CMSC 201 Spring 2017
Homework 5 – Practice Time

Assignment: Homework 5 – Practice Time
Due Date: Friday, April 7th, 2017 by 8:59:59 PM
Value: 40 points

Collaboration: For Homework 5, collaboration is allowed. Make sure to consult the syllabus about the details of what is and is not allowed when collaborating. You may not work with any students who are not taking CMSC 201 this semester.

If you work with someone, remember to note their name, email address, and what you collaborated on by filling out the Collaboration Log.

You can find the Collaboration Log at http://tinyurl.com/collab201-Sp17.

Remember that all collaborators need to fill out the log each time; even if the help was only "one way" help.

Make sure that you have a complete file header comment at the top of each file, and that all of the information is correctly filled out.

```
# File:    FILENAME.py
# Author:  YOUR NAME
# Date:    THE DATE
# Section: YOUR DISCUSSION SECTION NUMBER
# E-mail:  YOUR_EMAIL@umbc.edu
# Description:
#    DESCRIPTION OF WHAT THE PROGRAM DOES
```

Instructions

For each of the questions below, you are given a problem that you must solve or a task you must complete.

Objective

Homework 5 is designed to give you lots of practice with lists, functions, loops, tuples, and designing your own programs. You should think carefully about how to break each problem down into smaller functions.

Remember to enable Python 3 before running and testing your code:

```
scl enable python33 bash
```

Coding Standards

Prior to this assignment, you should be familiar with the Coding Standards, available on Blackboard under "Assignments" and linked on the course website at the top of the "Assignments" page.

For now, you should pay special attention to the sections about:

- Comments (Header and In-Line)
  - ***Function header comments***
- Constants
  - For Homework 5, you must use constants instead of magic numbers. Magic strings are also forbidden.
- Make sure to **read the last page of the Coding Standards document**, which prohibits the use of certain tools and Python keywords

Questions

Each question is worth the indicated number of points. Following the coding standards is worth 6 points. Failing to have complete file headers or failing to have correctly named files will cause a deduction of points.

hw5_part1.py **(Worth 6 points)**

Create a program that asks the user to enter two coordinate points, and then calculates and prints out the distance between them. The program should follow the sample output shown below, and should use the distance formula (specified on Wikipedia). The program ***may not*** import the math library.
(HINT: Raising a number to the power of 0.5 is the same as a square root.)

The program must contain a **`main()`** and a function called **`distance()`**, implemented as described in the function header comment given below. (You should include this function header comment in your own code.)

```
##############################################################
# distance() calculates the distance between two points
# Input:     point1; a tuple containing an X and a Y value
#            point2; a tuple containing an X and a Y value
# Output:    ans;    a float of the calculated distance
```

Here is some sample output, with the user input in **blue**.
(Yours does not have to match this word for word, but it should be similar.)

```
bash-4.1$ python hw5_part1.py
This program will ask for two points, and
will compute the distance between the two.
Please enter a value for x1: -1
Please enter a value for y1: 7
Please enter a value for x2: 8
Please enter a value for y2: 2
The distance between (-1, 7) and (8, 2) is 10.295630140987
```

(HINT: Printing a tuple will automatically print the parentheses around it too.)

hw5_part2.py **(Worth 10 points)**

Write a program that checks if a name given by the user is a common name based on its ending, using a few simple functions to break the task down.

The program must contain a **`main()`**, as well as the following 3 functions:

- **`isProperName()`**
  - Input: name, a string
  - Output: a Boolean
  - Task: check if name is all lowercase, with an uppercase first letter
    - Examples: **`"Hrabowski"`** → **`True`**
      `"McCrery"` → **`False`**

- **`isSameEnd()`**
  - Input: fullString and endString, two strings
  - Output: a Boolean
  - Task: check if fullString ends with endString (case sensitive)
    - Examples: **`"Kal-el", "el"`** → **`True`**
      `"Bermand", "man"` → **`False`**
      `"Son", "son"` → **`False`**
  - **This is a helper function for checkCommonEndings()**

- **`checkCommonEndings()`**
  - Input: name, a string
    commonEndings, a list of strings
  - Output: a string; the first common ending found in name;
    if no common ending found, return empty string
  - Tasks: check if name ends in any of the common endings
    - Examples: **`"Gibson", ["son", "on"]`** → **`"son"`**
      `"Smith", ["son", "on"]` → **`""`**

The program should ask the user for their name, re-prompting until they provide a "proper" name (all lowercase, except for an uppercase first character).
Once it receives a proper name, it should determine whether the name is common. If the name is common, it should state what ending it contains that makes it common; otherwise, it should state it is uncommon.

