CS 61A Structure and Interpretation of Computer Programs
Spring 2015

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 1 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	
Email (...@berkeley.edu)	
Login (e.g., cs61a-ta)	
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	Total
/12	/12	/14	/12	/50

1. (12 points) Mutater-tot

For each of the expressions in the table below, write the output displayed by the interactive Python interpreter when the expression is evaluated. **The output may have multiple lines**. Expressions are evaluated in order, and **expressions may affect later expressions**.

Whenever the interpreter would report an error, write Error. If execution would take forever, write Forever.

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

```
def ready(betty):
    print(len(betty))
    betty[0].append(betty)
    return betty[0:1]

def get_set(s):
    ready(s)
    return s.pop()

def go(on, up):
    if up:
        return go(on[0], up-1)
    else:
        return on

f = [1, [2]]
g = [[3, 4], [5], 6]
h = [g, g]
```


Expression	Interactive Output
f.pop()	[2]
h[1].pop()	6
g[g[1][0]-g[0][1]]	[5]
len(ready(g))	2 1
g[0][2][0][1]	4
ready(get_set(h))[0][0]	2 3 3
[len(go(h, k)) for k in range(3)]	[1, 3, 4]

2. (12 points) Vulcans

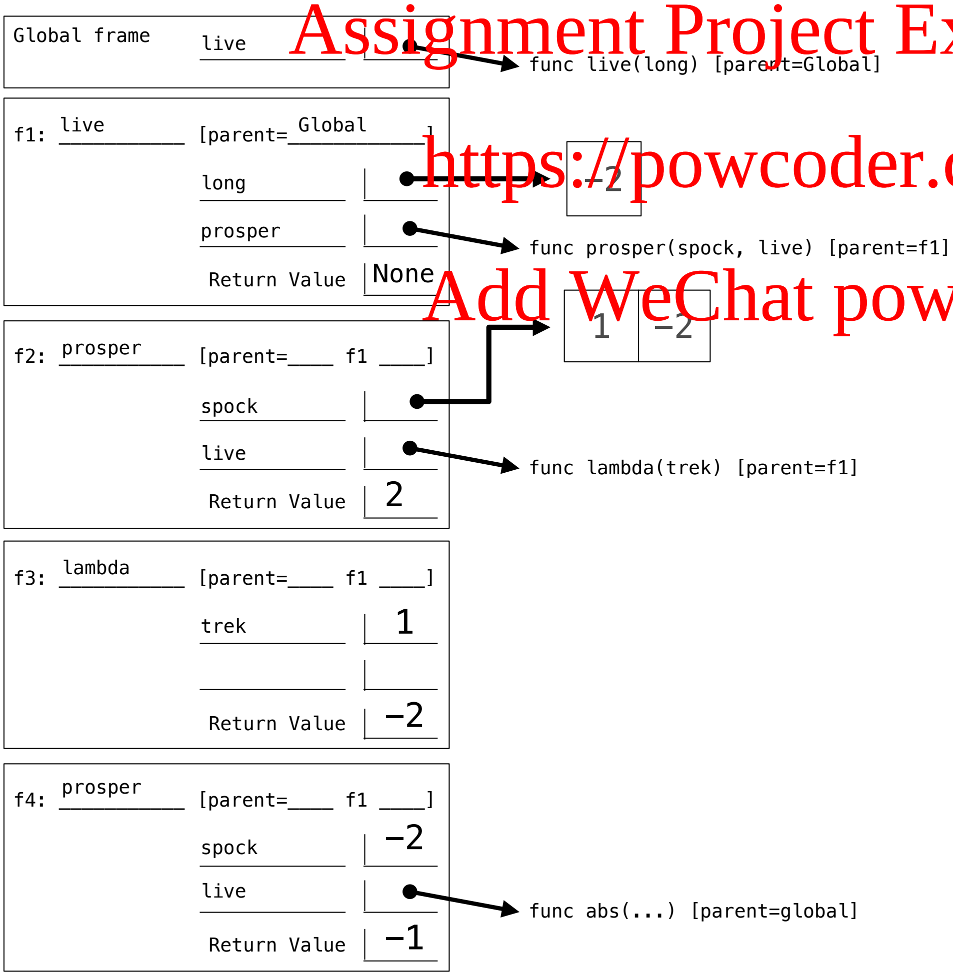
(a) **(8 pt)** Fill in the environment diagram that results from executing the code below until the entire program is finished, an error occurs, or all frames are filled. *You may not need to use all of the spaces or frames.*

A complete answer will:

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

Remember: Do not add a new frame when calling a built-in function (such as `abs`). The built-in `abs` function is always written as `func abs(...) [parent=Global]`.

```
def live(long):
    def prosper(spock, live):
        nonlocal long
        if len(long) == 1:
            return spock+1
        long[1] = live(long[0])
        long = long[1:]
        prosper(long[0], abs)
        return spock[0]+1
    prosper(long, lambda trek: trek-3)
live([1, 4])
```



(b) **(4 pt)** Fill in the environment diagram that results from executing the code below after the entire program is finished. No errors occur during the execution of this example.

