Computer Science 1 — CSci 1100
Fall Semester, 2015
Test 3 Overview and Practice Questions

REMINDERS

- Test 3 will be held **Monday, November 23, 2015** in the late afternoon and evening. The vast majority of you will take the test from 6:00-7:30 pm. You will be assigned to locations randomly. Please check the homework server on Thursday night for your exam location.

  You MUST show up to the correct exam location. We will have limited seating and exam copies in each section.
- Students who have provided Prof. Adalı with an accommodation letter and therefore require extra time will take the test starting at 4:30 pm in Darrin 236. Other students with additional conflicts are notified independently.
- You **MUST BRING YOUR ID** to the exam. Missing ids will cause an immediate 20 point penalty.
- We are sorry for any inconvenience this has caused.

Overview

- Primary coverage is Lectures 13-20, Labs 7-10, HW 6-8. Material from lists is also part of this test. **JSON format will not be on the test. Sorting will not be on the test.**
- You are likely to get set or file questions in this exam as well. Make sure you review the relevant exercise questions from Exam #2 Overview.
- No calculators, no textbook, no classnotes, no electronics of any kind! BUT, you may bring a one-page, double-sided, 8.5" x 11" "crib sheet" sheet with you. You may prepare this as you wish, and you may work in groups to prepare a common crib sheet. Of course, each of you must have your own copy during the test.
- Please refer back to the Test 1 and Test 2 practice problems for further instructions.

Questions

1. Write a function called `notused` that takes a list of words as its single parameter, and returns a set containing the letters of the English alphabet that are not used by any of the words in the input list. Your function must use sets. Here is an example of how your function should work:

   ```
   >>> notused([ "Dog", "pony", "elephant", "Tiger", "onyx", "Zebu" ])
   set(['c', 'f', 'k', 'j', 'm', 'q', 's', 'w', 'v'])
   ```

   Hint: you can use the following set in your solution:

   ```
   all_letters = set("abcdefghijklmnopqrstuvwxyz")
   ```

2. Given three sets, `s1`, `s2`, and `s3`, write a short segment of Python code to find the values that are in **exactly one** of the three sets. The result should be stored in a set called `s`. You may NOT use any loops.

3. Given three strings of words, with each word separated by a space, write code to output the number of words that appear in all three strings. Assume the strings are associated with the variables `w1`, `w2` and `w3`. For

   ```
   w1 = "the quick brown fox jumps over the lazy dog"
   w2 = "hey diddle diddle the cat and the fiddle the cow jumps over the moon"
   w3 = "jack and jill went over the hill to fetch a pail of water"
   ```

The output should be 2 because `the` and `over` appear in all three. **No loops are allowed.** You can solve this in one (long) line of code, although if you use just a few it is fine.

4. What is the output when the following code is run by Python? For sets, do not worry about getting the exact order of the output correct.

   ```
   s1 = set( [7,9, 12, 7, 9] )
   s2 = set( ['abc', 12, 'b', 'car', 7, 10, 12 ])
   s3 = set( [12, 14, 12, 'ab'] )
   print s1 & s2


   print s1 | s2


   print 'b' in s2


   print 'ab' in s2


   print 'ab' in s3

   s2.discard(12)
   print (s1 & s2) ^ s3
   ```

   Of course, you can make up many other questions about set operations. There are also extra questions at the end of the Lecture 13 notes.

5. You are given a dictionary containing reviews of restaurants. Each key is the name of the restaurant. Each item in the dictionary is a list of reviews. Each review is a single string. See the example below.

   ```
   rest_reviews = {"DeFazio's":["Great pizza", "Best in upstate"], \
       "I Love NY Pizza":["Great delivery service"], \
       "Greasy Cheese": [ "Awful stuff", "Everything was terrible" ] }
   ```

   Assuming `rest_reviews` has already been created, solve the following problems.

   (a) Write code to find all restaurants where the review contains at least one of the following words: `awful, terrible, dump`. For each restaurant found, output the name of the restaurant and the number of reviews that have at least one of the words.

   (b) Write code to find and print the name of the restaurant with the highest number of reviews. If there is more than one restaurant with the same number of reviews, print the names of each of these restaurants.

