CS 88 Spring 2016 Computational Structures in Data Science

PRACTICE FINAL

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 min(s):
    """Return the minimum of the values in sequence s."""
    min_s = s[0]
    min_s = 0
    min_s = s
    for e in range(len(s)) :
    for e in range(s) :
    for e in range(1,len(s)) :
    for e in s[1:]:
    for e in s:
        if s[e] < min_s:
        if e < min_s:
        if e > min_s:
            min_s = e
            min_s = s[e]
        min_s = e
    return min_s
```

(b) (3 pt) Count characters with recursion

```
def count_chars(ch, str):
    """Count the number of times charact ch appears in str

    >>> count_chars('a', 'aardvark')
    3
    """
    if len(str) == 0:
    if len(str) > 0:
    if not str:
    if len(str) == 1:
        return 0
        return 1
        return ch
        return count_char(ch, str[1:])
    else:
        match = 1 if str[0] == ch else 0
        if str[0] == ch:
            return 1 + count_chars(ch, str[1:] )
        return match + count_chars(ch, str[1:])
        if str[0] != ch:
            return count_chars(ch, str[1:] )
        return count_chars(ch, str[1:])
```

(c) (3 pt) Higher order functions

```
def summer(fun):
    """Return a function that sums function fun applied to elements of a seq.

    >>> summer(lambda x: x+x)([1, 2, 3])
    12
    """
    runsum(x)
    def runsum(x):
        psum = 0
        psum = x
        for e in x:
            psum += e
            psum += fun(e)
        return psum
        return runsum
    return x
    return runsum
```

(d) (3 pt) Class class

```
class Course:
    """Course enrollments class.

    >>> c = Course("cs88")
    >>> c.name
    'cs88'
    >>> c.add_student("David")
    >>> c.add_student("Cathy")
    >>> list(c.students)
    ['David', 'Cathy']
    """
    courses = []
    def __init__(self, course_name):
        Course.name = course_name
        self.name = course_name
        self.students = []
        self.courses.append(self)
        Course.courses.append(self)

    def add_student(self, student_name):
        if student_name not in courses.students
        if student_name not in self.students:
            self.students.append(student_name)

    def students(self):
        for student in self.students:
            yield(student)
            return(student)
        return students
```

2. (11 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 dictionary

```
def key_of_min_value(d):
    """Return the key associated with the minimum value. """
    min_value = 0
    min_key = 0
    for key in d:
        if d[key] < min_value:
            min_value = d[key]
            min_key = key
    return min_key
```

Bug: _____

_____

Error input: _____

_____

(b) (3 pt) Bug in sorting

```
def quicksort(s):
    """Return a sorted copy of sequence s using quicksort. """
    if s == []:
       return s
    pivot = s[0]
    smaller = []
    larger = []
    for x in s[1:]:
        if x < pivot:
            smaller += [x]
        if x > pivot:
            larger += [x]
     return quicksort(smaller) + [pivot] + quicksort(larger)
```

```
Bug: ____________________________________________

_____________________________________________________

Error input: ____________________________________

_____________________________________________________
```

(c) **(2 pt)** Bug fun

```
def make_adder(a):
    """Return a function that adds a to its argument."""
    def adder(b):
        return a + b
    return adder(a)


Bug: ____________________________________________

_____________________________________________________

Error input: ____________________________________

_____________________________________________________
```

(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 bug and an abstraction barrier violation. Identify both and fix them by marking up the code.

```
def tree(value, branches=[]):
    return {1: value, 2: branches}

def value(t):
    return t[1]

def branches(t):
    return t[2]

def is_heap(t):
            A heap is a tree where every value is larger than or equal to
    the value of its parent. The root of a heap is the smallest value
    in the tree.
```

```
    val = t[1]
    for b in branches(t):
        if b < val or !is_heap(b):
             return False
         return True
```

Bug: _____

Violation: _____

Correction: _____

---

3. (7 points) Class of dogs - WWPP

Consider the following class defintion

```
class Dog:
    count = 0

    def __init__(self, name):
        Dog.count += 1
        self.name = name
        self.sticks_fetched = 0

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

    def fetch(self):
        print(``OK!'')
        self.sticks_fetched += 1

class Puppy(Dog):

    def __init__(self, name):
        Dog.__init__(self, name)

    def fetch(self):
        print(self.name + `` can't fetch!'')


