CS 88
Spring 2016

Computational Structures in Data Science

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. *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, z):` `return x + y * z` `boo(2,3,4)`	14
`def hoo(x,y):` `while x > y:` `y = y*2` `return y` `hoo(10,3)`	12
`def f(ele, seq, ind):` `return ele/2` `listfun(range(5), f)`	[0.0, 0.5, 1.0, 1.5, 2.0]
`def f(ele, seq, ind):` `return ele+ind` `listfun(range(5),f)`	[0, 2, 4, 6, 8]
`def f(ele, seq, ind):` `return ele%2 or True` `listfun([1,2,3,4], f)`	[1, True, 1, True]

Expression	**Interactive Output**

`def f(ele, seq, ind):`
`    return seq[ind+1-ind%2]`
 	
`listfun([3, 4, 6, 7], f)`
	[4, 4, 7, 7]

`def f(ele, seq, ind):`
`    p = 0`
`    for x in seq[ind:]:`
`        p = p+x`
`    return p`
 	
`listfun([1,2,3,4,5], f)`
	[15, 14, 12, 9, 5]

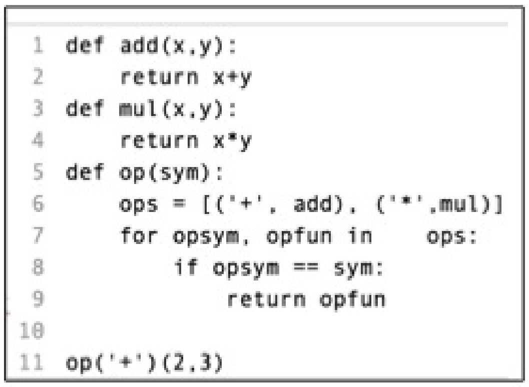
`def f():`
`    def g(e,s,i):`
`        return (i,e+1)`
`    return g`
 	
`listfun([1,2,3,4,5], f())`
	[(0, 2), (1, 3), (2, 4), (3, 5), (4, 6)]

2. (8 points) Environmental Policy

The environment diagram below reflects the state of execution indicated by the arrow, i.e., a call to *op* has been made and the first assignment statement executed. Fill in the diagram to show the 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.*

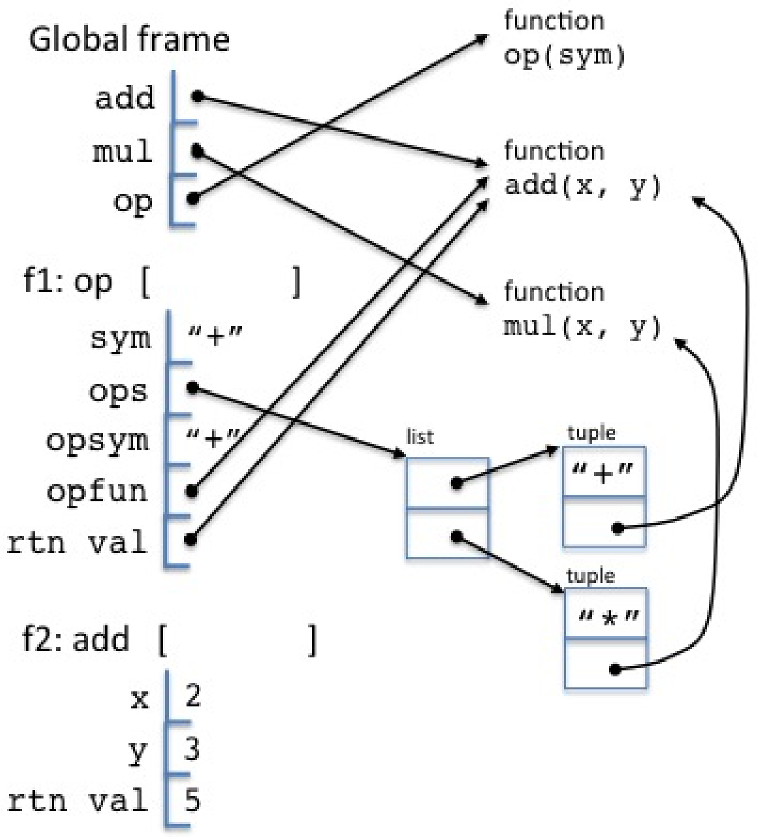
A complete answer will:

- 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 add(x,y):
    return x+y
def mul(x,y):
    return x*y
def op(sym):
    ops = [('+', add), ('*',mul)]
    for opsym, opfun in    ops:
        if opsym == sym:
            return opfun

op('+')(2,3)
```



3. (12 points) Function Junction

(a) (3 pt) Implement the `leading_chars` function, which takes a string `s` and character `ch`. It returns the number of leading `ch` characters in the string, i.e., at the start of the string. We have started the implementation for you. It should use a `while` loop.

