Name: _____

Email address (UW NetID): _____

CSE 160 Winter 2016: Midterm Exam

(closed book, closed notes, no calculators)

Instructions: This exam is closed book, closed notes. You have 50 minutes to complete it. It contains 11 questions and 10 pages (including this one), totaling 90 points. Before you start, please check your copy to make sure it is complete. After the 10 pages of the exam, there is a syntax cheat sheet that you may remove. Turn in all 10 pages of the exam, together, when you are finished. When time has been called you must put down your pencil and stop writing. **Points will be deducted from your score if you are writing after time has been called.**

Good Luck! Total: 90 points. Time: 50 minutes.

Problem	Points Possible
1	6
2	4
3	5
4	12
5	8
6	4
7	4
8	12
9	12
10	12
11	11
Total	90

1) [6 pts] For each of the if statements below, write the output when x = 15, x = 45, and x = 60 in the table below:

```
a)
if x < 30:
    print "line 1"
elif x < 60:
    print "line 2"
else:
    print "line 3"

b)
if x < 30:
    print "line 1"
if x < 60:
    print "line 2"
else:
    print "line 3"
```

	X = 15	X = 45	X = 60
Code a)			
Code b)			

2) [4 pts] Write the output of the code below in the box here:

```
sum = 0
for x in range(1, 5):
    for y in range(x):
        sum = sum + y
print 'sum:', sum
```

MY ANSWER:

3) [5 pts] The following function is supposed to take a list, and shift each element to the left by one location. The first element in the list should be placed at the last position of the list. For example, the list: `[1, 5, 4, 6, 8, 7]` should become: `[5, 4, 6, 8, 7, 1]`

```
def shift_one(lst):
    # Assumes lst contains at least one element.


    for i in range(0, len(lst) - 1):

        lst[i] = lst[i + 1]


    return lst
```

However there is a bug in the code. You should:

a) Identify and describe in plain English the bug

b) Modify the code above such that it does what is expected.

MY ANSWER:

a)

b) **Show your modifications in the code above.**

4) [12 pts] Suppose we have the following list:

```
mystery = [ [8], 4, [1, 5, 3, [2, 6]], [9, 7], 7]
```

Write the result of the following expressions. If an error is thrown, briefly describe the error.

a) `mystery[1]`	
b) `mystery[0]`	
c) `[8] in mystery`	
d) `8 in mystery`	
e) `mystery[2][1]`	
f) `mystery[4][0]`	
g) `mystery[-1]`	
h) `mystery[2:3]`	

i) Write code that modifies the original mystery list so that it contains the following (change in **bold**):

```
mystery = [ [8], 4, [1, 5, 3, [2, 6], 9], [9, 7], 7]
```

j) Write code that modifies the original mystery list so that it contains the following (change in **bold**):

```
mystery = [ [8], 0, [1, 5, 3, [2, 6]], [9, 7], 7]
```

5) [8 pts] For the following snippets of code indicate **the first error it produces and give the line number of the error**. If there is no error, state that. If there is more than one error, identify the first error that would be encountered. Assume each snippet of code is executed independently.

Possible Errors: (each may be used more than once)

- KeyError
- AssertError
- NameError
- TypeError
- IndentationError
- SyntaxError

a)

```
1 my_dict = {'a': 4, 'b': 5, 'd': 7}
2 my_dict['a'] = "red"
3 print my_dict['c']
```

a) Error:

Line number:

b)

```
1 for i in range(19):
2     if i % 2 == 0:
3     print 'even'
4     else:
5     print 'odd'
6 assert(i % 2 == 0)
```

b) Error:

Line number:

c)

```
1 lst = [3, 5, 2]
2 print "length" + len(lst)
3 assert(len(lst) == 3)
```

c) Error:

Line number:

d)

```
1 my_list = [1, 2, 3]
2 if 4 not in my_list:
3     the_list.append(4)
4 assert(4 in my_list)
```

d) Error:

Line number:

6) [4 pts] What output is produced after running the following piece of code?

```
a = [3, 1, 5]
b = a
c = b[:]
d = list(a)

a.append(9)
b.append(2)
c.append(6)
d.append(5)

print a
print b
print c
print d
```

MY ANSWER:

7) [4 pts] What output is produced after running the following piece of code?

```
from operator import itemgetter

levels = [ ('Ann', 4), ('Jenny', 5),('Greg', 3),
               ('Gray', 3), ('Sam', 5) ]

print sorted(levels, key=itemgetter(1), reverse=True)
```

MY ANSWER:

8) [12 pts] Write a function called `create_recip_dict` that takes no arguments and returns a dictionary that maps the integers 1 to 100 to their reciprocal as a **string**. For example the resulting commands in the interpreter should produce the output as shown:

```
>>> recip_dict = create_recip_dict()
>>> recip_dict[0]
```
(Error)

```
>>> recip_dict[1]
'1/1'
>>> recip_dict[2]
'1/2'
>>> recip_dict[100]
'1/100'
>>> recip_dict[101]
```
(Error)

MY ANSWER:

```
def create_recip_dict():

  # Your code here
```

9) [12 points] Write a function called `word_lengths` that takes a string argument as input. Assume the string has already been stripped of all punctuation and converted to lowercase. The function should split the string into individual words and return a dictionary mapping the number of letters in a word to a set of words of that length that appeared in the string. For example:

```
print word_lengths("this is a cool string eh")
```

Would print something like this:

```
{1: set(['a']), 4: set(['this', 'cool']), 2: set(['is', 'eh']), 6: set(['string'])}
```

MY ANSWER:

```
def word_lengths(input_string):
    # Assumes input_string contains at least one letter.

    words = input_string.split(' ')

    # Your code here
```

10) [12 points] Write a function called `find_common_sets` that takes two lists of sets as arguments. Assume the two lists are of equal length but the sets they contain may differ. The function should return a list containing the intersections of the two sets in the **corresponding positions** of each list. For example:

```
list_a = [{1}, {3, 4, 5}, {1, 2}]
list_b = [{3}, {3, 5, 6}, {2}]
print find_common_sets(list_a, list_b)
```

Would print something like this:

```
 [set([]), set([3, 5]), set([2])]
```

MY ANSWER:

```
def find_common_sets(lst_one, lst_two):
    # Assumes lst_one and lst_two contain at least one set each.

    # Your code here
```

11) [11 pts] a) **Draw** the entire environment, including all active environment frames and all user-defined values, **at the moment that the MINUS OPERATION IS performed**. Feel free to draw out the entire environment, but be sure to CLEARLY indicate what will exist at the moment the **MINUS** operation is performed (e.g. cross out frames that no longer exist).

b) When finished executing, **what is printed out by this code**?

MY ANSWER:

c) **How many different stack frames** (environment frames) are active when the call stack is DEEPEST/LARGEST? (Hint: The global frame counts as one frame.)

MY ANSWER:

```
x = 10

def happy(y):
    return fuzzy(fuzzy(y)) - y

def fuzzy(z):
    return z + 3

def sunny(x):
    val = fuzzy(x)
    return happy(val) + val

print sunny(x)
```

MY ANSWER: