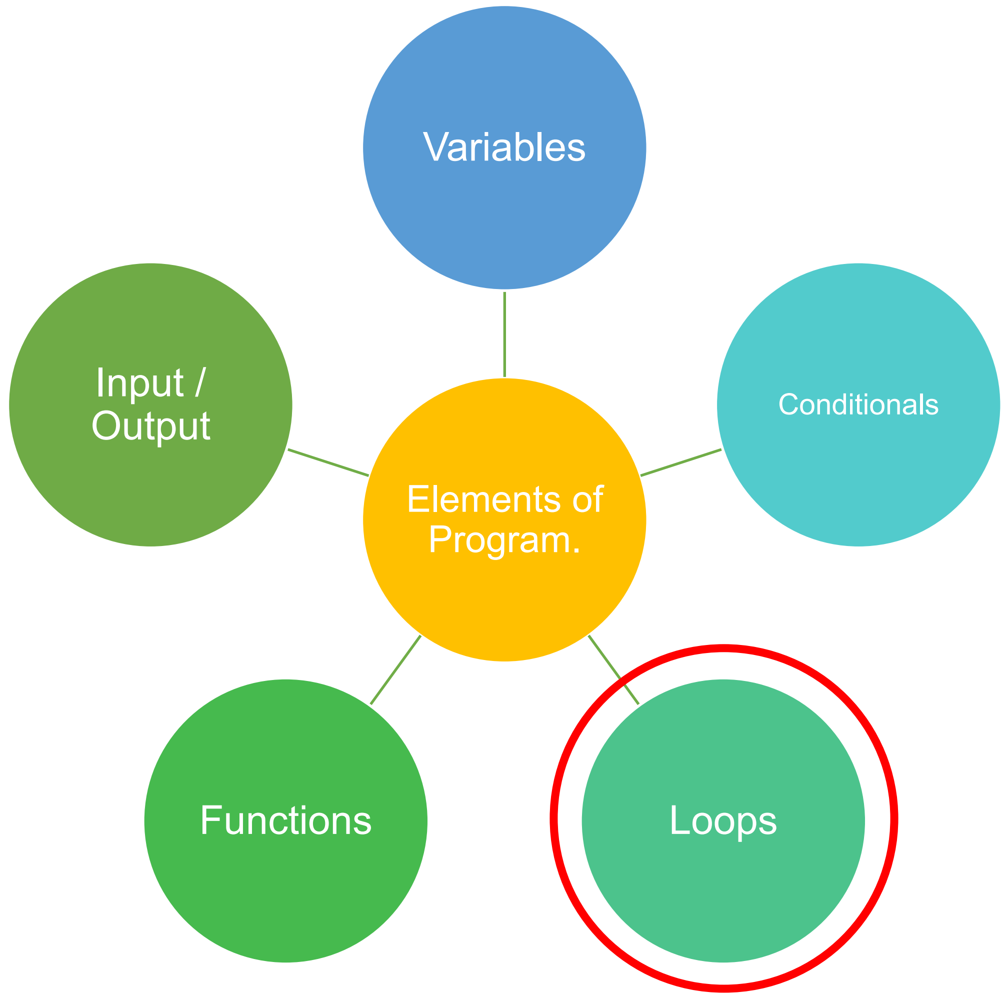
Loops and Arrays

A4 – Constructs and Techniques and Their Implementation in Programming

Ikromkhujaev Ilyoskhuja

Elements

- To develop any instruction there are some elements needed or we can essentially present in all language.
- So any programming language is made up of 5 basic elements of the instructions.

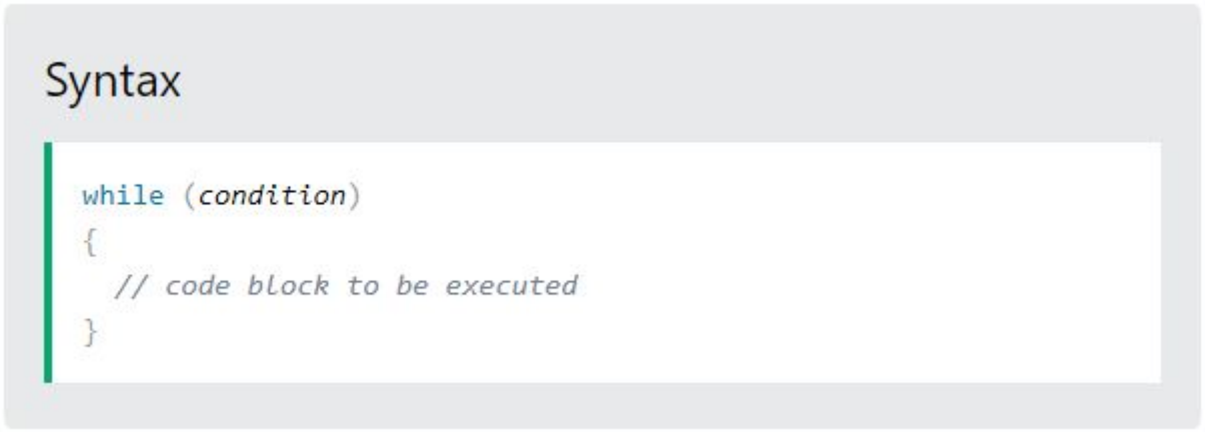


Loops

- Almost all the programming languages provide a concept called loop, which helps in executing one or more statements up to a desired number of times.
- A loop is a sequence of instruction s that is continually repeated until a certain condition is reached.
- Loops can execute a block of code as long as a specified condition is reached.
- Loops are handy because they save time, reduce errors, and they make code more readable.

Loops – While Loop

The `while` loop loops through a block of code as long as a specified condition is `True`:

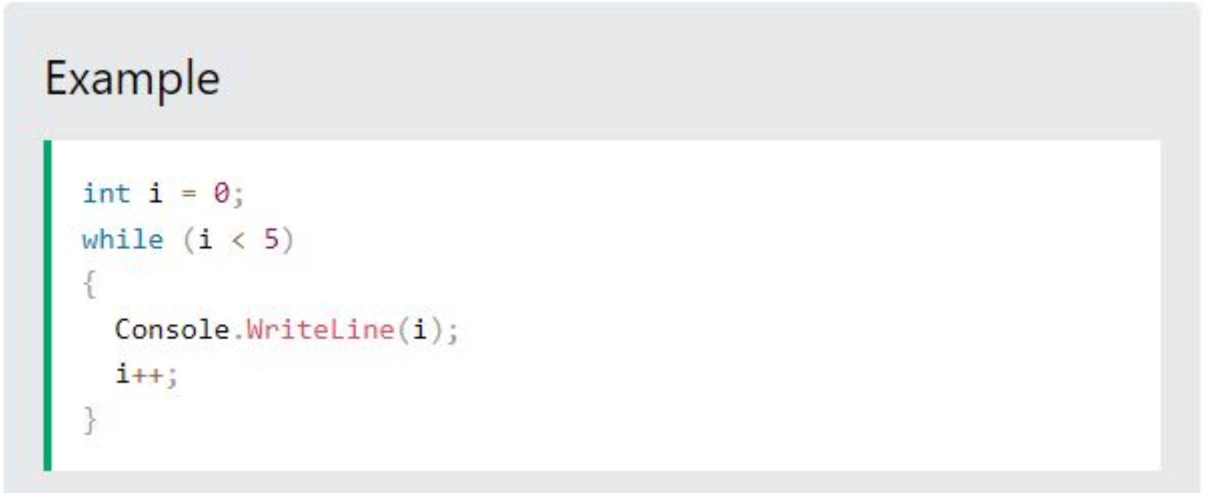


Syntax

```
while (condition)
{
  // code block to be executed
}
```

Loops – While Loop

In the example below, the code in the loop will run, over and over again, as long as a variable (i) is less than 5:



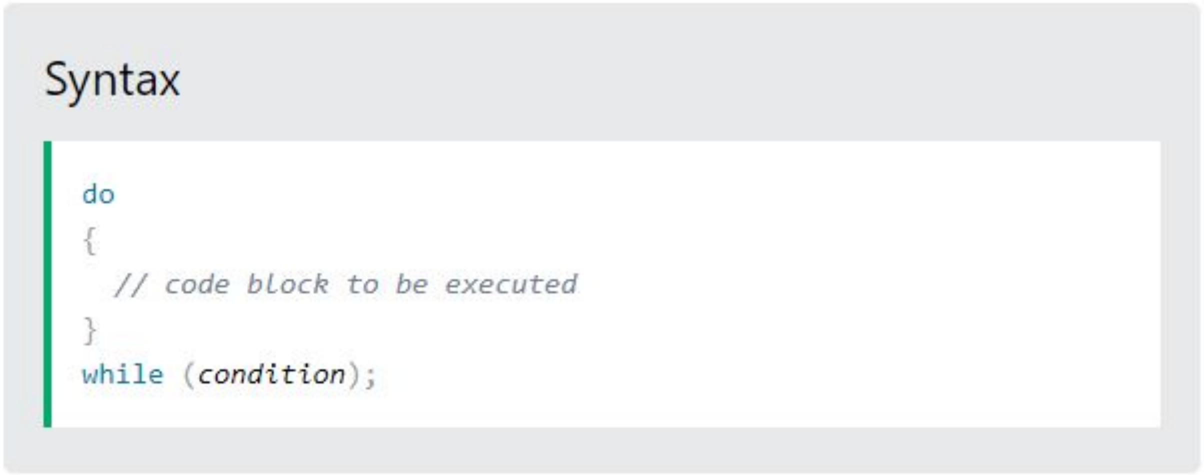
Example

```
int i = 0;
while (i < 5)
{
  Console.WriteLine(i);
  i++;
}
```

Note: Do not forget to increase the variable used in the condition, otherwise the loop will never end!

Loops – Do/While Loop

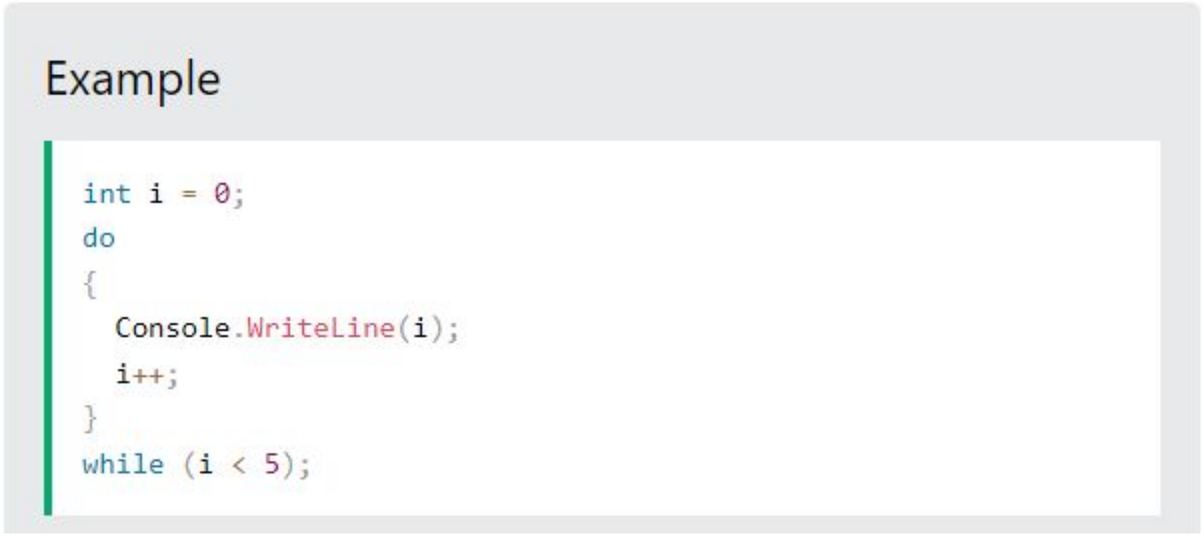
The `do/while` loop is a variant of the `while` loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.


Syntax

```
do
{
  // code block to be executed
}
while (condition);
```

Loops – Do/While Loop

The example below uses a `do/while` loop. The loop will always be executed at least once, even if the condition is false, because the code block is executed before the condition is tested:



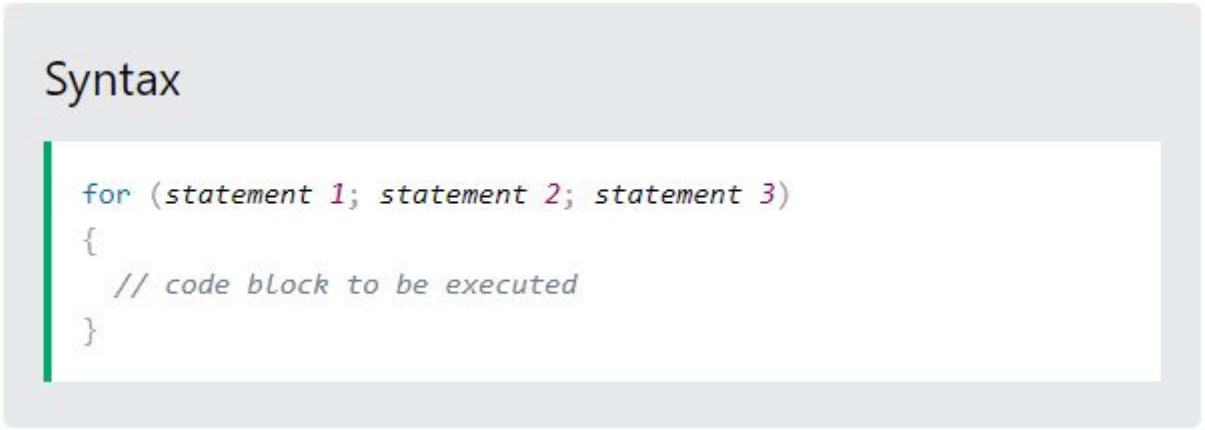
Example

```
int i = 0;
do
{
  Console.WriteLine(i);
  i++;
}
while (i < 5);
```

Note: Do not forget to increase the variable used in the condition, otherwise the loop will never end!

