CMSC 201 Spring 2016
Lab 04 – For Loops

Assignment: Lab 04 – For Loops
Due Date: During discussion, February 29th through March 3rd
Value: 10 points

Part 1: Lists

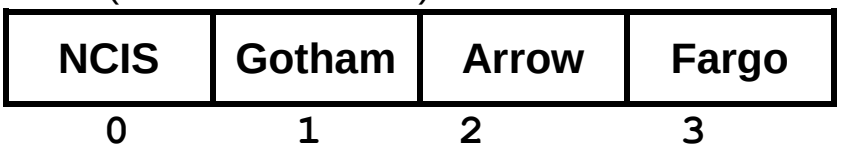
Lists are an easy way to hold lots of individual pieces of data without needing to make lots of variables. They are a type of *data structure*, which are specialized ways of organizing and storing data.

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 **showList**:

```
showList = ["NCIS", "Gotham", "Arrow", "Fargo"]
```

Which creates the list (called **showList**) below:



We can then use indexing to get individual elements of the list. For example:

```
print(showList[1], "returns February 29th.")
print("The show", showList[3], "is based on a movie.")
```

Part 2: Strings

In Python, we can represent text (a sequence of characters) using the *string* data type. A string in Python is created like so:

```
my_name = "John Doe"
```

Python will treat anything between two quotation marks (double quotes `""` or single quotes `''`) as a string. Strings can be *indexed* just like lists (because they are lists of characters).

For example:

```
my_name[0]                  # This evaluates to 'J'
my_name[3]                  # This evaluates to 'n'
my_name[-1]                 # This evaluates to 'e'
my_name[len(my_name)]       # Error!! Doesn't exist!!!
```

So far, we have learned a few *operations* that you can perform on strings:

- **`STRING.upper()`** gives the **`STRING`** back in all uppercase
- **`STRING.lower()`** gives the **`STRING`** back in all lowercase
- **`len(STRING)`** gives the length of **`STRING`**
- **`STRING * NUMBER`** repetition of **`STRING`** (**`NUMBER`** times)
- **`STRING1 + STRING2`** concatenates **`STRING1`** and **`STRING2`** together

For example:

```
# puts the words together without spaces in between
name = "joHn" + "DOe"

name.upper()     # => JOHNDOE (string in all uppercase)
name.lower()     # => johndoe (string in all lowercase)
len(name)        # => 7, how long the string is
print(name * 3)  # prints "joHnDOejoHnDOejoHnDOe"
```

Part 3: 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 see both below. *Iterating* over a list 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)
```

Output:

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

Notice how the above loop executes for each item in the list, replacing the value of the variable **`fruit`** with the next item in the list.
IMPORTANT: The variable **`fruit`** is a copy of the current element of the list, and changing the variable **`fruit`** will not change anything in the original list (**`list_of_fruits`**).

A **`for`** loop can also be used to iterate through the characters in a string.

For example:

```
name = "SAM"
for character in name:
   print("Give me an", character + "!")
```

Output:

```
Give me an S!
Give me an A!
Give me an M!
```

Notice how the loop executes for each character in **`name`**. It starts with the first character in the string, executes the for loop body (the print statement) with that character, and moves on to the next character.

If we want to change the original list, however, we need to use a **`for`** loop that loops over the *index* of each element of a list (or the index for each character of a string). To do that, we use the **`range()`** function.

For example:

```
greetings = ["Hello", "Hola", "Ciao", "Salut"]
for i in range(len(greetings)):
    greetings[i] = "Goodbye"
print(greetings)
```

Output:

```
['Goodbye', 'Goodbye', 'Goodbye', 'Goodbye']
```

We can also run a **`for`** loop a specific number of times. For example:

```
numToRun = 5
for n in range(numToRun):
    print("Executing for time #", n)
```

You can either define the number of times to run prior to using it in the **`for`** loop (like we did above with **`numToRun`**), with an expression in its place (like we did with **`len(greetings)`**), or with just a number (as we will see below).

The **`range()`** function can be used to make the loop variable change in very specific ways. If we give the **`range()`** function a single number, the loop variable will first be set to the value 0; on the next iteration of the loop, it will change to 1, 2, 3, etc. – all the way up to **one less** than the number specified by **`range()`**.

For example:

```
for i in range(4):
    print("The value of i is:", i)
```

Output:

```
The value of i is: 0
The value of i is: 1
The value of i is: 2
The value of i is: 3
```

You can also be much more specific in choosing the exact way that your **`for`** loop variable will update with each run of the loop.

For example:

```
for number in range(5, 8):
    print("This will execute 3 times:", number)
```

Output:

```
This will execute 3 times: 5
This will execute 3 times: 6
This will execute 3 times: 7
```

Notice how this loop will only execute 3 times, starting at the number 5 and ending at the number 7. **Specifying the starting number for the `range()` function is optional!** If you don't include it then 0 is chosen as the default start for the **`range()`** function. You can also optionally specify the increment that the **`range()`** function increases by for each iteration.

For example:

```
for num in range(1, 10, 2):
    print("The value increases by 2:", num)
```

Output:

```
The value of num is: 1
The value of num is: 3
The value of num is: 5
The value of num is: 7
The value of num is: 9
```

See how the above loop only executes 5 times? This time, the loop variable **`num`** will increase by 2 for each iteration, instead of by 1. **The third number, which specifies the increment of the range is also optional!** If you don't include it, an increment of 1 is chosen by default. For the **`range()`** function, the only hard requirement is a number to end the range on.

You can also use a negative increment in your **`range()`** function. This can be used to count down, or to iterate backwards through a list. If you use this, you need to **make sure that your starting number is higher than your ending number**, and make sure that the end of the range you specify is one less than where you actually want to stop.

For example:

```
for i in range(5, -1, -1):
    print("The value of i is:", i)
```

Output:

```
The value of i is 5
The value of i is 4
The value of i is 3
The value of i is 2
The value of i is 1
The value of i is 0
```

Part 4A: Writing Your Program

After logging into GL, navigate to the **`Labs`** folder inside your **`201`** folder. Create a folder there called **`lab4`**, and go inside the newly created **`lab4`** directory.

```
linux2[1]% cd 201/Labs
linux2[2]% pwd
/afs/umbc.edu/users/k/k/k38/home/201/Labs
linux2[3]% mkdir lab4
linux2[4]% cd lab4
linux2[5]% pwd
/afs/umbc.edu/users/k/k/k38/home/201/Labs/lab4
linux2[6]%
```

You will only be writing one python file for this assignment. The directions are split into two so that you only need to work on one part of the problem at a time.

To open the file for editing, type

`emacs palindrome.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:          palindrome.py
# Author:        YOUR NAME
# Date:          TODAY'S DATE
# Section:       YOUR DISCUSSION 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 4B and 4C.

Part 4B: Reversing a String

To practice using `for` loops and the `range()` function, you will be creating a program that takes in a string from the user and checks if it is a palindrome. (And you'll do it without using any of Python's built-in string functions!)

THINK: What is your input, output, and process for this problem?

For the first part of your code, you should ask the user to input a string, and store it. Then, ***using a `for` loop***, you program should figure out what that string is in reverse. Make sure that you save your result in a new variable, because you will need it for the next part of this lab.

YOU MUST USE A LOOP FOR THIS PART OF THE LAB.

Try to solve Part 4B on your own before you turn to these hints!

Are you stuck on how you're supposed to use the `for` loop?
Think about how can you use a `for` loop to iterate through the string backwards.

Did you get the error "`IndexError: string index out of range`" when trying to reverse the string?
If so, try printing out the values of your loop variable. This way you can see if it's starting and stopping where you want it to. (And remember that Python strings are indexed from 0 to their length-1.)

Do you not know how to store your reversed string once you have it?
Remember that we can concatenate two strings (or a string and a character) together by using the `+` operator. Don't forget to initialize the variable you want to use to store your reversed string!

Part 4C: Palindrome Checker

Once you have written a `for` loop to reverse the string given by the user (and stored the reversed version of the string in a separate variable from the original), you can write a simple palindrome checker!

As a reminder, a palindrome is a string that is written the same way forwards as it is backwards. Examples of strings that are palindromes are:

- `"tacocat"`
- `"racecar"`
- `"kayak"`
- `"rotor"`

Make sure that you tell the user whether their string is a palindrome or not by printing your result to the screen.

(HINT: We're making a decision (about whether the string is a palindrome or not), so we should use a decision structure!)

Part 5: Completing Your Lab

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

```
linux2[6]% scl enable python33 bash
bash-4.1$ python palindrome.py
Enter a string: racecar
racecar is a palindrome

bash-4.1$ python palindrome.py
Enter a string: kittycat
kittycat is not a palindrome
bash-4.1$
```

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!