Name: ___ **Sample Solution** _____

Email address (UW NetID): _____

CSE 160 Spring 2015: Final Exam

(closed book, closed notes, no calculators)

Instructions: This exam is closed book, closed notes. You have 50 minutes to complete it. It contains 7 questions and 9 pages (including this one), totaling 68 points. Before you start, please check your copy to make sure it is complete. Turn in all pages, together, when you are finished. Please write neatly; we cannot give credit for what we cannot read.

Good Luck!

Total: 68 points. Time: 50 minutes.

Problem	Max Points	Score
1	5	
2	3	
3	10	
4	6	
5	24	
6	10	
7	10	
Total	68	

1) [5 pts] You receive the following error messages after running your code:

```
Traceback (most recent call last):
  File "social_network.py", line 338, in <module>
    do_stuff(friends_list)
  File "social_network.py", line 200, in do_stuff
    result = recommend_by_influence(friendlist)
  File "social_network.py", line 107, in recommend_by_influence
    output = read_result()
NameError: global name 'read_result' is not defined
```

a) List the names of the stack frames that existed at the point the error was discovered.

Global, do_stuff, recommend_by_influence

b) What is the most recent stack frame (e.g. the last function we successfully called)?

recommend_by_influence

c) Describe how you would go about trying to find the cause of and fix this error.

Most likely this is due to a misspelling of the function name referred to as "read_result()" on line 107 of social_network.py. So a good start would be to search to see if there is a similarly named function in the file social_network.py. If that fails, maybe this function is defined in another namespace like we did before with Random or nx, requiring the function name to be prefaced with that module name.

2) [3 pts] Write code that would produce a "KeyError: " error

```
temp_dict = {2:"foo", 5:"bar"}
print temp_dict[3]
```

```
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    print temp_dict[3]
KeyError: 3
```

3) [10 pts] a) Given the function `sum_roll()` which returns the sum of two random 6-sided dice rolls, complete the following function to create a dictionary of sum frequencies. The dictionary should contain an entry for every possible sum of two dice. For example, `freq_of_sums(5)` might return:
`{2:1, 3:0, 4:0, 5:1, 6:0, 7:2, 8:0, 9:0, 10:1, 11:0, 12:0}`

```
def freq_of_sums(n=10):
    ''' Rolls two 6-sided dice n times and records the sum of
    each pair. Assumes n is a positive integer. Returns a
    dictionary mapping the sums to the number of times in n rolls
    that sum was rolled.'''
    # Your code here
    # One solution, others possible.
    # Also fine to use a loop for initializing the dictionary
    freq_dict = {i:0 for i in range(2, 13)}
    for roll in range(n):
        dice_sum = sum_roll()
        freq_dict[dice_sum] += 1
    return freq_dict
```

b) Describe in a couple of sentences your approach to testing this function.

Here are a couple of specific assert statements you could use:

```
n = 20
assert (sum(freq_of_sums(n).values()) == n)
assert len(freq_of_sums(n)) == 11
```

There is no need to test for values of n=0 or negative, or for n as a string or float, as that behavior is not specified in the docstring – the function can do whatever it wants there.

A couple of folks suggested running for very large numbers of n and seeing if the distribution approached the uniform distribution which might catch some egregious errors but otherwise is hard to use to check exact correctness. You could also create your own version of the function `sum_roll` and have it return a set of known values, such that you would know exactly what the distribution would be.

4) [6 pts] a) Write a docstring for the following function. Document the inputs and any outputs or side effects (meaning, what else does the function do?).

```
import matplotlib.pyplot as plt

def my_plot(slope, x_points):

    """Your docstring here."""

    y_points = []

    for x in x_points:

        y_points.append(x**slope)

    plt.clf()

    plt.plot(x_points, y_points)

    plt.show()
```

MY ANSWER:

```
''' Displays a plot of x**slope for all x_points.

Inputs:
    slope: a number that will be used as the exponent of each
        value in x_points
    x_points: a list of numbers

Outputs: None

Side effects:
    Clears any previous matplotlib graph before plotting.
    Displays a graph of x vs. x**slope for all x in x_points.
'''
```

b) Add one line of code to `my_plot` that would save the resulting figure as "using_val.png", where val is whatever value is stored in the variable `slope`. Draw arrows in the code above to show where you would add your line of code, be exact!

```
plt.savefig("using_" + str(slope) +".png")
```

This line should be inserted after the call to plot and BEFORE the call to show.

