CS 88 Fall 2018

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. Check that you have 5 double-sided pages (including cover) for 4 problems. Put your name on every page.

Last name	
First name	
Student ID number	
Berkeley 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. (20 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:

```
x = 2
y = 3
z = "hello"

def hofun(fun, seq):
    return [fun(seq, s) for s in seq]
```

Expression	**Interactive Output**

`def f(s, i):`
`  return s`
`hofun(f, [1, 2])`
	[[1, 2], [1, 2]]

`x**y`	8
`y / x`	1.5
`x % y`	2
`z if x // y else z*x`	'hellohello'
`len(z) > y and x`	2

`def fooz(x):`
`  x = x*x`
`  return x + y, x`
`a,b = fooz(y)`
`a`
	12

`def mooz(a, b, c ):`
`  return [i or a for i in range(b,c)]`
`mooz("a", 0, 3)`
	['a', 1, 2]

`def nooz(a, b, c):`
`  for x in range(c):`
`  a += b`
`  return a`
`nooz(1, y, 2)`
	7

`def zooz(s, i):`
`  if s[0] > i:`
`  return s`
`  else:`
`  return [i] + s`
`zooz([2, 3, 8, 1], 4)`
	[4, 2, 3, 8, 1]

`def f(s, i):`
`  return s[0]+i`
`hofun(f, [1, 3, 2])`
	[2, 4, 3]

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

`def f(s, i):`
`  for j in range(len(s)):`
`  if i == s[j]:`
`  return j`
`  return -1`
`hofun(f, [1, 3, 1, 2])`
	[0, 1, 0, 3]

2. (10 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:

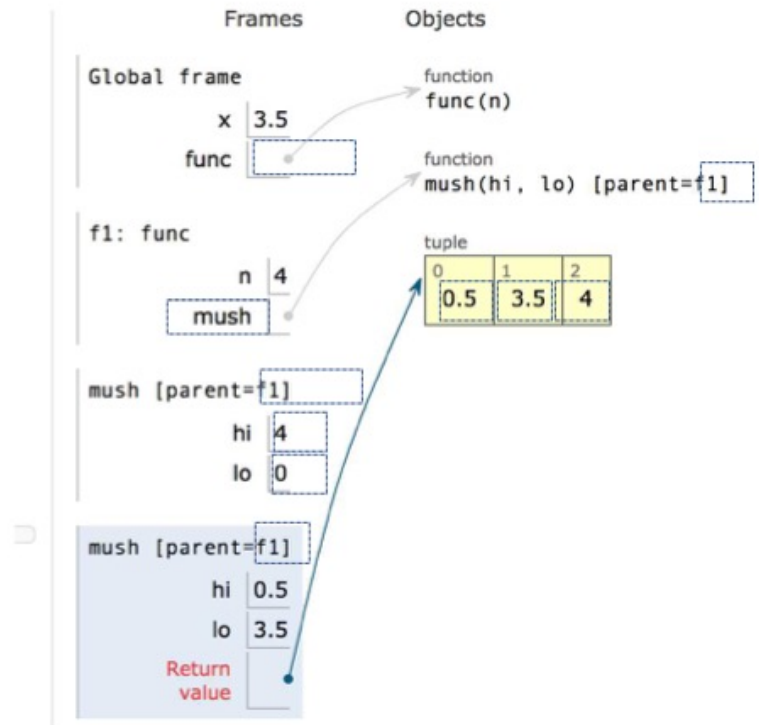
- Add all missing names and parent annotations to all local frames and objects.
- Add all missing values created or referenced during execution.

To help you, these are indicated by dashed boxes.

Python 3.6

```
1  x = 3.5
2  def func(n):
3      def mush(hi, lo):
4          if hi > lo:
5              return mush(hi-x, lo+x)
6          else:
7              return hi, lo, n
8      return mush(n, 0)
9  func(4)
```

Edit this code



3. (18 points) Letter fun

(a) (2 pt) Implement the `match` function, which takes two characters and returns 1 if they match and 0 otherwise

```
def match(a, b):
    """Detect a match (a==b).  Return 1 on match, otherwise 0

    >>> match('a', 'a')
    1
    >>> match('a', 'b')
    0
    """
    return 1 if a == b else 0
```

(b) (4 pt) Implement the `count_letter` function, which takes a string `s` and character `c`; it returns the number of times that the character appears in the string. *You must use iteration. You may not use recursive calls.*

```
def count_letter(s, c):
    """ Returns the number of times that character c occurs in string s

    >>> count_letter("banana", 'a')
    3
    >>> count_letter("banana", 'z')
    0
    """
    count = 0
    for ch in s:
        if ch == c:
            count += 1
    return count
```

(c) (4 pt) Re-implement the `count_letter` function recursively using the `match` function above.
You must use recursive calls. You may not use iteration. You must call ***`match`****.*

```
def count_letter(s, c):
    """ Returns the number of times that character c occurs in string s

    >>> count_letter2("banana", 'a')
    3
    >>> count_letter2("banana", 'z')
    0
    """
    if not s:
        return 0
    else:
        return match(s[0], c) + count_letter(s[1:], c)
```

(d) (4 pt) Implement the `match_maker` function, which takes a character `c` and returns a function that takes a single character as input and returns 1 if it matches `c` and 0 otherwise.

```
def match_maker(c):
    def match(x):
        return 1 if x == c else 0
    return match
```

(e) (4 pt) Re-implement the `count_letter` function using higher order function. You may use the `match_maker` function define above. You may further assume:

- `sum(s)`, which returns the sum of a sequence,
- `map(f, s)`, which maps function f over sequence s,
- `filter(f, s, p)`, which maps function f over sequence s filtered by predicate p
- `reduce(f, s, i)`, which reduces s using function f with identity i.

You must use higher order functions. You must call ***`match_maker`****.*

```
def count_letter(s, c):
    """ Returns the number of times that character c occurs in string s

    >>> count_letter3("banana", 'a')
    3
    >>> count_letter3("banana", 'z')
    0
    """

    return sum(map(match_maker(c), s))
```

4. (12 points) Cheap Eats

Implement the `cheapest` function according to the doc string. You may use any form of control you like.

```
def cheapest(price):
    """Returns a function that takes a list of integer food_prices and
    returns a list of the items in food_prices that are less than price.
    >>> cheap_foods = cheapest(50)
    >>> cheap_foods([70, 23, 27, 90])
    [23, 27]
    >>> cheaper_foods = cheapest(20)
    >>> cheaper_foods([21, 26, 90])
    []
    >>> cheapest_foods = cheapest(10)
    >>> cheapest_foods([-8, 2, 2, 4])
    [-8, 2, 2, 4]
    """
    def cheaper(price_list):
        return [p for p in price_list if p < price]
    return cheaper
```