CMSC 201 Fall 2017
Lab 05 – Lists

Assignment: Lab 05 – Lists
Due Date: **During discussion**, October 2nd through October 5th
Value: 10 points (8 points during lab, 2 points for Pre Lab quiz)

This week's lab will put into practice the concepts you learned about lists: indexing, mutating, and traversing. It will also make use of **`while`** loops, both to get input from the user, and to traverse the contents of the list.

(Having concepts explained in a new and different way can often lead to a better understanding, so make sure to pay attention as your TA explains.)

Part 1A: Review – Sentinel While Loops

One way to use a `while` loop is as a ***sentinel*** loop. A sentinel loop continues to process data until reaching a special value that signals the end of the data. The special value is called the ***sentinel***.

Here is the pseudocode for a sentinel 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.

```python
SENTINEL = "exit"

def main():
    # initialize the list to be empty
    grocery_list = []
    # 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 != SENTINEL:
        grocery_list.append(userVal)
        # get another value from the user
        userVal = input("Enter an item, or 'exit' to end: ")

    print("Remember to buy", grocery_list)

main()
```

Part 1B: Review – 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:

Aya	**Brad**	**Carlos**	**David**	**Emma**
0	1	2	3	4

Part 1C: Review – Traversing Lists

Looking at the contents of a list is also known as ***traversing*** the list, and can be done using a basic **`while`** loop. In the loop, we use a variable to keep track of which item in the list we are looking at by having it store the index of that item. As we move on to the next item, that variable is incremented, until we reach the end of the list.

The length of the list is an important property, as it is used to tell the **`while`** loop when to stop traversing the list. The length can be gotten by using the **`len()`** function.

For example, this code would traverse the **`names`** list above, printing out that each person is awesome:

```
# this variable can be called anything
# it starts at zero because that's the first index
index = 0

while index < len(names):
    print( names[index], "is awesome!")
    index += 1
```

Part 1D: Review – Mutating Lists

Lists can also be "mutated" – we can add and remove items from them as many times as we want. This means that we can start off with an empty list (denoted as two square brackets: **`newList = []`**) and fill it as necessary.

Adding to a list is easy to do: simply add the new item to the end of the list, using the **`.append()`** function. The following line of code adds a few items to a list called **`newList`**:

```
newList.append("A Thing")
newList.append(1.37)
newList.append(0)
newList.append(False)
```

After we run these lines of code, our list would look like this:

"A Thing"	**1.37**	**0**	**False**
0	**1**	**2**	**3**

To remove items from the list, we use the appropriately named **`.remove()`** function. The **`.remove()`** function takes in *what* we want to remove, not *where* it is in the list. For example, if we call it and ask it to remove 0, it will remove the third element, the integer 0, and not the string "A Thing", which is stored at index 0.

```
newList.remove(0)
```

"A Thing"	**1.37**	**False**
0	**1**	**2**

The **`.remove()`** function also updates the indexes of anything after the removed element, so that our list looks like a regular list after the element was deleted. (In other words, notice how the index at which **`False`** is stored changes from 3 before the removal to 2 afterwards.)

Part 2: Exercise

In this lab, you'll be creating one file, **`languages.py`**, but you'll be creating it in four steps. That way, you can focus on each of the steps needed one by one.

The program you'll be coding will display different options for programming languages, and will allow users to vote on which one they prefer, using the numbers printed next to each one. Once voting is over, your program will print out each programming language and how many votes is earned.

Tasks

- ☐ Create a **`languages.py`** file
- ☐ Write the code to print out the programming language choices
- ☐ Write the code to get votes from the user
- ☐ Write the code to "save" the votes in a list
- ☐ Write the code to print out the total votes
  - ☐ (You should run and test your **`languages.py`** file after each step)
- ☐ Show your work to your TA

Part 3A: Creating Your File

First, create the **`lab05`** folder using the **`mkdir`** command -- the folder needs to be inside your **`Labs`** folder as well. *(If you need a reminder of how to create and navigate folders, try asking a classmate next to you for help. If you're both stuck, ask the TA or refer to the instructions for Lab 1.)*

Next, create a Python file called **`languages.py`** using the "**`touch`**" command in GL.
The "`touch`" command creates a new blank file, but doesn't open it.
Once a file has been "touched", you can open and edit it using emacs.

```
touch languages.py
emacs languages.py
```

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

```
# File:          FILENAME.py
# Author:        YOUR NAME
# Date:          10/TODAY/2017
# Section:       YOUR SECTION NUMBER
# E-mail:        USERNAME@umbc.edu
# Description:   YOUR DESCRIPTION GOES HERE AND HERE
#                YOUR DESCRIPTION CONTINUED SOME MORE
```

Part 3B: Printing the Programming Language Choices

This is the first of four steps that must be written for this lab.
The first step is to copy in the list of programming langauges, and to write code that will print out the different choices.