Loops – For Loop

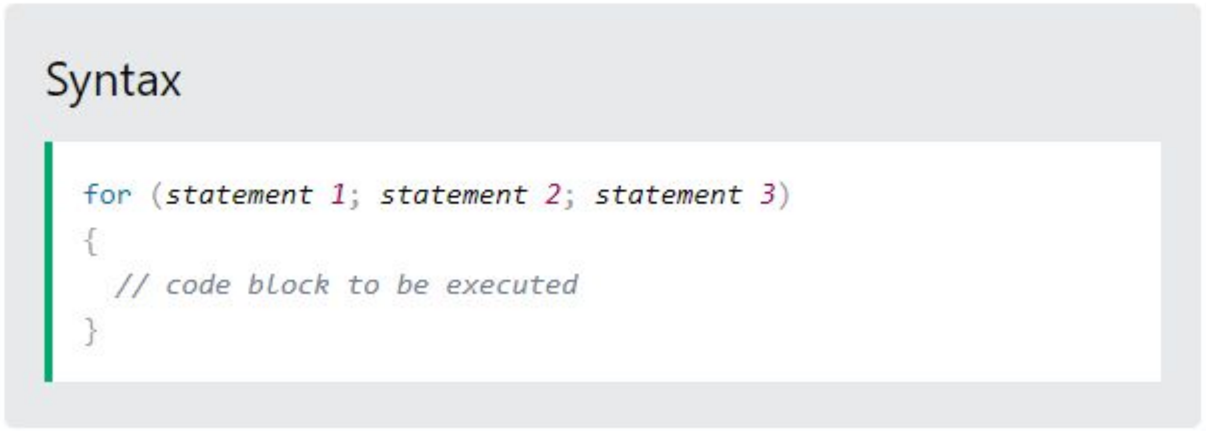
When you know exactly how many times you want to loop through a block of code, use the `for` loop instead of a `while` loop:



Syntax

```
for (statement 1; statement 2; statement 3)
{
  // code block to be executed
}
```

Loops – For Loop



Syntax

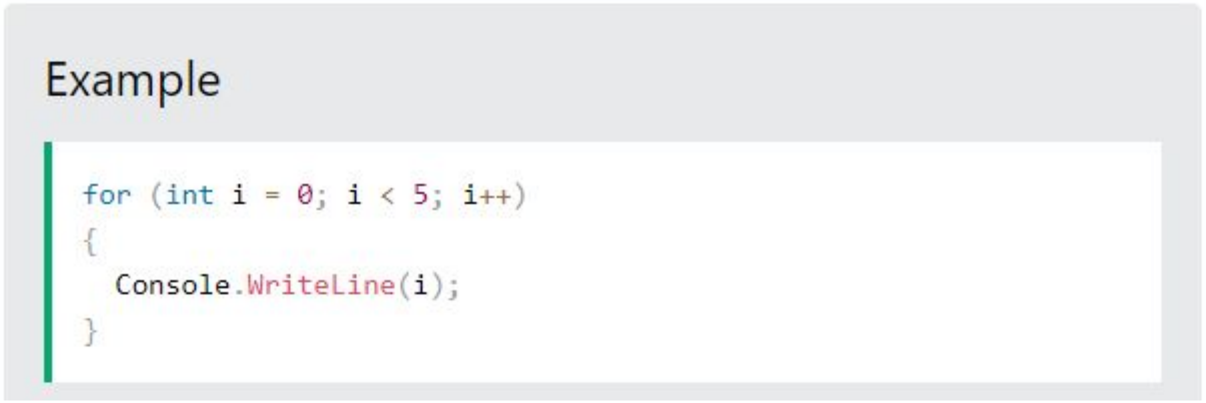
```
for (statement 1; statement 2; statement 3)
{
  // code block to be executed
}
```

Statement 1 is executed (one time) before the execution of the code block.

Statement 2 defines the condition for executing the code block.

Statement 3 is executed (every time) after the code block has been executed.

Loops – For Loop


Example

```
for (int i = 0; i < 5; i++)
{
  Console.WriteLine(i);
}
```

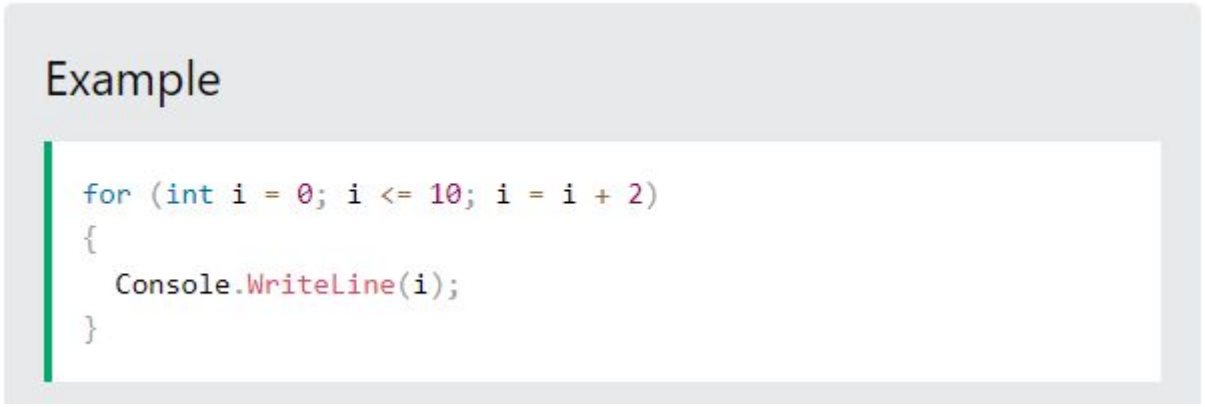
Statement 1 sets a variable before the loop starts (`int i = 0`).

Statement 2 defines the condition for the loop to run (`i` must be less than `5`). If the condition is `true`, the loop will start over again, if it is `false`, the loop will end.

Statement 3 increases a value (`i++`) each time the code block in the loop has been executed.

Loops – For Loop

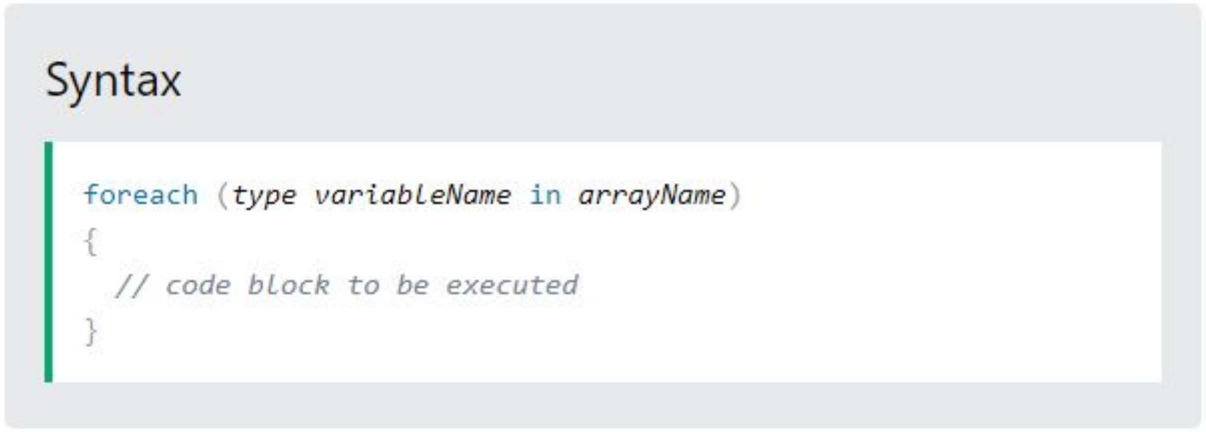
This example will only print even values between 0 and 10:


