Computer Science 1 — CSci 1100
Exam 2 Overview and Practice Questions
Spring Semester, 2016

REMINDERS

- Test 2 will be held **Monday, March 28, 2016** in the late afternoon and evening. The vast majority of you will take the test from 6:00-7:30 pm in Darrin 308 and 318. In particular, students in Jeramey Tyler's lab sections — 1 and 3 — with be in Darrin 318 and all other students will be in Darrin 308.

  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. Stewart with an accommodation letter and who 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 25 point penalty.

Overview

- Primary coverage is Lectures 6-13, Labs 3-6, HW 3-5. Material from tuples and modules are also part of this test. Images will not be covered.

- Some important topics to keep in mind: lists and tuples are very important for this exam. Make sure you are very comfortable with all the different list functions (append, insert, remove, concatenation, replication and slicing), and the differences between tuples, strings and lists (they are indexed similarly, but assignment creates copies for every object type we have seen so far except for lists and images. Study how to pass lists as arguments to functions (pass by reference). We will also ask elaborate questions on the use of if statements, while and for loops, ranges, splitting and slicing. Make sure you know how to do basic operations on lists using loops such as: compare items from two separate lists, find specific items in a list, find the index of a specific item in a list (min, max, last value with some property), add up values in a list. Learn to write functions over lists such as check if the list contains a specific type of item, return true if so and false otherwise. Know booleans, comparators and the boolean functions **and**, **or** and **not**. Know how to generate truth tables for boolean expressions. These are examples of the type of problems we have solved and you should become very comfortable with them before the test. They are by no means the full list of possible questions.

- 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.

- Below are **many** sample questions, far more than will be on the test. Solutions to most of the problems will be posted on-line on Friday, March 25. These posted solutions will **not** include problems involving output from Python — test these out yourself!

- Please note that your solution to any question that requires you to write Python code will be at most 10-12 lines long, and may be much shorter.

- The test questions will be closely related to these practice problems **AND** to problems from the homework, the labs, and the lecture exercises.

- We have not posted the actual exams from previous semesters because the material and coverage have changed somewhat. Relevant questions have been added here.

- *How to study?*
  - First, make sure you know the syntax of all the different programming constructs we learned. You cannot construct solutions if you do not know the building blocks. Practice them like you would a foreign language. Memorize them. The best approach is to keep typing them into the Python shell and trying new variations. This will also help when you are trying to track down syntax errors.
  - Review all the code from lectures. These have been posted online: Class modules
  - Review and re-do lecture exercises, lab and homework problems. Solve exercises we have not done in class. These are often a good way to test a specific topic.
  - Work through the sample questions writing out solutions! Read solutions only as a last resort. Remember: reading a solution is **much easier** than actually writing one. You are graded on the writing part.
  - You are encouraged to work with other students, but ask yourself if you understand a problem well enough to solve it on your own. Try to replicate a solution you worked on with someone a little while later, without looking at any solutions.
  - Identify the problems that cause you difficulty and review lecture notes and background reading on these topic areas. Go to office hours and ask to review any concepts you did not understand.
  - Use the Wing IDE / Python interpreter extensively to help you understand what is happening, but practice writing solutions using pencil and paper!
- Advanced warnings:
  - We will not answer questions from students in the middle of the test unless there is a mistake in a question.
  - Once the test starts, students may not leave until they turn in their tests, and once they leave they may not return.

Questions

1. Write a Python function called `compare_date` that takes as arguments two lists of two integers each. Each list contains a month and a year, in that order. The function should return -1 if the first month and year are earlier than the second month and year, 0 if they are the same, and 1 if the first month and year are later than the second. Your code should work for any legal input for month and year. Example calls and expected output are shown below:

```
>>> compare_date( [10,1995], [8,1995] )
1
>>> compare_date( [5,2010], [5,2010] )
0
>>> compare_date( [10,1993], [8,1998] )
-1
```