   (c) Write a function that takes as arguments the review dictionary, a new review, and the name of a restaurant. The function should add the review to the dictionary. If the restaurant is already in the dictionary, the function should add the review to the existing list of reviews for that restaurant. If the restaurant is not in the dictionary, the function should add a new item to the dictionary. Your function should be called by:

   ```
   add_review(rest_reviews, new_review, rest_name)
   ```

   (d) Write a function that takes the same arguments as `add_review`, but deletes the given review. Specifically, if the review is in the dictionary associated with the restaurant, the function should delete the review and return `True`. Otherwise, the function should return `False`. If the given restaurant is not in the dictionary, the function should also return `False`. The function should be called by

   ```
   del_review(rest_reviews, old_review, rest_name)
   ```

6. For each of the following sections of code, write the output that Python would generate:

Part a

```
x = {1:['joe',set(['skiing','reading'])],\
     2:['jane',set(['hockey'])]}

x[1][1].add('singing')
x[1][0] = 'kate'

for item in sorted(x.keys()):
    print x[item][0], len(x[item][1])
```

Part b

```
y = {'jane':10, 'alice':2, 'bob':8,\
     'kristin':10}

m = 0
for person in sorted(y.keys()):
    if y[person] > m:
        print "**", person
        m = y[person]

for person in sorted(y.keys()):
    if y[person] == m:
        print "!!", person
```

Part c: Note that this problem requires an understanding of aliasing.

```
L1 = [0,1,2]
L2 = ['a','b']
d = { 5:L1, 8:L2 }
L1[2] = 6
d[8].append('k')
L2[0] = 'car'
for k in sorted(d.keys()):
    print k,
    for v in d[k]:
        print v,
    print
```

Part d:

```
L1 = [0,1,2,4,1, 0]
s1 = set(L1)
L1.pop()
L1.pop()
L1.pop()
L1[0] = 5
s1.add(6)
s1.discard(1)
print L1
for v in sorted(s1):
    print v
```

7. Given a dictionary, `d`, whose keys are strings and whose values are integers, write a (very) short segment of code that finds and outputs the smallest (in lexicographic order) key and the smallest value in the dictionary. For example, if

   ```
   d = {'abc':5, 'def':2, 'ab':10}
   ```

   your code should output:

   ```
   Smallest key: ab
   Smallest value: 2
   ```

8. Suppose `Person` is a class that stores for each person their name, birthday, name of their mother and father. All of these are strings. The start of the class, including the initializer, is given below.

   ```
   class Person(object):
       def __init__( self, n, bd, m, f):
           self.name = n
           self.birthday = bd
           self.mother = m
           self.father = f
   ```

   Write a method for the Person class that takes as an argument `self` and another Person object and returns 2 if the two people are twins, 1 if they are siblings (but not twins), -1 if two people are the same, and 0 otherwise. Note that siblings or twins must have the same mother and the same father.

9. You are given dictionaries `D1` and `D2` where each key is a string representing a name and each value is a set of phone numbers. Write a function to merge `D1` and `D2` into a single dictionary `D`. `D` should contain all the information in both `D1` and `D2`. As an example,

   ```
   >>> D1 = {'Joe':set(['555-1111','555-2222']), 'Jane':set(['555-3333'])}
   >>> D2 = {'Joe':set(['555-2222','555-4444']), 'Kate':set(['555-6666'])}
   >>> merge_dict(D1,D2)
   {'Joe':set(['555-1111','555-2222','555-4444']), 'Jane':set(['555-3333']),\
   ... 'Kate':set(['555-6666']) }
   ```

10. Consider the following definition of a `Rectangle` class:

    ```
    class Rectangle(object):
        def __init__( self, u0, v0, u1, v1 ):
            self.x0 = u0
            self.y0 = v0
            self.x1 = u1
            self.y1 = v1
    ```

    where we can assume `(x0,y0)` is the lower left corner of the rectangle and `(x1,y1)` is the upper right corner of the rectangle. Your problem is to write a `Rectangle` class method called `inside` that takes an `x` and `y` coordinate of a point and returns `True` if `(x,y)` is inside the rectangle and `False` otherwise. For example

    ```
    >>> r1 = Rectangle( 1, 3, 5, 10 )
    >>> r1.inside( 2, 7 )
    True
    >>> r1.inside( 6, 7 )
    False
    ```

11. This question involves a class called `Student` that stores the student's name (a string), id number (a string), courses taken (list of strings), and major (a string). Write the Python code that implements this class, including just the following methods

   (a) An initializer having as parameters only the name and the id. This should initialize the list of courses to empty and the major to `"Undeclared"`. An example use of this method would be

   ```
   >>> p = Student( "Chris Student", "123454321" )
   ```

   (b) A method called `"add_courses"` to add a list of courses to the courses that the student takes. For example, the following should add three courses to `Chris Student`.

