CMSC 201 Spring 2019

Homework 4 – Lists (and More)

Assignment: Homework 4 – Lists (and More)
Due Date: Friday, March 8th, 2019 by 11:59:59 PM
Value: 40 points

Collaboration: For Homework 4, **collaboration is not allowed** – you must work individually. You may still come to office hours for help, but you may not work with any other CMSC 201 students.

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.

You should already be familiar with one-way, two-way, and multi-way decision structures. You should also be familiar with **`while`** loops and lists.

This assignment will focus on using lists to store information, as well as using while loops to traverse these lists and decision structures to control the flow of the program.

At the end, your Homework 4 files must run without any errors.

NOTE: Your filenames for this homework must match the given ones exactly.
And remember, filenames are case sensitive!

Additional Instructions – Creating the hw4 Directory

During the semester, you'll want to keep your different Python programs organized, organizing them in appropriately named folders (also known as directories).

Just as you did for previous homeworks, you should create a directory to store your Homework 4 files. We recommend calling it **`hw4`**, and creating it inside the **`Homeworks`** directory inside the **`201`** directory.

If you need help on how to do this, refer back to the detailed instructions in Homework 1. (You don't need to make a separate folder for each file. You should store all of the Homework 4 files in the same **`hw4`** folder.)

Coding Standards

Prior to this assignment, you should re-read 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:

- Naming Conventions
- Use of Whitespace
- Comments (specifically, File Header Comments)
- Line Length
- Constants
    - For Homework 4, 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

Additional Specifications

For this assignment, **you must use** `main()` as seen in your **`lab2.py`** file, and as discussed in class.

For this assignment, you should pay attention to each problem's instructions on using "input validation." For example, the user may enter a negative value, but your program may require a positive value. **Make sure to follow each part's instructions about input validation.**

If the user enters a different type of data than what you asked for, your program may crash. This is acceptable.

Do note that you do not have to use lists for every part of this homework, only those where it is explicitly specified. However, you may find other parts are easier with the use of lists.

Questions

Each question is worth the indicated number of points. Following the coding standards is worth 4 points. If you do not have complete file headers and correctly named files, you will lose points.

hw4_part1.py **(Worth 6 points)**

For this part of the homework you will write code to draw an isosceles right triangle. (A right triangle in which the height and base are the same size.)

Your program should prompt the user for these inputs, **in exactly this order:**

1. The height of their triangle
2. The symbol the triangle will be *outlined* in
3. The symbol the triangle will be *filled* with

For these inputs, you can assume the following:

- The height will be a positive integer (greater than zero)
- The symbols will be a single character each

Use the *first symbol* to draw an upside-down isosceles right triangle of the height chosen by the user. The triangle should be filled in with the *second symbol*, in the cases in which there is interior space to fill.

(See the next page for sample output.)

Here is some sample output for **hw4_part1.py**, with the user input in **blue**. (Yours does not have to match this word for word, but it should be similar.)

```
linux1[3]% python hw4_part1.py
Please enter the height of the triangle: 1
Please enter a symbol for the triangle outline: W
Please enter a symbol for the triangle fill: m
W

linux1[4]% python hw4_part1.py
Please enter the height of the triangle: 3
Please enter a symbol for the triangle outline: Q
Please enter a symbol for the triangle fill: q
QQQ
QQ
Q

linux1[5]% python hw4_part1.py
Please enter the height of the triangle: 7
Please enter a symbol for the triangle outline: x
Please enter a symbol for the triangle fill: O
xxxxxxx
xOOOOx
xOOOx
xOOx
xOx
xx
x
```

(NOTE: Because the text is taller than it is wide, the triangle may not appear to be an isosceles right triangle, but the number of characters in the height and the width are still the same.)

HINT: You can keep the `print()` *function from printing on a new line by using* `end=""` *at the end:* `print("Hello", end="")`. *If you do want to print a new line, you can call print without an argument:* `print()`.

hw4_part2.py **(Worth 6 points)**

Create a program that will have the user enter a list of the superpowers they would have. The user can continue entering powers indefinitely, stopping only when they enter the sentinel value "QUIT".

