# CyberCamp at UNK

## Loops

**What you will learn with this tutorial...**

*In this tutorial we will learn how to use an important tool present in the programming world: **loops**. First we will understand what it is, and after that we will study two different ways of building loops in Python. Once it is done, as always, you will have a short assignment to apply the new concepts you learnt with this tutorial!*

## 1 Loops

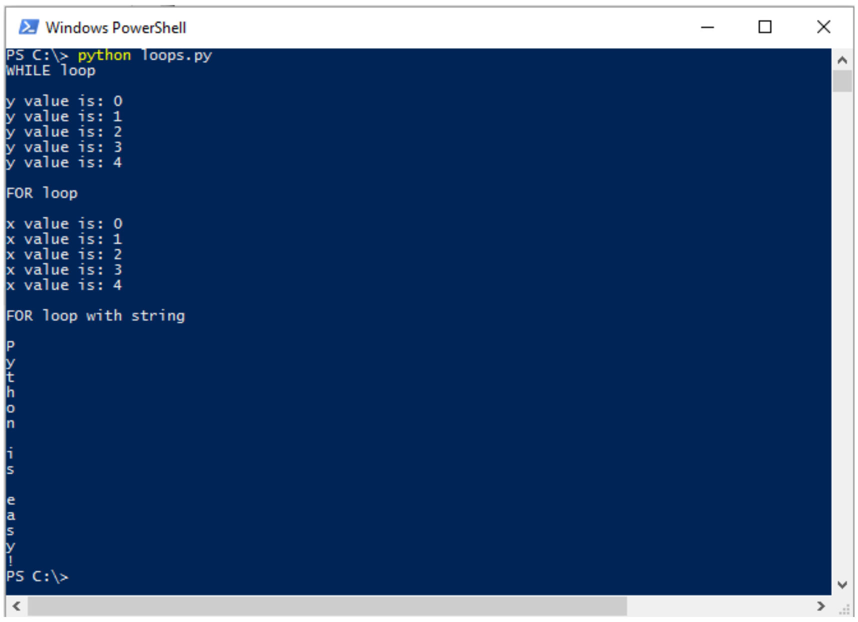
**Loop**, in the programming world, is a tool that programmers use whenever they want to execute some piece of code repeatedly. This tool is really important and heavily used in all programming languages. Loops in Python are really easy to use and code, and this is what we are going to learn now. We will see two different types of loop commands: **`while`** and **`for`** and understand what are the differences between them.

First of all, open your text editor and type the code below.

```
def main():

  print "WHILE loop\n"
  y = 0
  while y < 5:
    print "y value is: %d" % (y)
    y = y + 1;

  print "\nFOR loop\n"
  for x in range(0, 5):
    print "x value is: %d" % (x)

  print "\nFOR loop with string\n"
  sentence = "Python is easy!"
  for letter in sentence:
    print "%c" % (letter)

main()
```

Once again, remember, ***watch for the blocks indentation!***

After typing, save and execute your code. Your output should be something like this:



```
PS C:\> python loops.py
WHILE loop

y value is: 0
y value is: 1
y value is: 2
y value is: 3
y value is: 4

FOR loop

x value is: 0
x value is: 1
x value is: 2
x value is: 3
x value is: 4

FOR loop with string

P
y
t
h
o
n

i
s

e
a
s
y
!
PS C:\>
```

Figure 1: Loops executing in Python

Let's understand now what is happening with our code.

We coded three different loops. One **`while`** and two **`for`**. We will start discussing our **`while`**. Loops execute until one condition is met and stop them. The condition to stop our loop is placed right after the `while` word. In this case, we are defining that our loop must execute `while` the value of **`y`** is smaller than 5. For this reason, we must define an initial value for `y`, and that is made on line 4 of our code: `y = 0`. All the code inside the **while block** will be executed until the condition is met. What our loop does is simple print a string with the value of `y`. After that we **increment** our `y`. To increment its value is to add 1 to the previous value of the variable. For each time we run through our while our `y` values raises in 1. We must not forget this line, otherwise our `y` value will never change, hence we will never leave our loop. When `y = 5` the `while` condition is not met anymore, hence our **`while`** ends, and we leave our loop. This is how **`while`** works, pretty simple!

Let's move now to our second loop, our **`for`**. This type of loop is even easier to code than `while` loops. Note here that we do not need to start a variable. We start directly with the `for` command. We define the name of a variable that will **iterate** through our loop. In the previous example we used `y`, here we will use **`x`**. The `for` command will be executed in a range of numbers, that is defined inside the **`in range`** command. You put the initial desired value first, followed by comma, and then the final value. Here, as you can see, our `for` will be executed from 0 to 5. It means that **for each** iteration, `x` will assume the current value of range. Our print simply prints the current value of the variable. Note that here we do not need to explicitly increment our `x`, as we did with our variable in the `while` loop. The **`range`** automatically does that for us. This is how **`for`** works!