```
def lead_chars(s, ch):
    """Return the number of leading ch characters in string s.

    >>> lead_chars("abc", 'z')
    0
>>> lead_chars("110", '1')
    2
>>> lead_chars("", 'z')
    0
    """
    lead_chs = 0


    while s:


        if s[0] == ch:


            lead_chs = lead_chs + 1


            s = s[1:]



        else:


            return lead_chs


    return lead_chs
```

(b) **(3 pt)** Implement `mean_fun`, which computes the mean of the result of applying a function `f` to the integer values 1 through n. Recall the definition of mean of a sequence is obtained by taking the sum of the elements divided by the number of elements.

```
def identity(x):
    return x

def square(x):
    return x*x

def mean_fun(f, n):
    """Return the mean of f(1) ... f(n).

    >>> mean_fun(identity, 5)
    3.0
    >>> mean_fun(square, 5)
    11.0
    """
    psum = 0
    for i in range(1,n+1):
        psum = psum + f(i)
    return psum/n

#









# Keep down here
```

(c) **(3 pt)** Implement `binary` using recursive calls. It returns the list of binary strings, i.e., formed of the digits '0' and '1' of length n for positive number n. There are 2^n of these! We have started the recursive formulation for you. Complete it. Your solution should not do redundant work. (2^n is enough!)

```
def binary(n):
    """Return the binary strings of length n > 0.

    >>> binary(3)
    ['000', '001', '010', '011', '100', '101', '110', '111']
    """
    if n == 1:
        return ["0","1"]
    else:


        lsd = binary(n-1)








        return ["0"+d for d in lsd] + ["1"+d for d in lsd]
```

(d) (3 pt) Implement the function `finder_maker` which takes a input *val* and returns a predicate function which determines if its input is equal to *val*. Use this to implement the function `find`, which returns the indeces of occurences of *val* in *s*.

```
def finder_maker(val):
    """Return a function that determines if its input is equal to val.
    >>> finder_maker("hi")("hello")
    False
    >>> finder_maker("hi")("hi")
    True
    """
    def find(x):
        return x == val
    return find


def find(s, val):
    """Return the list of indeces of occurences of val in s.

    >>> find([0,1,2,3,1,2,4,1], 1)
    [1, 4, 7]
    """
    pred = finder_maker(val)
    return [i for i in range(len(s)) if pred(s[i])]

# Alternative solution
def find(s, val):
    """Return the list of indeces of occurences of val in s.

    >>> find([0,1,2,3,1,2,4,1], 1)
    [1, 4, 7]
    """
    pred = finder_maker(val)
    found = []
    for i in range(len(s)):
        if pred(s[i])]:
            found = found + [i]
    return found




# Keep to bottom of page
```

4. (6 points) Where oh where

Implement the function `pairs_where` which takes a sequence and a binary function over elements in the sequence and returns the pairs of elements in the sequence for which the function evaluates to True.

```
def divides(x,y):
    """Return whether x divides y."""
    return y%x == 0

def segment_intersect(s1, s2):
    """Return whether two segments, each a (start, length) tuple, intersect.

    >>> segment_intersect((1,4),(2,5))
    True
    >>> segment_intersect((1,4),(2,1))
    True
    >>> segment_intersect((3,1),(1,1))
    False
    """
    if s1[0] > s2[0]:
        return segment_intersect(s2, s1)
    return s1[0] + s1[1] >= s2[0]

# Solution that fits the docstring
def pairs_where(s, f):
    """Return list of pairs of distinct elements x in s and y in s such that f(x,y) i

    >>> pairs_where([2,3,4,5,6,7,8],divides)
    [(2, 4), (2, 6), (2, 8), (3, 6), (4, 8)]
    >>> pairs_where([(1,3),(2,3),(5,1)],segment_intersect)
    [((1, 3), (2, 3)), ((2, 3), (5, 1))]
    """
    pairs = []
    for i in range(len(s)-1):
        for j in range(i+1, len(s)):
            if f(s[i], s[j]):
                pairs = pairs + [(s[i], s[j])]
    return pairs

# Solution that fits the desription and corrected doc string
def pairs_where(s, f):
    """Return list of pairs of distinct elements x in s and y in s such that f(x,y) i

    >>> pairs_where([2,3,4,5,6,7,8],divides)
    [(2, 4), (2, 6), (2, 8), (3, 6), (4, 8)]
    >>> pairs_where([(1,3),(2,3),(5,1)],segment_intersect)
    [((1, 3), (2, 3)), ((2, 3), (1, 3)), ((2, 3), (5, 1)), ((5, 1), (2, 3))]
    """
    pairs = []
    for i in range(len(s)):
        for j in range(len(s)):
            if i != j and f(s[i], s[j]):
                pairs = pairs + [(s[i], s[j])]
    return pairs
```