Python Chapter 5: Repetition Structures

This guide has been written to reinforce your learning with the actual chapter lessons and not to serve as alternative way of learning python. I have tried my best to explain and provide explanations; however, if you find anything that needs to be corrected, please inform me.

- **Maruthi Basava**

BEFORE YOU START WRITING CODE, PLEASE BE AWARE THAT WHITESPACES MATTER IN PYTHON. LEAVING STRAY SPACES IN VARIOUS AREAS COULD CRASH YOUR PROGRAM. PLEASE MAKE SURE THAT THE SYNTAX IS CORRECT.

In the last chapter, we learned about booleans, relational operators, and if statements. In this chapter, we will learn about loops.

What are loops?

Loops just loop through the entire code block until the boolean turns false.

What the heck did I just say?

Don't worry, you'll understand it very soon!

Let's say that we need to create a program to give the cookie monster a cookie, we can do just that with this code here:

Python:

```
cookie_jar = 10

def giveCookie():
    global cookie_jar
    cookie_jar = cookie_jar - 1
    print "Cookies Left: " + str(cookie_jar)

giveCookie()
```

Terminal:

```
Cookies left: 9
```

By the way, I don't think I have reviewed the global keyword, i'll do it here. The global keyword allows any function to access a global variable by placing global keyword next to the global variable. If you don't, then the function can't access the global variable and it would crash.

Here we have a cookie_jar variable that stores a value of 10.
Then we have a function giveCookie()
Inside of that function, we first put global in front of the cookie_jar because it is a global variable. Then we set the cookie_jar equal to cookie_jar - 1 (basically subtracting 1).
Then we print the leftovers.

Ok cool, so every time we call the giveCookie function, we subtract one cookie from the cookie_jar.

But now, the cookie monster is hungry and wants to eat ALL 10 cookies.

What are we going to do?

Are we going to copy and paste the same giveCookie() line 10 times?

If we are, it is going to look like this:

Python:

```python
cookie_jar = 10

def giveCookie():
    global cookie_jar
    cookie_jar = cookie_jar - 1
    print "Cookies Left: " + str(cookie_jar)

giveCookie()
giveCookie()
giveCookie()
giveCookie()
giveCookie()
giveCookie()
giveCookie()
giveCookie()
giveCookie()
giveCookie()
```

Terminal:

```
Cookies Left: 9
Cookies Left: 8
Cookies Left: 7
Cookies Left: 6
Cookies Left: 5
Cookies Left: 4
Cookies Left: 3
Cookies Left: 2
Cookies Left: 1
Cookies Left: 0
```

Eww... thats nasty.

Imagine if the cookie monster wants to 100 cookies, or 1,000 cookies, or even 1,000,000 cookies.

Are you going to sit there all day and copy and paste the function giveCookie() a million times?

If you aren't going to, then good news!

Python has something called loops and it is used for this exact reason: do one thing but many times.

Python has two types of loops,
The while loop and the for loop.

The While Loop:

Remember the old example?

```
cookie_jar = 10

def giveCookie():
    global cookie_jar
    cookie_jar = cookie_jar - 1
    print "Cookies Left: " + str(cookie_jar)

giveCookie()
giveCookie()
giveCookie()
giveCookie()
giveCookie()
giveCookie()
giveCookie()
giveCookie()
giveCookie()
giveCookie()
```

It's ugly isn't it?

Well we can fix that with a while loop.

What is a while loop?

A loop is written just like how a function is written or an if statement.

