6CMSC 201 Fall 2018

Homework 4 – Lists (and More)

Assignment: Homework 4 – Lists (and More)
Due Date: Friday, October 5th, 2018 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 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 previous assignments.

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 a sideways pyramid with a character.

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

1. The symbol the pyramid will be *built with*
2. The width of their pyramid

You can assume the following the symbol will be a single character.

You must validate the width of the pyramid to be positive (bigger than zero).

Use the *symbol* to draw sideways pyramid of the width chosen by the user.

(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.)

```
linux2[7]% python3 hw4_part1.py
Please enter a single character: @
Enter the width of the pyramid: 0
Enter the width of the pyramid: -1
Enter the width of the pyramid: 1
@

linux2[8]% python3 hw4_part1.py
Please enter a single character: $
Enter the width of the pyramid: 3
$
$$
$$$
$$
$

linux2[9]% python3 hw4_part1.py
Please enter a single character: x
Enter the width of the pyramid: 4
x
xx
xxx
xxxx
xxx
xx
x
```

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 wish list of items they want to purchase. The user can continue entering names indefinitely, stopping only when they enter the sentinel value "Q".

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

Once the user has completed the list, the program should print out the total number of items on the list. It should then print out whether the wishlist has too many, too few, or a perfect number of items, based on a desired list size of 5 items. 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.)

```
linux2[9]% python3 hw4_part2.py
Enter an item you want ('Q' to quit): apples
Enter an item you want ('Q' to quit): pears
Enter an item you want ('Q' to quit): TV
Enter an item you want ('Q' to quit): phone
Enter an item you want ('Q' to quit): juice
Enter an item you want ('Q' to quit): Q
There are 5 items on your list.
You have the perfect number of items.

linux2[10]% python3 hw4_part2.py
Enter an item you want ('Q' to quit): Q
There are 0 items on your list.
There are not enough items on your list.

linux2[11]% python3 hw4_part2.py
Enter an item you want ('Q' to quit): headphones
Enter an item you want ('Q' to quit): wallet
Enter an item you want ('Q' to quit): headphones
The item headphones is already on your wishlist.
Enter an item you want ('Q' to quit): chips
Enter an item you want ('Q' to quit): Q
There are 3 items on your list.
There are not enough items on your list.

linux2[12]% python3 hw4_part2.py
Enter an item you want ('Q' to quit): chair
Enter an item you want ('Q' to quit): bed
Enter an item you want ('Q' to quit): sheets
Enter an item you want ('Q' to quit): pillow
Enter an item you want ('Q' to quit): blanket
Enter an item you want ('Q' to quit): comforter
Enter an item you want ('Q' to quit): Q
There are 6 items on your list.
There are too many items on your list.
```

hw4_part3.py **(Worth 8 points)**

You are tasked with writing the input validation component of a historical database. You enter a year into this database, and it should bring up a bunch of facts about that year. However, only certain years can be looked up. That's where you come in!

The program must re-prompt the user until they provide a year 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.*, year is odd, and it's bad year), the program must print out all of the things that are wrong, and how to fix them.

The program follows these rules for years:

1. The year must be after 1800.
2. The year must be before 2000.
3. The year must be even.
4. The year must not be a "bad year"
    a. Bad years include: `1812, 1863, 1918, 1929, 1932, 1939, 1941, 1963, 1968, 1979`

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 files posted for Lecture 7 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_part3.py**, with the user input in **blue**. (Yours does not have to match this word for word, but it should be similar.)

```
linux2[4]% python3 hw4_part3.py
Enter a year: 1900
Searching the database for... 1900

linux2[5]% python3 hw4_part3.py
Enter a year: 1932
That was a bad year!
Enter a year: 1968
That was a bad year!
Enter a year: 1969
Year is odd!
Enter a year: 1979
Year is odd!
That was a bad year!
Enter a year: 1892
Searching the database for... 1892

linux2[6]% python3 hw4_part3.py
Enter a year: 1789
Year is too small!
Year is odd!
Enter a year: 2013
Year is too big!
Year is odd!
Enter a year: 1804
Searching the database for... 1804
```

hw4_part4.py **(Worth 8 points)**

This program allows people to enter their favorite month, and then determine how many votes were cast for each of the four seasons. First, the program should display a menu of choices for the 12 months. Then, the user enters votes for their favorite month. When the votes are done being cast, the program displays the votes made by season.

