CS 61A Structure and Interpretation of Computer Programs
Fall 2015 Make-Up Midterm 1

INSTRUCTIONS

- You have 50 minutes to complete the exam.
- The exam is closed book, closed notes, closed computer, closed calculator, except one hand-written 8.5” $\times$ 11” crib sheet of your own creation and the official CS 61A midterm 1 study guide.
- Mark your answers **on the exam itself**. We will *not* grade answers written on scratch paper.

Last name	
First name	
Student ID number	
BearFacts email (`_@berkeley.edu`)	
TA	
All the work on this exam is my own. **(please sign)**	

1. (12 points) Do Not Be Alarmed

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. If an error occurs, write "Error".

Hint: No answer requires more than 5 lines. (It's possible that all of them require even fewer.)

The first two rows have been provided as examples.

Recall: The interactive interpreter displays the value of a successfully evaluated expression, unless it is `None`.

The `compose1` function appears on the left column of page 2 of your study guide. Assume it is defined.

Assume that you have also executed the following statements:

```
def fire(alarm, y):
    if vlsb(y):
        return alarm(y+1)

def vlsb(x):
    print(x)
    if x > 3:
        return vlsb(x-1)
    return True

siren = lambda loud: fire(print, y)
y = 4
```

Expression	**Interactive Output**
`pow(2, 3)`	8
`print(4, 5) + 1`	4 5 Error
`compose1(print, print)(5)`	
`fire(vlsb, 1)`	
`fire(siren, 2)`	

