CMSC 201 Spring 2016
Lab 06 – Lists

Assignment: Lab 06 – Lists
Due Date: During discussion, March 21st through March 25th
Value: 10 points

Lab 6 focuses on using lists and loops, but is also a review of much of the material we have covered so far. Although understanding individual concepts is very important, being able to put them together in new and interesting ways is what will allow you to create interesting computer programs.

To complete this lab you will need to use lists, indexing, for loops, user input, interactive while loops, decision structures (`if/elif/else`), and `print()` statements. (Because much of this material has been covered in previous labs, the pre-lab review will be shorter than normal, and may contain material from previous lab descriptions.)

Part 1: Lists and Indexing

Lists are an easy way to hold lots of individual pieces of data without needing to make lots of variables. They are a type of *data structure*, which are specialized ways of organizing and storing data.

In order to get a specific variable, or *element*, from a list, we need to access that *index* of the list. NOTE: Lists don't starting counting from 1 – the first element in the list is at index 0.

For example, the following line of code creates a list called **`names`**:

```
names = ["Aya", "Brad", "Carlos", "David", "Emma"]
```

Which creates the list (called **`names`**) below:

Part 2: For Loops

We can use **`for`** loops to perform two different actions: *iterating* over a list, or performing an action a certain number of times. We will focus on *iterating* over a list – this means moving through a list, one element at a time.

For example:

```
list_of_fruits = ["kiwi", "banana", "peach"]
for fruit in list_of_fruits:
    print("I ate a", fruit)
```

When run, the code above will print the following:

```
I ate a kiwi
I ate a banana
I ate a peach
```

Sometime we may want to display a list and number its contents. The easiest way to do this is to use **`range`** to generate a list of the list's indices (from 0 to the length of the list minus 1). We then use this in a **`for`** loop, and use the loop variable as both the index to access each element, and as a counter of how many elements have been accessed so far.

```
subjects = ["dragonology", "chem", "english", "bio"]

# we want to loop over the length of the list
for i in range( len(subjects) ):
    # numbering will start at 0
    print(i + 1, ":", subjects[i])
```

When run, the code above will print the following:

```
1 : dragonology
2 : chem
3 : english
4 : bio
```

Part 3: While Loops

A **`while`** loop statement in Python repeatedly executes a target statement as long as a given Boolean condition evaluates to **`True`**.

The syntax of a **`while`** loop in the Python programming language is:

```
while CONDITION:
    STATEMENTS(S)
```

Here, **`STATEMENT(S)`** may be a single statement or a *block* of statements. The condition can be any expression, as long as it evaluates to either **`True`** or **`False`**. (Remember, any non-zero value is seen as "**`True`**" by Python.) The **`while`** loop continues to run as long as (while) the condition is still **`True`**.

Part 4: Interactive While Loops

Another way to use a **`while`** loop is as an *interactive* or *sentinel* loop. An interactive (sentinel) loop continues to process data until reaching a special value that signals the end. The special value is called the *sentinel*.

Here is the pseudocode for an interactive loop in Python:

```
Get the first data item from the user
While data item is not the sentinel
    Process the data item
    Get the next data item from the user
```

One of the scenarios in which we can implement this type of loop is a version of our grocery list program that allows us to enter as many items as we like. Although it is similar to previous versions, the interactive (sentinel) while loop of the grocery list program allows us to enter as many items as we like until the sentinel value of **`"exit"`** is entered.

```
def main():
    grocery_list = []   # initialize the list to be empty
    # get the initial user value
    userVal = input("Enter an item, or 'exit' to end: ")

    # run the while loop until the user enters "exit"
    while userVal != "exit":
        grocery_list.append(userVal)
        # get another value from the user
        userVal = input("Enter an item, or 'exit' to end: ")

    # once the user is done with the list, print it out
    for g in grocery_list:
        print("Remember to buy", g)

main()
```

Part 5A: Writing Your Program

After logging into GL, navigate to the **Labs** folder inside your **201** folder. Create a folder there called **lab6**, and go inside the newly created **lab6** directory.

```
linux2[1]% cd 201
linux2[2]% cd Labs
linux2[3]% pwd
/afs/umbc.edu/users/k/k/k38/home/201/Labs
linux2[4]% mkdir lab6
linux2[5]% cd lab6
linux2[6]% pwd
/afs/umbc.edu/users/k/k/k38/home/201/Labs/lab6
linux2[7]%
```

To open the file for editing, type

emacs tv_shows.py

and hit enter.