2. Assume `v` is a list containing numbers. Write Python code to find and print the highest two values in `v`. If the list contains only one number, print only that number. If the list is empty, print nothing. For example, if we assigned

```
v = [ 7, 3, 1, 5, 10, 6 ]
```

then the output of your code should be something like

```
7 10
```

If we are given that

```
v = [ 7 ]
```

then the output of your code should be

7

3. Consider a simplified version of the Lab 4 data, where just the name of the restaurant, the type of restaurant, and the ratings are provided. Assume these values **have already been read into a list of lists** of the form below:

```
restaurants = [ [ 'Acme', 'Italian', 2, 4, 3, 5],
                [ 'Flintstone', 'Steak', 5, 2, 4, 3, 3, 4],
                [ 'Bella Troy', 'Italian', 1, 4, 5] ]
```

   Write a segment of Python code that prints all `Italian` restaurants in the `restaurants` list that have no ratings of value 1 and at least one rating of value 5. In the above example, `Acme` would be printed in the output, but `Flintstone` and `Bella Troy` would not. `Flintstone` is not Italian and `Bella Troy` has a 1 rating. Your code should work for any legal version of `restaurants`.

4. Continuing with Lab 4, assume that you have the code

```
in_file = open('yelp.txt')

for line in in_file:
    p_line = parse_line(line)
    print p_line
```

   and that the `parse_line` function will return a list that looks like

```
["Meka's Lounge", 42.74, -73.69, "Bars", [5, 2, 4, 4, 3, 4, 5], 3.857142857142857 ]
```

   where the last entry in the list is the average ratin. Modify the `for loop` above to create a list called `high` that stores the names of all restaurants that have an average rating of at least 4.0. You do not have to print `high`.

5. In the game of chess you can often estimate how well you are doing by adding the values of the pieces you have captured. The pieces are Pawns, Bishops, Knights, Rooks and Queens. Their values are

```
    P - (P)awn, value = 1
    B - (B)ishop, value = 3
    K - (K)night, value = 3
    R - (R)ook, value = 5
    Q - (Q)ueen, value = 9
```

   Write a Python function called `chess_score` that takes a single string as an argument and returns the combined values represented by the pieces in the string. You may assume that only `'P'`, `'B'`, `'K'`, `'R'`, and `'Q'` appear in the string. You may **not** use any if statements and you may **not** use any loops. As an example,

```
print chess_score('BQBP')
```

   should output the value 16 because there are 2 Bishops (3 points each), 1 Queen (9 points each), and 1 Pawn (1 point each).

6. Use DeMorgan's laws to rewrite the following boolean expressions using only one **not**.

```
(not ex1) and (not ex2)
(not ex1) or (not ex2)
```

7. Use the distributive law to rewrite the following boolean expressions.

```
ex1 or (ex2 and ex3)
ex1 and (ex2 or ex3)
```

8. Use truth tables to show that:

```
not (ex1 or ex2) == (not ex1) and (not ex2)
```

9. You are given a file that contains, on each line of input, three integers separated by commas. Write a Python program that sums all of the first integers, the second integers, and the third integers, outputting the resulting sums all on one line, separated by commas. As a simple example, if the input is

```
 2, 5,7
 3,  6,   10
   1, 2, -3
  2, 4, 1
```

   Then the output should be

```
 8, 17, 15
```

10. Write Python code to generate the following ranges
    (a) $(100, 99, 98, \ldots, 0)$
    (b) $(55, 53, 51, \ldots, -1)$
    (c) $(3, 5, 7, 9, \ldots, 29)$
    (d) $(-95, -90, -85, \ldots, 85, 90)$

11. Write a **while** loop to add all of the numbers in a list `v` until it reaches a negative number or until it reaches the end of the list. Store the sum in the variable `result`. Your code should work for any version of `v` containing only numbers. For example, the value of `result` should be 25 after the loop for both of the following lists:

```
  v = [ 10, 12, 3, -5, 5, 6 ]

  v = [ 0, 10, 3, 6, 5, 1 ]
```

12. Write Python code that takes a list of numbers, `v`, and outputs the *positive* values that are in `v` in increasing order, one value per line. If there are no positive values, then the output should be the string `'None'`. You may assume there is at least one value in the list. As an example,

```
v = [ 17, -5, 15, -3, 12, -5, 0, 12, 22, -1 ]
```