   ```
   >>> p.add_courses( [ "CSCI1100", "BASK4010", "GEOL1320" ] )
   ```

   (c) A method called `common_courses` that returns a list continains the courses two students have taken in common:

   ```
   >>> q = Student( "Bilbo Baggins", "545454545" )
   >>> q.add_courses( [ "MATH1240", "CSCI1100", "HIST2010", "BASK4010" ] )
   >>> print q.common_courses(p)
   [ "CSCI1100", "BASK4010" ]
   ```

12. Using the `Student` methods and attributes from the previous question, suppose you are given a list of student objects called `all`. Write a segment of code to output the names of all students who have taken at least two courses that start with `CSCI`.

13. Given a list `L` and a positive integer `k`, the problem under consideration here is to create a new list containing only the `k` smallest values in `L` list. For example, if

   ```
   L = [ 15, 89, 3, 56, 83, 123, 51, 14, 15, 67, 15 ]
   ```

   and `k=4`, then the new list should have the values

   ```
   Ls = [3, 14, 15, 15]
   ```

   (Note that one of the 15 is not here.)

   (a) Write a function, `k_smallest(L,k)`, that returns the desired list. It does this using sorting, but does not change `L`. Do this in 4 lines of code without writing any loops.

   (b) Write a second version of `k_smallest` that does NOT use sorting. This can be viewed as a slight variation on the idea of insertion sort. Starting by writing a function that inserts a value `x` into a list that contains `k` or fewer values, ordered. If there are `k` values already in this list and `x` is greater than the last value, the function should not do anything.

14. What is the output of the following two code segments?

   ```
   # Part A
   dt = { 1: [ 'mom', 'dad'], 'hi': [1, 3, 5 ]}
   print len(dt)
   print dt[1][0]
   dt['hi'].append(3)
   dt[1][0] = 'gran'
   print dt[1]
   ```

   ```
   # Part B
   ```

```
# Remember that pop() removes and returns the last value from the list.
LP = [2, 3, 5, 7]
LC = [4, 6, 8, 9]
nums = dict()
nums['pr'] = LP
nums['co'] = LC[:]
LP[1] = 5
print len(nums['co'])
v = LC.pop()
v = LC.pop()
v = LC.pop()
LC.append(12)
print len(LC)
print len(nums['co'])
v = nums['pr'].pop()
v = nums['pr'].pop()
print nums['pr'][1]
print len(LP)
```

15. Given a list of dictionaries, where each dictionary stores information about a person in the form of attribute (key) / value pairs. For example, here is a list of dictionaries reprsenting four people:

```
people = [ { 'name':'Paul', 'age' : 25, 'weight' : 165 }, \
    { 'height' : 155, 'name' : 'Sue', 'age' : 30, 'weight' : 123 }, \
    { 'weight' : 205, 'name' : 'Sam' }, \
    { 'height' : 156, 'name' : 'Andre', 'age' : 39, 'weight' : 123 } ]
```

Write code that finds and outputs, in alphabetical order, the names of all people whose age is known to be at least 30. You may assume that each dictionary in `people` has a `'name'` key, but not necessarily a `'age'` key. For the example above, the output should be

```
Andre
Sue
```

16. Here is the code from binary search from class, with added print statements.

```
def binary_search( x, L):
   low = 0
   high = len(L)
   while low != high:
       mid = (low+high)/2
       print "low: %d, mid: %d, high: %d" %(low,mid,high)
       if x > L[mid]:
           low = mid+1
       else:
           high = mid
   return low
```

Show the output for the following three calls:

```
# (a)
print binary_search( 11.6, [1, 6, 8.5, 11.6, 15, 25, 32 ] )
```

```
# (b)
print binary_search( 11.6, [1, 6, 8.5, 11.6, 15, 25, 32 ] )
```

```
# (c)
print binary_search( 75, [1, 6, 8.5, 11.6, 15, 25, 32 ] )
```

17. Given a dictionary that associates the names of states with a list of the names of (some of the) cities that appear in it, write a function that creates and returns a new dictionary that associates the name of a city with the list of states that it appears in. Within the function, output the cities that are unique — they appear in only one state. Do this in alphabetical order.