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

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

```
>>> Dog.count


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

>>> clifford.speak()


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

>>> fido.speak()


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

>>> clifford.fetch()


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

>>> fido.fetch()
```

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

>>> Dog.fetch(fido)


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

>>> fido.sticks_fetched


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

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

```
SELECT course_name
FROM Courses;
```

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

```
SELECT course_name, student_name, grade
FROM Courses, Students
WHERE course_id = c_id
AND course_name =   CS88  ;
```

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

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

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

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

(c) **(3 pt)** Fill in the blanks below for the SQL query that returns course name and number of students in each course.

```
SELECT ___________________________

FROM ____________________________

WHERE ____________________________

GROUP BY ________________________

ORDER BY course_name;
```

5. (10 points) Heard you like iterators

Implement an iterator class called `IteratorsIteratorThe` `__init__` method for `IteratorsIterator` takes in a sequence of iterables and constrators an iterator that will iterate over each of these iterables, i.e., stitch them all together into one iterable. An `IteratorsIterator` instance represents a sequence of the values in the iterables.

```
class IteratorsIterator:
    """
    >>> list(IteratorsIterator([ [1, 3, 5], [2, 4, 6], [88], [88] ]))
    [1, 3, 5, 2, 4, 6, 88, 88]

    >>> lotso_iterables = [ [1, 2], ['h', 'i'], ['c', 's', 8, '8'] ]
    >>> s = IteratorsIterator(lotso_iterables)
    >>> next(s)
    1
    >>> next(s)
    2
    >>> list(s)
    ['h', 'i', 'c', 's', 8, '8']
    """

    def __init__(self, iterables):
        self.iterators = [iter(iterable) for iterable in iterables]

    def __iter__(self):

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

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

    def __next__(self):

        if ----------------------------------

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

        try:

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

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


        except StopIteration:

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

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

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

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. 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 dictionary with the column name as the key in each item and the data sequence being the value associated with that key. We have used *collections.OrderedDict*, rather than *dict* because it keeps the items ordered according to how they were inserted. (You can ignore this and just think of it as a dictionary; it does not change any line of code.)

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.

```
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
    >>> list(x.rows())
    [[1, 4, 'f'], [2, 5, 'g'], [3, 6, 'h']]
    >>> x.summary()['b']
    [4, 5.0, 6, 15]
    """
    def __init__(self):
        """Initialize a table.  Do not change this."""
        self._columns = collections.OrderedDict()
        self._num_rows = 0


    def num_rows(self):

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


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


    def labels(self):

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


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

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

    def columns(self):

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

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


    def __len__(self):

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

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

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


    def __setitem__(self, column_label, column_data):
        """Special method to add or set a column to a data sequence
        using indexing.

        Raises a ValueError if the column_data is not a list of the
        same length as other columns.
        """

        if not isinstance(column_data, collections.Iterable) or
               isinstance(column_data, str):

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

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

        if not self._columns :

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

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

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

        else:
```

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

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

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

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

def with_columns(self, columns):

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

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

    for (column_label, column_data) in columns:
        new_table[column_label] = column_data
    return new_table

def apply(self, fun, column_label):

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

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

    return ___________________________________

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

def where(self, row_selector):

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

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

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


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


        return ------------------------------------
                                       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):
        """Return a generator for the rows in the table."""


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


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


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


    def summary(self, stats = [min, mean, max, sum]):
        """Return a table consisting of summary statistics on each
        column of a table with a row for each stats reducer function.
        Where the stats function is invalid, the summary value is 'NA'
        """
        def wrap(fun, seq):


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


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


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


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

```
    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
```