   Then the output of your code should be

```
12
12
15
17
22
```

   As a second example, if

```
v = [ -17, -5, -15, -3, -12, -5, 0, -12, -22, -1 ]
```

   then then output should be just

```
None
```

13. What is the output of the following operations:

```
>>> mylist = [1,4,8,12,6]
>>> x = mylist.sort()
>>> print x

>>> mylist = [1,4,8,12,6]
>>> slice1 = mylist[2:4]
>>> slice1[0] = 20
>>> print slice1

>>> print mylist
```

14. Write a Python `for loop` to print out the values from the list `v` that are positive (0 is NOT a positive number).

15. What is the output of the following program?

```
def spam(a1,b1,a2,b2):
    if (a1 == a2) and (b1 > b2):
        return 1
    else:
        return 0

def egg(a1,b1,a2,b2):
    if (a1 > a2) and (b1 == b2):
        return 0
    else:
        return 1


a1 = 3
b1 = 4
a2 = 6
b2 = 4

print spam(a2, b2, a1, b1)

print egg(a1, b1, a2, b2)

c = spam(a1, b2, a2, b1)

print c

c += egg(a1, b2, a2, b1)

print c
```

16. Write a function called `copy_half` that takes the name of two files as arguments. The function should copy the first, third, fifth, etc. lines (i.e. odd lines only) from the first file to the second file. For example, if the file names are `'in.txt'` and `'out.txt'` and if `'in.txt'` contains

```
starting line
  not this line
middle line is here
    skip this line too
     I like this line
```

then after the call

```
copy_half( 'in.txt', 'out.txt' )
```

the file `'out.txt'` should contain

```
starting line
middle line is here
     I like this line
```

17. Write a segment of code that reads integers from a file called `test2.txt` and stores the positive values in one list, the negative values in a second list, and skips blank lines and zeros. The order of the values in each list should match the order of the input. Each line of input will contain either spaces or spaces and an integer. For example, if `test2.txt` contains

```
      11
  -3

  5
    0
```

Then after your code, the list `P` should be `[ 11, 5 ]` and the list `N` should be `[ -3 ]`.

18. Give the output of each of the following

(a)

```
i = 4
L = [ 0, 12, 3, 5, 2, -1 ]
while 0 <= i and i < len(L):
    if L[i] < 0:
        break
    else:
        i = L[i]
    print i, L[i]
```

(b)

```
tough = 2
for i in range(2):
    s = 1
    for j in range(i, tough):
        s += tough
    print s
    print tough
    tough = s
print tough
```

19. Suppose a list of words in alphabetical order has been assigned to the variable called `words`. For example, we might have the assignment

```
words = [ 'aardvark', 'abaka', 'expedite', 'experience', 'shoetrees', 'tastetest', 'test' ]
```

Write code to find and output the first and the last string in `words` that start and end with the same letter and are at least 8 characters long. You may assume that at least one word in `words` satisfies this condition. You may write a function if you wish. For the above example, the output should be

```
expedite
tastetest
```

20. Please show the output from the following code?

```
def get_min(v):
    v.sort()
    return v[0]

def get_max(v):
    x = max(v)
    return x

v = [ 14, 19, 4, 5, 12, 8 ]
if  len(v) > 10 and get_min(v) > 6:
    print "Hello"
    print v[0]
    print v[4]
else:
    print "So long"
    print v[0]
    print v[-1]
    if len(v) < 10 or get_max(v):
        print get_max(v)
        print v[0]
        print get_min(v)
        print v[0]
```

21. Write code that uses a `range` (and NO loops) to generate the following lists:

```
v0 = [ 10, 9, 8, 7, 6, 5, 4, 3 ]


v1 = [ -10, -3, 4, 11, 18, 25, 32, 39 ]
```

22. Consider the following list of lists of strings:

```
wordy = [ [ 'impala', 'malibu', 'camry', 'jetta'],
          [ 'zebra', 'impala', 'lion', 'impala', 'malibu', 'zebra' ],
          [ 'tiger', 'lion', 'cowboy', 'jet', '49er' ],
          [ ],
          [ 'five', 'seven', 'nine']  ]
```

(a) Show the output:

```
print wordy[2][1]


print wordy[1][2][3]


print len(wordy)


print len(wordy[1])
```