    As an example, if the first dictionary looks like

    ```
    states = {'New Hampshire': ['Concord', 'Hanover'],\
              'Massachusetts': ['Boston', 'Concord', 'Springfield'],\
              'Illinois': ['Chicago', 'Springfield', 'Peoria']}
    ```

    then after the function the new dictionary call `cities` should look like

    ```
    cities = {'Hanover': ['New Hampshire'], 'Chicago': ['Illinois'],\
              'Boston': ['Massachusetts'], 'Peoria': ['Illinois'],\
              'Concord': ['New Hampshire', 'Massachusetts'],\
              'Springfield': ['Massachusetts', 'Illinois']}
    ```

    and the four unique cities output should be

    ```
    Boston
    Chicago
    Hanover
    Peoria
    ```

    Here is the function prototype:

    ```
    def create_cities(states):
    ```

18. Consider the Pig and the Bird classes, shown side-by-side here to save space:

    ```
    class Pig(object):
        def __init__( self, n, x0, y0, r0 ):
            self.name = n
            self.xc = x0
            self.yc = y0
            self.radius = r0
    ```

    ```
    from Pig import *
    class Bird(object):
        def __init__( self, n, x0, y0, r0, dx0, dy0 ):
            self.name = n
            self.x = x0
            self.y = y0
            self.radius = r0
            self.dx = dx0
            self.dy = dy0

        def pops_pigs( self, pig_list ):
            # You have to write this part
    ```

    Write `Bird` method `pops_pigs` that "pops" **all pigs** that intersect the bird. The pigs are `Pig` objects in the list `pig_list`. The function should output the names of the pigs popped, remove all popped pigs from the list of pigs, and return `True` if at least one pig is popped (and `False` otherwise). For example, suppose the names, positions and radii of the pigs in `pig_list` are

```
"Wilbur" position (10,10) radius 2
"Clarence" position (14,10) radius 1
"Porky" position (17,8) radius 1
"Thomas" position (18,3) radius 2
```

and the bird is at location `(12,10.5)` with radius 2. Then both `Wilbur` and `Clarence` should be popped and the list of pigs at the end of the function should only contain Porky and Thomas.

19. Consider the following definition of a `Rectangle` class:

```
class Rectangle(object):
    def __init__( self, u0, v0, u1, v1 ):
        self.x0 = u0      # x0 and y0 form the lower left corner of the rectangle
        self.y0 = v0
        self.x1 = u1      # x1 and y1 form the upper right corner of the rectangle
        self.y1 = v1
        self.points = []  # See part (b)
```

   (a) Write a `Rectangle` class method called `contains` that determines if a location represented by an `x` and a `y` value is inside the rectangle. For example

```
>>> r = Rectangle( 1, 3, 7, 10 )
>>> r.contains( 1, 4)
True
>>> r.contains( 2,11)
False
```

   (b) Suppose there is a class

```
class Point(object):
    def __init__( self, x0, y0, id0 ):
        self.x = x0
        self.y = y0
        self.id = id0
```

   and each `Rectangle` stores a list of `Point` objects whose coordinates are inside the rectangle. Write a `Rectangle` class method called `add_points` that adds a list of `Point` objects to the existing (initially empty) list of `Point` objects stored with the `Rectangle` object. If a point is outside the rectangle's boundary or if a point with the same id is already in the rectangle's point list, the point should be ignored. Otherwise, it should be added to the rectangle's point list. Your method must make use of the `contains` method from part (a).

