CS 88 Spring 2016 Computational Structures in Data Science

PRACTICE MIDTERM 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 CS 88 midterm 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	
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)**	

1. (14 points) Evaluators Gonna Evaluate

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 4 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`. Assume that you have started `python3` and executed the following statements:

```
def listfun(seq, fun):
    return [fun(x,seq,i) for x,i in zip(seq, range(len(seq))) ]
```

Expression	**Interactive Output**

`def f(ele,seq,ind):`
`  return ele`
 	
`listfun(range(5), f)`
	[0, 1, 2, 3, 4]

`pow(3, 2)`	9

`def boo(x,y):`
`  return x*x - y`
 	
`boo(3,4)`
	5

`def hoo(x,y):`
`  while x > y:`
`  x = x - y`
`  return x`
 	
`hoo(10,3)`
	1

`def f(ele, seq, ind):`
`  return ele//2`
 	
`listfun(range(5), f)`
	[0, 0, 1, 1, 2]

`def f(ele, seq, ind):`
`  return ele*ind`
 	
`listfun(range(5),f)`
	[0, 1, 4, 9, 16]

`def f(ele, seq, ind):`
`  return ele%2 and True`
 	
`listfun(range(5), f)`
	[0, True, 0, True, 0]

Expression	**Interactive Output**

`def h(ele, seq, ind):`
`  return seq[len(seq)-ind]`
 	
`listfun(range(5),h)`
	Error

`def f(ele, seq, ind):`
`  p = 0`
`  for x in seq[:ind]:`
`  p = p+x`
`  return p`
 	
`listfun([1,2,3,4,5], f)`
	[0, 1, 3, 6, 10]

`def f():`
`  def f(e,s,i):`
`  return (i,s[i])`
`  return f`
 	
`listfun([1,2,3,4,5], f())`
	[(0, 1), (1, 2), (2, 3), (3, 4), (4, 5)]

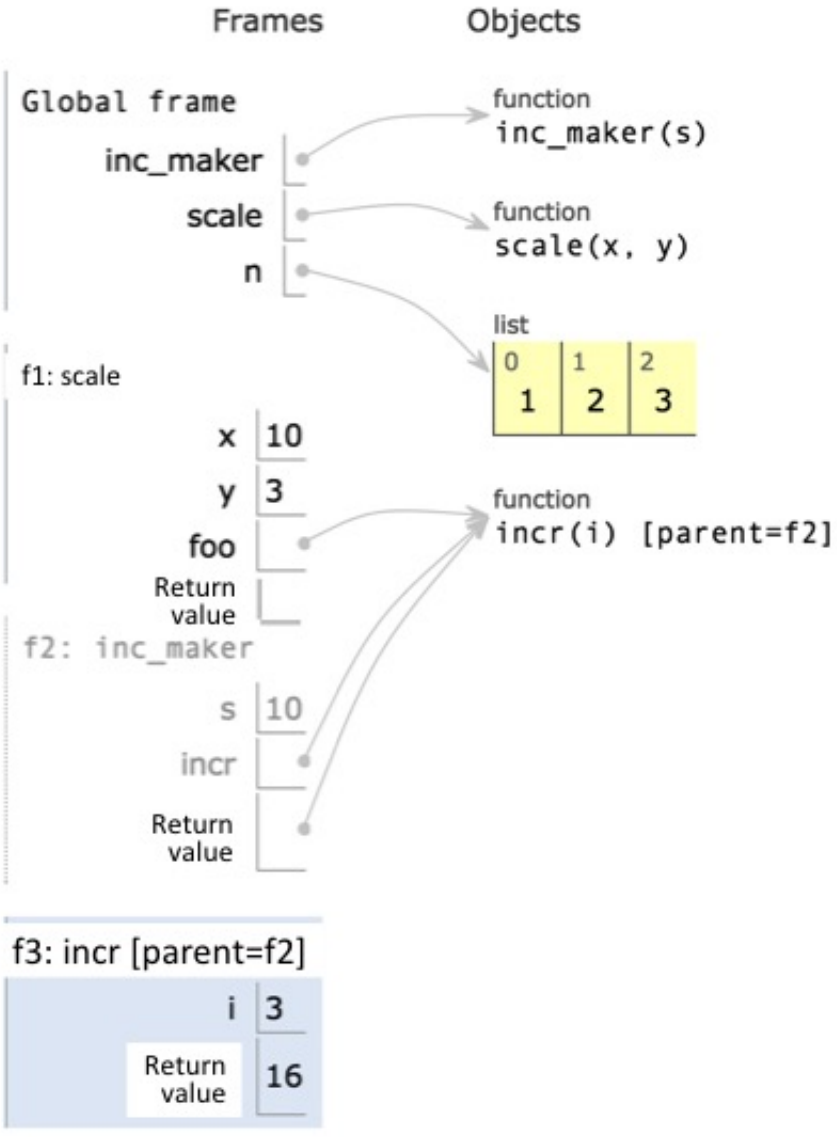
2. (8 points) Environmental Policy

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.

```
1 def inc_maker(s):
2     def incr(i):
3         return i+s+len(n)
4     return incr
5 def scale(x, y):
6     foo = inc_maker(x)
7     return foo(y)
8 n = [1,2,3]
9 scale(10, n[2])
```