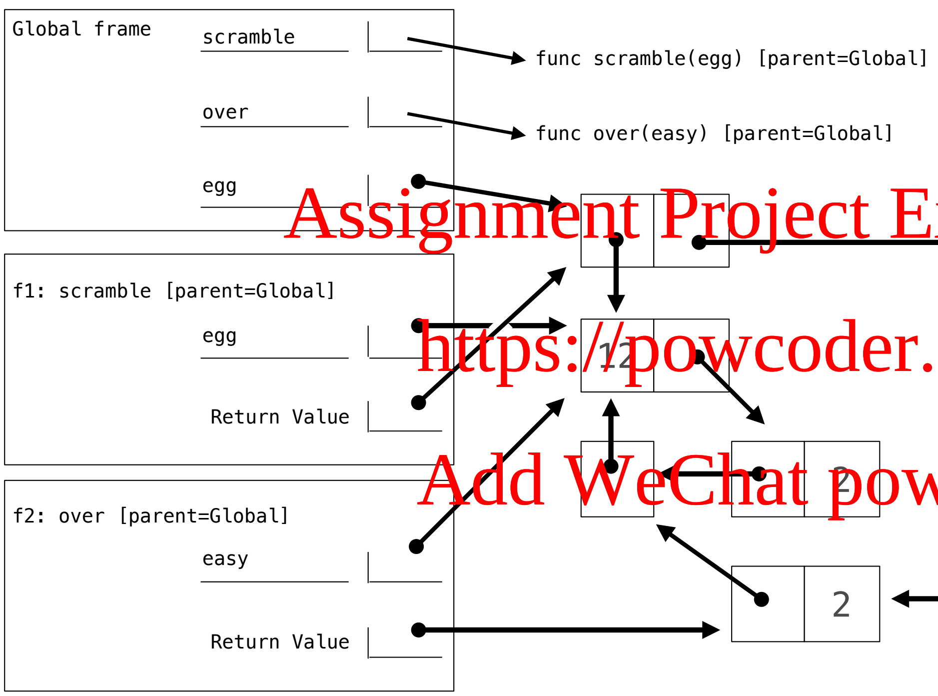
A complete answer will:

- Add all missing values created or referenced during execution.
- Show the return value for each local frame.

```
def scramble(egg):
    return [egg, over(egg)]

def over(easy):
    easy[1] = [[easy], 2]
    return list(easy[1])

egg = scramble([12, 24])
```



3. (14 points) Will Code for Points

(a) (2 pt) Implement `objectify`, which takes a tree data abstraction and returns an equivalent `Tree` instance. Both the `Tree` class and the tree data abstraction appear on the midterm 2 study guide.
Warning: Do not violate the tree data abstraction! (Exams are flammable.)

```
def objectify(t):
    """Return a Tree instance equivalent to a tree represented as a list.

    >>> m = tree(2)
    >>> m
    [2]
    >>> objectify(m)
    Tree(2)
    >>> r = tree(3, [tree(4, [tree(5), tree(6)]), tree(7, [tree(8)])])
    >>> r
    [3, [4, [5], [6]], [7, [8]]]
    >>> objectify(r)
    Tree(3, [Tree(4, [Tree(5), Tree(6)]), Tree(7, [Tree(8)])])
    """
    return Tree(root(t), [objectify(b) for b in branches(t)])
```


(b) (2 pt) Circle the Θ expression that describes the number of `Tree` instances constructed by calling `objectify` on a tree with n nodes.

