CMSC 201 Fall 2018

Lab 05 – Lists

Assignment: Lab 05 – Lists
Due Date: Thursday, October 4th by 8:59:59 PM
Value: 10 points

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.

```
SENTINEL = "exit"

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

    # run the while loop until the user enters "exit"
    while userVal != SENTINEL:
        grocery_list.append(userVal)
        # get another value from the user
        userVal = input(msg)

    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:

You can access individual elements of the list with similar bracket syntax:

```
# displays the string David to the screen
print( names[3] )
```

You can also make in-place edits to a single element of the list by using the bracket syntax, and an assignment operator:

```
names[3] = "Stella"
```

The above line of code of code will update **`names`** like so:

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
```

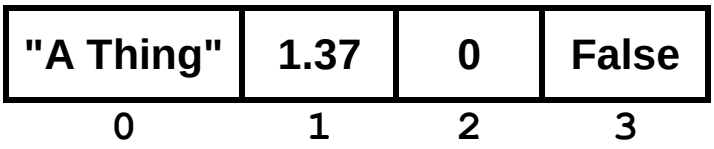
Part 1D: Review – Mutating Lists

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 with square brackets: `newList = []`) and fill it as necessary.

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

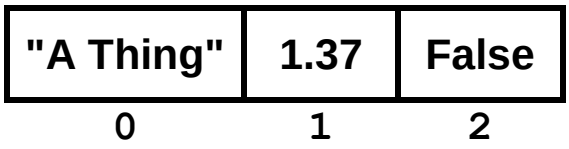
```
newList.append("A Thing")
newList.append(1.37)
newList.append(0)
newList.append(False)
```

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



To remove items from the list, we use the appropriately named `.remove()` method. The `.remove()` method 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)
```



The `.remove()` method 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, **`lights.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 light switches, and will allow users to flip any of these switches, using the numbers printed next to each one. Once the user is done flipping switches, your program will print out each switch and whether or not it was flipped.

Tasks

- ☐ Create a **`lights.py`** file
- ☐ Write the code to print out the light switches
- ☐ Write the code to get flips from the user
- ☐ Write code to actually flip the switch the user chooses
- ☐ Write the code to print out the switches when the user is done
    - ☐ (You should run and test your **`lights.py`** file after each step)
- ☐ Submit your work via the submit system

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, go to office hours or refer to the instructions for Lab 1.)*

Next, create a Python file called **`lights.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 lights.py
emacs lights.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:          TODAY'S DATE
# 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 Lights

This is the first of four steps that must be completed for this lab.
The first step is to copy in the list of switch states for five lights, and to write code that will print out the different choices.

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

```
# list of 5 switch states
switches = [False, False, False, False, False]
```

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 "before" state of all of the lights (whether or not they are on)
  - **`True`** means on, and **`False`** means off
  - For this part, they should all be off!

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

```
linux1[8]% python3 lights.py
Switches before:
Light 1 is off.
Light 2 is off.
Light 3 is off.
Light 4 is off.
Light 5 is off.
```

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: Flipping the Switch(es)

This is the second of four steps that must be completed for this lab.
Now that the code to set and display the switches is complete, we need to allow the user to actually flip them!

For now, we won't worry about actually changing the state of each switch. Just write the code that will allow the user to "flip" a switch by picking the number of the switch to flip, and that will stop when they enter a "0" instead. **For this lab, you can assume the user will always enter a valid light switch number (1 – 5 inclusive, or 0 to quit).**

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.)

```
linux1[9]% python3 lights.py
Switches before:
Light 1 is off.
Light 2 is off.
Light 3 is off.
Light 4 is off.
Light 5 is off.
Enter a switch to flip (0 to stop): 1
Enter a switch to flip (0 to stop): 2
Enter a switch to flip (0 to stop): 5
Enter a switch to flip (0 to stop): 4
Enter a switch to flip (0 to stop): 3
Enter a switch to flip (0 to stop): 2
Enter a switch to flip (0 to stop): 3
Enter a switch to flip (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 with a clear stop condition). You should use the same basic code setup to allow the user to keep flipping switches until they choose to quit by entering "0".

Part 3D: Storing Flips

This is the third of four steps that must be completed for this lab.
Now that you can accept flips, we need to store them. We already have a list that stores the state of each of the five switches, so you must now update that list in-place to store the correct state!

For example, if the user flipped the 2nd light three times, the 4th light once, and 5th light twice, the switch list would look like:

`switches=`	`False`	`True`	`False`	`True`	`False`
	index 0	index 1	index 2	index 3	index 4

Remember, list indexing starts at 0, but we're presenting the choices to the user starting at 1, so the way the program assigns flips to each light switch will need to compensate for this offset.

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

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

We've removed the list of switches 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 lights.py
[[ Switches should be displayed here ]]
Enter a switch to flip (0 to stop): 2
Enter a switch to flip (0 to stop): 2
Enter a switch to flip (0 to stop): 2
Enter a switch to flip (0 to stop): 4
Enter a switch to flip (0 to stop): 5
Enter a switch to flip (0 to stop): 5
Enter a switch to flip (0 to stop): 0
Switches: [False, True, False, True, False]
```

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 it once. For this step, we'll display the final switch states for each light. (If your code that asks for and stores switch flips 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 flipping, you need to go through the list one more time and print out final state of each one.

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

Here is some sample output, with the user input in **blue**.
We've removed the list of switches 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 lights.py
[[ Switches should be displayed here ]]
Enter a switch to flip (0 to stop): 3
Enter a switch to flip (0 to stop): 5
Enter a switch to flip (0 to stop): 4
Enter a switch to flip (0 to stop): 2
Enter a switch to flip (0 to stop): 2
Enter a switch to flip (0 to stop): 3
Enter a switch to flip (0 to stop): 1
Enter a switch to flip (0 to stop): 1
Enter a switch to flip (0 to stop): 0

Switches after:
Light 1 is off.
Light 2 is off.
Light 3 is off.
Light 4 is on.
Light 5 is on.
```

Part 4: Completing Your Lab

Since this is an online lab, you will need to use the **`submit`** command to complete your lab.

To submit your file (due **Thursday, October 4th, 2018 by 8:59:59 PM**), use the command:

```
linux1[4]% submit cs201 LAB5 lights.py
Submitting lights.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.

Tasks

- ☐ Create a **`lights.py`** file
- ☐ Write the code to print out the light switches
- ☐ Write the code to get flips from the user
- ☐ Write code to actually flip the switch the user chooses
- ☐ Write the code to print out the switches when the user is done
    - ☐ (You should run and test your **`lights.py`** file after each step)
- ☐ Submit your work via the submit system