Computer Science 1 — CSci 1100 — Spring 2016
Exam 2
March 28, 2016

RCS ID: ______________ @rpi.edu **Name:** ____________________

RIN # : ______________

Circle your lab section

Sec. 1	T 10 (Sage 2704), Jeramey
Sec. 2	T 12 (Sage 2112), Simon
Sec. 3	T 2 (Sage 3101), Jeramey
Sec. 4	W 10 (Eaton 216), Simon
Sec. 5	W 12 (Eaton 215), Antwane
Sec. 6	W 2 (Sage 2704), Antwane
Sec. 7	W 10 (Eaton 215), Simon
Sec. 8	W 12 (Eaton 216), Antwane

Problem	**Points**	**Score**
1	16	
2	12	
3	12	
4	25	
5	15	
6	20	
Total	100	

Instructions:

- You have 90 minutes to complete this test.
- You may use only one double-sided crib sheet. Otherwise, put away all books, laptop computers, and electronic devices.
- Please read each question carefully several times before beginning to work.
- We generally will not answer questions except when there is a glaring mistake or ambiguity in the statement of a question.
- There are **no Python syntax errors** anywhere on this test.
- Unless otherwise stated, you may use any technique we have covered thus far in the semester to solve any problem.
- Please state clearly any assumptions that you have to make in interpreting a question.
- There are **6 questions** on this test.
- When you are finished with this test please turn it in to the proctor, turn in your crib sheet with your name on it, and show your student id. You will then be free to leave the exam room.

1. (**16 points**) Show the output of the following:

Code:	**Answer:**
# **Part (a)**	

```
a = 73
b = 62
c = 62
b1 = a > b
b2 = c < b
print(b1)
print(b2)
print(not (b1 and b2))
```

# **Part (b)**

```
values=[-10, 20, 5.2, -57, 3, -9, 81, 7, 6, 4, -3]
for value in values:
    if value < 0:
        continue
    if abs(value) > 50:
        break
    print(value)
```

# **Part (c)**

```
def a(w, s, z):
    w *= 2
    s *= 2
    for l in range(len(z)):
        z[l] *= 2
    print(w)
    print(s)
    print(z)
i = 7
j = "Prestige"
k = range(5)
a(i, j, k)
print(i)
print(j)
print(k)
```

# **Part (d)**

```
a = [12, 4, 7, -2, 7]
b = a
c = a[::-1]
d = "Life Of "
e = d
a.sort()
b.remove(-2)
c[1] = c[4]
e += "Brian"
print("{}\n{}\n{}".format(a, b, c))
print("{}\n{}".format(d, e))
```

2. (**12 points**) For each of the following write a single line of Python code to solve the problem:

Code:	**Answer:**
(a) Write an expression that generate a list containing the values `[ 87, 82, 77, 72, ..., 2, -3, -8 ]`	

(b) Write an expression that starts from a list called `v` and creates a new list containing the values `v[3]`, `v[6]`, `v[9]`, ... This should work for any length list. For example if we have
`v = ['car', 'boy', 'bike', 'camel', \`
`     'horse', 'trolley', 'jet', 'truck']`
then your expression should produce the list
`[ 'camel', 'jet' ]`	

(c) Write an expression that is `True` if the first value in a list called `v` appears in **more than half** the locations in `v` and is `False` otherwise. Your code should work for both even and odd length lists. For example if
`v = [ 12, 9, 16, 3, 12, 12 ]`
your expression should evaluate to `False` and if
`v = [ 12, 9, 16, 3, 12, 12, 12 ]`
your expression should evaluate to `True`	

3. (**12 points**) Write a function called `is_earlier` that takes as arguments two times, each represented as a two-part tuple containing an integer hour and a string that is either `'AM'` or `'PM'` (assume this string is in all capital letters). The function should return -1 if the first time is earlier in the day than the second time. It should return 0 if they are the same time, and it should return 1 if the first time is later in the day than the second time. Note that midnight is 12AM and noon is 12PM.