The first thing you should do in your new file is create and fill out the comment header block at the top of your file. Here is a template:

```
# File:          tv_shows.py
# Author:        YOUR NAME
# Date:          TODAY'S DATE
# Section:       YOUR SECTION NUMBER
# E-mail:        USERNAME@umbc.edu
# Description:   YOUR DESCRIPTION GOES HERE AND HERE
#                YOUR DESCRIPTION CONTINUED SOME MORE
```

Now you can start writing your code for the lab, following the instructions in Parts 5B and 5C.

Part 5B: Printing the Television Shows

For Lab 6, you will be writing up a short program to help the user and their friends decide which television show to watch. The program will print out the list of possible shows, and then everyone can vote on which show they would like to watch. The program will close voting if someone enters "0" as their choice, and will output the total votes for each show.

You will be coding Lab 6 in an incremental manner – in other words, you will code up one piece of the lab and test that it works **before** moving on to the next piece. Incremental development is a very effective way of tackling a problem, in part because it makes it easier to pinpoint where an error occurs when you are only working on a small part of the code at a time.

The first piece of code we will write is printing the television shows. Copy the list below, **`shows`**, into your program.

```
shows = ["Daredevil", "Fargo","Limitless", "Elementary",
"Brooklyn 99", "Empire", "Supergirl"]
```

To print the inventory, you should write code that will print two different things on each line:

- **The number of the television show** (start counting from 1)
- **The television show's name** (*e.g.*, Daredevil, Fargo, etc.)

When you have completed this, the output should look something like this:

```
bash-4.1$ python tv_shows.py
1 - Daredevil
2 - Fargo
3 - Limitless
4 - Elementary
5 - Brooklyn 99
6 - Empire
7 - Supergirl
```

(There are hints on the next page if you need them.)

Try to solve Part 5B on your own before you turn to these hints!

Having trouble making the numbering start at 1 instead of 0?
Remember that the index of a list begins counting at 0. Also recall that the **`range()`** function starts at 0 by default. If you are printing the current index as the number, you will start at 0 – in order to start counting at 1, you will need to print something like **`index + 1`**.

Part 5C: Voting for a Television Show

Do not move on to this part until your program can print the television shows!

Now that you have one piece of your program working, we can focus on the next task. Now that the shows can be displayed, we need to allow the user and their friends to actually vote!

To complete your program, you will need to ask for and store the votes that the user makes for each television show. They can continue to vote until they enter "0" to quit. **If they make an invalid choice, you do not need to reprompt the user – simply ignore the invalid choice.**

You will need to store the votes for each television show in another, separate list of integers. Remember, list indexing starts at 0, but we're presenting the choices to the user starting at 1, so the way you store votes will need to compensate for this offset.

Here is a sample run of the store program, with user input in blue.
(If you need some help, hints are available after this sample output.)

```
bash-4.1$ python tv_shows.py
1 - Daredevil
2 - Fargo
3 - Limitless
4 - Elementary
5 - Brooklyn 99
6 - Empire
7 - Supergirl
You and your friends are voting on a show to watch.
Which show would you like to vote for?
Enter '0' to stop voting: -1
Enter '0' to stop voting: 11
Enter '0' to stop voting: 1
Enter '0' to stop voting: 3
Enter '0' to stop voting: 4
Enter '0' to stop voting: 4
Enter '0' to stop voting: 5
Enter '0' to stop voting: 2
Enter '0' to stop voting: 0
```

Try to solve Part 5C on your own before you turn to these hints!

Are you stuck on how to interact with the user?

Take a look at the example on page 4 of an interactive while loop. You should use the same basic code setup to allow the user to keep voting until they choose to quit by entering "0".

Stuck on how to store the user's votes?

You need a list of the same length as the number of television shows. It should be a list of integers, and since this is something we're using to count, they should all be initialized to zero.

Still stuck on how to store the user's votes?