$\Theta(1)$ $\Theta(\log n)$ $\Theta(n)$ $\Theta(n^2)$ $\Theta(2^n)$

(c) (4 pt) Implement `closest`, which takes a `Tree` of numbers `t` and returns the smallest absolute difference anywhere in the tree between an entry and the sum of the entries of its branches. The `Tree` class appears on the midterm 2 study guide. The built-in `min` function takes a sequence and returns its minimum value.
Reminder: A branch of a branch of a tree `t` is *not* considered to be a branch of `t`.


```
def closest(t):
    """Return the smallest difference between an entry and the sum of the
    root entries of its branches.

    >>> t = Tree(8, [Tree(4), Tree(3)])
    >>> closest(t) # |8 - (4 + 3)| = 1
    1
    >>> closest(Tree(5, [t])) # Same minimum as t
    1
    >>> closest(Tree(10, [Tree(2), t])) # |10 - (2 + 8)| = 0
    0
    >>> closest(Tree(3)) # |3 - 0| = 3
    3
    >>> closest(Tree(8, [Tree(3, [Tree(1, [Tree(5)])])])) # 3 - 1 = 2
    2
    >>> sum([])
    0
    """

    diff = abs(t.entry - sum([b.entry for b in t.branches]))

    return min([diff] + [closest(b) for b in t.branches])
```

(d) **(6 pt)** Implement `double_up`, which mutates a linked list by inserting elements so that each element is adjacent to an equal element. The `double_up` function inserts as few elements as possible and returns the number of insertions. The `Link` class appears on the midterm 2 study guide.

```
def double_up(s):
    """Mutate s by inserting elements so that each element is next to an equal.

    >>> s = Link(3, Link(4))
    >>> double_up(s) # Inserts 3 and 4
    2
    >>> s
    Link(3, Link(3, Link(4, Link(4))))
    >>> t = Link(3, Link(4, Link(4, Link(5))))
    >>> double_up(t) # Inserts 3 and 5
    2
    >>> t
    Link(3, Link(3, Link(4, Link(4, Link(5, Link(5))))))
    >>> u = Link(3, Link(4, Link(3)))
    >>> double_up(u) # Inserts 3, 4, and 3
    3
    >>> u
    Link(3, Link(3, Link(4, Link(4, Link(3, Link(3))))))
    """
    if s is Link.empty:

        return 0

    elif s.rest is Link.empty:

        s.rest = Link(s.first)

        return 1

    elif s.first == s.rest.first:

        return double_up(s.rest.rest)

    else:

        s.rest = Link(s.first, s.rest)

        return 1 + double_up(s.rest.rest)
```

4. (12 points) What color is it?

(a) (6 pt) Implement the `look` method of the `Dress` class. The `look` method returns a `Dress` instance's current `color` when the number of times that the instance's `look` method has ever been invoked evenly divides the total number times that the `look` method of any `Dress` instance has ever been invoked. Otherwise, the instance's `color` changes to the most recently returned color from any call to `look`, and `None` is returned.

```
class Dress:
    """What color is the dress?

    >>> blue = Dress('blue')
    >>> blue.look()
    'blue'
    >>> gold = Dress('gold')
    >>> gold.look()
    'gold'
    >>> blue.look()  # 2 does not evenly divide 3; changes to gold
    >>> Dress('black').look()
    'black'
    >>> gold.look()  # 2 does not evenly divide 5; changes to black
    >>> gold.look()  # 3 evenly divides 6
    'black'
    >>> Dress('white').look()
    'white'
    >>> gold.look()  # 4 evenly divides 8
    'black'
    >>> blue.look()  # 3 evenly divides 9
    'gold'
    """
    seen = 0
    color = None

    def __init__(self, color):
        self.color = color
        self.seen = 0

    def look(self):

        Dress.seen += 1

        self.seen += 1

        if Dress.seen % self.seen == 0:

            Dress.color = self.color

            return self.color

        else:

            self.color = Dress.color
```

(b) **(6 pt)** Implement `decrypt`, which takes a string `s` and a dictionary `d` that contains words as values and their secret codes as keys. It returns a list of all possible ways in which `s` can be decoded by splitting it into secret codes and separating the corresponding words by spaces.

```
def decrypt(s, d):
    """List all possible decoded strings of s.

    >>> codes = {
    ...     'alan': 'spooky',
    ...     'al': 'drink',
    ...     'antu': 'your',
    ...     'turing': 'ghosts',
    ...     'tur': 'scary',
    ...     'ing': 'skeletons',
    ...     'ring': 'ovaltine'
    ... }
    >>> decrypt('alanturing', codes)
    ['drink your ovaltine', 'spooky ghosts', 'spooky scary skeletons']
    """

    if s == '':

        return []

    messages = []

    if s in d:

        messages.append(d[s])

    for k in range(1, len(s)+1):

        first, suffix = s[:k], s[k:]

        if first in d:

            for rest in decrypt(suffix, d):

                messages.append(d[first] + ' ' + rest)

    return messages
```