

Name: _____

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.
- b) What is the most recent stack frame (e.g. the last function we successfully called)?
- c) Describe how you would go about trying to find the cause of and fix this error.

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

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
```

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

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:

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!

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.'''
```

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.'''
```

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.'''
```

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
```

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:
```

c) List one advantage of using a class

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:

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:

```
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))
```