20. When you wrote your code for HW 8 (past semeter) we allowed you to assume (among other things) that the birds were inside the game rectangle at the start. Our solution, however, included safety checks to make sure that we gave you valid input. Write a function `check_birds` that takes a list of birds as its only argument. The function should **change** the list of birds to remove any bird having any part of its circle on or outside the boundaries of the rectangle. The function should return the name of the birds that were removed. For example, after the following code

```
birds = []
birds.append( Bird('Tweety', 7, 95, 12, 2, 3) )
birds.append( Bird('Sylvestor', 50, 50, 10, -3, 6) )
birds.append( Bird('Daisy', 80, 20, 9, -1, 5) )
print check_birds(birds), len(birds)
```

The output should be

```
[ 'Tweety' ], 2
```

because Tweety has been removed from `birds` for being outside the rectangle.

21. Suppose you have a dictionary to represent your phone contacts. The keys are the names of people and the value associated with each key is the set of all that person's phone numbers, represented as strings (and only three digits for simplicity's sake). Here is an example:

```
contacts = {}
contacts['Bob'] = set(['100', '909'])
contacts['Alice'] = set(['505', '101'])
contacts['Chad'] = set(['999'])
contacts['Danielle'] = set(['123' ,'234', '345'])
```

(a) (**7 points of the total**) Write a function called `add_contact` that adds a phone number to a contact. If the contact does not exist, the function should create the contact. For example, the function call

```
add_contacts( contacts, 'Bob', '108')
```

would add a third number to the contact set for `'Bob'`

(b) Write a function `find_name` that takes the contact dictionary and a phone number and returns the name of the individual having that number. If the number is not in the dictionary, the function should return the string `"Unknown"`. For example, the call

```
find_name( contacts, '234' )
```

should return `'Danielle'`. You may assume each phone number is associated with at most one contact name.

(c) Write a function `reverse_contacts` that creates and returns a new dictionary where the keys are phone numbers and the values are the names of the people having the phone number. You may assume only one person has a given phone number. Using this new dictionary it will be much faster to search for the name associated with a number. For the `contacts` dictionary from the start of the problem the call

```
reverse_contacts( contacts )
```

Should produce the dictionary

```
{'101': 'Alice', '909': 'Bob', '999': 'Chad', '345': 'Danielle', \
 '123': 'Danielle', '234': 'Danielle', '100': 'Bob', '505': 'Alice'}
```

22. Show the output of the following code:

```
places = { 'OR':{'Portland' : set(['Pearl District', 'Alameda']), 'Eugene' : set()},
           'NY':{'Albany' : set(), 'NY' : set(['Chelsea', 'Harlem'])}  }

print places['OR']['Eugene']      # <---
a = []
for place in places:
    a += places[place].keys()
print a                           # <---

for x in a:
    if len(x) < 7:
        print x
        for place in places:
            if x in places[place]:
                print places[place][x]    # <---
```

23. Write a function called `smallest_in_common` that takes two different lists, both in increasing order, and returns the smallest value that appears in both lists. Importantly, you may not use a set, and you may not use a dictionary (these lead to simple code, but does more work than necessary). The most credit will be given for solutions that scan through each list no more than once. For example, if

```
l1 = [ 1, 3, 8, 12, 12, 15, 20 ]
l2 = [ 7, 9, 10, 11, 15, 30, 35 ]
l3 = [ 2, 4, 5, 13, 16, 17, 23, 25 ]
```

then

```
print smallest_in_common(l1,l2)   # should output 15
print smallest_in_common(l1,l3)   # should output None
```

24. Suppose you are given a file named `businesses.txt` in which each line contains the name of a business and its category (a single value), followed by a sequence of review scores, each separated by `|`.

Write a piece of code that reads this file, and prints the names of all businesses, their categories, and the average review score for each business. Also print the total number of unique categories in this file. For example, for the file below:

```
Dinosaur Bar-B-Que|BBQ|5|4|4|4|5|5|4|2
DeFazio's Pizzeria|Pizza|5|5|5|5|5|5|5|5|5|5|3|5|5|5
I Love NY Pizza|Pizza|4|5|5|3
```

Your program should print:

```
Dinosaur Bar-B-Que (BBQ): Score 4.125
DeFazio's Pizzeria (Pizza): Score 4.857
I Love NY Pizza (Pizza): Score 4.250
2 categories found.
```

