Fall 2018 Final Exam Clarifications

Question 1

Q1.b in Alternate - “\” means the code continues onto the next line

Class Exam question:

- Check must verify that if A is left of B and B is right of A.
- Change the following
  - `for pair in self.pairs` to `for pair in rights:`
  - `for pair in self.pairs()` to `for pair in rights():`

Q3. Each of the three parts is distinct (ie. you may not use your answer from part a for part b)

Q4. Each box is executed *after* the previous one.

Q4. If a box does not output anything, please write No Output.

Q5. Counting Class - “If any of the function evaluations throw **AN** (not and) exception…”

CS 88 Fall 2018 — Computational Structures in Data Science

Final

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 final study guide - attached to the end of the exam.
- Mark your answers **on the exam itself**. We will *not* grade answers written on scratch paper. *** WRITE YOUR NAME ON EVERY PAGE. ***

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	
By my signature, I certify that all the work on this exam is my own, and I will not discuss it with anyone until final session is over. **(please sign)**	

1. (16 points) Toe the line

Each of the functions below contain a docstring and several lines of code. Among them are at least one sequence of lines that correctly implements 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) Mean Iteration.

```
def mean(s):
    """Return the mean of a sequence s of numbers.

    >>> mean([2, 4, 3])
    3.0
    """
    psum = s[0]
    psum = 0
    psum = 1/len(s)
    n = 0
    for i in range(len(s)):
    for psum in s:
    for e in s:
    for e in s[1:]
        psum += e
        psum = e
        n += 1
        if e:
            psum += e
        psum += s[i]
    return psum/n
    return psum/len(s)
    return psum
```

(b) (4 pt) Where recursion

```
def where(s, p):
    """Return a list of the items in iteratable s that satisfy p.

    >>> where([1, 2, 3, 4], lambda x: x % 2)
    [1, 3]
    """
    return where(s[1:], p)
    if s:
    if not s:
        return s[0]
        return []
    elif s:
        return [s[0]] + where(s[1:], p)
        return where(s[1:], p)
    elif p(s):
        return [s + where(s[1:], p)
        return where(s[1:], p)
    else:
        return [s[0]] + where(s[1:], p)
        if p(s[0]):
            return [s[0]] + where(s[1:], p)
            return [s[0]] + where(s, p)
        if s[0]:
            return [s[0]] + where(s[1:], p)
        else:
            return [s[0]] + where(s[1:], p)
            return where(s[1:], p)
            return []
        return where(s[1:], p)
```

(c) (4 pt) Higher order function clamp down

A common data analysis step is to clean data by "clamping" it to a certain reasonable range, replacing outliers by the lower or upper limits of the range of values. Here we use HOF to construct such clamp function with a specified minimum and maximum value for the range.

```
def clamper(minv, maxv):
    """Return a function that clamps the elements of an
    iterator between minv and maxv.

    >>> c = clamper(0, 10)
    >>> c([1, -4, 5, 50])
    [1, 0, 5, 10]

    """
    def clamp(s):
    def clamp(minv, maxv, s):
        return [v if v > minv else minv for v in s]
        ss = [v if v < maxv else maxv for v in s]
        return [v if v > minv else minv for v in
                [v if v < maxv else maxv for v in s]]
        ss = [v if v < maxv else maxv for v in s]
        return [v if (v > minv and v < maxv) else minv
                 if v < minv else maxv for v in s]
        return [v if (v > minv and v < maxv) for v in s]
        return s
        return [v if v > minv else minv for v in s]
        return [v for v in s if v > minv and v < maxv]
    return s
    return clamp
    return clamp(minv, maxv)
    return clamp(minv, maxv, s)
```

(d) **(5 pt)** Class exam

```
class Exam_Seating:
    """Adjacency of seatings for an exam

    >>> x = Exam_Seating("CS88", "Final")
    >>> x.add("Randy", "Mark", None)
    >>> x.add("Joan", "Randy", "Mark")
    >>> x.add("", "Joan", "Randy")
    >>> x.pairs()
    [('Joan', 'Randy'), ('Randy', 'Mark')]
    >>> x.check()
    >>> x.seatings[1]
    {'student': 'Randy', 'right': 'Mark', 'left': 'Joan'}
    """
    def __init__(self, course, exam):
        name = course + exam
        self.name = course + exam
        Exam_Seating.seatings = []
        self.seatings = []

    def add(self, left_name, student, right_name):
        self.seatings += [(left_name, student, right_name)]
        self.seatings += [{'student':student, 'left':left_name,
                           'right':right_name}]
        seatings += [{'student':student, 'left':left_name,
                      'right':right_name}]
        seatings += [(left_name, student, right_name)]
        self.seatings = [{'student':student, 'left':left_name,
                          'right':right_name}]
        seatings += [(left_name, student, right_name)]
        return self.seatings

    def pairs(self):
        return [(s['left'], s['student']) for s in self.seatings if s['left']]
        return sorted([(s['left'], s['student'])
                       for s in self.seatings if s['left']])
        return sorted([(s['left'], s['student']) for s in self.seatings])
        return [(s[0], s[1]) for s in self.seatings if s[0]]
        return sorted([(s[0], s[1])
        for s in self.seatings if s[0]])
        return sorted([(s[0], s[1]) for s in self.seatings])
        return sorted([(s[0], s[1])
        for s in self.seatings if s[1]])

    def check(self):
        rights = [(s['student'], s['right']) for s in self.seatings if s['right']]
        rights = [(s[1], s[2]) for s in self.seatings if s[2]]
        for pair in self.pairs():
        for pair in self.pairs:
            assert pair in self.pairs
            assert pair in self.pairs()
            yield pair if pair not in self.pairs
        return [pair for pair in rights
                if pair not in self.pairs()]
```