Example

```
for (int i = 0; i <= 10; i = i + 2)
{
  Console.WriteLine(i);
}
```

Loops – Foreach Loop

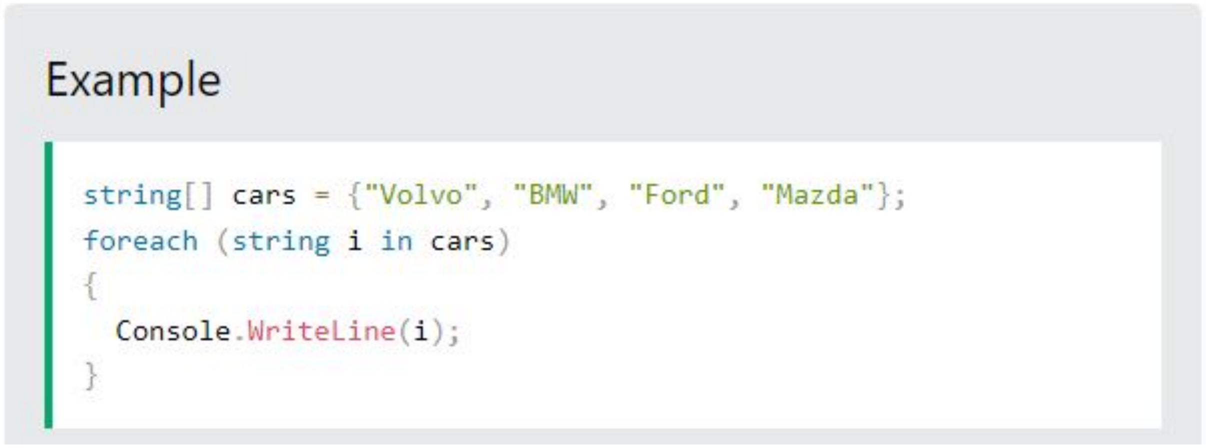
There is also a `foreach` loop, which is used exclusively to loop through elements in an **array**:


Syntax

```
foreach (type variableName in arrayName)
{
  // code block to be executed
}
```

Loops – Foreach Loop

The following example outputs all elements in the **cars** array, using a foreach loop:



Example

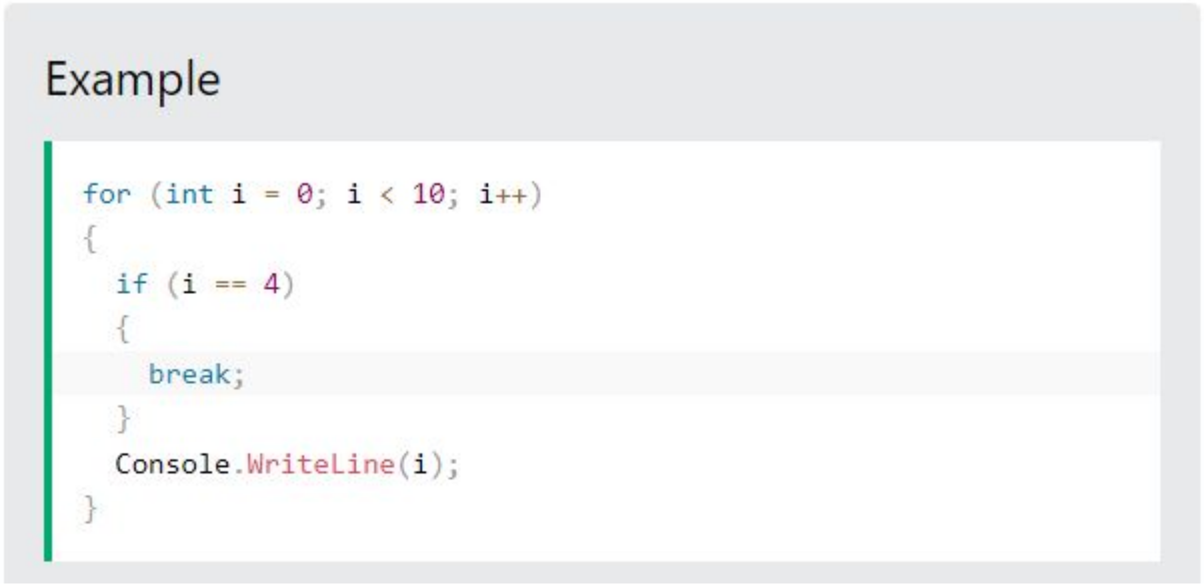
```
string[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
foreach (string i in cars)
{
  Console.WriteLine(i);
}
```

We will discuss Arrays soon.

Loops – Break

We have already seen the `break` statement used in an earlier slide of this lesson. It was used to "jump out" of a `switch` statement.

- The `break` statement can also be used to jump out of a loop.
- This example jumps out of the loop when `i` is equal to `4`:



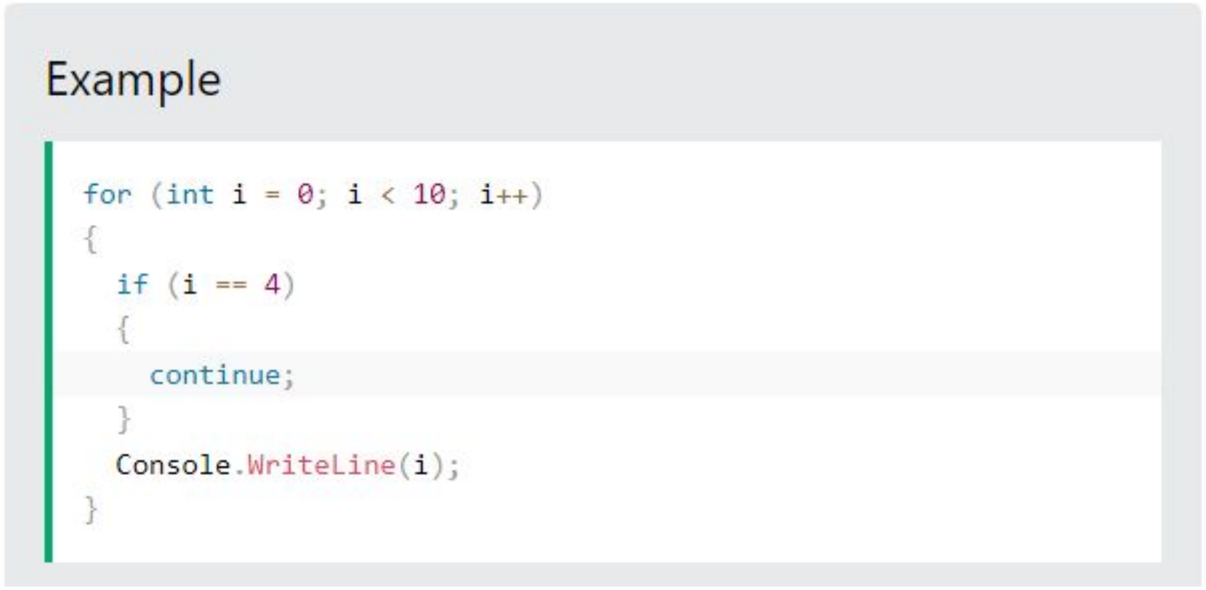
Example

```
for (int i = 0; i < 10; i++)
{
  if (i == 4)
  {
    break;
  }
  Console.WriteLine(i);
}
```

