CMSC 201 Fall 2018
Lab 06 – Strings

Assignment: Lab 06 – Strings
Due Date: **During discussion**, October 8th to October 11th
Value: 10 points (**no Pre Lab quiz**)

This week's lab will put into practice the concepts you learned about strings: indexing, traversing, splitting and joining. It will make use of `while` loops, to traverse the contents of the string.

(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 – Strings and Indexing

Strings are a sequence of characters surrounded by quotation marks. We've been using them to display information on the screen since day 1.

In order to get a specific character from a string, we need to access that ***index*** of the string. Just like lists! NOTE: Indexing into strings and lists doesn't starting counting from 1 – the first element in the sequence is at index 0.

For example, the following line of code creates a string called **`fruit`**:

```
fruit = "apple"
```

Which creates the string (called **`fruit`**) below:

a	p	p	l	e
0	1	2	3	4

Part 1B: 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.

Traversing strings works just the same as traversing lists.

For example, this code would traverse the **`fruit`** string above, printing out each letter:

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

while index < len(fruit):
    print("The next letter is:", fruit[index])
    index += 1
```

Part 1C: Review – Mutating Strings

Unlike Lists, we cannot directly mutate the contents of a string. Any operation that we do directly to a string in Python creates a copy of that string.

For example, we have methods that's allows us to create uppercase and lowercase versions of strings (`.upper()` and `.lower()`) .

If you want to save the result of applying these methods, you have to use a variable. Usually, you will just use the same variable.

```
fruit = "apple"
fruit = fruit.upper()
# now the variable fruit contains the string "APPLE"
```

You can index into strings, but you cannot edit the contents in place.

```
fruit[3] = "z" # this will fail
```

Part 1D: Review – Escape Sequences

There are a variety of special characters and whitespace that you may want to appear on the screen. To use these characters, we need to know their **escape sequence.**

`\t`	Tab character (like hitting the tab key)
`\n`	Newline character (like hitting enter)
`\\`	Backslash (escape the escape character)
`\"`	Quotation mark
`\'`	Apostrophe

Remember, these escape sequences are treated as a single character when used in a string. That means that the length of the string only counts an escape sequence as one character!

Part 1E: Review – Strip, Split, and Join

We have special methods we can use remove whitespace, to take strings apart, and to put them back together again. **`.strip()`** will remove outer whitespace, **`.split()`** will break a string apart into a list of substrings, and **.join()** will force a list of strings back into a single string again.

`.strip()` will give you back a copy of a string without any surrounding whitespace. Remember, it will **not** remove any **inner whitespace**.

```
sentence = "\t\tThis is a sample sentence.\n\n"
sentence = sentence.strip() # result: "This is a sample sentence."
```

`.split()` will give you a list of substrings created after removing what is inside the parentheses. Remember, if there is nothing inside the parenthesis it will split on whitespace.

```
testStr = "this is\ta test\n"
splitList = testStr.split() # result: ["this", "is", "a", "test"]
```

You can split on any character or substring you'd like.

```
testStr = "peter piper"
splitList = testStr.split("e") # result: ["p", "t", "r pip", "r"]
```

`.join()` will take a list of strings, and concatenate them together with whatever string you decide to use.

```
listOfStrings = ["anna", "ben", "caroline", "zoee"]
print( "&".join(listOfStrings) ) #result: anna&ben&caroline&zoee
```

Part 2: Exercise

In this lab, you'll be creating one file, **`palindrome.py`**.

The program you'll be coding will display whether or not a string is a palindrome. A palindrome is a string that can written the same forward as it can be backwards.

Tasks

- ☐ Create an **`palindrome.py`** file
- ☐ Write the code to get a string from the user, and remove all whitespace from it using strip, split, and join.
- ☐ Write the code to traverse the string one character at a time
- ☐ Add to your traversal code to check the matching character on the other end of the string
  - ☐ (You should run and test your **`palindrome.py`** file after each step)
- ☐ Show your work to your TA

Part 3A: Creating Your File

First, create the **`lab06`** 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 **`palindrome.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 palindrome.py
emacs palindrome.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: Removing Whitespace

This is the first of three steps that must be written for this lab.
The first step is to get a string from the user. This string might have any amount of whitespace in it, which could throw off our code for checking if this string is a palindrome.

Once you have a string from the user, you should use some combination of strip, split, and join to remove whitespace.

You should also force the string to be all lowercase letters using lower.

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

```
linux2[3]% python3 palindrome.py
Enter a string: this is a test
thisisatest

linux2[4]% python3 palindrome.py
Enter a string: tHIS   IS     a Test
thisisatest
```

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

Part 3C: Traversing the edited string

This is the second of three steps that must be written for this lab.
Now that the code to get rid of whitespace is complete, it is time to traverse the string one character at a time.

For now, we won't worry about checking whether or not a string is a palindrome. We will just make sure that we can go through the string one character a time. You should write a loop to do this, and if you want you can print out the index of each character along with it.

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.)

```
linux3[4]% python3 palindrome.py
Enter a string: bananas
0 b
1 a
2 n
3 a
4 n
5 a
6 s
```

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

Part 3D: Palindrome Check

This is the third and final step that must be written for this lab.

Now it's time to check if the string is a palindrome. For this part to work, you should think about how you can use your traversal loop to check two characters at the same time.

You already have the current index stored in a variable, but now you need to figure out how to check the character on the opposite side from the index that you have. Can you come up with a mathematical expression to do that?

Once you've figured out an expression, now you have to use it to check if the string is a palindrome. Since you are already traversing the string, all you have to do is check and make sure that each character pair is a match, and once you find a mismatch you know that string is not a palindrome. Something like a Boolean flag will be very useful here.

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

```
linux3[5]% python3 palindrome.py
Enter a string: this is a test
'thisisatest' is not a palindrome

linux3[6]% python3 palindrome.py
Enter a string: race car
'racecar' is a palindrome

linux3[7]% python3 palindrome.py
Enter a string: not so Boston
'notsoboston' is a palindrome
```

Stuck on the mathematical expression?
Given some **`index`** of the string, the character on the opposite side of the string can be found at **`len(string) – index - 1`**

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 an **`palindrome.py`** file
- ☐ Write the code to get a string from the user, and remove all whitespace from it using strip, split, and join.
- ☐ Write the code to traverse the string one character at a time
- ☐ Add to your traversal code to check the matching character on the other end of the string
  - ☐ (You should run and test your **`palindrome.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!