):
        return 1 + len(self.rest)

    def __getitem__(self, i):
        if i == 0:
            return self.first
        return self.rest[i - 1]
```

(a) Implement a `mutating_map` method that takes in a function and applies it to each element in an `Rlist`. This method should mutate the list in place, replacing each element with the result of applying the function to it. Do not create any new objects. You may assume that the input `Rlist` contains at least one element.

```
def mutating_map(self, fn):
    """Mutate this Rlist by applying fn to each element.

    >>> r = Rlist(1, Rlist(2, Rlist(3)))
    >>> r.mutating_map(lambda x: x + 1)
    >>> r
    Rlist(2, Rlist(3, Rlist(4)))
    """
    self.first = fn(self.first)
    if self.rest != Rlist.empty:
        self.rest.mutating_map(fn)
```

(b) The *sieve of Eratosthenes* is an ancient algorithm for finding prime numbers. It starts with a sequence of numbers between 2 and n, in order. The first number is a prime, and the algorithm removes all larger multiples of that number from the sequence. Then the next remaining number is a prime, and the algorithm removes all larger multiples of that number from the sequence, and so on, until the end of the sequence is reached. At that point, all remaining numbers in the sequence are prime.

Here is a more concrete illustration of this process:

Initial sequence:	2, 3, 4, 5, 6, 7, 8, 9, 10
Remove larger multiples of 2:	2, 3, 5, 7, 9
Remove larger multiples of 3:	2, 3, 5, 7
Remove larger multiples of 5:	2, 3, 5, 7
Remove larger multiples of 7:	2, 3, 5, 7
Done.	

In this problem, you will implement this algorithm on `Rlist`s. Assume that you have `map_rlist` and `filter_rlist` functions with the following signatures and docstrings:

```
def map_rlist(s, fn):
    """Return an Rlist resulting from mapping fn over the elements of s.

    >>> map_rlist(Rlist(1, Rlist(2, Rlist(3))), lambda x: x * x)
    Rlist(1, Rlist(4, Rlist(9)))
    """

def filter_rlist(s, fn):
    """Filter the elements of s by predicate fn.

    >>> filter_rlist(Rlist(1, Rlist(2, Rlist(3))), lambda x: x % 2 == 1)
    Rlist(1, Rlist(3))
    """
```

i. First, write a function `sequence_to_rlist` that converts a Python sequence into an `Rlist`. Elements in the resulting `Rlist` should be in the same order as in the original sequence.

```
def sequence_to_rlist(seq):
    """Converts a sequence to an Rlist, preserving order.

    >>> sequence_to_rlist((3, 2, 1))
    Rlist(3, Rlist(2, Rlist(1)))
    """
    if not seq:
        return Rlist.empty
    return Rlist(seq[0], sequence_to_rlist(seq[1:]))
```

An iterative version:

```
def sequence_to_rlist(seq):
    rlst = Rlist.empty
    for i in range(len(seq) - 1, -1, -1):
        rlst = Rlist(seq[i], rlst)
    return rlst
```

ii. Now fill in the following function `prime_sieve` that implements the sieve of Eratosthenes algorithm. This function takes in an `Rlist` of numbers between 2 and n and removes all composite numbers from the `Rlist`. You may assume that the input `Rlist` has at least one element. You may leave the last line blank if you do not need it.

```
def prime_sieve(rlst):
    """Remove all composite numbers from the input Rlist. Assumes
    that the input contains the numbers from 2 to len(rlst), in
    order.

    >>> seq = sequence_to_rlist(range(2, 15))
    >>> prime_sieve(seq)
    >>> seq
    Rlist(2, Rlist(3, Rlist(5, Rlist(7, Rlist(11, Rlist(13))))))
    """
    while rlst.rest != Rlist.empty:
        func = lambda x: x % rlst.first != 0
        rlst.rest = filter_rlist(rlst.rest, func)
        rlst = rlst.rest
```