5) [24 pts] For this problem you should write code in good style (as if you were submitting it for a homework assignment) and you **should use functions defined in earlier parts of the problem** if applicable.

You have a data structure which is a list of dictionaries as follows:

```
cities = [
       {'Name':'Vancouver','State':'WA','Population':161791},
       {'Name':'Salem','State':'OR','Population':154637},
       {'Name':'Seattle','State':'WA','Population':608660},
       {'Name':'Spokane','State':'WA','Population':208916},
       {'Name':'Portland','State':'OR','Population':583776},
       {'Name':'Bellingham','State':'WA','Population':80885},
       . . .
      ]
```

a) Complete the function `max_in_state` to return the city (as a dictionary) with the highest population in a given state. If `cities` contained only the dictionaries above, a call to `max_in_state(cities, 'WA')` would return:

```
{'Name':'Seattle','State':'WA','Population':608660}
```

```
def max_in_state(city_list, state):
  '''Return the dictionary for the city in city_list with the
  largest population for the given state.  Return None if there
  are no cities for the given state in city_list.'''
  max_city = None
  for city in city_list:
    if city['State'] == state:
      if max_city == None or city['Population'] > max_city['Population']:
        max_city = city
  return max_city
```

5) b) Complete the following function to return the total population for a given state. (Assume that all census locations for a state are included in the cities list). If `cities` contained only the following dictionaries:

```
cities = [
        {'Name':'Seattle','State':'WA','Population':50},
        {'Name':'Portland','State':'OR','Population':10},
        {'Name':'Spokane','State':'WA','Population':20},
        {'Name':'Vancouver','State':'WA','Population':7},
       ]
```

a call to `state_population(cities, 'WA')` would return: 77

```
def state_population(city_list, state):
    '''Return a number representing the total population of the
    given state based on the list of cities provided. Return None
    if there are no cities for the given state in city_list.'''

    total_pop = None
    for city in city_list:
        if city['State'] == state:
            if total_pop == None:
                total_pop = 0
            total_pop += city['Population']
    return total_pop
```

5) c) Complete the following function to return a dictionary mapping state to the total population for that state. (Assume that all census locations for a state are included in the cities list) If `cities` contained only the dictionaries shown in part b, a call to

`all_state_populations(cities)` would return:

```
{'WA':77, 'OR':10}
```

```
def all_state_populations(city_list):
    '''Returns a dictionary mapping state to total population
    for each state in city_list. Returns an empty dictionary if
    city_list is empty.'''

    state_set = set()
    for city in city_list:
        state_set.add(city['State'])
    # OR state_set = {city['State'] for city in city_list}
    all_states_dict = {}
    for state in state_set:
        all_states_dict[state] = state_population(city_list, state)
    # OR all_states_dict = {state: state_population(city_list, state)
    #                                          for state in state_set}
    return all_states_dict
```

6) [10 pts] You are given the following class definition:

```
class Car:
    def __init__(self, gas_tank_size, miles_per_gallon):
        ''' gas_tank_size and gas is in gallons '''
        self.gas = 0.0
        self.tank_size = gas_tank_size
        self.mpg = miles_per_gallon

    def refill_tank(self):
        '''Fills gas tank up to the tank size.
        tank_size and gas are in gallons.'''
        self.gas = self.tank_size
```

a) Fill in the code for the function below that is also a part of the class Car.

```
    def drive(self, miles):
        '''Drives the car the provided number of miles.
        Will deduct the required gas from the tank of the car.
        If there is not enough gas just print "Not enough gas!"
        and will not drive or deduct the gas.'''
        # Your code here


        gas_needed = float(miles) / self.mpg

        if gas_needed <= self.gas:

            self.gas = self.gas - gas_needed

        else:

            print "Not enough gas!"
```

b) Add code below to use the method that you wrote to drive `nicks_car` 16 miles.

```
nicks_car = Car(15, 30)
nicks_car.refill_tank()
# your code here:

nicks_car.drive(16)
```

c) List one advantage of using a class

Many possible answers. Collects the data representation and functions operating on that data in one place. Allows the implementer of the Car class to change the internal implementation while the client code can remain unchanged.

7) [10 pts] a) **Draw** the entire environment, including all active environment frames and all user-defined variables, **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:** **2**

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:** **4**

```
def sun(x):
    return x + 2

def water(x):
    y = sun(sun(x))
    return y - sun(x)

def summer(y):
    x = sun(y)
    return water(x * 10)

x = 7
print summer(sun(x))
```