CMSC 201 Fall 2015
Lab 06 – While Loops

Assignment: Lab 06 – While Loops
Due Date: During discussion, October 5th through October 8th
Value: 1% of final grade

Part 1: 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:
    STATEMENTS(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 (while) the condition is still **`True`**.

As soon as the condition evaluates to **`False`**, program control passes to the line immediately following the **`while`** loop. This is the first line of code after the **`while`** loop and its statements that it 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 a programming construct are considered to be part of a single *block* of code. Python uses indentation as its method of grouping statements.

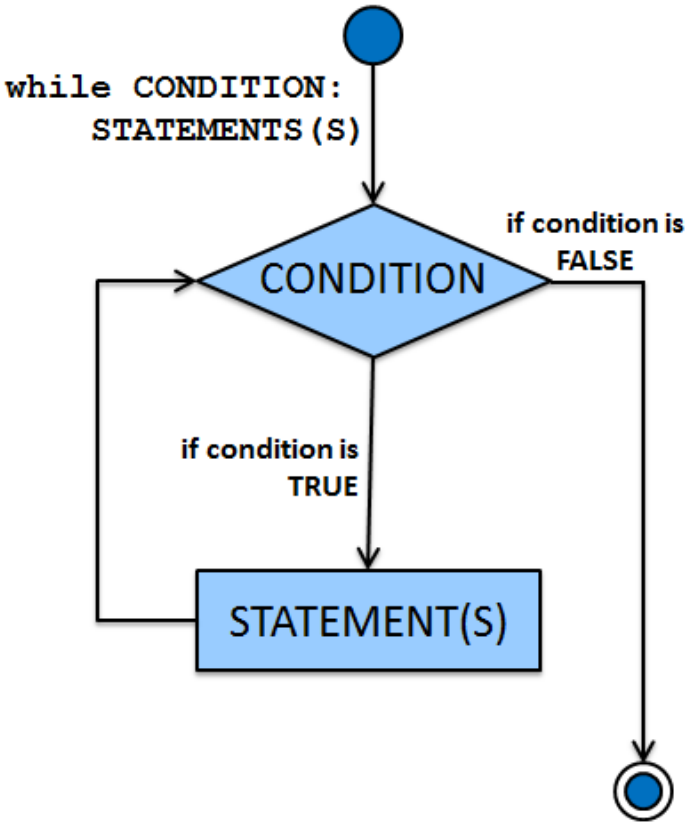

Figure1. A `while` loop in Python

It is also possible that a **`while`** loop might not ever run its conditional code (the "STATEMENTS" 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.

```
count = 0
while (count < 5):
    print ('The count is:', count)
    count = count + 1
print ("Good bye!")
```

When the above code is executed, it produces the following result :

```
The count is: 0
The count is: 1
The count is: 2
The count is: 3
The count is: 4
Good bye!
```

Part 2: Interactive (Sentinel) Loops

Another way to use a **`while`** loop is as an *interactive* or *sentinel* loop. An interactive (sentinel) loop continues to process data until reaching a special value that signals the end. The special value is called the *sentinel*.
You can choose to use any value for the sentinel. The only requirement is that it must be distinguishable from actual data values. The sentinel should also not be processed as part of the data.

The pseudocode for an interactive (sentinel) loop in the Python programming language is as follows:

```
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.

```
def main():
    grocery_list = []    # initialize the list to be empty
    userVal      = ""    # give our loop variable an initial
                         #   value so it will enter the loop

    # run the while loop until the user enters "exit"
    while userVal != "exit":
        userVal = input("Enter an item, or 'exit' to end: ")
        if userVal != "exit":
            grocery_list.append(userVal)
    # once the user is done with the list, print it out
    for i in grocery_list:
        print("Remember to buy", i)

main()
```

When the above code is executed, it produces the following result :

```
Enter an item, or 'exit' to end: candy
Enter an item, or 'exit' to end: cookies
Enter an item, or 'exit' to end: gummy bears
Enter an item, or 'exit' to end: exit
Remember to buy candy
Remember to buy cookies
Remember to buy gummy bears
```

Part 3A: Writing Your Program

After logging into GL, navigate to the **`Labs`** folder inside your **`201`** folder. Create a folder there called **`lab6`**, and go inside the newly created **`lab6`** directory.

```
linux2[1]% cd 201/Labs
linux2[2]% pwd
/afs/umbc.edu/users/k/k/k38/home/201/Labs
linux2[3]% mkdir lab6
linux2[4]% cd lab6
linux2[5]% pwd
/afs/umbc.edu/users/k/k/k38/home/201/Labs/lab6
linux2[6]%
```

(You will only be writing one python file for this assignment.)

To open the file for editing, type

`emacs password.py &`

and hit enter. (The ampersand at the end of the line is important – without it, your terminal will "freeze" until you close the emacs window. **Do not do this if you are not on a lab computer.**)

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:          password.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 3B.

Part 3B: Password Checker

To practice using **`while`** loops and string comparison, you will be creating a program that takes in a password guess from the user (a string) and check if their guess matches the actual password (a predefined value from your program). The user will get three chances to enter the correct password.

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

(Don't scroll down to the next page until you've thought about this!)

For the first part of your code, you should ask the user to input their guess at the password (a string). Then, **using a `while` loop**, your program should compare the user's input (their guess) to the PASSWORD (a constant that is set in your code). If the user guesses the password wrong 3 times, it tells the user and quits the program.

Input:
User enters a guess at the password (a string)

```
bash-4.1$ python password.py
Enter the password: password123
```

Process:
Using a **`while`** loop:

- Check guess vs. PASSWORD
  - (should be a constant variable **`PASSWORD = "UMBCrulz"`**)
- If they match, tell user guess was accepted
- Else, tell the user that the guess was incorrect.
- Tell user how many additional guesses they have left

If the number of incorrect guesses exceeds 3, tell the user that they cannot access the system (and the program should end). You will need to remember to keep track of how many tries they have left.

Output:
Prompt for password
Message for "Successful Login" or for "Unsuccessful Login"

YOU <u>MUST</u> USE A `while` LOOP FOR THIS LAB.

(There are hints on the next page if you need them.)

Try to solve Part 3B on your own before you turn to these hints!

Are you stuck on how to get started?
Start by creating the interactive (sentinel) loop that gets the user to enter their guess. Get the loop to work (just with accepting the correct password) before you worry about the number of tries.
If you accidentally made an infinite loop, use CTRL+C to stop it running!

Don't know how to store the correct password or number of tries?
You should use constants! One to hold the password "UMBCrulz" and one to hold the number of guesses that are allowed (3). Remember that the variable names constants should be in all uppercase.

Don't know how to print off the number of guesses left?
You have the number allowed (your constant, set to 3) and the number of attempts. Use these two variables to calculate the number of guesses left.

Don't know how (or where) to stop when the user has no guesses left?
You may want to use a Boolean to see if the user has exceeded the number of guesses allowed. This should be part of your while loop's condition.

Part 4: Completing Your Lab

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

```
bash-4.1$ python password.py
Enter the password: UMBCrulz
You have successfully logged into the system.

bash-4.1$ python password.py
Enter the password: HappyHalloween
WRONG.  You have 2 guesses left.
Enter the password: my_password
WRONG.  You have 1 guesses left.
Enter the password: MooCow
Too many incorrect guesses.  You are locked out of the system.
```

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!