CS 88
Spring 2016

Computational Structures in Data Science

RETAKE MIDTERM SOLUTIONS

INSTRUCTIONS

- You have 2 hours to complete the exam.
- The exam is closed book, closed notes, closed computer, closed calculator, except one 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. *Reminder*: Slicing with [1 :] gives the rest of the sequence, after the first element. *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 seqfun(seq, fun):
    return [ fun(seq[i:], i) for i in range(len(seq)) ]
```

Expression	Interactive Output

`def f(s, i):`
`    return i`
 	
`seqfun([1, 2, 3, 5, 10], f)`
	[0, 1, 2, 3, 4]

`pow(2, 3)`	8

`def moo(x, y, z):`
`    return (x-y)//z`
 	
`moo(9,3,2)`
	3

`def zoo(x, y, z):`
`    while (x > z):`
`        x = x - y`
`    return x`
 	
`zoo(13, 2, 6)`
	5

`def f(s, i):`
`    return s[0]`
 	
`seqfun([1, 2, 3, 5, 10], f)`
	[1, 2, 3, 5, 10]

`def f(s, i):`
`    x = s[0]`
`    if x > i:`
`        return x`
`    else:`
`        return i`
 	
`seqfun([1, 2, 3, 5, 10], f)`
	[1, 2, 3, 5, 10]

Expression	**Interactive Output**

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

`def f(s, i):`
`    return s`
 	
`seqfun([1, 2, 3, 4], f)`
	[[1, 2, 3, 4], [2, 3, 4], [3, 4], [4]]

`def f(x):`
`    def g(s, i):`
`        return x+i`
`    return g`
 	
`seqfun([2, 3, 4, 5], f(3))`
	[3, 4, 5, 6]

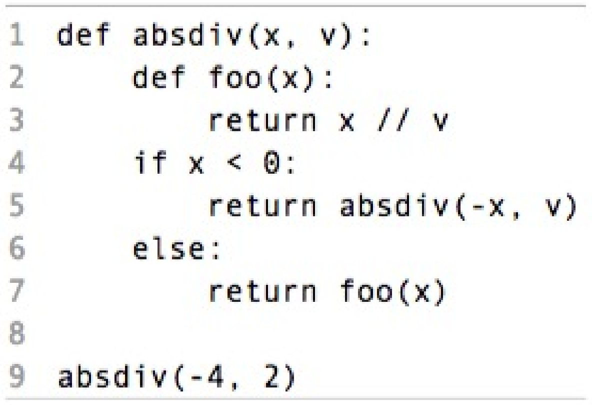
`def f(s,i):`
`    p = 0`
`    for x in s:`
`        p = p+x`
`    return p`
 	
`seqfun([2, 3, 4, 5], f)`
	[14, 12, 9, 5]

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.*

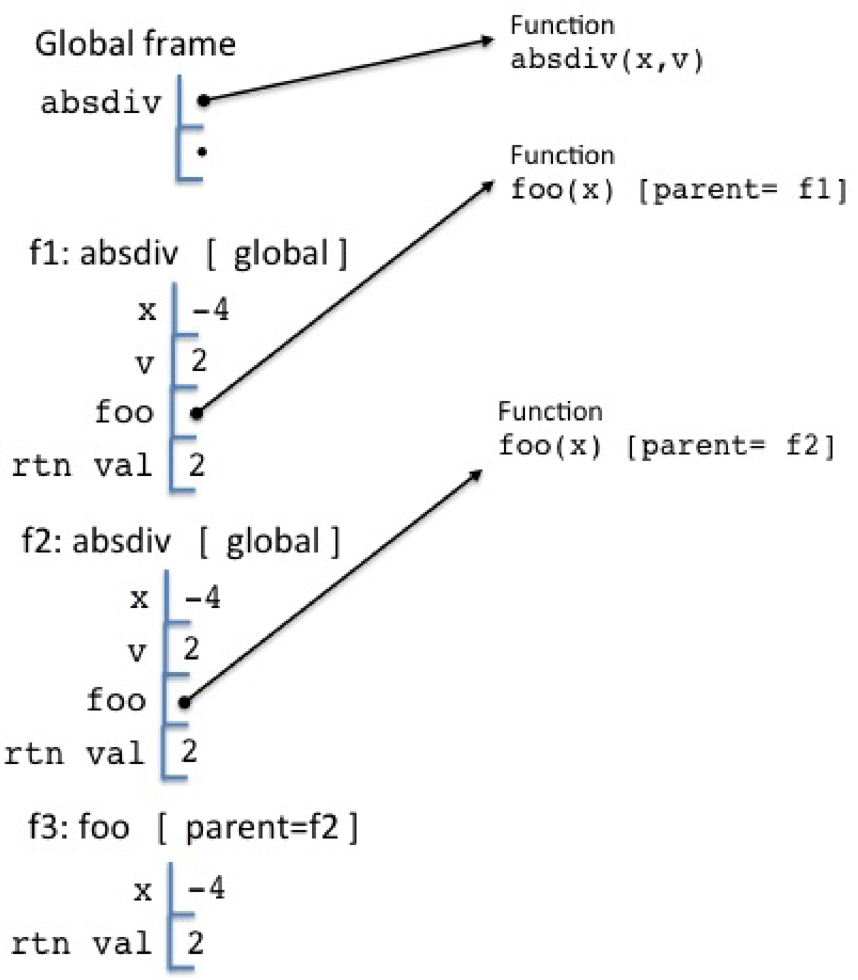
A complete answer will:

- 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.