Loops – Continue

The `continue` statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

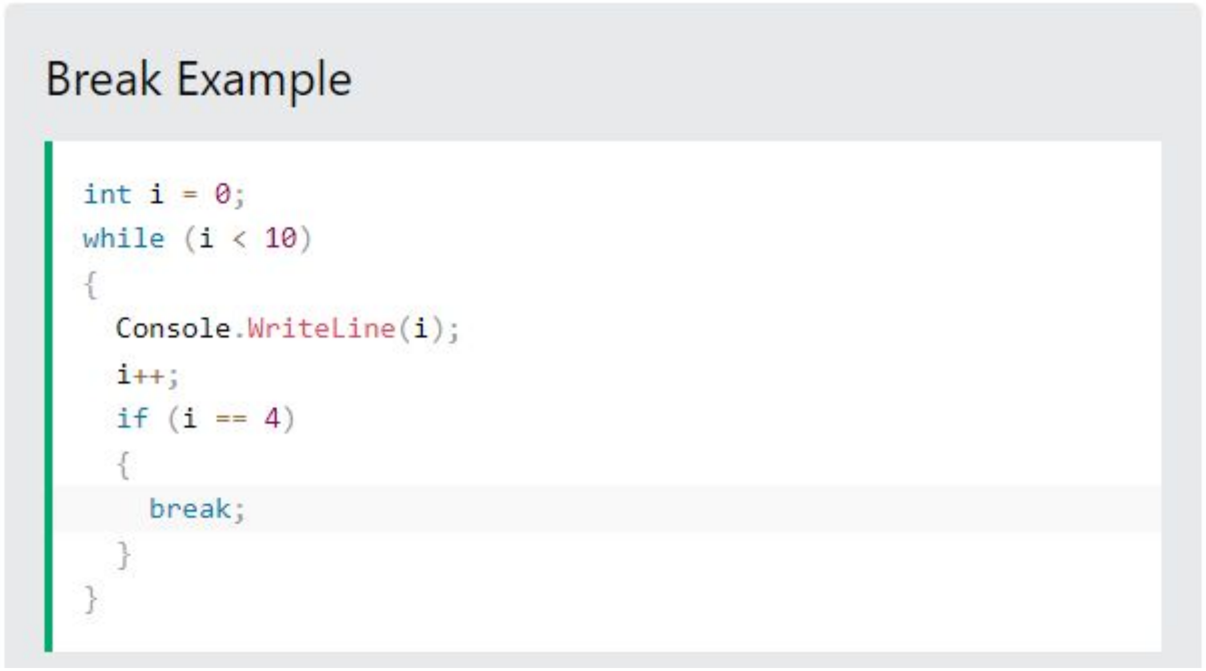
This example skips the value of `4`:


Example

```
for (int i = 0; i < 10; i++)
{
  if (i == 4)
  {
    continue;
  }
  Console.WriteLine(i);
}
```

Loops – Break and Continue in While

You can also use `break` and `continue` in while loops:

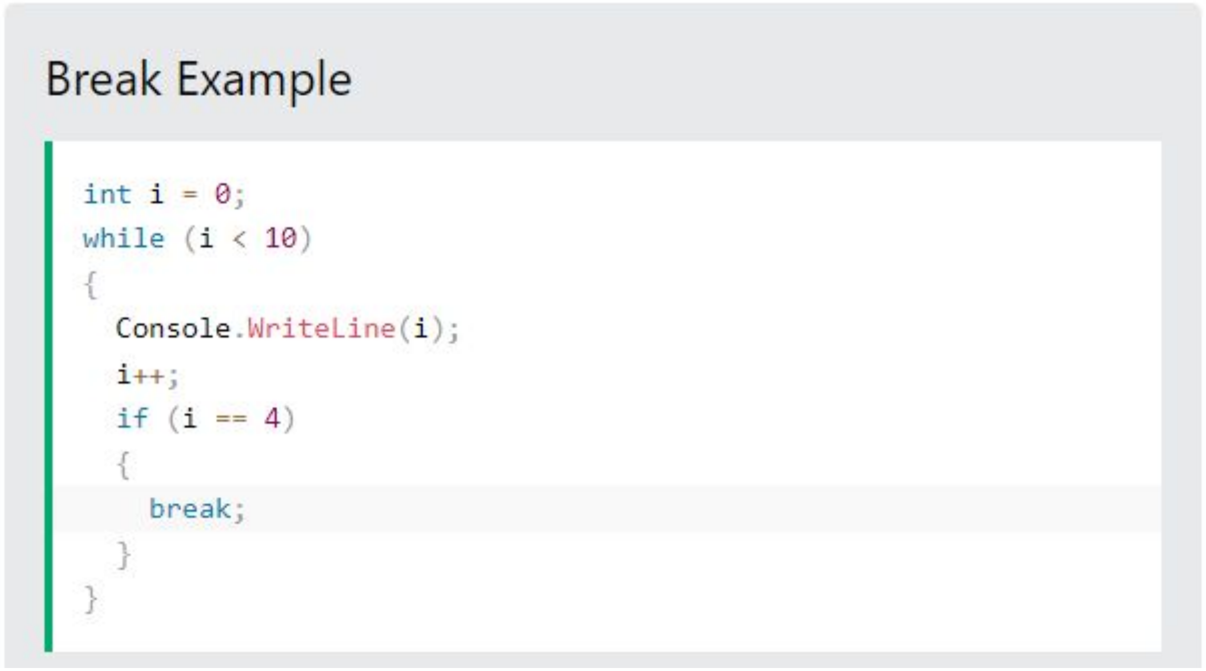


Break Example

```
int i = 0;
while (i < 10)
{
  Console.WriteLine(i);
  i++;
  if (i == 4)
  {
    break;
  }
}
```

Loops – Break and Continue in While

You can also use `break` and `continue` in while loops:

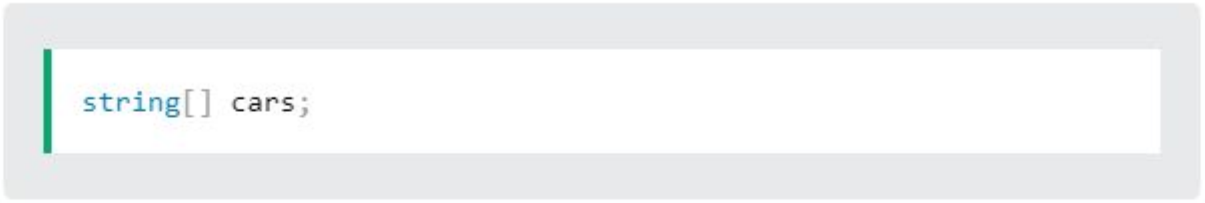


Break Example

```
int i = 0;
while (i < 10)
{
  Console.WriteLine(i);
  i++;
  if (i == 4)
  {
    break;
  }
}
```

Arrays

- Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.
- To declare an array, define the variable type with square **brackets**:

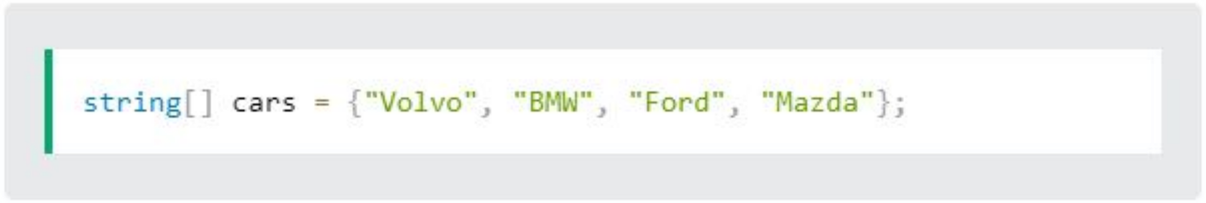


```
string[] cars;
```

- We have now declared a variable that holds an array of strings.

