CSc 110, Spring 2017
Programming Assignment #8: Animal Shelter (20 points)
Due Tuesday, March 21, 2017, 7:00 PM

Part 1:

```
Welcome to animal shelter management software
version 1.0!

Type one of the following options:
adopt:    adopt a pet
intake:   add more animals to the shelter
list:     display all adoptable pets
quit:     exit the program
save:     save the current data
transfer: transfer pets to another shelter
option? list
animal type? all
('alyson', 5.5, 'dog')
('chester', 1.5, 'cat')
('felice', 16.0, 'cat')
('jesse', 14.0, 'dog')
('merlin', 5.0, 'cat')
('percy', 12.0, 'cat')
('puppet', 18.0, 'cat')

Type one of the following options:
adopt:    adopt a pet
intake:   add more animals to the shelter
list:     display all adoptable pets
quit:     exit the program
save:     save the current data
transfer: transfer pets to another shelter
option? transfer
file name? transfered.txt

Type one of the following options:
adopt:    adopt a pet
intake:   add more animals to the shelter
list:     display all adoptable pets
quit:     exit the program
save:     save the current data
transfer: transfer pets to another shelter
option? list
cats, dogs or all? cat
('chester', 1.5, 'cat')
('felice', 16.0, 'cat')
('puppet', 18.0, 'cat')

Type one of the following options:
adopt:    adopt a pet
intake:   add more animals to the shelter
list:     display all adoptable pets
quit:     exit the program
save:     save the current data
transfer: transfer pets to another shelter
option? quit

7 pets currently in the shelter
0 adopted
2 transferred
```

This program focuses on using lists that change size and also demonstrates a type of program that maintains data in a manner useful in supporting typical volunteer organizations. Turn in a file named `animal_shelter.py` on the Homework section of the course web site.

Program Description:

In this assignment you will write a program that keeps track of animals at an animal shelter.

```
dog alyson 5.5
cat Chester 1.5
cat felice 16
DOG jesse 14
cat merLIN 5
cat percy 12
cat PUPPET 18
parrot sue 2.3
```

The hope is that all animals at a shelter will be adopted soon after entering. However, quick adoptions don't always happen. To speed up adoption and to avoid overcrowding, a shelter may transfer animals to another shelter.

When your program starts, it will read from three input files. Each file contains information about pets in the format of the file shown above. The file `pets.txt` contains the pets currently at the shelter, `transferred.txt` contains the pets that have been transferred from this shelter to another, and `adopted.txt` contains the pets that have been adopted from this shelter. The contents of the files are in alphabetical order by pet name.

Specification of Options:

`adopt`: When the user types this option, the program should prompt for the type of pet and then prompt for the pet's name as shown in the sample output. This pet should then be removed from the shelter and added to the list of pets that have been adopted. If a pet of this type and name does not exist, the program should print out a not found message as shown in the sample output.

`intake`: When the user types this option, the program should prompt the user for a file name as shown in the sample output. This file will be in the same format as the files read in at the beginning of the program and the pets will also be alphabetized by name. Your program should add all of the pets from this file to the shelter list, making sure to maintain the sorted order of the shelter list.

`list`: When the user types this option, the program should prompt the user for a type of pet (cat, dog, hamster, etc). The program should then display a list of all pets at the shelter of that type. If the user enters "all" instead of a specific type, all pets should be displayed.

`quit`: When the user types this option, the program should print the current shelter statistics, as shown in the sample output, and exit.

`save`: When the user types this option, the program should write the current contents of the shelter list, the transferred pet list, and the adopted pet list to the same files this program read the data from on startup. The data should be saved in the original format it was read in.

`transfer`: When the user types this option, the program should prompt the user for a file name as shown in the sample output. This file will be in the same format as the files read in at the beginning of the program and the pets will also be alphabetized by name. Your program should move all of the pets listed in the file the user specified out of the shelter and into the transferred list. You may assume all listed pets are in the shelter when the command is run. You may only loop through the shelter list one time every time the user selects this option.

If the user inputs anything besides the options above, the user should be re-prompted for a valid option as shown in the sample output.

All comparisons of strings should be case insensitive.

Implementation and Style Guidelines:

As always, a major focus of our style grading is **redundancy**. As much as possible, avoid redundancy and repeated logic in your code.

You will need to keep track of the pets in the shelter, the pets that have been transferred, and the pets that have been adopted. You may use lists of tuples to keep track of this data. You may alter these three lists throughout your program but you may not copy or replace them or create additional lists. You should use the list and its functions appropriately, and take advantage of its ability to "shift" elements as they are added or removed. Your list should not store any invalid or `None` elements as a result of removing pets.

You should introduce **constants** for any hardcoded values that appear in the code.

You should follow good general Python style guidelines such as: appropriately using control structures like loops and `if`/`else` statements; avoiding redundancy using techniques such as functions, loops, and factoring common code out of `if/else` statements; using good variable and function names; and not having any lines of code longer than 100 characters in length.

You should **comment** your code with a heading at the top of your class with your name, section, and a description of the overall program. Please also indicate how long you spent on this assignment in your heading by adding a comment like, `# hours spent: 7`. Also place a comment heading on top of each function, and a comment on any complex sections of your code. Comment headings should use descriptive complete sentences and should be written *in your own words*, explaining each function's behavior, parameters, return values, and assumptions made by your code, as appropriate.

Your solution should use only material taught in class and should use no material past lecture 23.

Part 2:

This part of the assignment will give you an opportunity to improve your testing skills. Turn in two files `whitebox.py` and `blackbox.py`.

Black Box Testing

Given the following function description, write a series of **up to 6 tests** as described in class, that test all important cases. DO NOT write the code to solve this problem!

Description

Write a function named `remove_consecutive_duplicates` that accepts as a list of integers, and modifies it by removing any consecutive duplicates. For example, if a list named `list` stores `[1, 2, 2, 3, 2, 2, 3]`, the call of `remove_consecutive_duplicates(list)` should modify it to store `[1, 2, 3, 2, 3]`.

White Box Testing

Given the following code, write **up to 6 tests** as described in class, that test all important cases. The code should find the longest sorted (increasing) sequence between the start and stop (inclusive) and return its length. The code may not be correct.

Code

```
def longest_sorted_sequence(list, start, stop):
    if (len(list) == 0):
        return 0
    max = 1
    count = 1
    for i in range(start, stop):
        if (list[i] >= list[i - 1]):
            count += 1
        else:
            count = 1
        if (count > max):
            max = count
    return max
```

Remember, each test should test one thing. It should have a descriptive error message. Each test should be able to discover a different type of error. Your testing code will be held to the same style standards as the rest of the code you have turned in this semester.

Note: There are many possible correct answers to part 2. We want to see that you are thinking about testing and following the principles we have covered but there are many, many good combinations of tests you can write.