```
def absdiv(x, v):
    def foo(x):
        return x // v
    if x < 0:
        return absdiv(-x, v)
    else:
        return foo(x)

absdiv(-4, 2)
```



3. (12 points) Function Junction

(a) (3 pt) Implement the `remove_first` function, which takes a sequence `s` and value `x`. It returns a new list consisting of the elements of `s` except the first occurence of `x`. You must use a loop, not recursion. Can you cannot call list methods, like `index` or `remove`.

```
def remove_first(s, x):
    """Remove the first occurence of x in sequence s.

    >>> remove_first(['a', 'b', 'c','b'], 'b')
    ['a', 'c', 'b']
    >>> remove_first(['a', 'b', 'c','b'], 'e')
    ['a', 'b', 'c', 'b']
    >>> remove_first(['a', 'b', 'c','b'], 'a')
    ['b', 'c', 'b']
    """
    res = []
    for i in range(len(s)):
        if s[i] != x:
            res = res + [s[i]]
        else:
            return res + s[i+1:]
    return res
```

(b) **(3 pt)** Implement `fixer`, which computes the sum of the result of applying a function `f` to the elements of a sequence, where that result is positive.

```
def identity(x):
    return x

def square(x):
    return x*x

def fixer(f, seq):
    """Return the sum of f(x) for x in seq where f(x) > 0.

    >>> fixer(identity, [1, 2, -3, 4])
    7
    >>> fixer(square, [1, 2, -3, 4])
    30
    """
    psum = 0
    for x in seq:
        if f(x) > 0:
            psum = psum + f(x)
    return psum
```

(c) **(3 pt)** Implement `minmax` using recursive calls. It returns a tuple consisting of the (minimum, maximum) of the elements of a sequence. We have started the recursive formulation for you. Complete it. Your solution should not do redundant work.

```
def minmax(s):
    """Return a tuple of the (min,max) of the elements in s.

    >>> minmax([1,-2, 3])
    (-2, 3)
    """
    if len(s) == 1:
        return (s[0],s[0])
    else:
        mn,mx = minmax(s[1:])
        return (min(s[0],mn), max(s[0],mx))
```

(d) (3 pt) Implement the function `scale_maker` which takes a input $scale_factor$ and returns a function which scales (multiplies) its input by $scale_factor$.

Use this to implement the function `sclaler`, which returns elements of s scaled by $scale_factor$.

```
def scale_maker(scale_factor):
    def fun(x):
        return x*scale_factor
    return fun

def scaler(s, scale_factor):
    """Scale each element of a by scale_factor using scale_maker.

    >>> list(scaler([1,2,3,5], 2))
    [2, 4, 6, 10]
    """
    return map(scale_maker(scale_factor), s)
```

4. (6 points) Match as you can

Implement the function `fun_match` which takes a unary function over elements in the sequence and returns the pairs of elements in the sequence such that the first not equal to the second, yet the function maps them to the same value.

```
def fun_match(f, s):
    """Return list of pairs (x,y) where x and y are in s, they are not equal but f ma

    >>> fun_match(abs, [-2, -1, 0, 1, -2])
    [(-1, 1)]
    >>> fun_match(lambda x: 3, [-2, -1, 0, 1])
    [(-2, -1), (-2, 0), (-2, 1), (-1, 0), (-1, 1), (0, 1)]

    """
    return [(x,y) for x in s for y in s if x != y and f(x) == f(y)]

# alternative solution

def fun_match(f,s):
    pairs = []
    for x in s:
        for y in s:
            if x != y and f(x) == f(y):
                pairs += [(x,y)]
    return pairs
```