

Data Structures

Create an Array

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.

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
```

Exercise (try it)

```
using System;

namespace MyApplication
{
    class Program
    {
```

```
static void Main(string[] args)
{
    string[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
    Console.WriteLine(cars[0]);
}
}
```

Note: 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
```

Exercise (try it)

```
using System;

namespace MyApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            string[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
            cars[0] = "Opel";
            Console.WriteLine(cars[0]);
        }
    }
}
```

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
```

Exercise (try it)

```
using System;

namespace MyApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            string[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
            Console.WriteLine(cars.Length);
        }
    }
}
```

Loop Through an Array

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]);
}
```

Exercise (try it)

```
using System;

namespace MyApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            string[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
            for (int i = 0; i < cars.Length; i++)
            {
                Console.WriteLine(cars[i]);
            }
        }
    }
}
```

The 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
}
```

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);
}
```

Exercise (try it)

```
using System;

namespace MyApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            string[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
            foreach (string i in cars)
            {
                Console.WriteLine(i);
            }
        }
    }
}
```

The example above can be read like this: **for each** `string` element (called **i** - as in **index**) in **`cars`**, print out the value of **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.

Sort Arrays

There are many array methods available, for example `Sort()`, which sorts an array alphabetically or in an ascending order:

Example

```
// Sort a string

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

Array.Sort(cars);

foreach (string i in cars)
{
    Console.WriteLine(i);
}
```

```
// Sort an int

int[] myNumbers = {5, 1, 8, 9};

Array.Sort(myNumbers);

foreach (int i in myNumbers)
{
    Console.WriteLine(i);
}
```

Exercise (try it)

```
using System;

namespace MyApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            // Sort a string
            string[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
            Array.Sort(cars);
            foreach (string i in cars)
            {
                Console.WriteLine(i);
            }

            // Sort an int
            int[] myNumbers = {5, 1, 8, 9};
            Array.Sort(myNumbers);
            foreach (int i in myNumbers)
            {
                Console.WriteLine(i);
            }
        }
    }
}
```

System.Linq Namespace

Other useful array methods, such as `Min`, `Max`, and `Sum`, can be found in the `System.Linq` namespace:

Example

```
using System;

using System.Linq;

namespace MyApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            int[] myNumbers = {5, 1, 8, 9};

            Console.WriteLine(myNumbers.Max()); // returns the largest value
            Console.WriteLine(myNumbers.Min()); // returns the smallest
value
            Console.WriteLine(myNumbers.Sum()); // returns the sum of
elements

        }
    }
}
```

Exercise (try it)

```
using System;
using System.Linq;

namespace MyApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            int[] myNumbers = {5, 1, 8, 9};
            Console.WriteLine(myNumbers.Max()); // largest value
            Console.WriteLine(myNumbers.Min()); // smallest value
        }
    }
}
```

```
        Console.WriteLine(myNumbers.Sum()); // sum of myNumbers
    }
}
```

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"};
```

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"};
```

Exercise (try it)

```
using System;

namespace MyApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            // Declare an array
            string[] cars;

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

            // This would cause an error: cars = {"Volvo", "BMW", "Ford"};

            Console.WriteLine(cars[0]);
        }
    }
}
```

C# - Multidimensional Arrays

You learned about the single-dimensional arrays in the previous section. C# also supports multidimensional arrays up to 32 dimensions.

The multidimensional array can be declared by adding commas in the square brackets. For example, [,] declares two-dimensional array, [, ,] declares three-dimensional array, [, , ,] declares four-dimensional array, and so on. So, in a multidimensional array, no of commas = No of Dimensions - 1.

The following declares multidimensional arrays.

Example: Multidimensional Arrays

```
int[,] arr2d; // two-dimensional array
int[, ,] arr3d; // three-dimensional array
int[, , ,] arr4d ; // four-dimensional array
int[, , , ,] arr5d; // five-dimensional array
```

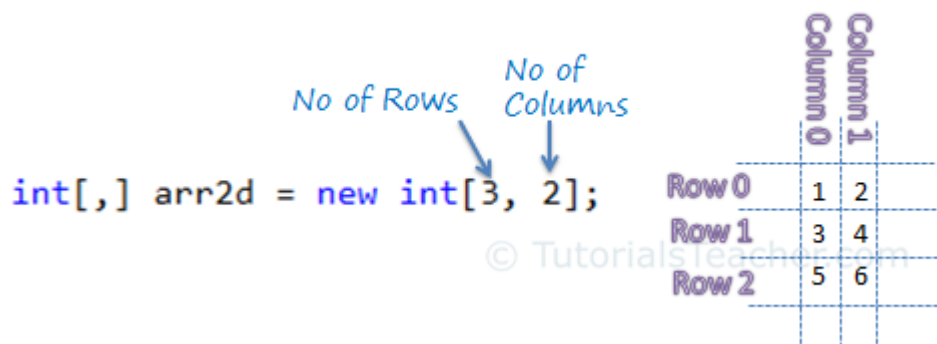
Let's understand the two-dimensional array. The following initializes the two-dimensional array.

Example: two-dimensional Array

```
int[, ] arr2d = new int[3,2]{
    {1, 2},
    {3, 4},
    {5, 6}
};

// or
int[, ] arr2d = {
    {1, 2},
    {3, 4},
    {5, 6}
};
```

In the above example of a two-dimensional array, `[3, 2]` defines the no of rows and columns. The first rank denotes the no of rows, and the second rank defines no of columns. The following figure illustrates the two-dimensional array divided into rows and columns.



Two-dimensional

Array

The following access values of the two-dimensional array.

Example: Access two-dimensional Array

```
int[, ] arr2d = new int[3,2]{
    {1, 2},
    {3, 4},
    {5, 6}
};

arr2d[0, 0]; //returns 1
arr2d[0, 1]; //returns 2
arr2d[1, 0]; //returns 3
arr2d[1, 1]; //returns 4
arr2d[2, 0]; //returns 5
arr2d[2, 1]; //returns 6
```

```
//arr2d[3, 0]; //throws run-time error as there is no 4th row
```

Exercise (try it)

```
using System;

public class Program
{
    public static void Main()
    {
        int[,] arr2d = new int[3,2]{
                                {1, 2},
                                {3, 4},
                                {5, 6}
                                };

        Console.WriteLine(arr2d[0, 0]);
        Console.WriteLine(arr2d[0, 1]);
        Console.WriteLine(arr2d[1, 0]);
        Console.WriteLine(arr2d[1, 1]);
        Console.WriteLine(arr2d[2, 0]);
        Console.WriteLine(arr2d[2, 1]);
    }
}
```

In the above example, the value of a two-dimensional array can be accessed by index no of row and column as [row index, column index]. So, `[0, 0]` returns the value of the first row and first column and `[1, 1]` returns the value from the second row and second column.

Now, let's understand the three-dimensional array. The following declares and initializes three-dimensional arrays.

Example: Three-dimensional Array

```
int[, ,] arr3d1 = new int[1, 2, 2]{
    { { 1, 2}, { 3, 4} }
};

int[, ,] arr3d2 = new int[2, 2, 2]{
    { {1, 2}, {3, 4} },
    { {5, 6}, {7, 8} }
};

int[, ,] arr3d3 = new int[2, 2, 3]{
    { { 1, 2, 3}, {4, 5, 6} },
    { { 7, 8, 9}, {10, 11, 12} }
};

arr3d2[0, 0, 0]; // returns 1
```

```
arr3d2[0, 0, 1]; // returns 2
arr3d2[0, 1, 0]; // returns 3
arr3d2[0, 1, 1]; // returns 4
arr3d2[1, 0, 0]; // returns 5
arr3d2[1, 0, 1]; // returns 6
arr3d2[1, 1, 0]; // returns 7
arr3d2[1, 1, 1]; // returns 8
```

Exercise (try it)

```
using System;

public class Program
{
    public static void Main()
    {
        int[, ] arr3d1 = new int[1, 2, 2]{
            {{1, 2}, {3, 4}} // 1 row of two-dimensional
array
        };

        int[, ] arr3d2 = new int[2, 2, 2]{ // 2 rows of two-
dimensional array
            {{1, 2}, {3, 4}},
            {{5, 6}, {7, 8}}
        };

        int[, ] arr3d3 = new int[2, 2, 3]{ // 2 rows of two-
dimensional array
            {{1, 2, 3}, {4, 5, 6}},
            {{7, 8, 9}, {10, 11, 12}}
        };

        Console.WriteLine("arr3d2 Values");
        Console.WriteLine(arr3d2[0, 0, 0]);
        Console.WriteLine(arr3d2[0, 0, 1]);
        Console.WriteLine(arr3d2[0, 1, 0]);
        Console.WriteLine(arr3d2[0, 1, 1]);
        Console.WriteLine(arr3d2[1, 0, 0]);
        Console.WriteLine(arr3d2[1, 0, 1]);
        Console.WriteLine(arr3d2[1, 1, 0]);
        Console.WriteLine(arr3d2[1, 1, 1]);
    }
}
```

As you can see in the above example, [1, 2, 2] of `arr3d1` specifies that it will contain one row of two-dimensional array [2, 2]. `arr3d2` specifies dimensions [2, 2, 2], which indicates that it includes two rows of two-dimensional array of

[2, 2]. Thus, the first rank indicates the number of rows of inner two-dimensional arrays.

C# - Jagged Arrays

A jagged array is an array of array. Jagged arrays store arrays instead of literal values.

A jagged array is initialized with two square brackets []. The first bracket specifies the size of an array, and the second bracket specifies the dimensions of the array which is going to be stored.

The following example declares jagged arrays.

Example: Jagged Arrays

```
int[][] jArray1 = new int[2][]; // can include two single-dimensional arrays
int[,] jArray2 = new int[3][,]; // can include three two-dimensional arrays
```

In the above example, `jArray1` can store up to two single-dimensional arrays. `jArray2` can store up to three two-dimensional arrays. `[,]` specifies the two-dimensional array.

Example: Jagged Array

```
int[][] jArray = new int[2][];

jArray[0] = new int[3]{1, 2, 3};

jArray[1] = new int[4]{4, 5, 6, 7};
```

You can also initialize a jagged array upon declaration like the below.

Example: Jagged Array

```
int[][] jArray = new int[2][]{
    new int[3]{1, 2, 3},
    new int[4]{4, 5, 6, 7}
};

jArray[0][0]; //returns 1
jArray[0][1]; //returns 2
jArray[0][2]; //returns 3
jArray[1][0]; //returns 4
jArray[1][1]; //returns 5
jArray[1][2]; //returns 6
jArray[1][3]; //returns 7
```

Exercise (try it)

```
using System;

public class Program
{
    public static void Main()
    {
        int[][] jArray = new int[2][]{
            new int[3]{1, 2, 3},

            new int[4]{4, 5, 6, 7}
        };

        Console.WriteLine(jArray[0][0]);
        Console.WriteLine(jArray[0][1]);
        Console.WriteLine(jArray[0][2]);
        Console.WriteLine(jArray[1][0]);
        Console.WriteLine(jArray[1][1]);
        Console.WriteLine(jArray[1][2]);
        Console.WriteLine(jArray[1][3]);
    }
}
```

Reference

www.w3schools.com