The third loop we will that a look is once again a **`for`**. But this is a little bit different from the previous one. On the previous one we used numeric values inside a range to iterate through our loop. Now we will see that we can iterate through **`strings`**. We first define a string named sentence, that has **`Python is easy!`** as its value. Then in our `for` we define a variable called **`letter`** that will iterate through our **`sentence`** variable. What we do is to print the current character `letter` is pointing to. The variable `letter` starts pointing to the first letter of our sentence, at each iteration it moves to the next letter, and keeps doing this, until the end of our string. Once again, we do not need to care about incrementing the `letter` variable, the `for` loop does that automatically for us! Iterate through strings is really easy and useful, you should keep this concept in mind!

These are the two types of loops we are going to see for now. If your code worked and if you understood it all you may move to the exercise on the next page!

**Now try for yourself!**

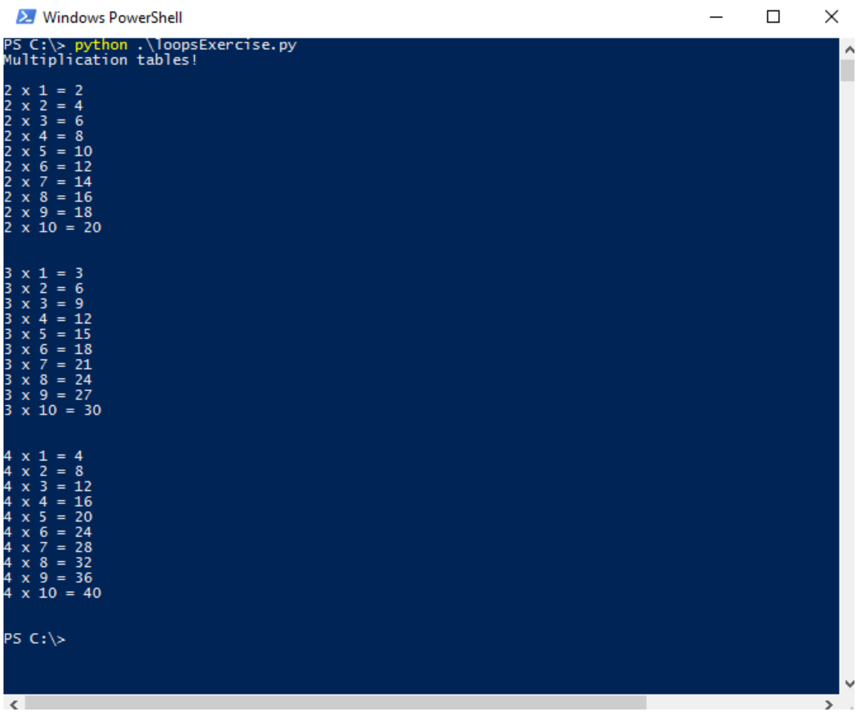
*It's your turn! Create a new file and name it **loopsExercise.py**.*

*One of the things that we did not mention above is that we can have **nested loops**. This is just a fancy way to say that we can put a loop inside another loop.*

*Programming is made to make our lives easier! Your job now is to use Python and `for loops` to generate the multiplication tables of 2, 3 and 4, so whenever you need to check some value you have just to look at your coded table! You should think a little about this problem to get it done, but here are some hints:*

- You should use two loops for this, the outside loop will iterate through which table you want to generate (2, 3 or 4), and the inside loop will iterate through the values from 1 to 10 for that table.
- To multiply two values in Python is really easy: `x * y` will multiply the value stored on variable `x` by the value stored on variable `y`.
- **Don't forget blocks indentation!** Each loop should have its own block.

*This may seem a little confusing but actually it is pretty simple. Think a little and you will get it done easily! In the image below you can see what is expected as output for your code:*



```
PS C:\> python .\loopsExercise.py
Multiplication tables!

2 x 1 = 2
2 x 2 = 4
2 x 3 = 6
2 x 4 = 8
2 x 5 = 10
2 x 6 = 12
2 x 7 = 14
2 x 8 = 16
2 x 9 = 18
2 x 10 = 20


3 x 1 = 3
3 x 2 = 6
3 x 3 = 9
3 x 4 = 12
3 x 5 = 15
3 x 6 = 18
3 x 7 = 21
3 x 8 = 24
3 x 9 = 27
3 x 10 = 30


4 x 1 = 4
4 x 2 = 8
4 x 3 = 12
4 x 4 = 16
4 x 5 = 20
4 x 6 = 24
4 x 7 = 28
4 x 8 = 32
4 x 9 = 36
4 x 10 = 40


PS C:\>
```

Figure 2: Exercise sample output

*Good luck!*