3. (12 points) Digit Fidget

(a) (3 pt) Implement the `num_str_base` function, which takes a integer `n` and integer base `b`. It returns a string representing **n** in base **b**. *You may not use recursive calls.*

```
def num_str_base(n, b):
    """Return a string representing number n in base b.

    >>> num_str_base(37, 10)
    '37'
    >>> num_str_base(37, 8)
    '45'
    >>> num_str_base(37, 16)
    '25'
    >>> num_str_base(37, 2)
    '100101'
    >>> num_str_base(-37, 8)
    '-45'
    """
    digits = "0123456789ABCDEF"

    if n < 0:


        return "-" + num_str_base(-n, b)


    if n == 0:


        return "0"


    else:


        nb_str = ""


        while n:


            nb_str = digits[n % b] + nb_str


            n = n // b


        return nb_str
```

(b) (3 pt) Implement `fold` using recursive calls. You can think of folding the list at its middle element and summing the pairs that line up. That is the first item of the folded list is the sum of the first and last items of the list. If the list has an odd number of items, the middle element is doubled.

```
def fold(s):
    """Return the list formed by folding s in the middle and adding the pairs.

    >>> fold([1, 2, 7, 9])
    [10, 9]
    >>> fold([1, 2, 4, 7, 9])
    [10, 9, 8]
    """
    if not s :

        return []

    elif len(s) == 1:

        return [s[0]*2]

    else:

        return [s[0]+s[-1]] + fold(s[1:-1])
```

(c) (3 pt) Implement the function `echo_maker` which takes an integer n as input and returns a function which takes a string and returns n concatenations of the string. Use this to implement the function `echo`.

```
def echo_maker(n):
    """Return a function that echos a string n times.

    >>> echo_maker(3)("hello")
    'hellohellohello'
    """
    def echofun(s):


        return s*n


    return echofun

def echo(s, n):
    """Return a list containing the elements of s, each echoed n times.

    >>> echo(["Hey", "Ho"], 2)
    ['HeyHey', 'HoHo']
    """
    return [echo_maker(n)(x) for x in s]
```

(d) (3 pt) Implement the function `tuple_slice` which takes list of tuples and returns a list of the i-th element of each tuple. It should be implemented using a list comprehension and require only a single line.

```
def tuple_slice(s, i):
    return [x[i] for x in s]
```

4. (6 points) Sneaky

Implement the function `inc_cv` which takes a (value, count) tuple *vc*and a target value, v. It returns a tuple with the orginal value, but the count incremented if the tuple value is equal to v. Otherwise, it returns the original tuple.

Implement the function `inc_count` which takes a list of (value, count) tuples and a value. It returns a list of (value, count) tuples with the tuple containing the specified target value having an incremented count.

Use these functions and assume that you have a working version of `tuple_splice` from the previous problem to implement `unique_count` which takes a list of values and returns a list of (value, count) tuples that contains a tuple for each value in the original list and the count of the times it occurs in that list.

```
def inc_vc(vc, v):
    """For (val, count) tuple vc, if val==v return tuple with incremented
    count, otherwise return the original tuple, vc.

    >>> inc_vc((1,2), 1)
    (1,3)
    >>> inc_vc((1,2), 2)
    (1,2)
    """
    if vc[0] != v:
        return vc
    else:
        return (vc[0], vc[1]+1)

def inc_count(vc_seq, v):
    """Return list of (val,count) tuples where tuple with val==v has incremented
    count.

    >>> inc_count([(1,2),(3,4)],3)
    [(1, 2), (3, 5)]
    """
    return [inc_vc(vc,v) for vc in vc_seq]

def unique_count(s):
    """Return list of (val, count) tuples where val appears in s count times.

    >>> unique_count([1,2,3,4,2,3,2])
    [(1, 1), (2, 3), (3, 2), (4, 1)]
    """
    unique_counts = []
    for x in s:
        if x not in tuple_slice(unique_counts, 0):
            unique_counts = unique_counts + [(x, 1)]
        else:
            unique_counts = inc_count(unique_counts, x)
    return unique_counts
```