CMSC 201 Fall 2018
Lab 04 – While Loops

Assignment: Lab 04 – While Loops
Due Date: **During discussion**, September 24th through September 27th
Value: 10 points (8 points during lab, 2 points for Pre Lab quiz)

In Lab 3, you used conditionals and simple decisions structures to control the "flow" of a program. This week's lab will put into practice some of the material learned in class, including while loops and nesting decision structures. (Having concepts explained in a new and different way can often lead to a better understanding, so make sure to pay attention.)

Part 1A: Review – While Loops

A **`while`** loop statement in the Python programming language repeatedly executes a target statement as long as a given Boolean condition is **`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 the condition is still **`True`**. In other words, it runs *while* the condition is **`True`**.

As soon as the condition evaluates to **`False`**, program control passes to the line of code immediately following the statements inside the **`while`** loop. This is the first line of code after the **`while`** loop and its statements. It is indented to the same depth as the "**`while <condition>:`**" line of code.

Remember that in Python, all the statements indented by the same number of character spaces after **`while`** (or **`if`**, etc.) statements are considered to be part of a single ***block*** of code. Python uses indentation as its method of grouping statements.

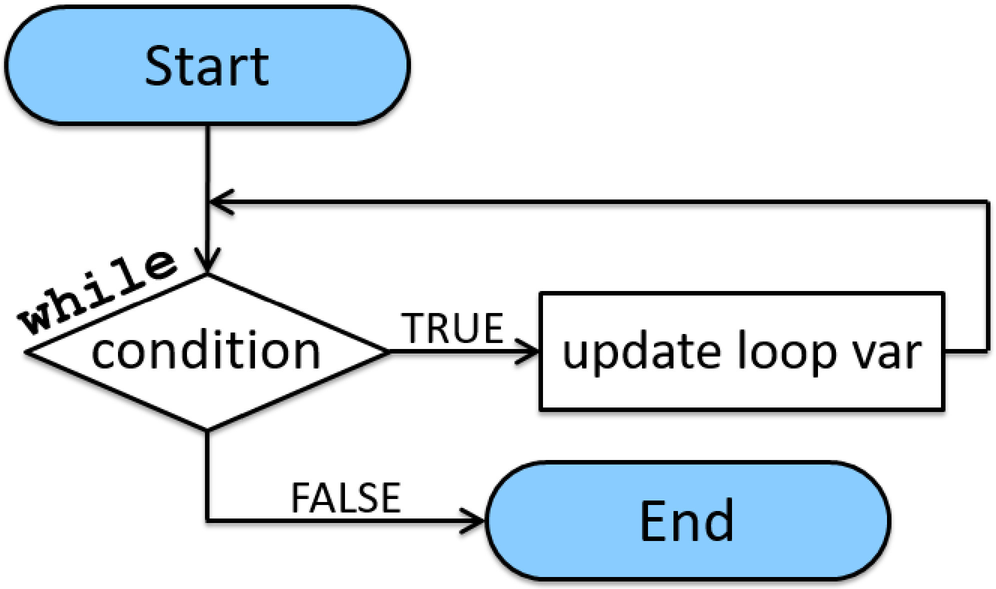

Figure1. A `while` loop in Python

```
while <condition>:
    <statement(s)>
```

Each time Python finishes executing the "**`<statement(s)>`**" inside the **`while`** loop, it returns to and reevaluates the "**`<condition>`**". This helps it decide: if the condition evaluates to **`True`**, it will execute the statements again; if the condition evaluates to **`False`**, it will stop running the **`while`** loop.

It's as if Python is asking itself "OK, I'm done – should I go again?" If the condition always evaluates to **`True`**, the program gets stuck in an ***infinite loop***.

It is possible that a **`while`** loop might not ever run the "**`<statement(s)>`**" inside the **`while`** loop. If the condition is tested and the result is **`False`**, the loop body (the statements) will be skipped and the first line of code after the while loop will be executed.

Part 1B: Review – Interactive Loops

One of the major uses of a **`while`** loop is to interact with the user of the program. Users are unpredictable, and we can't always rely on them to act in the correct way, or to follow the rules or directions we give.

Our program may need to ask a user for something over and over and over (and over and over) before it is satisfied. The user may be entering multiple pieces of information, or they may be putting in invalid data (such as a negative score on a quiz, or an email address with no "@" symbol in it).

Since we don't know how many times we'll have to reprompt the user, it makes the most sense to use a **`while`** loop when interacting with the user in this way.