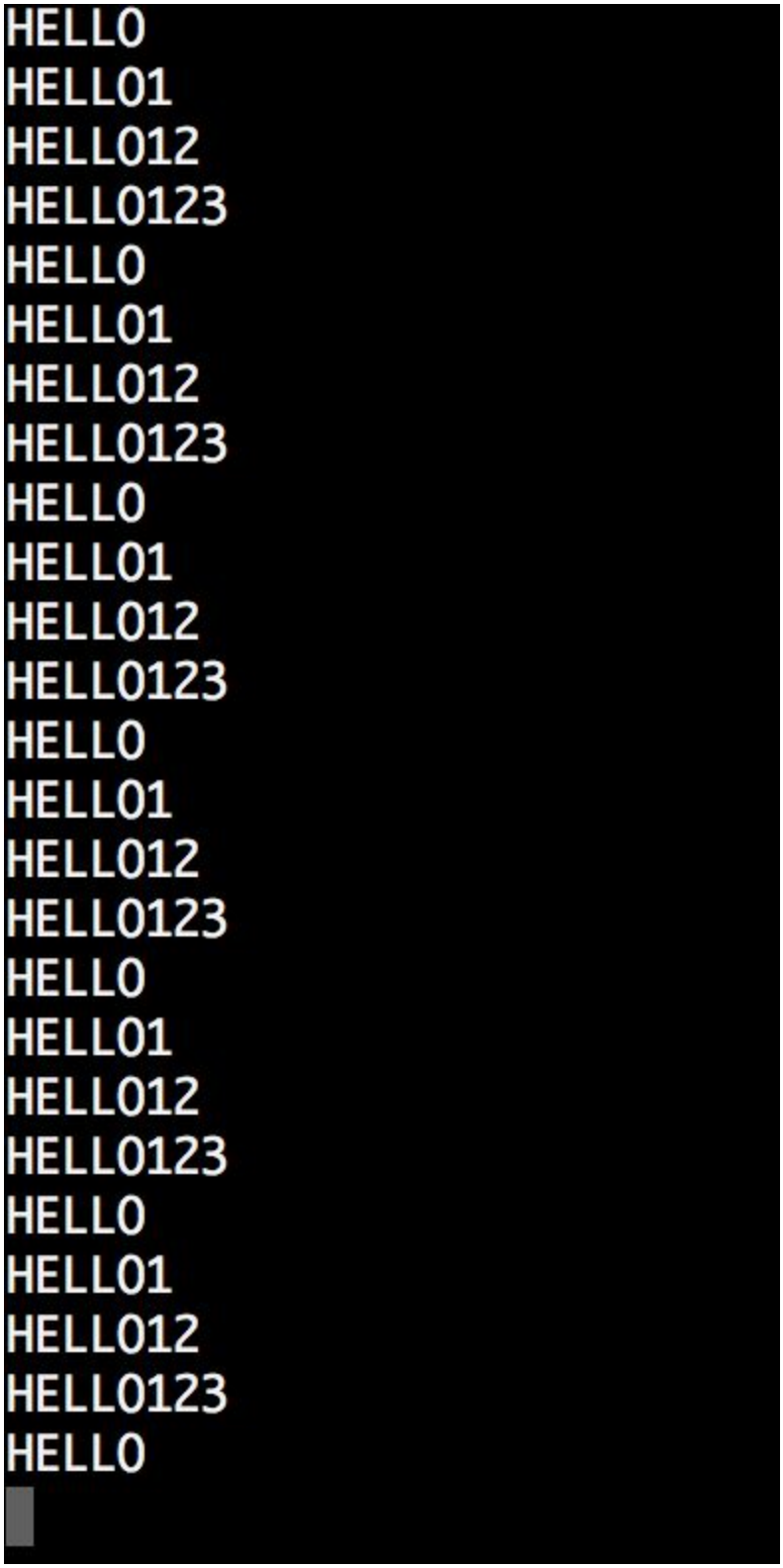
while CONDITION:
LINE OF CODE
LINE OF CODE
LINE OF CODE
...

Essentially, you type the while keyword in front of a conditional statement (remember last chapter? A conditional statement is anything that returns a true or false) then a color (:) Then you indent to write code inside of the block. Now we just created a while loop block.

Here how it looks in code:

Because we put a boolean of value False, the loop will not run. The loop will only and always run if the conditional statement is True. However, we can use this to our advantage by allowing the loop to run when we need it run then stop after.

CAREFUL! IF YOU PUT TRUE (LIKE IN THE EXAMPLE BELOW) IN A WHILE LOOP, YOU CAN CRASH YOUR PROGRAM OR YOUR COMPUTER BECAUSE IT RUNS FOREVER.


HELLO
HELLO1
HELLO12
HELLO123
HELLO
HELLO1
HELLO12
HELLO123
HELLO
HELLO1
HELLO12
HELLO123
HELLO
HELLO1
HELLO12
HELLO123
HELLO
HELLO1
HELLO12
HELLO123
HELLO
HELLO1
HELLO12
HELLO123
HELLO

See what I mean?

For a While loop to be functional, you must be able to stop it once it is done with its task.

Let's rewrite the cookie monster program using a while loop.

BEFORE:

```
cookie_jar = 10

def giveCookie():
    global cookie_jar
    cookie_jar = cookie_jar - 1
    print "Cookies Left: " + str(cookie_jar)

giveCookie()
giveCookie()
giveCookie()
giveCookie()
giveCookie()
giveCookie()
giveCookie()
giveCookie()
giveCookie()
giveCookie()
```

AFTER:

```
cookie_jar = 10

def giveCookie():
    global cookie_jar
    while cookie_jar > 0:
        cookie_jar = cookie_jar - 1
        print "Cookies Left: " + str(cookie_jar)


giveCookie()
```

Both give the same result, but this time, the one with the while loop is much cleaner.

But what is actually happening in the code?

1. **First we call giveCookie()**
2. **We ourselves access to the global variable cookie_jar by placing global in front of it.**
3. **Then we create a while loop with the conditional statement cookie_jar > 0 (basically we are asking it to see if cookie_jar is greater than 0, which means that we still have cookies)**
4. **Inside of the loop, we set cookie_jar to itself but minus 1**
5. **Then we print**
6. **Then the code inside the while loop block repeats until cookie_jar > 0 is False.**

By the way, we should probably change the function name to giveAllCookies than giveCookie because it makes more sense that the function is giving the cookie monster all the cookies than just 1.

```
cookie_jar = 10

def giveAllCookies():
    global cookie_jar
    while cookie_jar > 0:
        cookie_jar = cookie_jar - 1
        print "Cookies Left: " + str(cookie_jar)

giveAllCookies()
```