CMSC 201 Spring 2019
Lab 05 – While Loops and Boolean Flags

Assignment: Lab 05 – While Loops and Boolean Flags
Due Date: During discussion, February 25th through February 28th
Value: 10 points (8 points during lab, 2 points for Pre Lab quiz)

In Lab 4, you implemented some basic while loops. This week's lab will put into practice some of the more complex types of loops, such as using a Boolean flag to control a loop that has complex conditions for ending. (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 – Boolean Flags

Boolean flags can be used to help us use a **`while`** loop to check conditions that are too complex to fit in a regular conditional. Most commonly, we'll use them when verifying user input for various requirements.

Instead of the traditional conditional (using **`x != 5`**, **`count < 10`**, etc.), a **`while`** loop that uses a Boolean flag will look something like:

```
while not validName:
        or
while correctAns == False:
        or
while wrongNum == True:
```

where **`validName`**, **`correctAns`**, and **`wrongNum`** are all Boolean variables. The value of the Boolean will be changed inside the loop itself, where we have the ability to use more complex methods for checking things. It also means that it's possible to give the user feedback on any specific item they did incorrectly.

Remember, a **`while`** loop statement in the Python programming language repeatedly executes a target statement as long as a given Boolean condition is **`True`**. In other words, the loop will continue to run until the condition evaluates to **`False`**. So as long as the things we're checking for haven't been satisfied, the loop should keep running.

There are two common uses for Boolean flag-controlled loops:

1. Multiple requirements must be satisfied
2. A requirement can be satisfied in more than one way

Let's talk about each in turn.

Part 1B: Review – Multiple Requirements

If there are multiple requirements, the simplest thing to do after getting the user input is to assume that they have satisfied all the requirements, and then check each individually. If one of the requirements isn't satisfied, only then do we mark their input as invalid.
In other words, they need to satisfy ***all*** requirements to "pass."

The steps you would take inside the loop are

1. Get an input from the user
2. Set your Boolean flag so the loop will exit
3. Check each requirement
    a. If it's not satisfied, set the Boolean flag so the loop won't exit
    b. Inform the user what requirement they failed

Part 1C: Review – Multiple Means of Satisfaction

If, instead, there are multiple ways of satisfying the requirements, the scenario will play out but "reversed." After getting the user input, we will assume they didn't fulfill any of the possible ways to satisfy the requirements. Then we'll check each way of satisfying it separately. If they satisfy the requirements at least one of the possible ways, only then do we mark the input as valid.
In other words, they only need to match ***one*** possible way to "pass."

The steps you would take inside the loop are

1. Get an input from the user
2. Set your Boolean flag so the loop won't exit (don't change it)
3. Check each possible means of satisfaction
    a. If it's been satisfied, set the Boolean flag so the loop will exit

Part 2: Exercise

In this lab, you'll be finishing a file that has been started for you, called **`pickyNum.py`**.

Tasks

- ☐ Download the **`pickyNum.py`** file
- ☐ Complete the lines of code that are missing from the file
- ☐ Run and test your **`pickyNum.py`** file
- ☐ Show your work to your TA

Part 3A: Downloading the File

First, create the **`lab05`** folder using the **`mkdir`** command – the folder needs to be inside your **`Labs`** folder as well.

Next, copy a file into your **`lab05`** folder using the **`cp`** command. (The command should be all on one line.)

```
cp /afs/umbc.edu/users/k/k/k38/pub/cs201/pickyNum.py .
```

This will copy the file **`pickyNum.py`** from Dr. Gibson's public folder into your current folder, and will give it the same name.

Part 3B: Completing the Program

For Lab 5, you will be implementing a program that helps the user (you!) to select a number that will satisfy their very picky friend.

Follow the instructions inside the **`pickyNum.py`** file to complete the lab, including replacing the three **`???`** parts with actual code.

Here is some sample output of a working file, with the user input in **blue**. (Yours does not have to match this word for word, but it should be similar.)

```
linux1[9]% python3 pickyNum.py
You have a friend who is very picky about their numbers.  They've
asked you to choose a number that meets certain requirements, but
you're a lazy computer scientist, so you decided to code a program
that will check it for you, instead of thinking about it too hard.

The requirements are that the number satisfy the following:
     positive (greater than zero), less than 1000,
     divisible by 4, end in an 8, NOT end in a 2
Please enter your number: 0
The number must be greater than zero.
The number must end in an 8.
Please enter your number: 1000
The number must be less than 1000.
The number must end in an 8.
Please enter your number: 998
The number must be divisible by 4.
Please enter your number: 992
The number must end in an 8.
The number must NOT end in a 2.
Please enter your number: 998
The number must be divisible by 4.
Please enter your number: 988
Congrats!  Your friend accepts the number 988
```

You can find another sample run on the following page.

```
linux1[9]% python3 pickyNum.py
You have a friend who is very picky about their numbers.  They've
asked you to choose a number that meets certain requirements, but
you're a lazy computer scientist, so you decided to code a program
that will check it for you, instead of thinking about it too hard.

The requirements are that the number satisfy the following:
     positive (greater than zero), less than 1000,
     divisible by 4, end in an 8, NOT end in a 2
Please enter your number: 1
The number must be divisible by 4.
The number must end in an 8.
Please enter your number: 2
The number must be divisible by 4.
The number must end in an 8.
The number must NOT end in a 2.
Please enter your number: 3
The number must be divisible by 4.
The number must end in an 8.
Please enter your number: 4
The number must end in an 8.
Please enter your number: 5
The number must be divisible by 4.
The number must end in an 8.
Please enter your number: 6
The number must be divisible by 4.
The number must end in an 8.
Please enter your number: 7
The number must be divisible by 4.
The number must end in an 8.
Please enter your number: 8
Congrats!  Your friend accepts the number 8
```

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

- ☐ Download the **`pickyNum.py`** file
- ☐ Complete the lines of code that are missing from the file
- ☐ Run and test your **`pickyNum.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!