(See the next page for sample output and the list of common endings.)

You ***must*** use this list of common endings in your program, **in this order**:
"son", "stern", "man", "in", "er", "en", "el", “on”

Here is some sample output for **hw5_part2.py**, with the user input in **blue**.
(Yours does not have to match this word for word, but it should be similar.)

```
bash-4.1$ python hw5_part2.py
Welcome!  This program checks if your name is common.
Please enter your name: Gibson
Your name is common, since it ends in son
Thank you for using the common name checker!

bash-4.1$ python hw5_part2.py
Welcome!  This program checks if your name is common.
Please enter your name: McDonald
Sorry, but a proper name starts with a capital letter.
Please enter your name: ANGRY VIKING JR
Sorry, but a proper name starts with a capital letter.
Please enter your name: Ben
Your name is common, since it ends in en
Thank you for using the common name checker!

bash-4.1$ python hw5_part2.py
Welcome!  This program checks if your name is common.
Please enter your name: King
Your name is uncommon!  Congratulations!
Thank you for using the common name checker!

bash-4.1$ python hw5_part2.py
Welcome!  This program checks if your name is common.
Please enter your name: Mansfield
Your name is uncommon!  Congratulations!
Thank you for using the common name checker!

bash-4.1$ python hw5_part2.py
Welcome!  This program checks if your name is common.
Please enter your name: Man-man
Your name is common, since it ends in man
Thank you for using the common name checker!
```

hw5_part3.py **(Worth 18 points)**

This program allows the user to create a grocery list, with each food categorized by the food group that it belongs to.
The program must use a two dimensional list to accomplish this!

The user can continue entering items indefinitely, stopping only when they enter the sentinel value "STOP". After entering each item, they should be asked what food group the item belongs to (mentioning the food's name in the question), using the following options:

0. Veggie
1. Fruit
2. Protein
3. Grain
4. Dairy

Once the food item's name and the food group it belongs to have been received, the food should be added to the grocery list, categorized by the food group.

After the user indicates they are done, their complete grocery list should be printed back out to them, grouped by food group. The number of items in each food group should be displayed, followed by the foods on the list.

Think carefully about what functions you should create, and what constants you may need. (For simplicity's sake, the strings "Veggie", "Fruit", "Protein", "Grain", and "Dairy" do NOT need to be made constants.)

Your program *must* contain at least two user-created functions, not including the function for **`main()`**.

(HINT: To create an empty 2D list, try starting with an empty list, and appending additional empty lists (what will become rows/the food groups) to it, until you have the number of rows/food groups that you need. Think carefully about how to add an item to one of the "interior" food group lists.)

(See the next pages for sample output.)

Here is some sample output for **hw5_part3.py**, with the user input in **blue**. (Yours does not have to match this word for word, but it should be similar.)

```
bash-4.1$ python hw5_part3.py
Welcome!  This program allows you to create a grocery list

What would you like to buy? Enter 'STOP' to quit: bread
Please select the food group that bread belongs to
The food groups are:
0 - Veggie
1 - Fruit
2 - Protein
3 - Grain
4 - Dairy
Enter a number between 0 and 4 (inclusive): 3

What would you like to buy? Enter 'STOP' to quit: cheese
Please select the food group that cheese belongs to
The food groups are:
0 - Veggie
1 - Fruit
2 - Protein
3 - Grain
4 - Dairy
Enter a number between 0 and 4 (inclusive): 4

What would you like to buy? Enter 'STOP' to quit: STOP

You are buying 0 Veggie items:
You are buying 0 Fruit items:
You are buying 0 Protein items:
You are buying 1 Grain items:
         bread
You are buying 1 Dairy items:
         cheese
```

(See more sample output on the following page.)

