CS 88 Spring 2016 — Computational Structures in Data Science

Final Solutions

INSTRUCTIONS

- You have 3 hours to complete the exam.
- The exam is closed book, closed notes, closed computer, closed calculator, except one 8.5" × 11" crib sheet (2 sided) of your own creation and the official CS 88 final 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. **(12 points) Draw a line in the sand**

   Each of the functions below contain a docstring and a bunch of lines of code. Among of them are at least one sequence of lines that correctly implement the function. You are to cross out, i.e., remove, lines that are not needed in a correct implementation. The remaining lines should implement the function with no extraneous lines.

(a) (3 pt) Iteration.

```
def max(s):
    max_s = s[0]
    for e in s[1:]:
        if e > max_s:
            max_s = e
    return max_s
```

(b) (3 pt) Recursion

```
def minr(s):
    """Return the minimum element of s."""
    if len(s) == 1:
        return s[0]
    else:
        x = minr(s[1:])
        return x if x < s[0] else s[0]
```

(c) **(3 pt)** Higher order functions

```
def winner(fun, sun):
    """Return a function that counts the number of times fun(ch) occurs in sun."""
    def count(ch):
        return sum([1 for c in sun if c == fun(ch)])
    return count
```

(d) **(3 pt)** Warriors got class

```
class NBA_playoffs:
    def __init__(self, teams):
        self.teams = teams

    def team(self, name):
        return self.teams.index(name)

class Division(NBA_playoffs):
    """
    >>> west = Division("west", ["GSW", "HOU", "LAC", "POR", "OKC", "DAL", "SAS", "
    >>> west.team('POR')
    3
    """
    def __init__(self, division, teams):
        self.division = division
        NBA_playoffs.__init__(self, teams)
```

2. (12 points) Rubber baby buggy bumpers

Each problem contains a bug that will cause the function to fail, throw an exception, or hang on certain input. Identify the bug, produce a sample input that exhibits the bug.

Here is an example.

```
def fib(n):
    if n == 0:
        return 1
    else:
        return fib(n-1) + fib(n-2)
```

Bug: Goes into an infinite loop if passed a negative value because base test should be `if n <= 1:`.

Error input: `fib(1)`.

(a) (3 pt) Bug in iteration over a list

```
def index(s, c):
    """Return the index of c in s."""
    ind = 0
    for i in range(len(s)):
        if s[i] == c:
            ind = i
    return ind

Bug: ______________________________________________


____________________________________________________


Error input: _______________________________________


____________________________________________________
```

Bug: returns 0 if s is not in c

(b) (3 pt) Bug in sorting

```
def mergesort(s):
    if len(s) <= 1:
        return s
    middle = len(s)//2
    first = mergesort(s[:middle])
    second = mergesort(s[middle:])

    merged = []
    while len(first) > 0 and len(second) > 0:
        if first[0] < second[0]:
            merged += [first[0]]
            first = first[1:]
        elif first[0] > second[0]:
            merged += [second[0]]
            second = second[1:]
    return merged + first + second
```

```
Bug: ----------------------------------------------

----------------------------------------------------

Error input: ---------------------------------------

----------------------------------------------------
```

Bug: when the values at the front of the two lists are equal we don't move either one to merged.
`mergesort([3, 2, 2, 4, 5, 1]) => inf loop`

(c) **(3 pt)** Bug fun
What does this do?

```
def make_adder(a):
    def adder(a, b):
        return a + b
    return adder
adder(2)(3)
```

```
Result: ---------------------------------------

----------------------------------------------------

Bug: ----------------------------------------------

----------------------------------------------------
```

`adder` should take only b

(d) **(3 pt)** Spot the abstraction violation.
The following provides an implementation of an abstract data type of a tree in which each node contains a value and zero or more sub-trees as its branches. It contains a an abstraction barrier violation. Identify it and fix it by marking up the code.

```
def tree(value, branches=[]):
    return [branches, value]

def root(t):
    return t[1]

def branches(t):
    return t[0]

def count_nodes(t):
    num_child_nodes = 0
    for b in t[0]:
        num_child_nodes += count_nodes(b)
```

```
            return 1 + num_child_nodes
```

```
Violation: ____________________________________________

__________________________________________________

Correction: _____________________________________

__________________________________________________
```

Violation: `t[0]` should be `branches(t)`

3. (5 points) Class of dogs - WWPP

Consider the following class defintion

```
class Dog:
    dogs = []
    def __init__(self, name):
    Dog.dogs += [self]
    self.name = name

    def speak(self):
        return('Bark! My name is ' + self.name + '.')

class Puppy(Dog):
    def __init__(self, name):
        self.puddles = 0
        Dog.__init__(self, name)

    def piddle(self):
        self.puddles += 1

clifford = Dog('Clifford')
fido = Puppy('fido')
```

For each of following, fill in what would Python print.

```
>>> clifford.speak()

________________________
```

'Bark! My name is Clifford.'

```
>>> fido.speak()

_________________________
```

'Bark! My name is fido.'

```
>>> clifford.piddle()
```

```
--------------------------
```

AttributeError 'Dog' object has no attribute 'piddle'

```
>>> map(lambda x:x.name, Dog.dogs)


--------------------------
```

¡map object at 0x...¿

```
>>> list(map(lambda x:x.name, Dog.dogs))


--------------------------
```

['Clifford', 'fido']

4. (8 points) SQL sequel

The questions below refer to the following tables.

```
Courses

| course_id | course_name | sem         |
| 1         | Chem1A      | Fall 2015   |
| 2         | AstroC10    | Fall 2015   |
| 3         | CS88        | Spring 2016 |
| 4         | Math1B      | Spring 2016 |

Grades

| student_id | student_name | c_id | grade |
| 1          | Lucy Jones   | 2    | 95    |
| 2          | John Doe     | 3    | 97    |
| 3          | Jimmy Smith  | 4    | 77    |
| 4          | Carol White  | 3    | 88    |
| 3          | Jimmy Smith  | 1    | 85    |
| 1          | Lucy Jones   | 4    | 90    |
| 2          | John Doe     | 1    | 75    |
```

