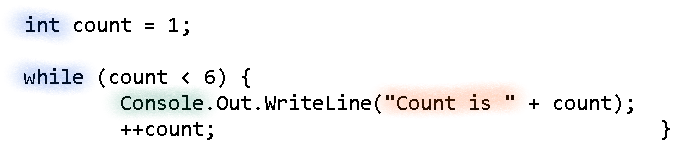
There are two types of loops in C#, called the while loop and the for loop.

A **while loop** tests for a certain condition, and if that condition is true, one or more lines of code can be executed.

When the interior code of a while loop is finished, the condition will be checked again.

You can also use a **do-while** loop to make sure that the condition is checked only after executing the contained code, which guarantees that the code will always execute at least once.

Let’s see a few examples.



Here we have an integer named count, which starts with a value of one. While count is less than 6, we will print a message and increment count by 1.

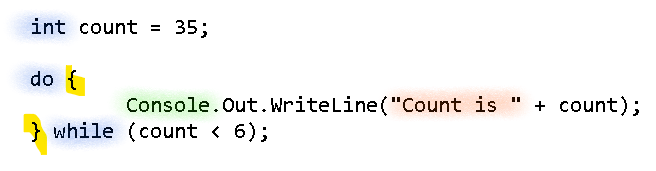
When the loop is over, a message will print notifying us.

When we run the program, we see that this performs as expected.

But what if we removed the increment? Now the loop will run forever.

This is a risk you need to be aware of when programming loops – *always make sure the looping condition will eventually be violated.*

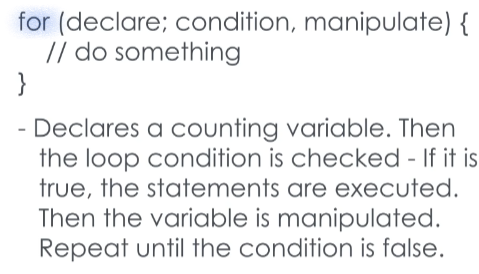
Let’s change this to a **do-while loop**, and change the condition to something that will never be met.



Now when we run the program again, we can confirm that it will still print the looping message at least once.

You’ll have noticed that a while loop requires some kind of manipulation to make sure the looping condition is met after some time.

If you want to make sure a loop only runs a certain number of times, or loops over every item in a collection, you need to do something like increment a counting variable.

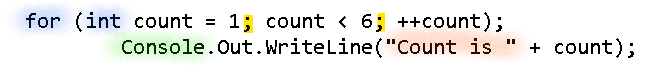
There’s a better way to do that, and that’s a for loop.

A **for loop** is constructed with an optional counting variable declaration, a loop condition, and a manipulation statement.

Before the for loop runs, the counting variable is initialized.

Then the loop condition is checked, and if it’s valid, the loop code runs.

When the loop code is complete, the manipulation statement automatically runs, and the looping condition is checked again.



Here we have a reconstruction of the while loop example we showed before, but structured as a for loop.

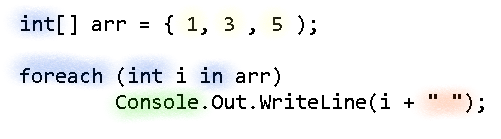
Note that the count variable is actually initialized inside of the for loop.

Two semicolons divide the initialization, the comparison, and the manipulation statements.

Inside the for loop, we have a print statement, but that’s it.

When we run this application, we’ll see that it runs exactly the same as the while loop.

There is also the **foreach loop**, shown here.



The foreach loop iterates over every item in a collection.

This usefulness of this will become more apparent as you become more familiar with collections, but here’s a quick example.

Here we have an array of integers, prepopulated with three different numbers.

The foreach loop declares a temporary variable here, which is ‘i’.

For each loop iteration, a value from the collection will be stored in the temporary variable, which you can use to manipulate it.

Here we simply iterate through the array and print the numbers inside