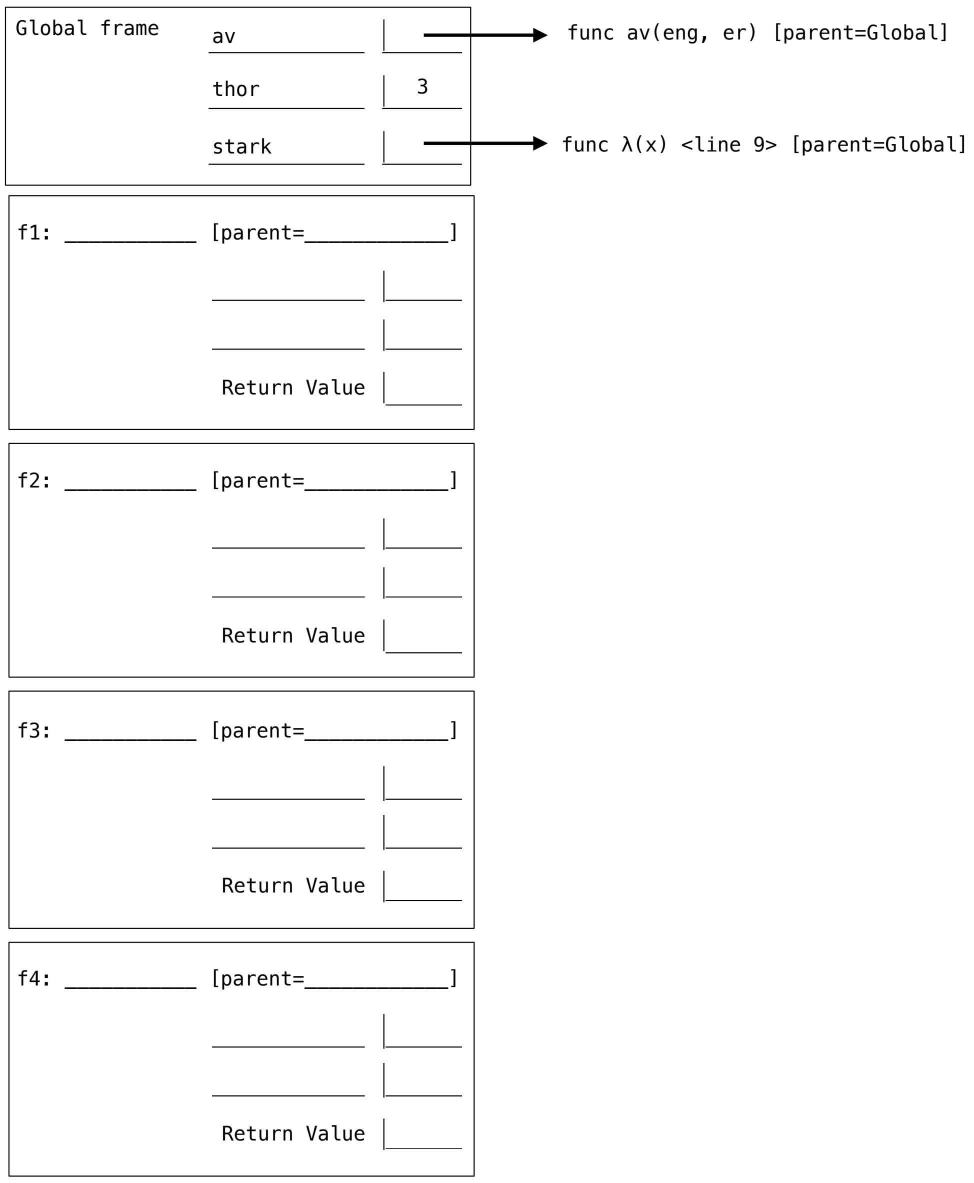
2. (12 points) Avengers

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.
- **Indicate the line number for each lambda function value.**

```
 1  def av(eng, er):
 2      er = lambda: eng
 3      def eng(er):
 4          eng = 2
 5          return lambda eng: eng + thor
 6      return er()
 7
 8  thor = 3
 9  stark = lambda x: lambda y: 4
10  av(stark, lambda: 5)(6)(7)
```



Global frame
av
func av(eng, er) [parent=Global]
thor
3
stark
func λ(x) <line 9> [parent=Global]
f1: ___________ [parent=___________]
Return Value
f2: ___________ [parent=___________]
Return Value
f3: ___________ [parent=___________]
Return Value
f4: ___________ [parent=___________]
Return Value

3. (16 points) Deja Vu

****IMPORTANT DEFINITION**** (Same as last exam) Each digit in a non-negative integer `n` has a *digit position*. Digit positions begin at 0 and count from the right-most digit of `n`. For example, in 568789, the digit 9 is at position 0 and digit 7 is at position 2. The digit 8 appears at both positions 1 and 3.

(a) (6 pt) Implement `luhn_sum` (again, but in a different way). The *Luhn sum* of a non-negative integer *n* adds the sum of each digit in an *even* position to the sum of doubling each digit in an *odd* position. If doubling an odd digit results in a two-digit number, those two digits are summed to form a single digit.

```
def luhn_sum(n):
    """Return the Luhn sum of n.

    >>> luhn_sum(135)     # 1 + 6 + 5
    12
    >>> luhn_sum(175)     # 1 + (1+4) + 5
    11
    >>> luhn_sum(138743)  # From lecture: 2 + 3 + (1+6) + 7 + 8 + 3
    30
    """

    even = lambda d: d

    odd = lambda d: ________________________________________________________

    return alt(even, odd, n)

def alt(f, g, x):

    if x == 0:

        return x

    else:

        return _______________________ + alt(g, f, _________________________)
```

(b) **(6 pt)** Implement `subsum`, which takes two non-negative integers `n` and `k` as arguments. It returns the largest possible sum of up to `k` consecutive digits in `n`. You may use the built-in `max` function. You **may not** make recursive calls to `subsum`.

```
def subsum(n, k):
    """Return the maximum sum of up to k consecutive digits in n.

    >>> subsum(162553, 1) # 6
    6
    >>> subsum(162553, 2) # 5 + 5
    10
    >>> subsum(162553, 3) # 6 + 2 + 5 OR 5 + 5 + 3
    13
    >>> subsum(5, 0)
    0
    >>> subsum(5432, 100) # 5 + 4 + 3 + 2
    14
    """
    recent, total, largest = 0, 0, 0


    w = pow(10, k)  # Raises 10 to the kth power. E.g., pow(10, 3) is 1000.


    while n:


        n, d = n // 10, n % 10


        recent = 10 * recent + d


        total = ______________________________________________________________


        recent = recent _______________________  # The last k digits of recent


        ________________ = ___________________________________________________


    return largest
```

(c) **(4 pt)** Circle all positive integers less than 20 that can be placed in the blank below so that the final expression evaluates to 20. Assume that `subsum` is implemented correctly. The `make_adder` function appears on the left column of page 2 of your study guide. *Hint:* 4 + 2 + 5 + 6 = 17.

```
x = make_adder( ______________ - 3 )

subsum(x(x(4256)), 4)
```

1 2 3 4 5 6 7 8 9 10

11 12 13 14 15 16 17 18 19