CS 88 Fall 2018 Computational Structures in Data Science

FINAL 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 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
    """
    n, psum = 0, 0
    for e in s:
        psum += e
        n += 1
    return psum/n
```

(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]
    """
    if not s:
        return []
    else:
        if p(s[0]):
            return [s[0]] + where(s[1:], p)
        else:
            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.

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

    """
    def clamp(s):
        return [v if v > minv else minv for v in
                [v if v < maxv else maxv for v in s]]
    return clamp

# OR

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

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

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

(d) (5 pt) Class exam

```
class Exam_Seating:
    """Adjacency of seats 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.seats[1]
    {'student': 'Randy', 'right': 'Mark', 'left': 'Joan'}
    """
    def __init__(self, course, exam):
        self.name = course + exam
        self.seats = []

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

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

    def check(self):
        rights = [(s['student'], s['right'])
                  for s in self.seats if s['right']]
        for pair in self.pairs():
            assert pair in rights
```

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`	'Tatooine'
`>>> luke.name()`	'Luke child of Anakin child of Shmi Skywalker of Tatooine'
`>>> luke.move("Ahch-To")` `>>> vader.name()`	'Anakin child of Shmi Skywalker of Ahch-To'
`>>> luke.name`	¡bound method Child.name of ...

`>>> chewy = Citizen("Chew",`
`...  "Bacca")`
`>>> chewy.home`
	'Ahch-To'

`>>> chewy.play()`	AttributeError: 'Citizen' object has no attribute '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]

    """
    items = []
    for ele in i:
        if ele not in items:
            items += [ele]
    return items
```

(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]

    """
    items = []
    for ele in i:
        if ele not in items:
            items += [ele]
            yield ele
```

(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__(self, i):
        self.items = []
        self.it = iter(i)

    def __iter__(self):
        return self

    def __next__(self):
        ele = next(self.it)
        while ele in self.items:
            ele = next(self.it)
        self.items += [ele]
        return ele
```

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`	[['Green', 'Fuji', 'gala'], ['jeans', 'shirts', 'socks'], ['books', 'toys']]

`>>> ship = boat[:]`
`>>> amazon[1] = 'secret weapon'`
`>>> ship`
	[['Green', 'Fuji', 'gala'], ['jeans', 'shirts', 'socks'], ['books', 'secret weapon']]

`>>> ship is boat`	False
`>>> ship == boat`	True
`>>> ikea = ('bed', 'chairs', 'desks')` `>>> ikea[2] = 'drawers'`	'tuple' object does not support item assignment

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)
    """
    _count_ = 0
    @classmethod
    def calls(cls):
        return Count._count_

    def __init__(self):
        self._exceptions_ = 0

    def exceptions(self):
        return self._exceptions_

    def comparator(self, f, g, arg, comp):
        Count._count_ += 1
        try:
            f_res = f(arg)
            g_res = g(arg)
            c_res = comp(f_res, g_res)
            return c_res
        except:
            self._exceptions_ += 1
            return False
```

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?

Length of input: zero, one, two, three, larger.

Order of input: sorted, reverse, random

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

```
def sort_test(res):
    if res:
        prev = res[0]
        for e in res[1:]:
        # Could do assertion or conditional
            assert e >= prev
            prev = e
    return true
```

(b) (2 pt) Respecting Abstraction Boundaries

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

When an operation method accesses the internal representation of the of the object rather than building on selectors (getters) and setters

How does this guide the design of a Class?

The class follows the methodology if it provides creation, accessors (selectors and settors) and operation methods, rather than exposing the instance attributes.

(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?

It provides a specification to write to.

It helps you understand the behavior of the code before you write it. In particular, to think about the different cases that need to be handled.

The tests are based on how you conceive of the class, i.e., what it supposed to do, rather than the code you wrote for the class.