

Name: _____

Email address (UW NetID): _____

CSE 160 Spring 2018: 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 70 points. Before you start, please check your copy to make sure it is complete. A syntax sheet will be provided separately. 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.** You should only use parts of Python that have been covered in the class so far.

Total: 70 points. Time: 50 minutes.

Problem	Max Points	Score
1	4	
2	4	
3	4	
4	12	
5	10	
6	12	
7	24	
Total	70	

0) [0 points] Question: What do you call a snake that is 3.14 meters long?

Answer: A "Pi-thon"

(get it? This is not a real question, just a joke. Smile and move on to question 1! 😊)

1) [4 pts] Grab Bag:

a) When should you write tests: before, during, or after writing the code? Why?

b) What is Data Abstraction and why is it useful? Be specific.

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

```
def mystery():
    result = 0
    for i in range(20, 0, -2):
        result += i
        if i % 8 == 0:
            result -= 8
        elif i % 4 == 0:
            result -= 4
        if i % 9 == 0:
            return result
    return result
```

```
print mystery()
```

MY ANSWER:

3) [4 pts] Indicate (circle) if there is an Error or No Error. ****For any credit, if there is an error, you MUST very briefly say what the problem is.**

a) `d = {}`

Error / No Error

`d[{0, 1, 2}] = {0, 1, 2}`

If an Error, describe briefly:

b) `d = {}`

Error / No Error

`d[[0, 1, 2]] = {0, 1, 2}`

If an Error, describe briefly:

c) `d = {}`

Error / No Error

`d["0, 1, 2"] = (0, 1, 2)`

`d["0, 1, 2"][1] = 0`

If an Error, describe briefly:

d) `d = {}`

Error / No Error

`d["0, 1, 2"] = {0, 1, 2}`

`print d.keys()["0, 1, 2"][0]`

If an Error, describe briefly:

4) [12 pts] 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:
```

5) [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 banana(x):
    y = orange(x)
    return y - orange(x + 2)

def apple(y):
    x = 2
    return banana(orange(y) + x)

def orange(x):
    return x + 5

x = 400
y = 3
print apple(orange(y))
```

6) [12 pts] 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:
```

6) (continued)

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

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
```

7) [24 pts total] You want to learn cake-baking from your friend by following along with them over video chat, but you're not sure which cake recipes you can both make. Given the following:

```
alice_pantry = {'egg': 12, 'cup of sugar': 4, 'cup of flour': 4}
bob_pantry = {'egg': 12, 'cup of sugar': 4, 'chocolate': 5}
recipes = {
    'dusty cake' : {'cup of sugar': 4, 'cup of flour': 4},
    'eggy cake'  : {'egg': 1 },
    'eggier cake' : {'egg': 10 },
    'mega egg'   : {'egg': 100 },
    'chocolate cake' : {'egg': 2, 'cup of sugar': 2, 'chocolate': 1}
}
```

- A pantry is a dictionary mapping ingredients to counts of those ingredients available in that kitchen pantry.
- `recipes` is a nested dictionary mapping cake names to the ingredients and amounts they require.
- A cake recipe is considered **bakeable** if you have at least enough of every ingredient required by the recipe.

a) Write a function `find_bakeable(my_pantry, friend_pantry, recipes)` that returns an **alphabetically-sorted list** of all cake names that you and your friend can both make using the ingredients you both have available in your own kitchen pantries. For example, the function call:

```
find_bakeable(alice_pantry, bob_pantry, recipes)
```

should return the following list:

```
['eggier cake', 'eggy cake']
```

****For full credit you should be sure to call the function `single_bakeable` inside of `find_bakeable`. Read its description ([shown on the next page](#) in part b) NOW. ****

```
def find_bakeable(my_pantry, friend_pantry, recipes):
    # Your code here
```


7) (continued)

b) Now, write the helper function `single_bakeable(pantry, recipes)` that returns a set of cake names that can be made from that pantry. For example, the function call:

```
single_bakeable(alice_pantry, recipes)
```

should return the set containing these three cake names:

```
{'eggier cake', 'eggy cake', 'dusty cake'}
```

```
def single_bakeable(pantry, recipes):  
    # Your code here
```