CMSC 201 Spring 2017

Homework 4 – Lists (and Loops and Strings)

Assignment: Homework 4 – Lists (and Loops and Strings)
Due Date: Friday, March 3rd, 2017 by 8:59:59 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, lists, and strings.

This assignment will focus on using lists to store information, as well as using while loops to traverse these lists. Slicing of strings will also be needed for some parts, and concatenation needed for using **`input()`** with variables.

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).

You should create a directory in which 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 four Homework 4 files in the same **`hw4`** folder.)

Objective

Homework 4 is designed to help you practice using **`while`** loops, lists, control statements like **`if/else`**, concatenation needed for using **`input()`**, and algorithmic thinking.

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

```
scl enable python33 bash
```

Task

For all of the programs below (except for part 2), you must use a list to store information. For part 2, you will need to use a **`while`** loop, most likely one that involves a Boolean flag.

Specification

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:

- Naming Conventions
- Use of Whitespace
- Comments (Header and In-Line)
- ***Constants***
    - ***For Homework 4, you must 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

You should start forming good habits now. Make sure to pay attention to your TA's feedback when you receive your Homework 4 grade back.

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 do not need to worry about any "input validation."

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

For example, if your program asks the user to enter a whole number, it is acceptable if your program crashes if they enter something else like "dog" or "twenty" or "88.2" instead.

Here is what that might look like:

```
Please enter a number: twenty
Traceback (most recent call last):
  File "test_file.py", line 10, in <module>
    num = int(input("Please enter a number: "))
ValueError: invalid literal for int() with base 10: 'twenty'
```

Questions

Each question is worth the indicated number of points. Following the coding standards, having complete file headers, and having correctly named files is worth 6 points.

hw4_part1.py **(Worth 4 points)**

Create a program that allows the user to add items to a "to do" list.

The user can continue entering items indefinitely, and the program should only stop re-prompting them when they enter the sentinel value "QUIT".

After their list is complete, it should be printed back out to them, with each to do item on a separate line. Each line must start with a "box" for them to use to check the item off (the box is two square brackets and a space: `[ ]`).

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 hw4_part1.py
Please enter an item (QUIT to quit): create homework
Please enter an item (QUIT to quit): release homework
Please enter an item (QUIT to quit): create midterm
Please enter an item (QUIT to quit): look at dog pictures
Please enter an item (QUIT to quit): sleep
Please enter an item (QUIT to quit): go to the gym
Please enter an item (QUIT to quit): QUIT

Here is your to do list:
[ ] create homework
[ ] release homework
[ ] create midterm
[ ] look at dog pictures
[ ] sleep
[ ] go to the gym
```

hw4_part2.py **(Worth 14 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.

*(**WARNING**: This part of the homework is the most challenging, so budget plenty of time and brain power. And read the instructions carefully!)*

The program follows these three rules for passwords:

1. If the password is all *lowercase*, it must contain an exclamation mark.
2. The password cannot be less than eight characters long.
3. The password cannot be longer than twenty characters long.

The program must re-prompt the user until they provide a password that satisfies all of the conditions above. 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.*, too long, and all lowercase with no exclamation mark), the program must print out both of the things that are wrong and how to fix them.

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.)

(PRO TIP: The livecoding file posted for creating a valid password from Lecture 8 may be a good place to start if you're stuck.)

(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.)

```
bash-4.1$ python hw4_part2.py
Please enter a password: science
The password is all lowercase, so it must contain
the character ! to be secure.
The password is too short.  Must be at least 8 characters.
Please enter a password: sciencerules
The password is all lowercase, so it must contain
the character ! to be secure.
Please enter a password: science!rules
Thanks for picking the super-secure password science!rules

bash-4.1$ python hw4_part2.py
Please enter a password: scienceIsTheCoolestThingEver
The password is long.  Must be no more than 20 characters.
Please enter a password: scienceIsPrettyCool
Thanks for picking the super-secure password
scienceIsPrettyCool

bash-4.1$ python hw4_part2.py
Please enter a password: 1234567
The password is all lowercase, so it must contain
the character ! to be secure.
The password is too short.  Must be at least 8 characters.
Please enter a password: 12345678
The password is all lowercase, so it must contain
the character ! to be secure.
Please enter a password: 12345678!
Thanks for picking the super-secure password 12345678!

