Tutorial – Loops

This tutorial will look at **while loops** and **for loops**, some of their uses and how to code them.

What Is A Loop

A loop is a chunk of code that will run multiple times, either until a particular condition has been met or until it's run a set number of times. They are often used to search for data in groups, random generation or to perform the same task multiple times.

Structure of While Loops

A while loop will keep running while the condition is true. Once it's not, the loop will stop.

```
while (condition is true)
{
        // Run this code
}
```

Whatever code is inside the **curly braces** of the loop will continue to be run until the loop condition is no longer true.

Creating A While Loop

While loops are probably the simplest loop to make. Create a new C# Console project in Visual Studio and call it **Loops.**

Then, in your **Main** function, write the following code:

```
        static void Main(string[] args)
        {
            int count = 0;

            while (count < 3)
            {
                count++;
                Console.WriteLine("The loop has run " + count + " times.");
            }
            Console.ReadKey();
        }
```

We create an int called **count.** We will use this to control how many loops we do.
We create a **while loop**, and the condition for it is, while **count** is **less than 3.**
Inside the loop, we add 1 to count **every time we run the loop.** We also print out how many times the loop has run by using the value of **count.**

Run the program and see what happens.

```
The loop has run 1 times.
The loop has run 2 times.
The loop has run 3 times.
```

You can clearly see that every time we run the loop **count** has been increased by 1. Once **count is 3**, the loop stops.

See what happens when you don't increase count by 1 every loop:

```
        static void Main(string[] args)
        {
            int count = 0;

            while (count < 3)
            {
                //count++;
                Console.WriteLine("The loop has run " + count + " times.");
            }
            Console.ReadKey();
        }
```

Run the program. This is called an **infinite loop** because it never ends - be careful to avoid them. In some progams, like **Unity**, if you accidentally create an infinite loop, the program can freeze up and you'll lose unsaved work. Every programmer can get caught out by this, so it's another reason to save frequently.

Remember in the **functions** tutorial how we got the monster to attack the player 3 times? But we copy and pasted the function call 3 times to achieve it. With a loop, we could just run the loop 3 times, like this:

```
        static void Main(string[] args)
        {
            int count = 0;

            while (count < 3)
            {
                MonsterAttacks();
            }
            Console.ReadKey();
        }
```

Creating A For Loop

With our while loop, we created a variable called **count** to store how many times the loop ran.
We then kept looping a set number of times, based on the value of **count.**
Each loop we ran some code, and increased the value of **count** by 1.

This isn't the only way to use a **while loop**, you can use many other conditions, but this was demonstrated so that you could see how the structure was the same in a for loop:

For (variable to count; how many times to run the loop; change the value of count**)**
{
    // Code to Run
}

Delete the code from your **Main** function and write the following:

```
        static void Main(string[] args)
        {
            for( int count = 0;
                 count < 3;
                 count++)
            {
                Console.WriteLine("The loop has run " + count + " times.");
            }
            Console.ReadKey();
        }
```

Hopefully you can see how similar the above loop is to our previous **while loop.**

We start the loop with the keyword **for.**
We create a variable to count how many times our loop has run called **count.**
Then we say that, at the end of each loop, we want to increase **count** by 1.
In between the **curly braces,** as before, we print how many times the loop has run.

Run the program and see what happens.

```
The loop has run 0 times.
The loop has run 1 times.
The loop has run 2 times.
```

The loop runs exactly the same amount of times as before, although the values we see is slightly different. This is because in our **while loop,** we changed the value of **count** manually ourselves at the start of the loop. In a **for loop**, you define how you want to modify the value **count** inside the **parentheses** of the loop, which means the **for loop** automatically modifies **count** for you **at the end** of the loop.

So even though the loop ran the same amount of times, the variable **count** doesn't actually change until after each time the loop has run.

For loops are typically laid out in this format:

```
        static void Main(string[] args)
        {
            for( int count = 0; count < 3; count++)
            {
                Console.WriteLine("The loop has run " + count + " times.");
            }
            Console.ReadKey();
        }
```

You declare the variable **count** (also commonly referred to as **i**), define how many times you want the loop to run and then specify how you want to change the **count** variable all in one line inside the parentheses. Splitting it across multiple lines, however, made it easier for you to read while you're learning.

Count doesn't have to start at 0.
Count can be increased or decreased, and you can do it by values greater than 1.
You can use any of the relational operators that you use to check if statements to check the value of count in the middle condition. Example:

```
for( int count = 3; count >= 1; count -= 2)
 {
     Console.WriteLine("The value of count is now " + count + ".");
 }
```

Try changing your loop to the above example. What happens?
We decrease the value of count by 2, we start count at 3, and we keep running the loop while count is greater than **or** equal to a value of 1.

Breaking Out Of A Loop

If for some reason you want to end a loop early, you can break out of one at any time with the key word **break.** Modify your loop as follows:

```
for( int count = 3; count >= 0; count -= 1)
 {
     Console.WriteLine("The value of count is now " + count + ".");
     if (count == 2)
     {
         break;
     }
 }
 Console.WriteLine("The loop is now finished.");
 Console.ReadKey();
```

Run the program and see what happens.

We put an **if statement** checking to see if **count is 2**. If it is, we use the command **break;**
Break is a **keyword** that literally means **"break out of the loop now."**
After you reach **break** in a loop, no more code will run – it stops the loop immediately and run whatever code comes after the loop, which is why you see the text on the next line print.

Returning From A Loop

Remember how functions can return values? If you use **return** inside a loop, instead of just exiting the loop, it will exit the entire function:

```
for( int count = 3; count >= 0; count -= 1)
 {
     Console.WriteLine("The value of count is now " + count + ".");
     Console.ReadKey();
     if (count == 2)
     {
         return;
     }
 }
 Console.WriteLine("The loop is now finished.");
 Console.ReadKey();
```

We add **ReadKey** this time to pause the application every time the loop runs to help demonstrate this lesson. Run the program and see what happens.

Once **count** reaches a value of 2 and the **if statement** is true, the loop reaches **return** and exits out of the **Main** function, never printing the "**Loop is now finished"** text. Because **Main** is our only function, when we exit out of it, the program ends, which is why the application suddenly shuts down.

Further Uses

So far you have only seen very basic loops without much practical use. In your game projects you will make use of loops in lots of different ways, the most common being to search through groups of data for a specific item of data. For example, if you were searching through a list of enemies for the one with the lowest health. For now, it's enough to understand how they work, and later you will learn more of their uses.

Conclusion

Try experimenting with both types of loops by creating some of your own. Try experimenting with different conditions for while loops, such as making a **bool** true or false. Try using **for loops** in different ways, by changing **count** by values other than 1, or try making some of your own functions and if statements and combining them with loops.

Remember if you want to make other functions to put them inside a new class, as in the **Functions & Intro To Classes** tutorial.