# 08 What Are Loops?

May 12, 2022

## 1 What Are Loops?

Loops are a way to repeat an action multiple times. You can use `while` to do that. > Try out this first loop:

[1]:
```
int counter = 0;
while (counter < 10)
{
  Console.WriteLine($"Hello World! The counter is {counter}");
  counter++;
}
```

```
Hello World! The counter is 0
Hello World! The counter is 1
Hello World! The counter is 2
Hello World! The counter is 3
Hello World! The counter is 4
Hello World! The counter is 5
Hello World! The counter is 6
Hello World! The counter is 7
Hello World! The counter is 8
Hello World! The counter is 9
```

So what happened here? `while` checks if the condition ('counter < 10') is true. If it is, it goes through the loop. Then it checks it again. It will keep going through the loop until the condition is false. When figuring out branches and loops, it can be helpful to put code into human language. This code is saying: "Set up counter to equal 0. While the counter is less than 10, print 'Hello World! the counter is {counter}' and then increase the counter by 1".

### 1.1 ++

`++` is a quick quick way to add one to a variable. You can also do the same with `--`, which subtracts one from the variable.

### 1.2 Infinite loops

It's easy to accidentally make an infinite loop. **Editors note: You can't tell them to create an infinite loop, because the notebook doesn't ever stop itself** If you hit an infinite loop, you have two options: you can always exit out of your program, or you can hit CTRL+C.

> Challenge: You can't show one on this notebook, because the code doesn't catch itself, but as a challenge, try making an infinite loop in Visual Studio! What are the different ways you might accidentally make it infinite? It can be scary to make a mistake, but always know you can CTRL+C or exit the program.

## 1.3 do

You can also make a loop with `do`. > Try out the following code and see if there's anything different between this and the `while` loop.

[3]:
```
int counter = 0;
do
{
  Console.WriteLine($"Hello World! The counter is {counter}");
  counter++;
} while (counter < 10);
```

```
Hello World! The counter is 0
Hello World! The counter is 1
Hello World! The counter is 2
Hello World! The counter is 3
Hello World! The counter is 4
Hello World! The counter is 5
Hello World! The counter is 6
Hello World! The counter is 7
Hello World! The counter is 8
Hello World! The counter is 9
```

Hmm, nothing much seems to different, right?

> Try changing the conditional to `(counter < 0)` on both codes. What happens?

In the `while` loop, nothing is printed out, but in the `do` loop, you get one print out. The `do` loop does the action first, then checks the conditional. In contrast, the `while` loop checks the conditional first, then does an action.

## 1.4 for loop

`for` is one the most common loops. It will repeat an action for a specific amount of turns. It can look a little intimidating. > Before you learn all the ins and outs of the code, run the following for loop just to see it working:

[4]:
```
for (int counter = 0; counter < 10; counter++)
{
  Console.WriteLine($"Hello World! The counter is {counter}");
}
```

```
Hello World! The counter is 0
Hello World! The counter is 1
Hello World! The counter is 2
Hello World! The counter is 3
Hello World! The counter is 4
Hello World! The counter is 5
Hello World! The counter is 6
Hello World! The counter is 7
Hello World! The counter is 8
Hello World! The counter is 9
```

### 1.4.1 What's in the parentheses?

There are three parts in the `for` loop parentheses: `int counter = 0`, `counter < 10`, and `counter++`. You can think of them as the start point, the end point, and the step size. `int counter = 0` is just setting up the counter. You're starting at 0. `counter < 10` is the conditional that gets checked at the start of every loop. Once the conditional is false (in this case, once counter is not less than 10), the loop is done and the code moves on. `counter++` is increasing counter by one, taking one step closer to the end. The step is taken at the end of each loop. > Try messing with the `for` loop set up. How does it change?

[ ]:
```

```