   As examples, the following two calls should return -1:

   ```
   is_earlier( (12, 'AM'), (4, 'AM') )
   is_earlier( (5, 'AM'), (3, 'PM') )
   ```

   while the following call should return 0:

   ```
   is_earlier( (2, 'PM'), (2, 'PM') )
   ```

   and finally, the following calls should return 1:

   ```
   is_earlier( (3, 'PM'), (12, 'PM') )
   is_earlier( (5, 'PM'), (5, 'AM') )
   ```

4. (**25 points**) This question examines a variation on our randomly walking turtle. We will consider a turtle walking on a one-dimensional board with possible locations `0` through `N-1`. The problem is in two parts.

   For **Part (a)** write function `run_turtle( N )` that takes a board size `N`, assigns the turtle random steps using `random.random()`, and sees how long it takes the turtle to move off the left end of the board (reaches position -1) or the right end of the board (reaches position `N`). The turtle always starts at the center of the board at position `N/2` and starts out moving to the right (increasing position). At each step, generate a random value between 0.0 and 1.0. If the value is less than 0.4, the turtle reverses its direction **and** takes a step in this new direction. Otherwise (the random value is at least 0.4), the turtle takes a step in the direction it already is headed.

   As an example, if the turtle is at position 5, headed right, and the random number is 0.372, then the turtle will turn around so that it is headed left and its new position will be 4.

   Function `run_turtle` should end when the turtle moves off the board. It should return a tuple containing two values: (1) the number of steps taken by the turtle and (2) the side of the board the turtle stepped off ("left" or "right").

For **Part (b)**, write code to ask the user for a board size and then call your `run_turtle()` function 1,000 times consecutively for that board size. (Do not worry about seeding the random number generator.) After this is finished, the code should print out (1) the average number of steps taken per run, (2) the number of times the turtle stepped off the left side, and (3) the number of times the turtle stepped off the right side.

5. (**15 points**) Recalling Lab 5, write a function that returns `True` if a word appears on a diagonal going down and to the right, and `False` otherwise. (**NOTE 10/16/16: Lab 5 was different this fall, but solving this problem is still good practice.**)

   For example, given the letter board:

```
bd = [['c', 'i', 'w', 'i', 'z', 'z', 'e', 'k', 's', 'm'],
      ['g', 't', 'u', 'p', 'e', 'i', 'p', 'r', 'f', 'j'],
      ['e', 'n', 'j', 'p', 'd', 't', 'b', 'f', 'd', 'r'],
      ['m', 'e', 'r', 'k', 'v', 'x', 'e', 's', 'u', 'd'],
      ['r', 'm', 'k', 's', 'q', 'j', 'q', 'l', 'p', 'r'],
      ['d', 'j', 'x', 'f', 'p', 'o', 's', 'm', 'w', 'a'],
      ['c', 'k', 'c', 'b', 'i', 'a', 'h', 'x', 'm', 'm'],
      ['d', 'r', 'l', 's', 'k', 't', 'm', 'e', 'a', 'n'],
      ['h', 'i', 'j', 's', 'd', 't', 'z', 'i', 'y', 'j'],
      ['g', 'f', 'k', 'f', 'h', 'p', 'w', 'f', 'u', 'h']]

==> print(is_on_diagonal(bd, 4, 3, 'spam'))
True
==> print(is_on_diagonal(bd, 3, 7, 'spam'))
False
```

   Answer:

6. (**20 points**) Suppose you are given an input file that should contain a table of grades. The first line of the file is the number of grades per line. (Call it N.) Each of the remaining lines is supposed to have N grades on it, separated by commas. Each grade should be at least 0 and at most 100. Write a program that checks the input file for correctness, printing out to the screen any line that does not contain N integers or that contains a grade outside the legal range. Assume these are the only possible formatting errors (i.e. no letters and no punctuation other than `','`). For example, if the file contains:

```
5
12, 4, 7, 99, 100
78, 82, 85, 94
-5, 93, 95, 99, 80
78, 99, 87, 88, 100
89, 98, 78, 87, 102
```

The program would print:

```
78, 82, 85, 94
-5, 93, 95, 99, 80
89, 98, 78, 87, 102
```

Please solve this in two parts. In Part (a), write a function called `all_ok` that takes two arguments: a string representing a line from the input file and the integer `N`. It should return `True` if the input line is correct — `N` grades, all in the range $0 \ldots 100$ — and `False` otherwise.

In Part (b) write the program to solve the problem by reading the file and printing each line that has an error. It **must** use `all_ok` to check for errors.

There are no more questions.