Q2) You are given the following class definition:

```
class City:
    def __init__(self, name, population, area):
        '''name: a string representing the name of a city
           population: an integer representing the number of
                       people in the city
           area: a number representing area in square miles '''
        self.name = name
        self.pop = population
        self.area = area

    def add_people(self, num_new_people):
        '''num_new_people: an integer representing the number of
           people to be added to the current population of the city '''
        self.pop = self.pop + num_new_people

    def get_pop_density(self):
        '''Returns a float representing the population density
           in the city.  Population density is defined as the
           number of people per square mile. '''
        # Code not visible
```

a) Write code in the `main` function, using methods from the `City` class, to:

- Add 177 new people to the population of `sea`.
- Print the population density of `lax`

This code is outside of the class **`City.`**

```
def main():
    sea = City("Seattle", 704352, 83.78)
    lax = City("Los Angeles", 3976000, 503)
    # Your code here:

    sea.add_people(177)

    print(lax.get_pop_density())
```

Q2) (continued)

b) Describe your overall approach to testing `get_pop_density`. Be as specific as you can (as close to actual code as possible).

Create multiple `City` objects with different values for area and population and write `assert` statements that call `get_pop_density()` on those objects. Since `get_pop_density` returns a float, you should use something like the `eq()` function we used in hw5 which checks to see if you are within some epsilon of the desired value. You want tests that determine not just that a float is returned, but that floating point division is being done (as opposed to converting the result of integer division to a float). Although the specification is not clear about whether or not population or area should be allowed to be zero, it was a good idea to think about checking those edge cases. Just checking large or small values for population and area is not specific enough. Here are a few example tests:

```
# Should be eq() .1 (checks that fp division is being done)
ts1 = City("Test City 1", 1, 10)
assert eq(ts1.get_pop_density(), 0.1)
# Varying types on area (population must be an integer)
ts2 = City("Test City 2", 10, 2.5)
assert eq(ts2.get_pop_density(), 4.0)
```

c) Finally, write the code for the `get_pop_density` method below. As shown above, this method is a part of the class **`City`**:

```
def get_pop_density(self):
    '''Returns a float representing the population density
       in the city.  Population density is defined as the
       number of people per square mile. '''
    # Your code here

    return self.pop/self.area
```

Q3) Write a function called `remove_words` that takes two arguments: the name of a file and a list of undesirable words that should be removed from the contents of that file. **The function should not modify the original file or create a new file**, but instead it should read in the file and return a single list containing the words from the original file, with all occurrences of the undesirable words removed. For example, if the input file named "`cool_essay.txt`" contained these 4 lines:

```
like happy like birthday

yep summer is totally here
lol happy summer
```

and you had this list of words:

```
words_to_remove = ['like', 'whatever', 'lol', 'yep', 'totally']
```

The function call:

```
remove_words("cool_essay.txt", words_to_remove)
```

would return this single list:

```
["happy", "birthday", "summer", "is", "here", "happy", "summer"]
```

You may assume that the input file contains no punctuation and all words in the input file and in the list `words_to_remove` are in lowercase.

```
def remove_words(filename, words_to_remove):
    # Your code here:

    clean_list = []
    infile = open(filename)
    for line in infile:
        line_list = line.split()
        for word in line_list:
            if word not in words_to_remove:
                clean_list.append(word)
    infile.close()
    return clean_list
```