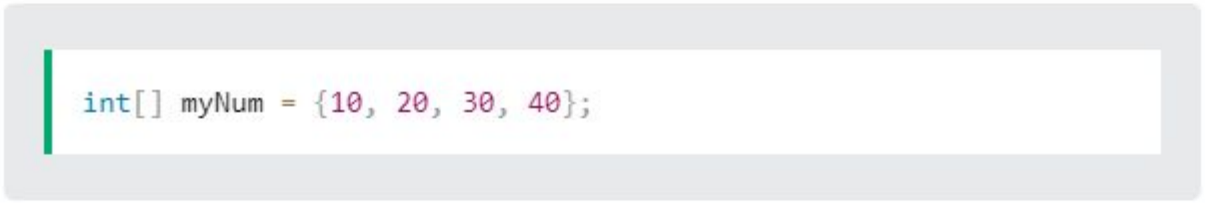
Arrays

- To insert values to it, we can use an array literal - place the values in a comma-separated list, inside curly braces:



```
string[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
```

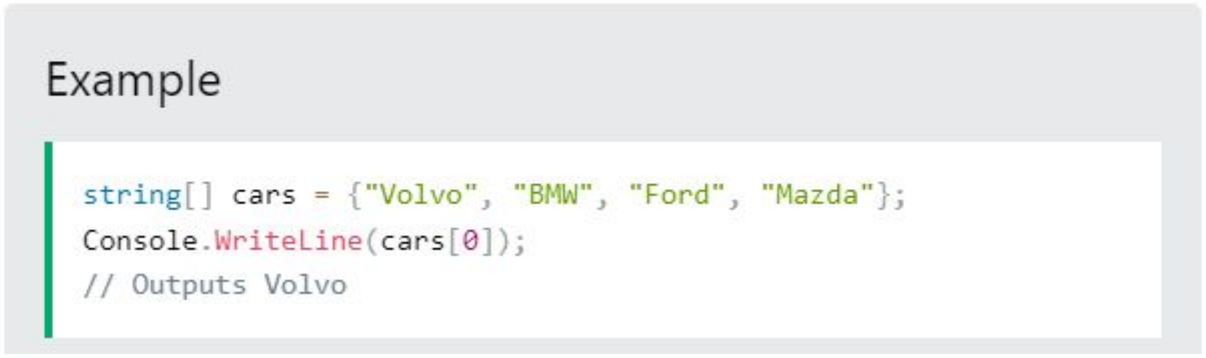
- To create an array of integers, you could write:



```
int[] myNum = {10, 20, 30, 40};
```

Access the Elements of an Array

- You access an array element by referring to the index number.
- This statement accesses the value of the first element in cars:



Example

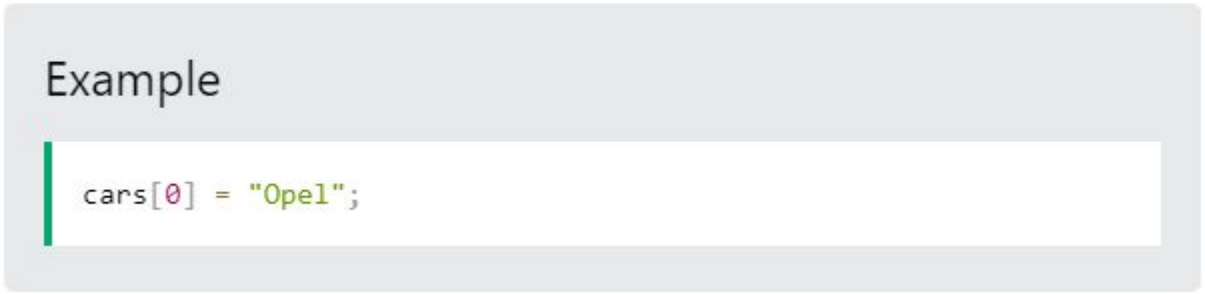
```
string[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
Console.WriteLine(cars[0]);
// Outputs Volvo
```

Array indexes start with 0: [0] is the first element.

[1] is the second element, etc.

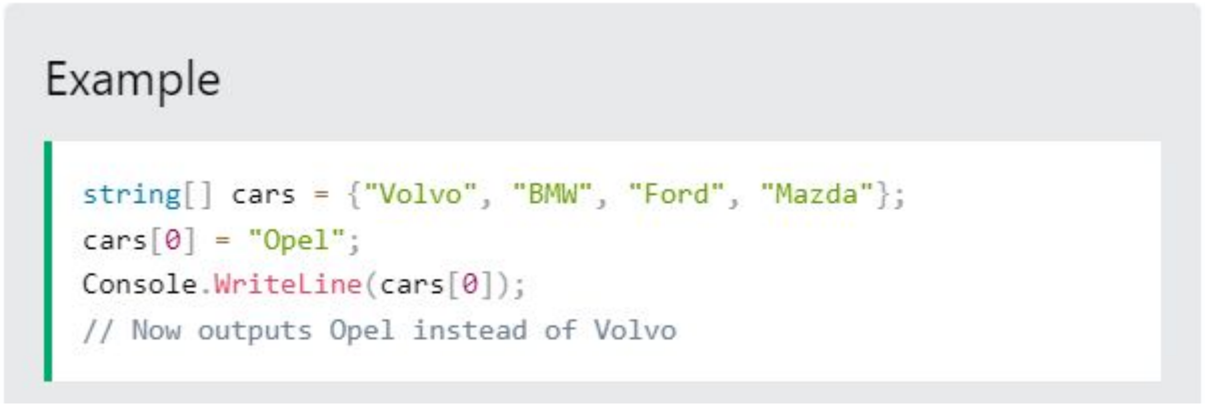
Change an Array Element

To change the value of a specific element, refer to the index number:



Example

```
cars[0] = "Opel";
```

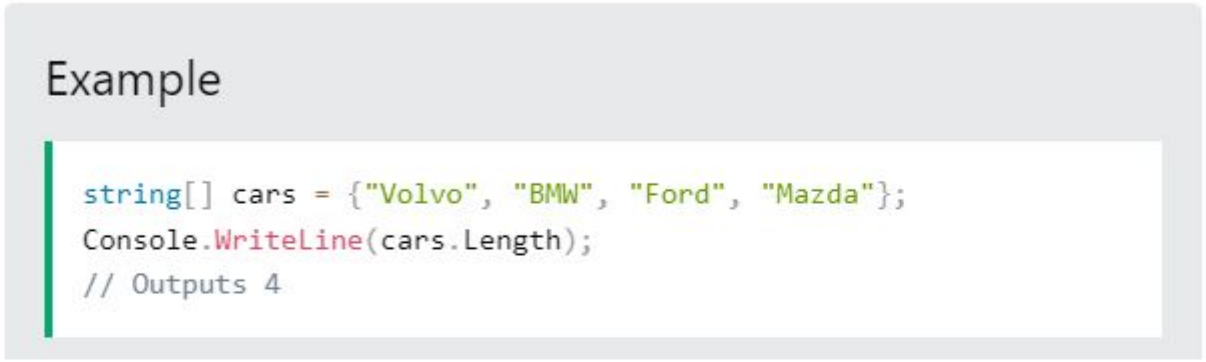


Example

```
string[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
cars[0] = "Opel";
Console.WriteLine(cars[0]);
// Now outputs Opel instead of Volvo
```

Array Length

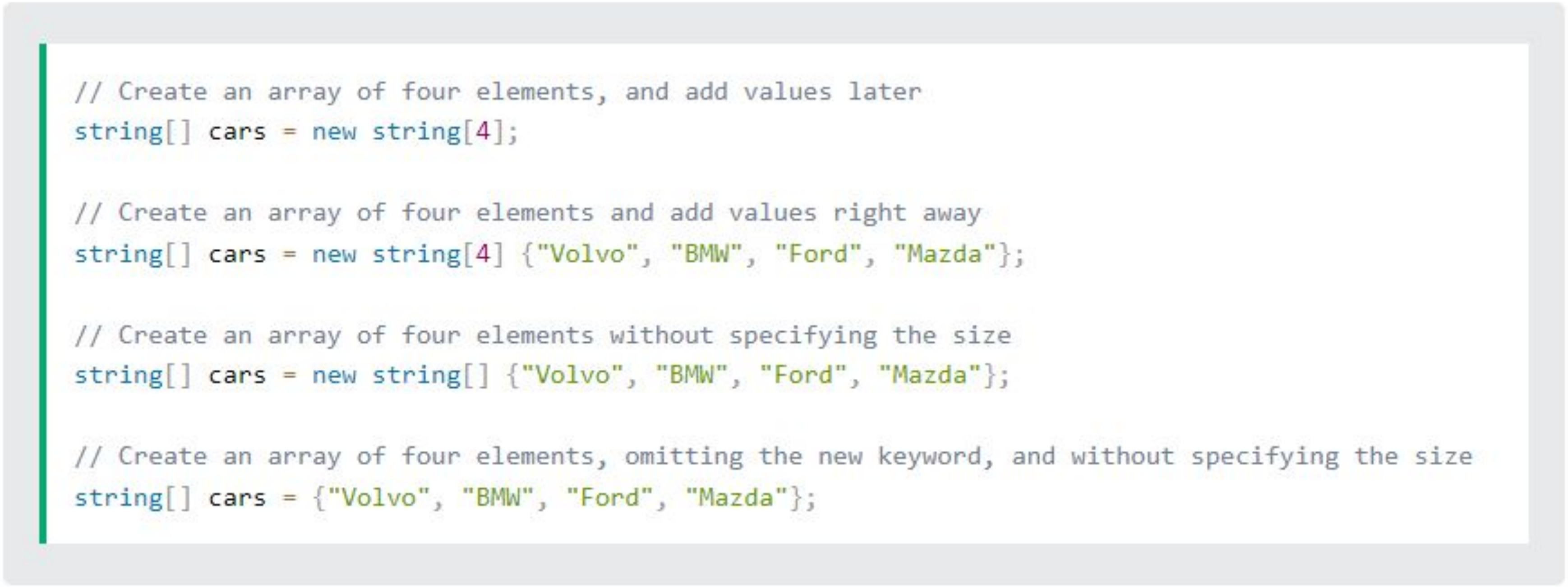
To find out how many elements an array has, use the Length property:


Example

```
string[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
Console.WriteLine(cars.Length);
// Outputs 4
```

Other Ways to Create an Array

If you are familiar with C#, you might have seen arrays created with the `new` keyword, and perhaps you have seen arrays with a specified size as well. In C#, there are different ways to create an array:

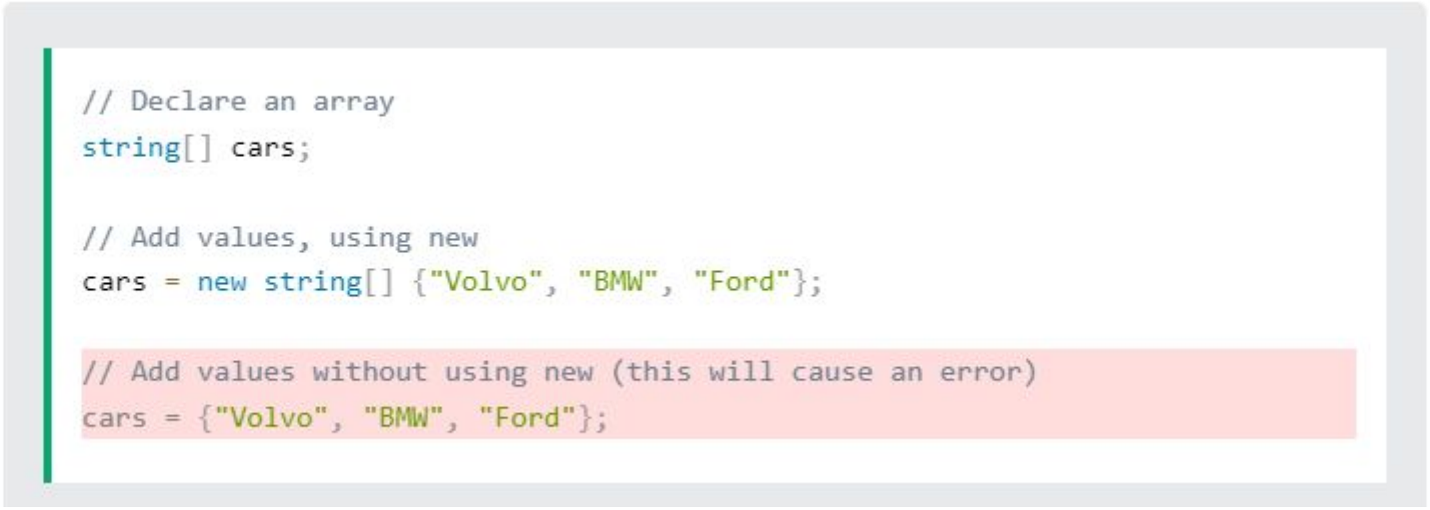


```
// Create an array of four elements, and add values later
string[] cars = new string[4];

// Create an array of four elements and add values right away
string[] cars = new string[4] {"Volvo", "BMW", "Ford", "Mazda"};

// Create an array of four elements without specifying the size
string[] cars = new string[] {"Volvo", "BMW", "Ford", "Mazda"};

// Create an array of four elements, omitting the new keyword, and without specifying the size
string[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
```

Other Ways to Create an Array

It is up to you which option you choose. In our tutorial, we will often use the last option, as it is faster and easier to read.