The first type of interactive loop is one that verifies input from the user. In this case, we continually reject the user's answer while it is unsatisfactory (in other words, until it is satisfactory). The pseudocode for one of these loops might look like this:

```
    Ask the user for their input
    While the input is not valid:
        Print out what sort of input IS valid
        Reprompt the user for new input
```

In an input-verifying loop, it is very important that you tell the user what is unacceptable about their input, and how to fix it.

```python
def main():
    age = int(input("How old are you?"))

    while age < 0:
        print("Please enter a positive age.")
        age = int(input("How old are you?"))

    print("Happy", age, "th birthday!")

main()
```

Part 2: Exercise

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

By the time the **`store.py`** program is complete, it will ask the user to enter sales, and will keep track of the total sales entered for two items.

Tasks

- ☐ Create a **`store.py`** file
- ☐ Write the code to get a positive number from the user
- ☐ Run and test your **`store.py`** file
- ☐ Write the code to get the cookies and tea sold from the user
- ☐ Run and test your **`store.py`** file
- ☐ Modify the code to store compute the money made from each sale
- ☐ Run and test your **`store.py`** file
- ☐ Show your work to your TA

Part 3A: Creating Your File

First, create the **`lab04`** 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 **`store.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 store.py
emacs store.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: Getting a Positive Number

This is the first of three steps that must be written for this lab.
This first step uses a simple **`while`** loop to get a positive number from the user. If they enter a non-positive number (negative or zero), the program must re-prompt them until they enter a positive number.

The program you are writing will be used to compute the money made from the sale of tea and cookies in a small cafe. The first question we are asking the user is how many sales they would like to enter.

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.)

```
linux2[4]% python3 store.py
How many sales would you like to enter? 0
You must enter a number greater than zero.
How many sales would you like to enter? -9
You must enter a number greater than zero.
How many sales would you like to enter? 4
```

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

Part 3C: Getting the Sales

This is the second of three steps that must be written for this lab.
This second step will use another **`while`** loop to get the individual sales from the user, and will keep asking for another sale until the user enters as many sales as they previously indicated. In other words, if the user answered that they wanted to enter "4" sales, the program must ask for four sales.

You program should respond to each sale with either "You sold a cookie!" if they entered **`'c'`**, or "You sold some tea!" if they entered **`'t'`**

The program does not need to worry about the user entering invalid sales. Your program can just ignore characters that are not 't' or 'c'

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.)

```
linux2[5] python3 store.py
How many sales would you like to enter? 4
Enter 'c' for cookie, or 't' for tea: c
You sold a cookie!
Enter 'c' for cookie, or 't' for tea: t
You sold some tea!
Enter 'c' for cookie, or 't' for tea: r
Enter 'c' for cookie, or 't' for tea: c
You sold a cookie!
```

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

Part 3D: Which sale is bigger?

This is the last of three steps that must be written for this lab.
This last step requires that you use decision structures to determine money made from each sale that the user entered, and which item made you more money.

The best way to do this is to have two separate variables that keep track of the total made from both cookies and tea. The price of a single cookie is $0.75, and the price of a single cup of tea is $1.25.

Think carefully about how and when to initialize your variables that will store the totals. If you initialize the totals inside the while loop, they will just be overwritten with each turn of the loop. Your totals would then be incorrect!

After you have computed the totals made from both items, you should display to the user both totals as well as the overall total. Then, you should output which item they made the most money on. You don't have to worry about what to do if they made the same amount.

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.)

```
linux2[6] python store.py
Enter a positive number: 6
Enter 'c' for cookie, or 't' for tea: c
You sold a cookie!
Enter 'c' for cookie, or 't' for tea: c
You sold a cookie!
Enter 'c' for cookie, or 't' for tea: c
You sold a cookie!
Enter 'c' for cookie, or 't' for tea: t
You sold some tea!
Enter 'c' for cookie, or 't' for tea: t
You sold some tea!
Enter 'c' for cookie, or 't' for tea: c
You sold a cookie!
You made 3.0 from cookies.
You made 2.5 from tea.
You made more money from cookies!
In total, you made 5.5 in sales.
```

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 **`store.py`** file
- ☐ Write the code to get a positive number from the user
- ☐ Run and test your **`store.py`** file
- ☐ Write the code to get all the items sold from the user
- ☐ Run and test your **`store.py`** file
- ☐ Modify the code to total made in sales, and which item made more
- ☐ Run and test your **`store.py`** file
- ☐ 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!