CST0006 – Computer Programming Foundations

INTRODUCTION to WEEK07

In this lecture and lab section we are going to talk about loops. Many times in programming, you will want to repeat a task over and over again. Sometimes you will want to iterate though a list and act on each item, or check something to see if it's true or false.

While Loops

The while loop is the easiest type of loop to understand. The logic simply breaks down to "while a condition is true, do something again and again". So, following last week's topic. If the condition is true, the loop runs, until it isn't true.

While Loop Example in Python

```
i = 0
while i < 10:
    #do something
    print(i)
    i += 1
```

While Loop Example in C

```
int i = 0;
while(i < 10)
{
    /* do something */
    printf("%d\n",i);
    i++;
}
```

For both of these examples:

- we create a variable i and store a zero (i is short for iterator)
- begin the while loop
- test the condition, if true, move into the block of code
- do something like print the value of i
- add 1 to the iterator variable
- loop until condition is false

Notice the code blocks within each example. Python uses indentation to mark a block of code, where as C uses curly braces to mark a block of code. In C programming, where those curly braces are places is considered a personal preference and indentation is cosmetic.

For Loops

The for loop is very much like a while loop, but can change depending on which language you are using. In either case, the general rule of thumb is a for loop is usually used when you want to iterate though known values, where as a while loop is usually used when you don't know the values you want to iterate though.

Some languages like Python, use a style where the variable you are checking is the name used for the items in a list it's best to think of this as a "for each" loop, and for languages like C, you define a variable and iterate on that variable.

Ex 1. For Loop Example in C

```
int i = 0;

for(i = 0; i < 10; i++)
{
    /* do something */
    printf("%d\n",i);
}
```

Ex 2. For Loop Example in Python

```
for i in range(10):
    #do something
    print(i)
```

Here are more advanced examples of a for loop in python

Ex 3. Print each items in a list

```
animals = ['dog', 'cat', 'monkey']

for animal in animals:
   print(animal)
```

Ex 4. Print each letter in a string

```
string = 'animals'

for letter in string:
    print(letter)
```

Ex 5. Print each value calling the index of an array of characters (what we call a string)

```
string = 'animals'

for i in range(len(string)):
    print("The value at index[" + str(i) + "] is " + string[i])
```

Questions to be covered in lecture:

Ex. 1: What are the three arguments doing in the for loop?

Ex. 2: What does the range function do?

What other arguments you use with range?

Ex. 3: What is the difference between animal and animals?

Ex. 4: What is a string?

Ex. 5: What does the *len()* function do?

What does the *range()* function do?

What is the difference between *str(i)* and *string[i]*?

Why doesn't doesn't the + operator work without using *str()*?

Recursion

Recursion is an extremely powerful form of repetition. Recursion is a bit more of an advanced topic for a beginner programming course, but it is still important that you have a general understanding of what recursion means. The easiest way to describe recursion is Recursion is a function that calls ***itself*** as an argument. The return value of the function becomes the argument of the function.

Another way of describing recursion is with the movie Inception. It's like a dream within a dream.

Factorials are a perfect way to explain recursion.

5! = 5 * 4 * 3 * 2 * 1 or 5! = 5 * 4!

So 5! is the same as 5 * 4!, which we can make into a programmable function.

Here is an example of recursion as a function:

```
def fact(n):
    if n == 1:
        return 1
    else:
        return n * fact(n-1)

print(fact(5))
```

Try running this code in debug mode so you can see how it breaks down.

Google "recursion". It's pretty funny.