2. (12 points) CLASSic movies

Consider the following Class definition and object instantiations to answer the sequence of ”What would Python Print” questions. For each, assume that the additional sequence of statements is executed. If the result is an error or object, explain what it would be.

```
class Citizen:
    home = "Tatooine"

    def __init__(self, first, last):
        self.firstname = first
        self.lastname = last

    def name(self):
        return self.firstname + " " + self.lastname + " of " + self.home

    def move(self, dest):
        Citizen.home = dest

class Child(Citizen):
    def __init__(self, first, parent):
        Citizen.__init__(self,first, parent.lastname)
        self.parent = parent

    def name(self):
        return self.firstname + " child of " + self.parent.name()

    def move(self, dest):
        self.parent.move(dest)

    def play(self):
        return self.firstname + " played outside."

shmi = Citizen("Shmi", "Skywalker")
vader = Child("Anakin", shmi)
luke = Child("Luke", vader)
```

Expression	**Interactive Output**
`>>> vader.play()`	'Anakin played outside.'
`>>> vader.home`	
`>>> luke.name()`	

`>>> luke.move("Ahch-To")`
`>>> vader.name()`	

`>>> luke.name`	

`>>> chewy = Citizen("Chew",`
`...                  "Bacca")`
`>>> chewy.home`	

`>>> chewy.play()`	

3. (11 points) Generate Distinctive Iterators

(a) (3 pt) Implement the `distinct` function, which takes an iterable `i` and returns a list containing the distinct elements of it, i.e., like `numpy.unique`

```
def distinct(i):
    """Return list of distinct elements in iterable i.

    >>> distinct([1,2,3, 2])
    [1, 2, 3]

    """
    ____________________________________________________

    ____________________________________________________

    ____________________________________________________

    ____________________________________________________

    ____________________________________________________

    ____________________________________________________

    ____________________________________________________
```

(b) (3 pt) Implement the `distinct_G` generate, which takes an iterable `i` and generates the distinct elements of it.

```
def distinct_G(i):
    """Return a generator of distinct elements in iterable i.

    >>> list(distinct_G([1,2,3,2]))
    [1, 2, 3]

    """
    ____________________________________________________

    ____________________________________________________

    ____________________________________________________

    ____________________________________________________

    ____________________________________________________

    ____________________________________________________

    ____________________________________________________
```

(c) (5 pt) Implement the `DistinctIter` iterator, which takes an iterable `i` and returns an iterator for the distinct elements of it.

```
class DistinctIter:
    """return an iterator of distinct elements in iterable i

    >>> list(DistinctIter([1, 2, 3, 2]))
    [1, 2, 3]

    """
    def __init__(                               ):

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

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

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

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

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

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

    def __iter__(                               ):

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

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

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

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

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

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

    def __next__(                               ):

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

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

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

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

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

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

4. (6 points) Mutants

Answer the sequence of "What would Python Print" questions. For each, assume that the additional sequence of statements is executed. If the result is an error or object, explain what it would be.

```
apple = ['Green', 'Fuji', 'Poison']
old_navy = ['jeans', 'shirts', 'socks']
amazon = ['books', 'toys']

boat = [apple, old_navy, amazon]
```

Expression	**Interactive Output**
`>>> old_navy`	['jeans', 'shirts', 'socks']

`>>> apple[2] = 'gala'`
`>>> boat`	

`>>> ship = boat[:]`
`>>> amazon[1] = 'secret weapon'`
`>>> ship`	

`>>> ship is boat`	
`>>> ship == boat`	

`>>> ikea = ('bed', 'chairs', 'desks')`
`>>> ikea[2] = 'drawers'`	

5. (10 points) Counting Class

Implement the Count class to meet the following specifications. Its objects have a `comparator` method that takes a function `f` and function `g` and argument value `arg` and a comparison function `comp`. It returns whether the results of applying the two functions to the arg satisfy the comparison, i.e., `comp(f(arg), g(arg)`. If any of the function evaluations throw and exception the result is False. In addition, Count objects have a 'exceptions' access method that returns the number of invocations that resulted in an exception. Count has a 'calls' classmethod that returns the number of calls to the `comparator` methods of all objects. It should have a private class attribute for keeping track of calls and a private object attribute for keep track of exceptions.

```
class Count:
    """ Counting class

    >>> ctrA = Count()
    >>> ctrA.comparator(lambda x: 1/x, lambda x: x, 1, lambda x, y: x==y)
    True
    >>> ctrA.comparator(lambda x: 1/x, lambda x: x, 0, lambda x, y: x==y)
    False
    >>> ctrB = Count()
    >>> ctrB.comparator(lambda x: 1/x, lambda x: x-2, 2, lambda x, y: x == 1/y)
    False
    >>> ctrA.exceptions(), ctrB.exceptions(), Count.calls()
    (1, 1, 3)
    """

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

6. (5 points) Short Answer

For each of the following, provide a sentence or two of concise explanation.

(a) (2 pt) Testing sorts

You have just written a new sort function and need to produce a set of test cases to make sure it works. Briefly explain what you would seek to cover in a good test set, i.e, in `sort_test(sort(inp))` what set of inputs `inp` would you use to test your sort function?

Write a small function `sort_test` to test the output of a call to sort.

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

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

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

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

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

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

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

(b) (2 pt) Respecting Abstraction Boundaries

When following an Abstract Data Types methodology, what constitutes an abstraction violation?

How does this guide the design of a Class?

(c) (1 pt) More test thoughts

What are some reasons you might want to write the tests for a function before you even build the function?