Lecture 3

Steven Walton
PS 232 Computational Methods
Department of Physics
Embry-Riddle Aeronautical University
Prescott, AZ 86301

September 4, 2014

Loops

Loops are one of the most powerful and useful things in programming. They are pretty self explanatory in nature. This control structure will execute the code that is contained and then loop back to the start of the structure. In this manner we are able to repeat the same line of code many times quickly, and without having to write many lines of code. The first loop that we'll introduce is the for loop. This is useful for executing a set of code a specific number of times. So let's try an example.

```
print ''Hi!''
for x in range(0,3):        # start of the loop, make sure you include :
   print ''My name is...'' # This will print three times

print ''Slim Shady          # This will print once
```

Pay attention the colon. If you forget this then your code will not execute. The other thing to pay attention to is the indenting. While this is usually good coding practice in python this is necessary. Python forces you to have to have more readable code, whitespace (spaces, tabs, etc) are important in python. This is the full code here, notice that we did not define x. Python knows that we are using x for the numbers in the range(0,3). Notice how easy this is to read. One of the key aspects of python is its readability. This is one of the reasons that the language is so popular. You'll soon fine that you will be able to read python code, even if you aren't able to write it.
Now if you know this Eminem song you know that there is more to is. We know there is the pause and a changing word/scratches. To fill out the lyrics we need to do that. Loops are great for this actually, because we have that variable x updating in the for loop. So let's add those lines.

```
subVocals = ['(what?', '(who?)', '[scratches]'] # sub vocals in an array
print ''Hi!''
for x in range(0,3):
   print ''My name is...'' + subVocals[x]       # prints with sub vocals

print ''Slim Shady''
```

Perfect! Now we are printing out the first few lines of the song.
The next thing I want to introduce you to is the `if` statement. This is pretty straight forward, because it

is the same as English. The structure is "IF (condition) THEN (do something)" So let's continue with the lyrics.

```
print "Hi!"
subVerse = ['(what?)', '(who?)', '[scratches]', '(huh?)', '(what?)',
           '[scratches]']          # this is fine being on another line
for x in range(0,6):
  print "My name is..." + subVerse[x]
  if (x == 2) or (x == 5):         # first nested if
    print "Slim Shady"
    if (x==2):                     # second nested if
        print "Hi!"
```

Now we have printed out the first verse of "My Name Is". For an exercise, I'll let you figure out a way to remove the first line (the "Hi!" line), and keep everything but subVerse in the loop.
But you're probably wondering what happened here in this code. We increased size of subVerse to include the rest of the first verse. Pay careful attention to the whitespace in here. The for loop is the same, but we introduced three concepts here. The if statement says that if x is 2 OR x is 5 then print out the following. An or statement is the same as it is in English and logic. A more common example might be "If you go to the store then buy wheat bread or white bread." The logic is the same here as it is in code, and the readability makes it easy to understand. So this means that if you go to the store then buy either white bread or wheat bread, but not both. We can also use AND to require that both conditions are met before the following lines are executed. The third concept here is something called nesting. Nesting is when you layer lines of code. So here we have an if statement nested in a for loop, and an if statement nested within the other if statement. Nesting will become a powerful tool in your coding strategy. It will allow you to run loops that aren't exactly the same and to run different subroutines.
Of course, we can do the same thing with numbers. If you remember to the previous lecture we had talked about the append and pop we can use this to update arrays. In MATLAB we can have a lot of control over vectors and can multiply matrices. In python we are going to deal with these differently and do them with loops. If you did problem 3 correctly on the first homework assignment then you will have done something like the following.

```
x = []
for i in range(0, len(phi)):
   x.append((a+b) * cos(phi[i] - h * cos((a+b)/b) * phi[i]))
```

Obviously this is a snippet of the code (I'm not going to give everything away). As you can see here we have an empty array and are updating the array each time. If our resolution for phi is 0.00001 (step sizes), then this code takes 1.561 seconds to execute. This is pretty fast (faster than MATLAB), but this isn't the fastest way we can write the code. Instead we can change this snippet to

```
x = np.zeros(len(phi))
for i in range(0, len(phi)):
   x[i] = (a+b) * cos(phi[i] - h * cos((a+b)/b) * phi[i])
```

With the same resolution this code takes 1.369 seconds. This isn't much different in time, but we do see that there is a difference in timing. A computer can more easily initialize an array of 0 than it can append an array. If you are running multiple loops or larger loops these valuable seconds will save you a lot of time. It is important to keep your code quick and simple.