Try creating a votes variable that contains exactly as many zeroes as the number of television shows. Something like this would work:

```
votes = [0] * len(shows)
```

Having trouble seeing the "big picture" of how your program should work?

Try drawing a quick flowchart or planning out what needs to happen on paper in pseudocode. Don't worry about the specific details, just try to visualize what needs to happen overall. How do you stop once the user wants to quit? When do you need to ignore a user's vote? How are the votes stored? When do variables need to be initialized?

Stuck on what to do for your program?

Here is some basic pseudocode for what your program should be doing:

- Print the choice for television shows

- Prompt the user for an show to vote for (or "0" to stop voting)
- Store the user's vote in a separate list for votes
    - Check first – if the choice isn't valid, don't try to store the vote
- Repeat all of the above… until they choose "0" to quit voting

Part 5D: Displaying the Votes

Do not move on to this part until your program can ask the user to vote!

All that's left now is to display the final votes for each television show. (You might also have to do some debugging if your code to ask for and store votes doesn't work correctly. That's how programming works, sometimes!)

Once the user has entered "0" in order to stop voting, you need to go through the list and print out the number of votes each show earned. You will need to iterate through both of the lists in order to print out the show name and the number of votes it received.

(If you need some help, the hints are available after this sample output.)

```
bash-4.1$ python tv_shows.py
1 - Daredevil
2 - Fargo
3 - Limitless
4 - Elementary
5 - Brooklyn 99
6 - Empire
7 - Supergirl
You and your friends are voting on a show to watch.
Which show would you like to vote for?
Enter '0' to stop voting: 1
Enter '0' to stop voting: 3
Enter '0' to stop voting: 4
Enter '0' to stop voting: 4
Enter '0' to stop voting: 5
Enter '0' to stop voting: 2
Enter '0' to stop voting: 0

Here are the final votes:
Daredevil has    1 votes
Fargo has        1 votes
Limitless has    1 votes
Elementary has   2 votes
Brooklyn 99 has  1 votes
Empire has       0 votes
Supergirl has    0 votes
```

Try to solve Part 5D on your own before you turn to these hints!

Is your program counting the user's vote for the wrong show?
Remember, the user's numbering starts at 1, but the indexing in a list starts at 0. If a user chooses to vote for show #3, the votes for that show are stored at `votes[2]`, not `votes[3]`.

Are you stuck on how to print elements from two lists at the same time?
Because we want to print two lists at once, we cannot use a loop that iterates over a single list's contents. Instead, we can look at the same index in both lists to print out the television show and the votes it received.

Still stuck on how to print two lists at once?
You will want to use a loop similar to the one at the top of page 3. It should iterate over a range that is the length of one of the lists.

Wondering how to make the votes in the sample output neatly line up?
Remember that the tab character in Python (`\t`) works similar to a tab in a word processing program. It doesn't indent a specific number of spaces, but rather indents to a specific point. Putting a tab in front of the number of votes should line the number of votes up at the same place on the line.

IMPORTANT NOTE: You do not need to worry about figuring out which show won (although this would be great practice to do later on your own!)

Part 6: Completing Your Lab

To test your program, first enable Python 3, then run **`tv_shows.py`**. Try a few different inputs to see how well your program works.

```
linux2[7]%  scl enable python33 bash
bash-4.1$ python tv_shows.py
1 - Daredevil
2 - Fargo
3 - Limitless
4 - Elementary
5 - Brooklyn 99
6 - Empire
7 - Supergirl
You and your friends are voting on a show to watch.
Which show would you like to vote for?
Enter '0' to stop voting: 1
Enter '0' to stop voting: 1
Enter '0' to stop voting: 2
Enter '0' to stop voting: 0

Here are the final votes:
Daredevil has    2 votes
Fargo has        1 votes
[etc]
```

Since this is an in-person lab, you do not need to use the **`submit`** command to complete your lab. Instead, raise your hand to let your TA know that you are finished.

They will come over and check your work – they may ask you to run your program for them, and they may also want to see your code. Once they've checked your work, they'll give you a score for the lab, and you are free to leave.

IMPORTANT: If you leave the lab without the TA checking your work, you will receive a **zero** for this week's lab. Make sure you have been given a grade before you leave!