25. Write a function that takes as input a file name, reads the file line by line and prints all the lines that follow one or more blank lines. A blank line is a line that has nothing but spaces and a new line. You should also print the first line in the file. For example, given the file called `av.txt` with the following content:

```
Avengers Assemble!
Some assembly required.
Every team needs a Captain.
Throw down the hammer.
```

```
Hulk, SMASH!

Humans.
They are not the cowering wretches we were promised.
They stand. They are unruly, and therefore cannot be ruled.
To challenge them is to court death.
```

`first_lines('av.txt')` should print:

```
Avengers Assemble!
Hulk, SMASH!
Humans.
```

26. Write a function that takes as input a list of numbers and generates a histogram. The histogram prints a star (*) for each occurrence of a number in the list. For example, if the list was:

    `numbers = [5, 4, 1, 1, 3, 1, 2, 2, 4, 1]`

    your function should print (sorted by the values):

    ```
    1: ****
    2: **
    3: *
    4: **
    5: *
    ```

    (a) Write the function using a dictionary. You may not use a set.

    (b) Write the same function using a set. You may not use a dictionary (hint: use `count` for the unique items in the list).

27. You are given three sentences in three strings `str1,str2,str3`. Assume the sentences contain no punctuation and have a single space between the words. Write a piece of code that prints the following:

    (a) The set of words that appear in `str1` but not in `str2` and not in `str3`.

    (b) The set of words that appear in all three sentences.

    (c) The set of words that appear in exactly two of the three sentences (hint. if you found words in `str1` and `str2`, but not in `str3`, that would be a third of the job!).

    Remember, words are not case sensitive. For example, if you are given the sentences:

    ```
    str1 = "Hail dear old Rensselaer the college of our heart"
    str2 = "For dear old Rensselaer we all must do our part"
    str3 = "True to old Rensselaer we will always strive to be"
    ```

    Your code should print (order of the words is not important):

    ```
    a) set(['heart', 'the', 'college', 'hail', 'of'])
    b) set(['old', 'rensselaer'])
    c) set(['dear', 'our', 'we'])
    ```

28. You are given a list of RPI Alumni as shown below. Each item in the list is a dictionary containing information about an alumnus and all items have the same keys. Write a piece of code that prints the name and addresses of each person who graduated before 2013. For example, given the list:

```
alums = [{'fname':'Abed', 'lname':'Nadir', 'graduated':2012, \
          'addresses':['Troy&Abed apt.','Abed&Annie apt.']}, \
         {'fname':'Troy', 'lname':'Barnes', 'graduated':2013, 'addresses':['Troy&Abed apt.']}, \
         {'fname':'Britta', 'lname':'Perry', 'graduated':2012, 'addresses':['1 Revolution lane']} ]
```

Your code should print (all information is printed in the order it appears in the list):

```
Abed Nadir
    Troy&Abed apt.
    Abed&Annie apt.
Britta Perry
    1 Revolution lane
```

29. You are writing a class called `Date` which stores the month, day and year of a date as integers. The `__init__` function for this class is given below.

    In your class, assume that a month is always 30 days. Write member functions for the following:

    (a) Write the `__str__` function that prints the day as `MM/DD/YYY`.

    (b) Write a function `__sub__` (for subtracting dates) that returns the number of days between two dates (not including the two input days). Your function should not modify the existing date objects and it should return an integer.

    (c) Use your methods to create date objects and print the number of days between today (4/21/2014) and the end of the semester (5/16/2014). The output of your code should be:

    ```
    24 days left between 4/21/2014 and 5/16/2014.
    ```

    Do not hard code values, use the above functions you created.

```
class Date(object):
    def __init__(self, mon, day, year):
        self.day = day
        self.mon = mon
        self.year = year
```

30. You are given the following function.

```
def trinary_search(L,x):
    low = 0
    high = len(L)-1
    while low != high:
        mid0 = low + (high-low)/3
        mid1 = low + 2*(high-low)/3
        print "(%d [%d,%d] %d)" %(low, mid0, mid1, high)
        if x > L[mid1]:
            low = mid1+1
        elif x > L[mid0]:
            low = mid0+1
            high = mid1
        else:
            high = mid0

    if x != L[low]:
        return None
    return low
```

Write the output of the following calls for this function:

`x = trinary_search([1,3,5,7,9,11], 3)`
`print "Call 1 returned:", x`
	`x = trinary_search([1,3,5,7,9,11], 12)`
	`print "Call 2 returned:", x`