The program must use two separate lists to accomplish this!

The user can continue entering favorite months indefinitely, stopping only when they enter the sentinel value `"-1".` Each time a vote is cast, it should print out the name of the month that was voted for. After they stop entering in votes, the program should print out how many votes were made in each season

After their list is complete the votes cast by season should be printed ack out to them. Here is the breakdown for months by season:

- Fall: **September, October, November**
- Winter: **December, January, February**
- Spring: **March, April, May**
- Summer: **June, July, August**

You can assume the user will enter a valid month (1-12 inclusive).

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

```
linux2[4]% python3 hw4_part4.py
1 - January
2 - February
3 - March
4 - April
5 - May
6 - June
7 - July
8 - August
9 - September
10 - October
11 - November
12 - December
What is your favorite month? (-1 to quit): -1
0 votes for winter
0 votes for spring
0 votes for summer
0 votes for fall

linux2[5]% python3 hw4_part4.py
[[ MENU OMITTED TO FIT PAGE ]]
What is your favorite month? (-1 to quit): 1
What is your favorite month? (-1 to quit): 1
What is your favorite month? (-1 to quit): 1
What is your favorite month? (-1 to quit): 1
What is your favorite month? (-1 to quit): 5
What is your favorite month? (-1 to quit): 4
What is your favorite month? (-1 to quit): 6
What is your favorite month? (-1 to quit): 5
What is your favorite month? (-1 to quit): 9
What is your favorite month? (-1 to quit): -1
4 votes for winter
3 votes for spring
1 votes for summer
1 votes for fall
```

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

Finally, create a program that isolates the vowels in a sequence of letters entered one at a time.

First, the program must ask the user to enter an indefinite number of letters (they should enter all caps "`QUIT`" to stop) and store those letters in a list. Once the user has stopped entering letters, the program should output those letters on the same line, with emphasis on the vowels. It should count how many vowels and consonants there are, and report those numbers at the end as well.

For these inputs, you can assume the following:

- The letters entered will be in all lowercase
- Only valid letters will be entered
    - no special characters, numbers, etc.

You place emphasis on the vowels in this problem by printing them out surrounded by square brackets.

(See the next page for sample output.)

Here is some sample output for **hw4_part5.py**, with the user input in **blue**.

(Your output must match this word for word.)

```
linux2[4]% python3 hw4_part5.py
Please enter letter of your word ('QUIT'to exit) a
Please enter letter of your word ('QUIT'to exit) p
Please enter letter of your word ('QUIT'to exit) p
Please enter letter of your word ('QUIT'to exit) l
Please enter letter of your word ('QUIT'to exit) e
Please enter letter of your word ('QUIT'to exit) QUIT
[ a ]ppl[ e ]
There were a total of 2 vowels and 3 consonants

linux2[5]% python3 hw4_part5.py
Please enter letter of your word ('QUIT'to exit) q
Please enter letter of your word ('QUIT'to exit) u
Please enter letter of your word ('QUIT'to exit) i
Please enter letter of your word ('QUIT'to exit) c
Please enter letter of your word ('QUIT'to exit) k
Please enter letter of your word ('QUIT'to exit) f
Please enter letter of your word ('QUIT'to exit) o
Please enter letter of your word ('QUIT'to exit) x
Please enter letter of your word ('QUIT'to exit) QUIT
q[ u ][ i ]ckf[ o ]x
There were a total of 3 vowels and 5 consonants

linux2[6]% python3 hw4_part5.py
Please enter letter of your word ('QUIT'to exit) QUIT

There were a total of 0 vowels and 0 consonants

linux2[7]% python3 hw4_part5.py
Please enter letter of your word ('QUIT'to exit) y
Please enter letter of your word ('QUIT'to exit) e
Please enter letter of your word ('QUIT'to exit) l
Please enter letter of your word ('QUIT'to exit) l
Please enter letter of your word ('QUIT'to exit) o
Please enter letter of your word ('QUIT'to exit) w
Please enter letter of your word ('QUIT'to exit) QUIT
y[ e ]ll[ o ]w
There were a total of 2 vowels and 4 consonants
```

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[3]% ls
hw4_part1.py  hw4_part3.py  hw4_part5.py
hw4_part2.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 hw4_part5.py** and press enter.

```
linux1[4]% 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[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.**