If the user enters a power that they have already entered (and therefore already exists in the list), the user must be notified, and the power should not be added to the list again.

Once the user has completed the list, the program should print out the total number of superpowers they have. It should then print out whether they are underpowered, perfect, or too strong, based on a desired superpower count of 3 total powers. The program **must** make use of a list to accomplish these tasks!

(See the next page for sample output.)

Here is some sample output for **hw4_part2.py**, with the user input in **blue**. (Yours does not have to match this word for word, but it should be similar.)

```
linux1[6]% python3 hw4_part2.py
Please enter a superpower ('QUIT' to stop): talk to dogs
Please enter a superpower ('QUIT' to stop): call dogs to me
Please enter a superpower ('QUIT' to stop): dogs live forever
Please enter a superpower ('QUIT' to stop): QUIT
You have 3 superpowers.
You're a perfect superhero!

linux1[7]% python3 hw4_part2.py
Please enter a superpower ('QUIT' to stop): heat vision
Please enter a superpower ('QUIT' to stop): QUIT
You have 1 superpowers.
You are underpowered!

linux1[8]% python3 hw4_part2.py
Please enter a superpower ('QUIT' to stop): flight
Please enter a superpower ('QUIT' to stop): telekinesis
Please enter a superpower ('QUIT' to stop): super speed
Please enter a superpower ('QUIT' to stop): super strength
Please enter a superpower ('QUIT' to stop): precognition
Please enter a superpower ('QUIT' to stop): QUIT
You have 5 superpowers.
You're too strong!
```

hw4_part3.py **(Worth 10 points)**

Write a program that asks the user to enter a password, and then checks it for a few different requirements before approving it as secure and repeating the final password to the user.

The program must re-prompt the user until they provide a password that satisfies all of the conditions. It must also tell the user each of the conditions they failed, and how to fix it.
If there is more than one thing wrong (*e.g.*, no lowercase, and longer than 15 characters), the program must print out all of the things that are wrong, and how to fix them.

The program follows these rules for passwords:

1. The password must contain at least one lowercase letter.
2. The password must contain at least one uppercase letter.
3. The password must be between 6 and 20 characters, inclusive.
    a. If the password is between 6 and 13 characters, inclusive, it must contain a "7" somewhere in the password.
    b. If the password is between 14 and 20 characters, inclusive, it must contain a "2" somewhere in the password.
4. The password cannot contain the characters "0" and "O" (zero and uppercase o) at the same time. (It can contain a "0" or an "O", just not both at the same time. It may also contain neither.)

For this part of the homework, you ***must*** have an in-line comment at the top of **each** of your program's individual `if`, `elif`, and `else` statements, explaining what is being checked by that conditional.

(HINT: Think carefully about what your conditionals should look like. If necessary, draw a truth table to help figure out what different inputs will do. Using a Boolean flag will also likely make this easier.)

(See the next page for sample output.)

Here is some sample output for **hw4_part3.py**, with the user input in **blue**. (Yours does not have to match this word for word, but it should be similar.)

```
linux1[8]% python hw4_part3.py
Please enter a password: dogs
Password must have an uppercase character
Password must be at least 6 characters
Please enter a password: DOGS
Password must have a lowercase character
Password must be at least 6 characters
Please enter a password: Dogs
Password must be at least 6 characters
Please enter a password: Doggos
Shorter passwords must contain a 7
Please enter a password: 7Doggos
Thank you for picking the secure password 7Doggos

linux1[9]%  python hw4_part3.py
Please enter a password: thisMustBeSecureItsLongAlso27
Password must be no longer than 20 characters
Please enter a password: abcdefghijklmnopqrst
Password must have an uppercase character
Longer passwords must contain a 2
Please enter a password: 2and7EQUALSnine
Thank you for picking the secure password 2and7EQUALSnine

linux1[10]%  python hw4_part3.py
Please enter a password: O_and_0
Shorter passwords must contain a 7
Password cannot contain a O and a 0 at the same time
Please enter a password: O_and_7
Thank you for picking the secure password O_and_7

linux1[11]%  python hw4_part3.py
Please enter a password: greatPassword7!
Longer passwords must contain a 2
Please enter a password: greatPassword2!
Thank you for picking the secure password greatPassword2!
```

hw4_part4.py **(Worth 6 points)**

This program allows the user to create a grocery list, which will give them a breakdown of each item they want to buy, including the amount that should be purchased.
The program must use two separate lists to accomplish this!