(b) Write a loop to print the last word of each list in `wordy`, stopping when either an empty list is found or when there are no more lists. For the above example, the output should be

```
    jetta
    zebra
    49er
```

23. Show the output from the following code:

```
def elephant(height):
    time_step = 1
    steps = 0
    while steps < height:
        steps += time_step
        steps -= time_step/3  # note: this is integer division
        time_step += 1
    print "%d, %d" %(time_step, steps)

elephant(0)
elephant(5)
elephant(6)
```

24. Show the output of the following code. Make sure we can determine what is output and what is scratch work.

```
def remove_something(z):
    z.remove( z[z[0]] )

v = [ 1, 8, [12, 8], 'hello', 'car' ]
x = 'salad'

if len(v[2]) >= 2:
    if x > v[3]:
        print 'One'
        if v[0] == 1:
            print 'Three'
    else:
        print 'Two'
elif len(v) == 5:
    print 'Six'
else:
    v.append('five')
    print 'Ten'

remove_something(v)
print v[1]
print v[2]
v.append(x)
print len(v)
```

25. You are given in variable `x` a list of lists represented as an NxN grid in which each list corresponds to one row of the grid. For example, a 4x4 grid is given by:

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

    Write a piece of code to print the grid in the following format with a vertical and horizontal line right in the middle:

```
1 2 | 3 4
4 3 | 2 1
----|----
2 1 | 4 2
2 1 | 4 5
```

26. Write a piece of code that repeatedly asks the user for numbers using `raw_input` until the user enters 'stop'. Then, the program reports the sum of the values entered by the user and the total number of values strictly greater than zero. You can assume that the user enters a valid number until she enters stop.

    An example run of this code is given below.

```
Enter a value ==> 1.2
Enter a value ==> 0
Enter a value ==> 2
Enter a value ==> -1
Enter a value ==> stop
Sum: 2.2
Values > 0: 2
```

27. Write a function `remove_val(l,val)` that removes all copies of `val` from list `l`.

    Suppose you are given a variable `x` containing numbers as shown below:

    `x = [1, 4, 2, 1, 2, 4, 4, 2, 5, 5, 2]`

    Then, your function should work as follows:

```
>>> remove_val(x,4)
>>> x
[1, 2, 1, 2, 2, 5, 5, 2]
```

    Note: if your function returns a new list with this content instead of modifying it as given, you will lose 3 points.

28. Suppose you are given the scores of two athletes in various competitions, given in two separate lists. Assume there are unknown number of competitions numbered 1,2,3, etc. and the length of the two lists is the same.

```
a1 = [11,8,11,9]
a2 = [11,9,8,12]
```

    For example according this to list, both athletes got a score of 11 in competition 1. Print the index of all the competitions in which `a2` did better. For example, for the above lists, we would print:

```
a2 is better in 2 4
```

    If there is no value in which `a2` is better, then you should print:

```
a2 is never better
```

29. What is the output from the following code:

```
>>> L1 = ['cat', 'dog', 'hawk', 'tiger', 'parrot']
>>> print L1[1:-1]
>>> print L1[1:-2]
>>> print L1[1:-4]
>>> print L1[1:0]
>>> print L1[1:10]
>>> print L1[::-1]
>>> print L1[1:4:2]
>>> print L1[::-2]
```

30. What is the output of the following programs:

Part a	Scratch area:

```
a = 25
b = 11
while True:
    print a, b
    if a <= 0 or b <= 0:
        break
    if a > b:
        a = a - b
    else:
        b = b - a
    b -= 1
    a += 1
```
 Output:	

Part b	Scratch area:

```
mylist = [10, -5, 4, 8, 1000, -1, -120, 18, 5.2]
for item in mylist:
    if item < 0:
        continue
    print item
```
 Output:	

Part c	Scratch area:

Part c

```
def spam(l,s):
    m = len(s)/2
    s1 = s[:m]
    s2 = s[m:]
    if l.count(s1) == 0:
        l.append(s1)
    if l.count(s2) == 0:
        l.append(s2)


l = ['ab','cd','de','fg']
s1 = 'abcde'
s2 = 'fghi'
spam(l,s1)
print s1
print l
l = spam(l,s2)
print s2
print l
```

Output: