

## LESSON 02 – 1D ARRAY METHODS



In this lesson we explore the various array methods that are available in C#, which makes using arrays much easier.

## Sections:

- I. [C# METHODS VS FUNCTIONS? ..... PAGE 1](#)
- II. [ARRAY SORTING METHOD..... PAGE 3](#)
- III. [ARRAY MIN, MAX, SUM, AND AVERAGE METHODS..... PAGE 4](#)
- IV. [.SPLIT\(\) METHOD ..... PAGE 5](#)
- V. [OTHER USEFUL ARRAY METHODS..... PAGE 6](#)

## I. C# METHODS VS FUNCTIONS:

Up until this point we have been using the term built-in ‘**functions**’. However, in C#, the proper terminology is usually ‘**method**’. The main difference (among others that we won’t get into) is that a method is attached to a **variable** or **object** and a function is not (for the most part). For example:

- `Console.WriteLine();`
  - The `WriteLine()` **method** is attached to the `Console` **object**.
- `myString.ToUpper();`
  - The `ToUpper()` **method** is attached to the `myString` **variable**.

A **function** is not attached to a **variable** or **object**. For example:

- `MyFunction();`

We have not really seen any use of such functions on their own (i.e., not attached to any variable or object), and most of the time you will be using methods anyway. Therefore, from this point forward we will use the term **method**, except for when we are accessing a property. For example:

- `myString.Length;`
  - `.Length` is a **property** that is attached to the `myString` **variable**.
  - **Recall:** Properties have no brackets ( ) at the end.

You may still see the terms 'function' and 'method' used interchangeably if you are looking at source code on the internet. Some programming languages, like JavaScript, exclusively use the term 'function'. And even some C# programmers will still refer to methods as functions. Either way, the difference in terminology should not affect your understanding of the code. We'll stick with the term '**method**' since that is the proper C# terminology.

**Note: 'Objects'** have yet to be discussed in detail, but for now you can consider them as something that contains a set of methods and properties.

## II. ARRAY SORTING METHOD:

To sort an array, we can use the **Array.Sort()** method:

```
string[] fruits = { "banana", "apple", "watermelon", "pear", "grape" };
Array.Sort(fruits);
for(int x = 0; x < fruits.Length; x++)
{
    if(x < fruits.Length - 1)
        Console.Write(fruits[x] + ", ");
    else
        Console.Write(fruits[x]);
}

int[] myInts = {3, 5, 1, 12, 10};
Array.Sort(myInts);
Console.WriteLine('\n');
for(int x = 0; x < myInts.Length; x++)
{
    if(x < myInts.Length - 1)
        Console.Write(myInts[x] + ", ");
    else
        Console.Write(myInts[x]);
}
```

### Sample Output:

apple, banana, grape, pear, watermelon

1, 3, 5, 10, 12

Notice the use of the keyword **'Array'** before the **.Sort()** method. The 'Array' keyword is a special object allowing us access to the .Sort() method where we can pass in an array to be sorted.

As you can see, the **.Sort()** method is pretty straightforward. Pass in an array as a parameter, and the array gets sorted! The example above demonstrates the sorting of a string array and integer array, but you can also use the **.Sort()** method for **double arrays** as well.

**Note:** Notice how we used an if-statement within our for-loop to determine when to output a comma with a space and when not to (i.e., only the last element in the array should have no comma or space beside it). This is a common technique when outputting in such a manner.

### III. ARRAY MIN, MAX, SUM, AND AVERAGE METHODS:

The **.Min()**, **.Max()**, **.Sum()**, **.Average()** array methods do exactly as indicated. To use these array methods, we call it from our array variable, for example:

```
int[] myInts = {3, 5, 1, 12, 10};  
Console.WriteLine(myInts.Min());  
Console.WriteLine(myInts.Max());  
Console.WriteLine(myInts.Sum());  
Console.WriteLine(myInts.Average());
```

Sample Output:

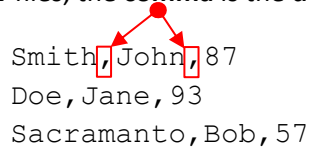
```
1  
12  
31  
6.2
```

Notice that these methods are called directly from our **'myInts'** array above. We are not using the special **'Array'** object that we used with **.Sort()** in the last section.

You can use **all** the above methods for **double arrays** as well. However, only **.Min()** and **.Max()** will work for **string arrays**.

**IV. .SPLIT() METHOD:**

A very useful method is the **.Split()** method. This method allows you to split a string into pieces based on a **delimiter**. A **delimiter** is a character that separates fields in a string. For example, with **.csv** files, the **comma** is the **delimiter**:



Smith, John, 87  
Doe, Jane, 93  
Sacramento, Bob, 57

The **.Split()** method is actually a string method, but we use it with arrays. Here is an example:

```
string mySentence = "hello world how are you";
string[] myWords;
myWords = mySentence.Split();

for(int x = 0; x < myWords.Length; x++)
{
    Console.WriteLine(myWords[x]);
}
```

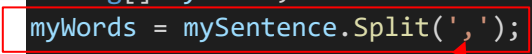
**Sample Output:**

```
hello
world
how
are
you
```

In the above code we have a string named **'mySentence'** containing a sentence where each word is obviously separated by a space. The **'space'** in this case is our **delimiter**. We then create a string array named **'myWords'** which we make equal to **mySentence.Split()**:

```
myWords = mySentence.Split();
```

The **.Split()** method has the **'space'** as a **default delimiter**, so when this statement executes, all the words get stored separately into our array **'myWords'**. We can see from the output that the words did in fact split into the array. If we wish to indicate a different delimiter we can pass it as a parameter to the **.Split()** method, for example:



```
string mySentence = "hello,world,how,are,you";  
string[] myWords;  
myWords = mySentence.Split(',');  
  
for(int x = 0; x < myWords.Length; x++)  
{  
    Console.WriteLine(myWords[x]);  
}
```

Sample Output:

```
hello  
world  
how  
are  
you
```

In the above code our '**mySentence**' has all the words separated by commas. Therefore, we pass a **comma** character as our **delimiter** to the **.Split()** method.

**OTHER USEFUL ARRAY METHODS:**

There are many useful array methods available in C#. It can be helpful to research others on the internet. Here are some other ones to consider:

- `Array.Reverse(Array)`
- `Array.BinarySearch(Array, Object)`
- `Array.Clear(Array)`
- `Array.IndexOf(Array, Object)`

**Note:** With the above methods, the parameter 'Array' is the array you wish to apply the method to. The parameter 'Object' is a particular item in the array.