Arrays & Strings

Arrays

C# provides numerous built-in classes to store and manipulate data.
One example of such a class is the **Array** class.
An array is a data structure that is used to store a collection of data. You can think of it as a collection of variables of the **same type**.
For example, consider a situation where you need to store 100 numbers. Rather than declare 100 different variables, you can just declare an array that stores 100 **elements**.
To declare an array, specify its element types with square brackets:

```
int[ ] myArray;
```

This statement declares an array of integers.
Since arrays are **objects**, we need to instantiate them with the **new** keyword:

```
int[ ] myArray = new int[5];
```

This instantiates an array named myArray that holds 5 integers.

> Note the **square brackets** used to define the number of elements the array should hold.

Arrays

After creating the array, you can assign values to individual elements by using the **index number**:

```
int[ ] myArray = new int[5];
myArray[0] = 23;
```

This will assign the value 23 to the first element of the array.

> Arrays in C# are zero-indexed meaning the first member has index 0, the second has index 1, and so on.

Arrays

We can provide initial values to the array when it is declared by using curly brackets:

```
string[ ] names = new string[3] {"John", "Mary", "Jessica"};
double[ ] prices = new double[4] {3.6, 9.8, 6.4, 5.9};
```

We can omit the size declaration when the number of elements are provided in the curly braces:

```
string[ ] names = new string[ ] {"John", "Mary", "Jessica"};
double[ ] prices = new double[ ] {3.6, 9.8, 6.4, 5.9};
```

We can even omit the **new** operator. The following statements are identical to the ones above:

```
string[ ] names = {"John", "Mary", "Jessica"};
double[ ] prices = {3.6, 9.8, 6.4, 5.9};
```

Array values should be provided in a comma separated list enclosed in {curly braces}.

Arrays

As mentioned, each element of an array has an index number.
For example, consider the following array:

```
int[ ] b = {11, 45, 62, 70, 88};
```

The elements of **b** have the following indexes:

11	45	62	70	88
[0]	[1]	[2]	[3]	[4]

To access individual array elements, place the element's index number in square brackets following the array name.

```
Console.WriteLine(b[2]);
//Outputs 62

Console.WriteLine(b[3]);
//Outputs 70
```

Try It Yourself

Remember that the first element has index 0.

Arrays & Loops

It's occasionally necessary to iterate through the elements of an array, making element assignments based on certain calculations. This can be easily done using loops.
For example, you can declare an array of 10 integers and assign each element an even value with the following loop:

```
int[ ] a = new int[10];
for (int k = 0; k < 10; k++) {
  a[k] = k*2;
}
```

We can also use a loop to read the values of an array.
For example, we can display the contents of the array we just created:

```
for (int k = 0; k < 10; k++) {
  Console.WriteLine(a[k]);
}
```

Try It Yourself

This will display the values of the elements of the array.

The variable **k** is used to access each array element.
The last index in the array is 9, so the for loop condition is k<10.

The foreach Loop

The **foreach** loop provides a shorter and easier way of accessing array elements.
The previous example of accessing the elements could be written using a **foreach** loop:

```
foreach (int k in a) {
  Console.WriteLine(k);
}
```

Try It Yourself

The **foreach** loop iterates through the array a and assigns the value of the current element to the variable **k** at each iteration of the loop. So, at the first iteration, k=a[0], at the second, k=a[1], etc.

The data type of the variable in the **foreach** loop should match the type of the array elements.
Often the keyword **var** is used as the type of the variable, as in: **foreach (var k in a)**. The compiler determines the appropriate type for **var**.

Arrays

The following code uses a **foreach** loop to calculate the sum of all the elements of an array:

```
int[ ] arr = {11, 35, 62, 555, 989};
int sum = 0;

foreach (int x in arr) {
  sum += x;
}

Console.WriteLine(sum);
//Outputs 1652
```

Try It Yourself

To review, we declared an array and a variable **sum** that will hold the sum of the elements.
Next, we utilized a **foreach** loop to iterate through each element of the array, adding the corresponding element's value to the **sum** variable.

The **Array** class provides some useful methods that will be discussed in the coming lessons.

Multidimensional Arrays

An array can have multiple dimensions. A **multidimensional** array is declared as follows:

```
type[, , ... ,] arrayName = new type[size1, size2, ..., sizeN];
```

For example, let's define a two-dimensional 3x4 integer array:

```
int[ , ] x = new int[3,4];
```

Visualize this array as a table composed of 3 rows and 4 columns:

	Column 1	Column 2	Column 3	Column 4
Row 1	x[0][0]	x[0][1]	x[0][2]	x[0][3]
Row 2	x[1][0]	x[1][1]	x[1][2]	x[1][3]
Row 3	x[2][0]	x[2][1]	x[2][2]	x[2][3]

Array indexing starts from 0.