However, you should note that if you declare an array and initialize it later, you have to use the `new` keyword:

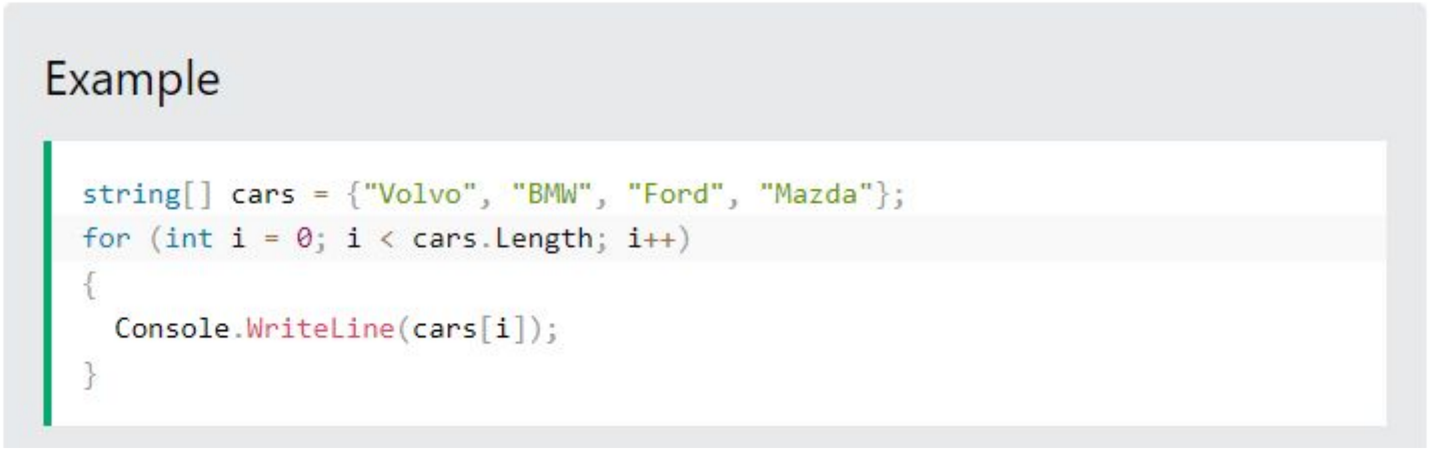


```
// Declare an array
string[] cars;

// Add values, using new
cars = new string[] {"Volvo", "BMW", "Ford"};

// Add values without using new (this will cause an error)
cars = {"Volvo", "BMW", "Ford"};
```

Loop Through Arrays

You can loop through the array elements with the for loop, and use the Length property to specify how many times the loop should run.

The following example outputs all elements in the **cars** array:

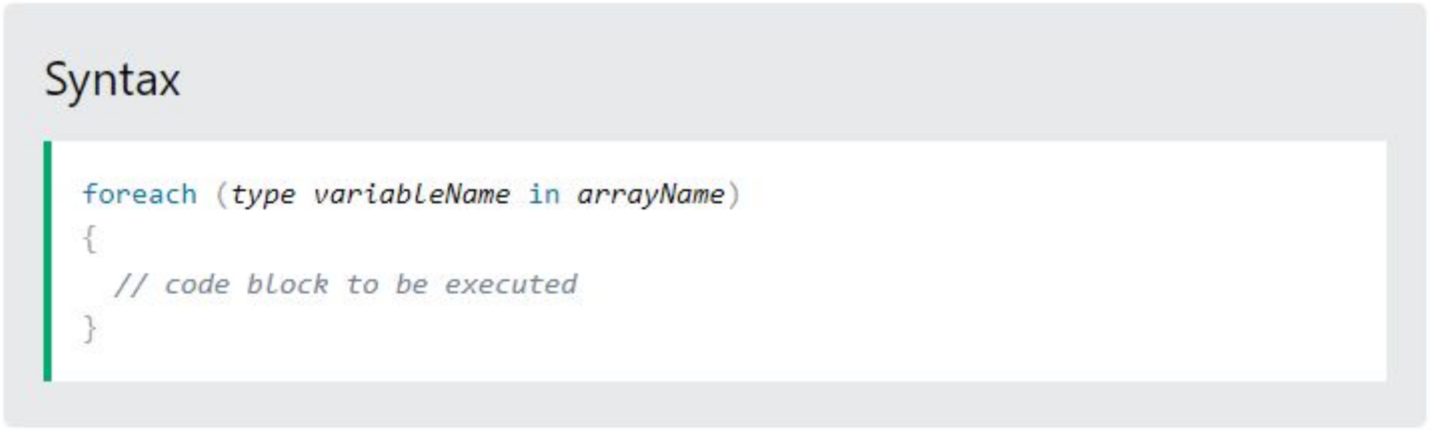


Example

```
string[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
for (int i = 0; i < cars.Length; i++)
{
  Console.WriteLine(cars[i]);
}
```

Loop Through Arrays – Foreach Loop

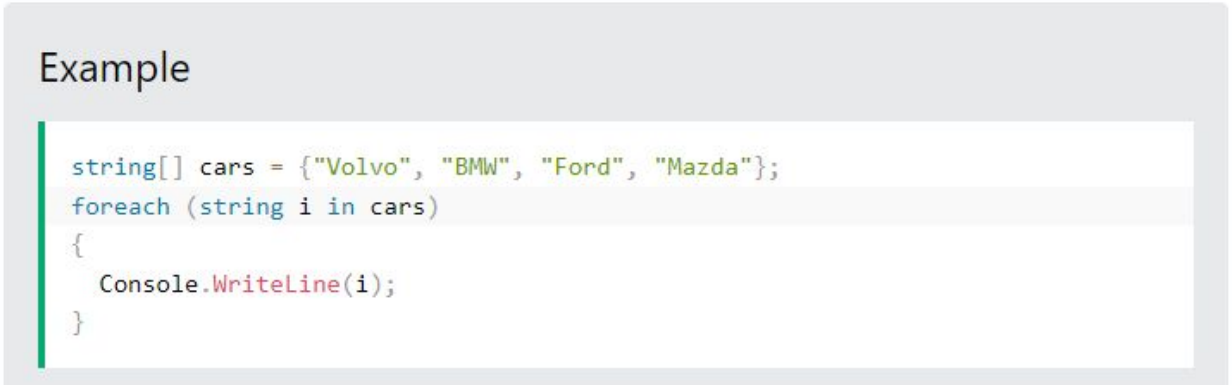
There is also a `foreach` loop, which is used exclusively to loop through elements in an **array**:


Syntax

```
foreach (type variableName in arrayName)
{
  // code block to be executed
}
```

Loop Through Arrays – Foreach Loop

The following example outputs all elements in the **cars** array, using a `foreach` loop:



Example

```
string[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
foreach (string i in cars)
{
  Console.WriteLine(i);
}
```

If you compare the `for` loop and `foreach` loop, you will see that the foreach method is easier to write, it does not require a counter (using the `Length` property), and it is more readable.

Next lecture

- In the next lecture we continue focusing on «Constructs and techniques and their implementation»
- “Microsoft Visual C# Step by Step” Microsoft Press