Copy the list below into your program's **`main()`**:

```
# types of programming languages to vote on
languages = ["Python", "Java", "C++", "Ruby", "C", "PHP",
"Shakespeare"]
```

To print the choices, you should write a **`while`** loop that will print out the following two things on each line:

- The number of the choice (with the count starting at 1, not 0!)
- The programming language's name

Here is some sample output for this part of the program.
(Yours should match this word for word.)

```
bash-4.1$ python languages.py
1 - Python
2 - Java
3 - C++
4 - Ruby
5 - C
6 - PHP
7 - Shakespeare
```

Once this part of the program works correctly, move on to the next step.

Having trouble making the numbering start at 1 instead of 0?
Remember that the index of a list begins counting at 0. If you are printing the current index as the number, your printing will start at 0. In order to start counting at 1, you will need to print something like **`index + 1`**.

Part 3C: Voting for a Programming Language

This is the second of four steps that must be written for this lab.
Now that the code to display the programming languages is complete, we need to allow the user and their friends to actually vote!

For now, we won't worry about storing the votes. Just write the code that will allow the user to vote, and will stop when they enter a "0" to quit. **If they make an invalid choice, simply ignore it and ask again.**

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 languages.py
1 - Python
2 - Java
3 - C++
4 - Ruby
5 - C
6 - PHP
7 - Shakespeare
What language do you like? (Enter 0 to stop): 1
What language do you like? (Enter 0 to stop): 1
What language do you like? (Enter 0 to stop): 4
What language do you like? (Enter 0 to stop): 4
What language do you like? (Enter 0 to stop): 9
What language do you like? (Enter 0 to stop): 7
What language do you like? (Enter 0 to stop): 0
```

Once this part of the program works correctly, move on to the next step.

Are you stuck on how to interact with the user?
Take a look at the example on page 2 of a sentinel loop (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".

Part 3D: Storing Votes

This is the third of four steps that must be written for this lab.
Now that you can accept votes, we need to store them. We'll store the votes for the programming languages in **another, separate list of integers**.

So if, for example, the user voted for Python (choice 1), Shakespeare twice (choice 7) and C++ four times (choice 3), the vote list would look like this:

`votes =`	1	0	4	0	0	0	2
	index 0	index 1	index 2	index 3	index 4	index 5	index 6

In other words, the votes stored at a given index in **`votes`** should be for the programming language stored at that same index in the **`languages`** list.

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. You'll also need to make sure you ignore invalid input – don't try to count a vote for option #8, when there are only 7 choices!

At the end, print out the list of votes, so you can ensure your program is working correctly. (Simply use **`print("votes:", votes)`** in your code.)

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

We've removed the list of genres at the beginning to save space, but it should still be present in your output.

(Yours does not have to match this word for word, but it should be similar.)

```
bash-4.1$ python languages.py
[[ programming languages should be displayed here ]]
What language do you like? (Enter 0 to stop): 3
What language do you like? (Enter 0 to stop): 3
What language do you like? (Enter 0 to stop): 4
What language do you like? (Enter 0 to stop): 6
What language do you like? (Enter 0 to stop): 0
votes: [0, 0, 2, 1, 0, 1, 0]
```

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

Once this part of the program works correctly, move on to the next step.

Stuck on how to store the user's votes?
You need a list of the same length as the number of programming languages. 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 programming languages. Something like this would work:

```
votes = [0, 0, 0, 0, 0, 0, 0]
```

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

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?

Part 3E: Printing Out the Results

This is the last of four steps that must be written for this lab.
This last step is relatively simple, as you've already done all of the hard work. For this step, we'll display the final votes for each programming language. (If your code that asks for and stores votes doesn't work correctly, you might also have to do some debugging. 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 programming language earned. You will need to iterate through **both** of the lists, in the same while loop, in order to print out the programming language and the number of votes it received.

Also, remove the line of code that prints out the list of votes from the last step!

IMPORTANT NOTE: You do not need to worry about figuring out which language won. **Although this would be great practice to try out!**
(If you need some help, hints are available after this sample output.)

Here is some sample output, with the user input in **blue**.
We've removed the list of genres at the beginning to save space.
(Yours does not have to match this word for word, but it should be similar.)

```
bash-4.1$ python languages.py
[[ programming languages should be displayed here ]]
What language do you like? (Enter 0 to stop): 3
What language do you like? (Enter 0 to stop): 3
What language do you like? (Enter 0 to stop): 4
What language do you like? (Enter 0 to stop): 6
What language do you like? (Enter 0 to stop): 0
Python has 0 votes.
Java has 0 votes.
C++ has 2 votes.
Ruby has 1 votes.
C has 0 votes.
PHP has 1 votes.
Shakespeare has 0 votes.
```

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

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 must use the same while loop to access their contents. Remember that both lists are the same length, and the indexes of programming languages and votes match between the two lists.

Still stuck on how to print two lists at once?
You will want to use a loop similar to the one at the bottom of page 3. (The two lists in your program should be the same length, so it doesn't matter which one you use for the length.)

Part 4: Completing Your Lab

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.

Tasks

- ☐ Create a **`languages.py`** file
- ☐ Write the code to print out the programming language choices
- ☐ Write the code to get votes from the user
- ☐ Write the code to "save" the votes in a list
- ☐ Write the code to print out the total votes
  - ☐ (You should run and test your **`languages.py`** file after each step)
- ☐ Show your work to your TA

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!