INTRODUCTION TO COMPUTER SCIENCE I
Spring 2014
LAB 4
Loops (and random numbers!)

This week, we are going to practice using loops. Along the way, we will introduce the ability to have the program randomly choose values, allowing us to have an "unpredictable" part of our programs. Finally, we'll combine the two into a little game for the user to play.

1 Loop practice

Begin by practicing your use of loops by writing a couple of simple ones. Because we are now using more complex structures like *conditional statements* (`if-then-else`) and *loops* (`while`), we will write this code into a `.py` module and run that module, rather than type the Python commands directly into the shell window. So, to write these loops, do the following:

1. Open IDLE.
2. Create a new module by going to the `File:New Window`. Within the new window, select `File:Save`, naming the new module `practice-loops.py` and placing it wherever you have been storing your work for this course.
3. Within this new module, write code that **counts down from 10 to 1**, printing each value as it counts. You should use a `while`-loop here.
4. Next, add to the module code that obtains an integer from the user (let's call it `v`), and then **counts toward 0** from `v`, again printing each value as it counts. That is, if $v = 7$, then we expect to see the numbers *decrease* from 7 to 0; if $v = -12$, then we should see the numbers *increase* from -12 to 0.

Test your code with various input values to be sure that it works correctly, and then move on to the next section.

2 Picking random numbers

Computers are deterministic machines. If you twice use the same program, and you provide it the same inputs each time, then you will see the same results. However, we sometimes want our programs to behave differently each time we use them. For example, imagine a program that is supposed to simulate the rolling of a pair of dice.

In order to have our programs behave to some extent unpredictably, Python provides a *random number generator* to pick numbers whose values we cannot predict.[1] Let's try using this capability...

[1] Strictly speaking, Python provides a *pseudo-random number generator*. The computer remains a deterministic device, and if we looked up the exact code for the number generator, and we knew enough about what a program had done so far, we *could* predict the values that come from this generator. It's simply written in a way to make that prediction difficult. We will discuss these ideas more in upcoming classes.

1. Go to the IDLE shell window.
2. Enter the following line:

```
>>> import random
```

   Note that if you want to use random number generation, you must use this `import` command first. For any module that you create that uses random numbers, this line should appear **at the top** of your module.

3. Enter the following two lines and see what you get:

```
>>> x = random.randrange(0, 100)
>>> print(x)
```

   You will see some value between 0 and 99 (but never 100 itself). The function `random.randrange(a, b)` will randomly select a value between (and including) `a` and (not including) `b`. It then returns that value; here, we assigned whatever that value was into `x`.

4. Do it again. Type the same two lines as above. Although you perform the same steps, you will (likely) see a different value. If you keep performing these two steps, you will see a different value each time.

You may use `random.randrange(a, b)` to select random values in any of your programs as needed. In fact, let's do that now...

3 Your assignment

Your assignment, should you choose to accept it,[2], is to write a small program that plays a common kids' game: *guess the number*. Once again, create a new module, and save it with the name `guess.py` into your usual folder for COSC-111 work. Then, write the code for this program to do the following:

1. Prompt the user for a *highest value* for this game, which we'll call `upper`. That is, the user wants to guess numbers between 1 and `upper`.
2. Randomly select `secret`, the value between 1 and `upper`.
3. Repeatedly allow the user to guess a number, `guess`. Depending on what the guess is, do one of the following:
   - If the guess is out of the valid range of possibilities, print an error message and let the user try again.

[2] You should.

- If the guess is correct, print a congratulatory message and end the program.
- If the guess is too low, print a message that the secret is a higher value and let the user try again.
- If the guess is too high, print a message that the secret is a lower value and let the user try again.

An extra challenge: Once you have this program working, can you change it to (a) count how many valid guesses the user enters before guessing correctly, and (b) only allows the user $\lceil log_2(\texttt{upper}) \rceil$ guesses before revealing the secret and ending the program.[3]

4 Submitting your work

Go to the CS submission system to submit your work for this lab. You need only submit your `guess.py` module.

This assignment is due on Sunday, Feb-16, 11:59 pm

[3] Why $\lceil log_2 \rceil$ guesses? Why shouldn't the user get more guesses than that?