Additional sample output for **hw5_part3.py**, with the user input in **blue**.
(Yours does not have to match this word for word, but it should be similar.)

For this page of sample output, we removed the listing of food groups – your own code should print them out every time the question is asked.

```
bash-4.1$ python hw5_part3.py
Welcome!  This program allows you to create a grocery list

What would you like to buy? Enter 'STOP' to quit: tuna
Please select the food group that tuna belongs to: 2
What would you like to buy? Enter 'STOP' to quit: bread
Please select the food group that bread belongs to: 3
What would you like to buy? Enter 'STOP' to quit: milk
Please select the food group that milk belongs to: 4
What would you like to buy? Enter 'STOP' to quit: bananas
Please select the food group that bananas belongs to: 1
What would you like to buy? Enter 'STOP' to quit: donuts
Please select the food group that donuts belongs to: 3
What would you like to buy? Enter 'STOP' to quit: cheese
Please select the food group that cheese belongs to: 4
What would you like to buy? Enter 'STOP' to quit: cereal
Please select the food group that cereal belongs to: 3
What would you like to buy? Enter 'STOP' to quit: yogurt
Please select the food group that yogurt belongs to: 4
What would you like to buy? Enter 'STOP' to quit: STOP

You are buying 0 Veggie items:
You are buying 1 Fruit items:
         bananas
You are buying 1 Protein items:
         tuna
You are buying 3 Grain items:
         bread
         donuts
         cereal
You are buying 3 Dairy items:
         milk
         cheese
         yogurt
```

(See more sample output on the following page.)

Additional sample output for **hw5_part3.py**, with the user input in **blue**.
(Yours does not have to match this word for word, but it should be similar.)

```
bash-4.1$ python hw5_part3.py
Welcome!  This program allows you to create a grocery list

What would you like to buy? Enter 'STOP' to quit:
happiness
Please select the food group that happiness belongs to
The food groups are:
0 - Veggie
1 - Fruit
2 - Protein
3 - Grain
4 - Dairy
Enter a number between 0 and 4 (inclusive): -1
That number is not allowed.  Please try again!
Enter a number between 0 and 4 (inclusive): 5
That number is not allowed.  Please try again!
Enter a number between 0 and 4 (inclusive): 42
That number is not allowed.  Please try again!
Enter a number between 0 and 4 (inclusive): 0

What would you like to buy? Enter 'STOP' to quit: STOP


You are buying 1 Veggie items:
         happiness
You are buying 0 Fruit items:
You are buying 0 Protein items:
You are buying 0 Grain items:
You are buying 0 Dairy items:
```

Submitting

Once your **`hw5_part1.py`**, **`hw5_part2.py`**, and **`hw5_part3.py`** files are complete, it is time to turn them in with the **`submit`** command. (You may also turn in individual files as you complete them. To do so, only **`submit`** those files that are complete.)

You must be logged into your account on GL, and you must be in the same directory as your Homework 5 Python files. To double-check you are in the directory with the correct files, you can type **`ls`**.

```
linux1[3]% ls
hw5_part1.py  hw5_part2.py  hw5_part3.py
linux1[4]%
```

To submit your Homework 5 Python files, we use the **`submit`** command, where the class is **`cs201`**, and the assignment is **`HW5`**. Type in (all on one line) **`submit cs201 HW5 hw5_part1.py hw5_part2.py hw5_part3.py`** and press enter.

```
linux1[4]% submit cs201 HW5 hw5_part1.py hw5_part2.py
hw5_part3.py
Submitting hw5_part1.py...OK
Submitting hw5_part2.py...OK
Submitting hw5_part3.py...OK
linux1[5]%
```

If you don't get a confirmation like the one above, check that you have not made any typos or errors in the command.

You can check that your homework was submitted by following the directions in Homework 0. Double-check that you submitted your homework correctly, since **an empty file will result in a grade of zero for this assignment.**