CMSC 201 Spring 2017
Lab 04 – While Loops

Assignment: Lab 04 – While Loops
Due Date: **During discussion**, February 20th through February 23rd
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.

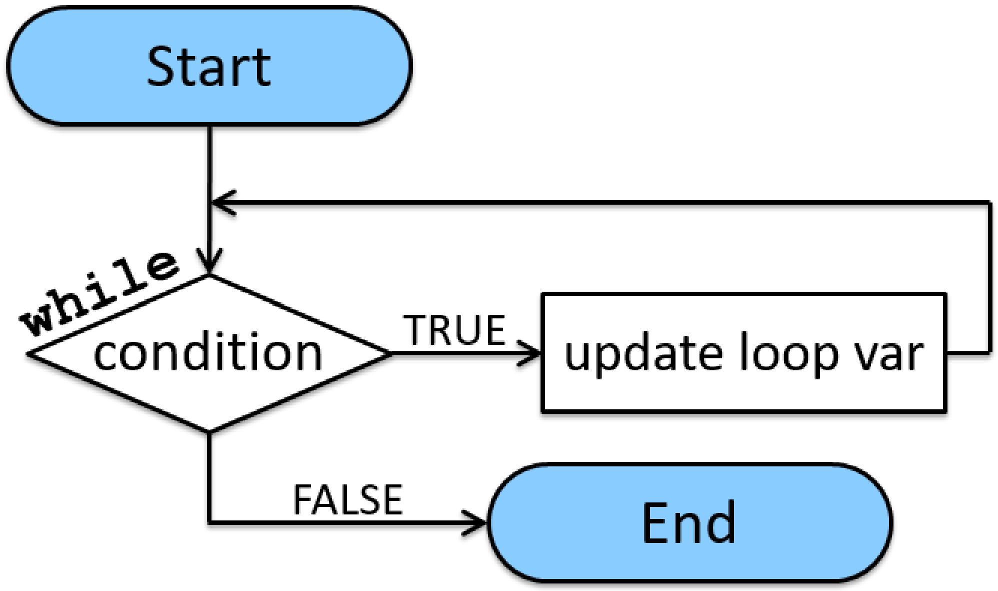

Figure1. A `while` loop in Python

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 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.

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

Tasks

- ☐ Create an **`avg.py`** file
- ☐ Write the code to get a positive number from the user
- ☐ Run and test your **`avg.py`** file
- ☐ Write the code to get the grades to be average
- ☐ Run and test your **`avg.py`** file
- ☐ Write the code to average and print out the grades
- ☐ Run and test your **`avg.py`** file
- ☐ Show your work to your TA

Part 3A: Creating Your File

First, create the **`lab4`** 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 **`avg.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 avg.py
emacs avg.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:          2/TODAY/2017
# 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 reprompt them until they enter a positive number.

The program you are writing will be used to average grades together, so the question we are asking the user is how many grades 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.)

```
bash-4.1$ python avg.py
How many grades would you like to enter? 0
You must enter a number greater than zero.
How many grades would you like to average? -9
You must enter a number greater than zero.
How many grades would you like to average? 4
```

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

Part 3C: Getting the Grades

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 grades from the user, and will keep asking for grades until the user enters as many grades as they previously indicated. In other words, if the user answered that they wanted to enter "4" grades, the program must ask for four grades.

The program does not need to worry about the user entering negative or invalid grades. If they enter a negative grade, just accept it as valid.

Once you've gotten all of the grades from the user, print out the total before ending the program.

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.)

```
bash-4.1$ python avg.py
How many grades would you like to enter? 4
Enter a grade: 75
Enter a grade: 85
Enter a grade: 95
Enter a grade: 60
Your total is 315
```

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

(HINT: You will likely need two new variables for this part: one to count the number of grades entered by the user so far, and one to keep track of the grade total so far. Both variables will need to be initialized before the while loop begins. By the end of the run above, the grade total would be 315.)

Part 3D: Calculating the Average

This is the last of three steps that must be written for this lab.
This last step is relatively simple, as it doesn't involve any more **`while`** loops. Instead, use the total calculated in the previous step, along with the number originally entered by the user, to calculate and print out the average.

Make sure to remove the print out of the total value from step 3C! It was included to ensure that the second step was done correctly, but it doesn't need to be in the final program.

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.)

```
bash-4.1$ python avg.py
How many grades would you like to enter? 4
Enter a grade: 75
Enter a grade: 85
Enter a grade: 95
Enter a grade: 60
The average of your 4 grades is 78.75

bash-4.1$ python avg.py
How many grades would you like to enter? 5
Enter a grade: 33
Enter a grade: 70
Enter a grade: 85
Enter a grade: 42
Enter a grade: 98
The average of your 5 grades is 65.6
```

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 **`avg.py`** file
- ☐ Write the code to get a positive number from the user
- ☐ Run and test your **`avg.py`** file
- ☐ Write the code to get the grades to be average
- ☐ Run and test your **`avg.py`** file
- ☐ Write the code to average and print out the grades
- ☐ Run and test your **`avg.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!