Multidimensional Arrays

We can initialize multidimensional arrays in the same way as single-dimensional arrays.
For example:

```
int[ , ] someNums = { {2, 3}, {5, 6}, {4, 6} };
```

This will create an array with three rows and two columns. Nested curly brackets are used to define values for each row.
To access an element of the array, provide both indexes. For example **someNums[2, 0]** will return the value **4**, as it accesses the first column of the third row.
Let's create a program that will display the values of the array in the form of a table.

```
for (int k = 0; k < 3; k++) {
  for (int j = 0; j < 2; j++) {
    Console.Write(someNums[k, j]+" ");
  }
  Console.WriteLine();
}
```

Try It Yourself

We have used two nested **for** loops, one to iterate through the rows and one through the columns. The Console.WriteLine(); statement moves the output to a new line after one row is printed.

> Arrays can have any number of dimensions, but keep in mind that arrays with more than three dimensions are harder to manage.

Jagged Arrays

A **jagged** array is an array whose elements are arrays. So it is basically an **array of arrays**. The following is a declaration of a single-dimensional array that has three elements, each of which is a single-dimensional array of integers:

```
int[ ][ ] jaggedArr = new int[3][ ];
```

Each dimension is an array, so you can also initialize the array upon declaration like this:

```
int[ ][ ] jaggedArr = new int[ ][ ]
{
  new int[ ] {1,8,2,7,9},
  new int[ ] {2,4,6},
  new int[ ] {33,42}
};
```

You can access individual array elements as shown in the example below:

```
int x = jaggedArr[2][1]; //42
```

Try It Yourself

This accesses the second element of the third array.

> A **jagged array** is an array-of-arrays, so an int[ ][ ] is an array of int[ ], each of which can be of different lengths and occupy their own block in memory.
> A **multidimensional array** (int[,]) is a single block of memory (essentially a matrix). It always has the same amount of columns for every row.

Arrays Properties

The Array class in C# provides various properties and methods to work with arrays.
For example, the **Length** and **Rank** properties return the number of elements and the number of dimensions of the array, respectively. You can access them using the dot syntax, just like any class members:

```
int[ ] arr = {2, 4, 7};
Console.WriteLine(arr.Length);
//Outputs 3

Console.WriteLine(arr.Rank);
//Outputs 1
```

Try It Yourself

The **Length** property can be useful in **for** loops where you need to specify the number of times the loop should run.
For example:

```
int[ ] arr = {2, 4, 7};
for(int k=0; k<arr.Length; k++) {
  Console.WriteLine(arr[k]);
}
```

Try It Yourself

Tap **Try It Yourself** to play around with the code!

Array Methods

There are a number of methods available for arrays.
Max returns the largest value.
Min returns the smallest value.
Sum returns the sum of all elements.

For example:

```
int[ ] arr = { 2, 4, 7, 1};
Console.WriteLine(arr.Max());
//Outputs 7

Console.WriteLine(arr.Min());
//Outputs 1

Console.WriteLine(arr.Sum());
//Outputs 14
```

Try It Yourself

C# also provides a static Array class with additional methods. You will learn about those in the next module.

Strings

It's common to think of strings as arrays of characters. In reality, strings in C# are objects. When you declare a **string** variable, you basically instantiate an object of type **String**.
String objects support a number of useful properties and methods:
Length returns the length of the string.
IndexOf(value) returns the index of the first occurrence of the value within the string.
Insert(index, value) inserts the value into the string starting from the specified index.
Remove(index) removes all characters in the string after the specified index.
Replace(oldValue, newValue) replaces the specified value in the string.
Substring(index, length) returns a substring of the specified length, starting from the specified index. If length is not specified, the operation continues to the end of the string.
Contains(value) returns true if the string contains the specified value.

The examples below demonstrate each of the String members:

```
string a = "some text";
Console.WriteLine(a.Length);
//Outputs 9

Console.WriteLine(a.IndexOf('t'));
//Outputs 5

 a = a.Insert(0, "This is ");
Console.WriteLine(a);
//Outputs "This is some text"

a = a.Replace("This is", "I am");
Console.WriteLine(a);
//Outputs "I am some text"

if(a.Contains("some"))
  Console.WriteLine("found");
//Outputs "found"

a = a.Remove(4);
Console.WriteLine(a);
//Outputs "I am"

a = a.Substring(2);
Console.WriteLine(a);
//Outputs "am"
```

Try It Yourself

You can also access characters of a string by its index, just like accessing elements of an array:

```
string a = "some text";
Console.WriteLine(a[2]);
//Outputs "m"
```

Try It Yourself

Indexes in strings are similar to arrays, they start from 0.

Working with Strings

Let's create a program that will take a string, replace all occurrences of the word "dog" with "cat" and output the first sentence only.

```
string text = "This is some text about a dog. The word dog appears in this text a number of times. This is the end.";

text = text.Replace("dog", "cat");
text = text.Substring(0, text.IndexOf(".")+1);

Console.WriteLine(text);
//Outputs: "This is some text about a cat."
```

Try It Yourself

The code above replaces all occurrences of "dog" with "cat". After that it takes a substring of the original string starting from the first index until the first occurrence of a period character.
We add one to the index of the period to include the period in the substring.

> C# provides a solid collection of tools and methods to work and manipulate strings. You could, for example, find the number of times a specific word appears in a book with ease, using those methods.

End.