(a) (3 pt) What would the query return? Write down all output values and column names.

```
SELECT course_name FROM Courses
ORDER BY course_name;


---------------------------

---------------------------

---------------------------

---------------------------

---------------------------

---------------------------
```

```
---------------------------
```

`course_name`
AstroC10
Chem1A
CS88
Math1B

(b) **(3 pt)** What would the following query return? Write down all output values and column names.

```
SELECT   c_id, max(grade) as max_grade
FROM           Students
GROUP BY       c_id
ORDER BY       c_id;
```

```
---------------------------

---------------------------

---------------------------

---------------------------

---------------------------

---------------------------

---------------------------
```

c_id max_grade 1 85 2 95 3 97 4 90

(c) **(2 pt)** Draw the box and pointer diagram for the following code.

```
x = [3, 6, 7]
y = list(x)
y[0] = x[0] == 3
z = x[1:]
y.append(z)
```

5. (10 points) Heard you like iterators

Implement an iterator class called `MinSortIterator`The `__init__` method for `MinSortterator` takes in a sequence of integers. A `MinSortIterator` instance represents a sorted sequence of the integers in the initial sequence.

See the doctests for expected behavior. Note that your solution is not allowed to modify the initial sequence.

```
class MinSortIterator:
    """
    >>> list(MinSortIterator([5, 3, 1, 6, 2]))
    [1, 2, 3, 5, 6]
    >>> seq = [5, -3, 19, 33, -5, 0]
    >>> sorted_iter = MinSortIterator(seq)
    >>> next(sorted_iter)
    -5
    >>> next(sorted_iter)
    -3
```

```
        >>> seq
        [5, -3, 19, 33, -5, 0]
        >>> list(sorted_iter)
        [0, 5, 19, 33]
        """

        def __init__(self, sequence):
            self.sequence = sequence[:]

        def __iter__(self):
            return self

        def __next__(self):
            if len(self.sequence) == 0:
                raise StopIteration
            minimum_element = min(self.sequence)
            self.sequence.remove(minimum_element)
            return minimum_element
```

Why is `list( ... )` required in the first doctest? What would happen without it?

6. (12 points) Data Science Tables

In data8 you have use the Table abstraction in a zillion different ways. In cs88, you have learned enough to build Tables yourself - again. It is an example of a container class. Below is a skeleton implementation of a subset of tables using the ADT design pattern. Just like the Tables you have used in data8, it is an ordered collection of named columns. All columns must be the same length. But a column is a sequence, rather than an numpy array. The internal representation is a list of column names and a list of column sequences.

Your job is to fill in the blanks for the incomplete methods. You will want to refer to the doctest for detail on method behavior.

```
from reduce_soln import *
import collections

class Table88:
    """Data science tables an ordered collection of named columns.

    >>> x = Table88().with_columns([('a', [1, 2, 3]), ('b', [4, 5, 6])])
    >>> x['a']
    [1, 2, 3]
    >>> len(x)
    2
    >>> x.num_rows()
    3
    >>> list(x.labels())
    ['a', 'b']
    >>> x['c'] = ['f', 'g', 'h']
    >>> len(x)
    3
    >>> x.row_as_dict(1)['c']
    'g'
    """
    def __init__(self):
        self._columns = []
        self._labels = []
        self._num_rows = 0

    def num_rows(self):
        if self._columns :
            return len(self._columns[0])
        return 0

    def labels(self):
        return self._labels

    def columns(self):
        return self._columns

    def __len__(self):
        return len(self._columns)

    def __setitem__(self, column_label, column_data):
        """Special method to add or set a column to a data vector using indexing."""
        if not isinstance(column_data, collections.Iterable) or isinstance(column_dat
            raise ValueError('Column data must be list')
        if self._columns and len(column_data) != self.num_rows():
            raise ValueError('Column length mismatch')

        if column_label in self.labels():
            self._columns[self.labels().index(column_label)] = column_data
        else:
            self._columns.append(column_data)
            self._labels.append(column_label)

    def __getitem__(self, column_label):
        return self._columns[self.labels().index(column_label)]

    def with_columns(self, columns):
        new_table = Table88()
        for (column_label, column_data) in columns:
            new_table[column_label] = column_data
        return new_table

    def apply(self, fun, column_label):
        return map(fun, self[column_label])

    def select(self, column_labels):
        return Table88().with_columns([(label, self[label]) for label in column_label

    def where(self, row_selector):
        return Table88().with_columns([(label, [self[label][i] for i in range(self.nu
                                       for label in self.labels()])

    def row(self, index):
        return [self[label][index] for label in self.labels()]

    def row_as_dict(self, index):
        return {label:self[label][index] for label in self.labels()}

    def rows(self):
        for i in range(self.num_rows()):
            yield self.row(i)

    def summary(self, stats = [min, mean, max, sum]):
        def wrap(fun, seq):
            try:
                return fun(seq)
            except:
                return "NA"
        stat_table = Table88()
        stat_table['stats'] = [stat.__name__ for stat in stats]
        for label in self.labels():
            stat_table[label] = [wrap(stat, self[label]) for stat in stats]
        return stat_table

    def __repr__(self):
        lines = "Table88:\n"
        if self.columns() :
            lines += "| " + " |".join(self.labels()) + " |"
            for i in range(self.num_rows()):
                lines += "\n| "
                for label in self.labels() :
                    lines += str(self[label][i]) + " | "
        return lines
```