Computer Science 1 — CSci 1100
Test 3 Overview and Practice Questions
Fall Semester, 2019

Important Logistical Instructions:

- Test 3 will be held **Thursday, November 21, 2019**.
- Most students will take the exam from 6:00 - 7:30 pm (90 minutes). Room assignments will be posted on Submitty by Tuesday night, November 19. For most students these assignments **will be different** from previous exams.
- Students who provided Professors Turner or Mushtaque with an accommodation letter indicating the need for extra time or a quiet location will be given extra time beyond 7:30. Shianne Hulbert will send you a reminder for your time and location. Use whatever information she sends you. It overrides any assignments given you on Submitty. If you show up at your Submitty location or time, you will be allowed to take the exam, but you will lose the accommodations.
- Students MUST:
  - **Go to their assigned rooms AND sit in their assigned sections.**
  - **Bring their IDs to the exam.**

  Failing to do one of these may result in up to a **20 point** penalty on the exam score. Failure to do both may cost up to **40 points**.

Overview

- Primary coverage is Lectures 15-19, Labs 7-10, HW 5-7.
- You do not need to know the intricacies of tkinter GUI formatting, but you should understand the GUI code structure we outlined, be able to trace through event driven code and write small methods that are invoked by the GUI. Consider the lecture exercises for Lecture 22 and the modifications you made to the BallDraw class during Lab 11 as types of questions you might need to answer.
- Please review lecture notes, class exercises, labs, homework, practice programs, and tests, working through problems on your own before looking at the solutions.
- Some problems will be related to material covered in Exam 2:
  - Lists and files
  - List and string splitting; ranges

  You should review relevant material from the Exam 2 practice problems if you are not yet comfortable with these topics.
- 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. You will need to turn in a signed copy of your crib sheet at the end of the test. If you need to keep a copy for your records, please photocopy it prior to coming to the exam.
- Please refer back to the Test 1 and Test 2 practice problems for further instructions.

Questions

1. 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 or ifs.

2. 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. In this case, it is acceptable to use more than one line, but make sure you understand the one-line solution when you see it.

3. 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.

4. 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. Be careful about capitalization. `'Awful'` and `'awful'` should match.

   (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)
```

5. 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(str(k) + ' ', end='')
    for v in d[k]:
        print(str(v) + ' ', end='')
    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)
```

6. 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.

7. 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']) }
```

8. 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 containing 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" ]
   ```

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

10. Given a list `L` and a positive integer `k`, 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 15s is not here.)

    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.

11. 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))
    ```

12. Given is 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 representing 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
```

13. 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: {} mid: {}, high: {}".format(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( 0, [1, 6, 8.5, 11.6, 15, 25, 32 ] ))



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

14. 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):
```

15. 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. Note, that for this example, on the boundary counts as in 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 second 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).

16. 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])    # <---
```

17. 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|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.
    ```

18. 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).

19. 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
    ```

20. 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("({} [{},{}] {})".format(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)`

21. Write a function called `get_line(fname,lno,start,end)` that takes as input a file name `fname`, a line number `lno`, and starting and end points (`start,end`) on the given line. The function should return the string containing all the characters from the starting point up to but not including the end point on that line (same as it would be with string slicing!).

    Line numbers start at 1; characters in a line are counted starting with zero and include the new line character at the end. If there are fewer than `lno` lines, your function should return None. If the line at `lno` has fewer than `end` characters, return an empty string.

    Given the following contents of file `hpss.txt`:

    ```
    Nearly ten years had passed since the Dursleys had
    woken up to find their nephew on the front step.

    Privet Drive had hardly changed at
    all.
    ```

    The following program:

    ```
    print('1:', get_line('hpss.txt', 2, 9, 15))
    print('2:', get_line('hpss.txt', 5, 5, 9))
    print('3:', get_line('hpss.txt', 5, 0, 4))
    print('4:', get_line('hpss.txt', 8, 0, 10))
    ```

    should print (notice for 3, the newline is also included in the returned string):

    ```
    1: to fin
    2:
    3: all.
    4: None
    ```

22. Suppose you are given the mean temperature for Troy in the month of December in a dictionary `temp` as shown below. The keys of the dictionary are years and the values are the mean temperature for that year. Write a piece of code that finds and prints the top three coldest years according to this dictionary. Note: If there are ties in values, any ordering of years is acceptable.

    For example, given the dictionary below:

    ```
    temp = { 2001: 36.4, 2002: 27.4, 2003: 29.3, 2004: 28.6, 2005: 27.8,
             2006: 37.3, 2007: 28.1, 2008: 30.2, 2010: 26.0, 2011: 35.4,
             2012: 33.8, 2013: 27.9, 2014: 32.8}
    ```

    Your program should output:

    ```
    2010: 26.0
    2002: 27.4
    2005: 27.8
    ```

23. Suppose you are given three variables `t1,t2,t3` in your program. Each variable is a set containing the menu for a different Thanksgiving dinner you are invited to.

    First, print items that are in the menu for all three dinners. Then, print the items that are in the menu for exactly one dinner. All items should be listed in alphabetical order.

    For example, if you are given menus:

    ```
    t1 = set(['Turkey', 'Potatoes', 'Green Beans', 'Cranberry', 'Gravy'])
    t2 = set(['Turkey', 'Yams', 'Stuffing', 'Cranberry', 'Marshmallows'])
    t3 = set(['Turkey', 'Gravy', 'Yams', 'Green Beans', 'Cranberry', 'Turducken'])
    ```

    your program must print the following (your output should match ours):

    ```
    Items in all three dinners: Cranberry, Turkey
    Items in exactly one dinner: Marshmallows, Potatoes, Stuffing, Turducken
    ```

24. What are the running times of the following algorithms:

    (a) Linear search through a list.
    (b) Binary search through a list.
    (c) insertion sort
    (d) merge sort
    (e) python sort
    (f) combination sort a list then binary seach to find a value
    (g) membership test in a list (list.index(value))
    (h) membership test in a set (value in set)
    (i) nested for loops over an entire list