The user can continue entering items indefinitely, stopping only when they enter the sentinel value "END". After entering each item, they should be asked how many of that item they want to buy.

After their list is complete, it should be printed back out to them in order (with each line containing the amount to be purchased and the item) followed by the total number of items to buy at the grocery store.

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.)

```
linux1[12]%  python3 hw4_part4.py
Please enter an item, or 'END' to stop: cheese
Please enter the amount to buy: 4
Please enter an item, or 'END' to stop: pizza
Please enter the amount to buy: 2
Please enter an item, or 'END' to stop: soda
Please enter the amount to buy: 12
Please enter an item, or 'END' to stop: avocado
Please enter the amount to buy: 6
Please enter an item, or 'END' to stop: pickles
Please enter the amount to buy: 70
Please enter an item, or 'END' to stop: END
Here is your grocery list:
Purchase 4 of cheese
Purchase 2 of pizza
Purchase 12 of soda
Purchase 6 of avocado
Purchase 70 of pickles
Total of 94 items to be purchased
```

hw4_part5.py **(Worth 8 points)**

Finally, create a program that outputs a specific response based on whether a word starts with "pre" or "post" (or neither).

First, the program must ask the user to enter ten different words, and store those words in a list.
The program **must** make use of a list to accomplish these tasks!

Once the list contains the ten words, the program must use the following rules to determine how each word should be printed back out to the user.

1. If the word starts with "**`pre`**" print out:
   `You should <word without pre> early`
2. If the word starts with "**`post`**" print out:
   `You should <word without post> later`
3. Otherwise, simply print the item:
   `You can <word> whenever!`

For these inputs, you can assume the following:

- The words entered will be in all lowercase
- The words entered will be at least 5 characters long

(HINT: You will want to use string slicing to check if the string begins with "pre" or "post" and to print out the word without those parts. Review Lecture 09 (Strings) for details on how to use slicing.)

You may **not** use any built-in Python methods, such as **`beginswith()`**, to check if the string begins with "pre"/"post" or not.

(See the next page for sample output.)

Here is some sample output for **hw4_part5.py**, with the user input in **blue**. (Yours does not have to match this word for word, but it should be similar.)

```
linux1[13]% python3 hw4_part5.py
Please enter a word: prepare
Please enter a word: postmodern
Please enter a word: pretest
Please enter a word: postpone
Please enter a word: super
Please enter a word: reprep
Please enter a word: ripost
Please enter a word: imposter
Please enter a word: premium
Please enter a word: posterboard

You should pare early
You should modern later
You should test early
You should pone later
You can super whenever!
You can reprep whenever!
You can ripost whenever!
You can imposter whenever!
You should mium early
You should erboard later
```

Submitting

Once your **hw4_part1.py**, **hw4_part2.py**, **hw4_part3.py**, **hw4_part4.py**, and **hw4_part5.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 4 Python files. To double-check you are in the directory with the correct files, you can type **ls**.

```
linux1[14]% ls
hw4_part1.py  hw4_part3.py  hw4_part5.py
hw4_part2.py  hw4_part4.py
linux1[15]%
```

To submit your Homework 4 Python files, we use the **submit** command, where the class is **cs201**, and the assignment is **HW4**. Type in (all on one line) **submit cs201 HW4 hw4_part1.py hw4_part2.py hw4_part3.py hw4_part4.py hw4_part5.py** and press enter.

```
linux1[16]% submit cs201 HW4 hw4_part1.py hw4_part2.py
hw4_part3.py hw4_part4.py hw4_part5.py
Submitting hw4_part1.py...OK
Submitting hw4_part2.py...OK
Submitting hw4_part3.py...OK
Submitting hw4_part4.py...OK
linux1[17]%
```

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.**