bash-4.1$ python hw4_part2.py
Please enter a password: SOEXCITED
Thanks for picking the super-secure password SOEXCITED
```

hw4_part3.py **(Worth 6 points)**

This program allows the user to create a shopping checklist that will remind them not only of the item they want to purchase, but also the quantity desired for each item.

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 "STOP". 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. Each line must contain the item name and the quantity of that item.

(Your program must NOT prompt for a quantity after the user enters "STOP".)

(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.)

```
bash-4.1$ python hw4_part3.py
Please enter an item ('STOP' to quit): eggs
Please enter the quantity you want to purchase: 24
Please enter an item ('STOP' to quit): ramen
Please enter the quantity you want to purchase: 3
Please enter an item ('STOP' to quit): milk
Please enter the quantity you want to purchase: 2
Please enter an item ('STOP' to quit): bagels
Please enter the quantity you want to purchase: 6
Please enter an item ('STOP' to quit): STOP

Here is your shopping checklist:
eggs    ( 24 )
ramen   ( 3 )
milk    ( 2 )
bagels  ( 6 )


bash-4.1$ python hw4_part3.py
Please enter an item ('STOP' to quit): python book
Please enter the quantity you want to purchase: 1
Please enter an item ('STOP' to quit): calc book
Please enter the quantity you want to purchase: 1
Please enter an item ('STOP' to quit): chocolate
Please enter the quantity you want to purchase: 19
Please enter an item ('STOP' to quit): STOP

Here is your shopping checklist:
python book     ( 1 )
calc book       ( 1 )
chocolate       ( 19 )
```

(HINT: If you want your numbers to line up like in the sample output, try putting a "`\t`" (backslash T) before your opening parentheses.)

hw4_part4.py **(Worth 10 points)**

Finally, create a program that determines whether a subject can be studied or not.

First, the program must ask the user to enter ten different subjects, and store those subjects in a list.

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

1. If the subject ends in "**`ology`**" print out:
   `You can study SUBJECT`
2. Otherwise, simply print the subject:
   `SUBJECT is not real!`

For these inputs, you can assume the following:

- The words entered will be in all lowercase
- The words entered will be at least 6 characters long

(HINT: You will want to use string slicing to check if the string ends with "ology". Review Lecture 08 (Strings and More) for details on how to use slicing.)

You may **not** use any built-in Python methods, such as **`endswith()`**, to check if the string ends in "ology" or not.

(See the next page for sample output.)

Here is some sample output for **hw4_part4.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 hw4_part4.py
Please enter a subject of study: dragonology
Please enter a subject of study: biology
Please enter a subject of study: ology
Please enter a subject of study: astropsychology
Please enter a subject of study: pteridology
Please enter a subject of study: computer science
Please enter a subject of study: english
Please enter a subject of study: biology rocks
Please enter a subject of study: sleeping
Please enter a subject of study: running

You can study dragonology
You can study biology
You can study ology
You can study astropsychology
You can study pteridology
computer science is not real!
english is not real!
biology rocks is not real!
sleeping is not real!
running is not real!
```

Submitting

Once your **hw4_part1.py**, **hw4_part2.py**, **hw4_part3.py**, and **hw4_part4.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[3]% ls
hw4_part1.py  hw4_part2.py  hw4_part3.py  hw4_part4.py
linux1[4]%
```

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** and press enter.

```
linux1[4]% submit cs201 HW4 hw4_part1.py hw4_part2.py
hw4_part3.py hw4_part4.py
Submitting hw4_part1.py...OK
Submitting hw4_part2.py...OK
Submitting hw4_part3.py...OK
Submitting hw4_part4.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.**