CMSC 201 Fall 2016

Lab 07 – Lists

Assignment: Lab 07 – Lists
Due Date: Thursday, October 20th by 8:59:59 PM
Value: 10 points

Lab 7 focuses on using lists and loops, but is also a review of much of the material we have covered so far. **Completing Lab 7 prior to the midterm exam is highly recommended.**

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 (sentinel 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:

Aya	Brad	Carlos	David	Emma
0	1	2	3	4

Part 2: For Loops

We can use **`for`** loops to ***iterate*** 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
```

Sometimes 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 indexes (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:
    statement(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: Sentinel Loops

Another 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.

```
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 **lab7**, and go inside the newly created **lab7** 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 lab7
linux2[5]% cd lab7
linux2[6]% pwd
/afs/umbc.edu/users/k/k/k38/home/201/Labs/lab7
linux2[7]%
```

To open the file for editing, type

emacs games.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:          games.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
# Collaboration: During lab, collaboration between
# students is allowed, although I understand I still
# must understand the material and complete the
# assignment myself.
```

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

Part 5B: Printing the Game Options

For Lab 7, you will be writing up a short program to help the user and their friends decide what game to play. The program will print out the list of possible games, and then everyone can vote on which game they would like to play. The program will close voting if someone enters "0" as their choice, and will output the total votes for each game.

You will be coding Lab 7 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, since 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 games. Copy the list below, **`games`**, into your program.

```
games = ["Twister", "Dodgeball", "Capture the Flag",
"Hide and Seek", "Croquet", "Ring Toss", "Ping Pong"]
```

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

- The **number** of the game (start counting from 1)
- The game's **name** (*e.g.*, Twister, Dodgeball, etc.)

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

```
bash-4.1$ python games.py
1 - Twister
2 - Dodgeball
3 - Capture the Flag
4 - Hide and Seek
5 - Croquet
6 - Ring Toss
7 - Ping Pong
```

(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 Game

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

Once you have one piece of your program working, we can focus on the next task. Now that the games can be displayed, 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, you do not need to reprompt the user – simply ignore the invalid choice.**

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 games.py
1 - Twister
2 - Dodgeball
3 - Capture the Flag
4 - Hide and Seek
5 - Croquet
6 - Ring Toss
7 - Ping Pong

What game would you like to play? (0 to quit): 7
What game would you like to play? (0 to quit): 6
What game would you like to play? (0 to quit): 9
What game would you like to play? (0 to quit): 1
What game would you like to play? (0 to quit): 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 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 5D: Storing Game Votes

Do not move on to this part until your program can accept votes correctly!

Now that you can accept votes, we need to store the votes. We'll store the votes for the games in **another, separate list of integers**. The list of votes should correspond directly to the list of games: the votes at a given index should be for the game stored at that same index in the `games` 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 handle invalid input correctly – don't try to count a vote for option #282, when there are only 7 choices!

At the end, print out the list of votes, so you can ensure your program is working correctly.

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 games.py
1 - Twister
2 - Dodgeball
3 - Capture the Flag
4 - Hide and Seek
5 - Croquet
6 - Ring Toss
7 - Ping Pong

What game would you like to play? (0 to quit): 9
What game would you like to play? (0 to quit): -1
What game would you like to play? (0 to quit): 5
What game would you like to play? (0 to quit): 3
What game would you like to play? (0 to quit): 1
What game would you like to play? (0 to quit): 2
What game would you like to play? (0 to quit): 2
What game would you like to play? (0 to quit): 7
What game would you like to play? (0 to quit): 0
Votes list is: [1, 2, 1, 0, 1, 0, 1]
```

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

Stuck on how to store the user's votes?

You need a list of the same length as the number of games. 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 games. Something like this would work:

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

Is your program counting the user's vote for the wrong game?

Remember, the user's numbering starts at 1, but the indexing in a list starts at 0. If a user chooses to vote for game #3, the votes for that game 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 5E: Displaying the Votes

Do not move on to this part until your program can store the votes correctly!

Now let's display the final votes for each games. (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 game earned. You will need to iterate through both of the lists in order to print out the game name and the number of votes it received.

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

```
bash-4.1$ python games.py
1 - Twister
2 - Dodgeball
3 - Capture the Flag
4 - Hide and Seek
5 - Croquet
6 - Ring Toss
7 - Ping Pong

What game would you like to play? (0 to quit): 2
What game would you like to play? (0 to quit): 2
What game would you like to play? (0 to quit): 7
What game would you like to play? (0 to quit): 6
What game would you like to play? (0 to quit): 2
What game would you like to play? (0 to quit): 1
What game would you like to play? (0 to quit): 0

Twister has 1 votes
Dodgeball has 3 votes
Capture the Flag has 0 votes
Hide and Seek has 0 votes
Croquet has 0 votes
Ring Toss has 1 votes
Ping Pong has 1 votes
```

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

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 game 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. (The two lists in your program should be the same length, so it doesn't matter which one you use for the range.)

Part 6: Completing Your Lab

To test your program, make sure you've enabled Python 3, then run **`games.py`**. Try a few different inputs to see how well your program works.

Submitting

Since this lab is not an in-person lab, you will need to use the **`submit`** command to turn your completed lab in.

Once your **`games.py`** file is complete, it is time to turn it in with the **`submit`** command, where the class is **`cs201`**, and the assignment is **`LAB7`**. Type in (all on one line) **`submit cs201 LAB7 games.py`** and press enter.

```
linux1[4]% submit cs201 LAB7 games.py
Submitting games.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 lab assignment was submitted by following the directions in Homework 0. Double-check that you submitted your lab assignment correctly, since **an empty file will result in a grade of zero for this assignment.**