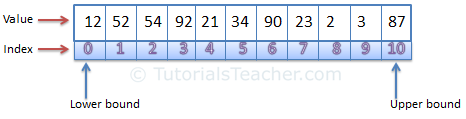
**C# Array:**

You have learned that a variable can hold only one literal value, for example int x = 1;. Only one literal value can be assigned to a variable x. Suppose, you want to store 100 different values then it will be cumbersome to create 100 different variables. To overcome this problem, C# introduced an array.

An array is a special type of data type which can store fixed number of values sequentially using special syntax.

The following image shows how an array stores values sequentially.

[](http://www.tutorialsteacher.com/Content/images/csharp/array.png)

Array Representation

As you can see in the above figure, index is a number starting from 0, which stores the value. You can store a fixed number of values in an array. Array index will be increased by 1 sequentially till the maximum specified array size.

**Array Declaration:**

An array can be declare using a type name followed by square brackets [].

Example: Array declaration in C#

int[] intArray; // can store int values

bool[] boolArray; // can store boolean values

string[] stringArray; // can store string values

double[] doubleArray; // can store double values

byte[] byteArray; // can store byte values

Student[] customClassArray; // can store instances of Student class

**Initialization:**

An array can be declared and initialized at the same time using the new keyword. The following example shows the way of initializing an array.

// defining array with size 5. add values later on

int[] intArray1 = new int[5];

// defining array with size 5 and adding values at the same time

int[] intArray2 = new int[5]{1, 2, 3, 4, 5};

// defining array with 5 elements which indicates the size of an array

int[] intArray3 = {1, 2, 3, 4, 5};

In the above example, the first statement declares & initializes int type array that can store five int values. The size of the array is specified in square brackets. The second statement, does the same thing, but it also assigns values to each indexes in curly brackets { }. The third statement directly initializes an int array with the values without giving any size. Here, size of an array will automatically be number of values.

Initialization without giving size is NOT valid. For example, the following example would give compile time error.

int[] intArray = new int[]; // compiler error: must give size of an array

**Late initialization:**

Arrays can be initialized after declaration. It is not necessary to declare and initialize at the same time using new keyword. Consider the following example.

string[] strArray1, strArray2;

strArray1 = new string[5]{ "1st Element",

"2nd Element",

"3rd Element",

"4th Element",

"5th Element"

};

strArray2 = new string[]{ "1st Element",

"2nd Element",

"3rd Element",

"4th Element",

"5th Element"

};

However, in the case of late initialization, it must be initialized with the new keyword as above. It cannot be initialize by only assigning values to the array.

The following initialization is NOT valid:

string[] strArray;

strArray = {"1st Element","2nd Element","3rd Element","4th Element" };

Accessing Array Elements:

As shown above, values can be assigned to an array at the time of initialization. However, value can also be assigned to individual index randomly as shown below.

Example: Assigning values to array index

int[] intArray = new int[5];

intArray[0] = 10;

intArray[1] = 20;

intArray[2] = 30;

intArray[3] = 40;

intArray[4] = 50;

In the same way, you can retrieve values at a particular index, as below:

Example: Accessing Array elements

intArray[0]; //returns 10

intArray[2]; //returns 30

Use a for loop to access the values from all the indexes of an array by using length property of an array.

Example: Accessing Array elements using for loop

int[] intArray = new int[5]{10, 20, 30, 40, 50 };

for(int i = 0; i < intArray.Length; i++)

Console.WriteLine(intArray[i]);

**Array properties and methods:**

| Method Name | | Description | |
| --- | --- | --- | --- |
| GetLength(int dimension) | | Returns the number of elements in the specified dimension. | |
| GetLowerBound(int dimension) | | Returns the lowest index of the specified dimension. | |
| GetUpperBound(int dimension) | | Returns the highest index of the specified dimension. | |
| GetValue(int index) | | Returns the value at the specified index. | |
| Property | Description | |
| Length | Returns the total number of elements in the array. | |

**Array Helper Class:**

.NET provides an abstract class, Array, as a base class for all arrays. It provides static methods for creating, manipulating, searching, and sorting arrays.

For example, use the Array.Sort() method to sort the values:

int[] intArr = new int[5]{ 2, 4, 1, 3, 5};

Array.Sort(intArr);

Array.Reverse(intArr);

You can create an instance of an Array that starts with index 1 (not default starting index 0) using Array class as shown below:

Array array = Array.CreateInstance(typeof(int),new int[1]{5},new int[1]{1});

array.SetValue(1, 1);

array.SetValue(2, 2);

array.SetValue(3, 3);

array.SetValue(4, 4);

array.SetValue(5, 5);

for (int i = 1; i <= array.Length; i++)

Console.WriteLine("Array value {0} at position {1}", array.GetValue(i), i);

**Multi-dimensional Array:**

You have learned about single dimensional arrays in the previous section. C# also supports multi-dimensional arrays. A multi-dimensional array is a two dimensional series like rows and columns.

int[,] intArray = new int[3,2]{

{1, 2},

{3, 4},

{5, 6}

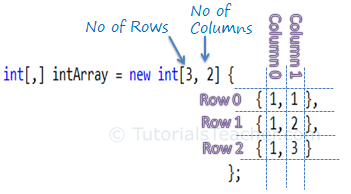
};

// or

int[,] intArray = { {1, 1}, {1, 2}, {1, 3} };

As you can see in the above example, multi-dimensional array is initialized by giving size of rows and columns. [3, 2] specifies that array can include 3 rows and 2 columns.

The following figure shows a multi-dimensional array divided into rows and columns:



Multi-dimensional Array

The values of a multi-dimensional array can be accessed using two indexes. The first index is for the row and the second index is for the column. Both the indexes start from zero.

int[,] intArray = new int[3,2]{

{1, 2},

{3, 4},

{5, 6}

};

intArray[0,0]; //Output: 1

intArray[0,1]; // 2

intArray[1,0]; // 3

intArray[1,1]; // 4

intArray[2,0]; // 5

intArray[2,1]; // 6

In the above example, intArray[2,1] returns 3. Here, 2 means the third row and 1 means the second column (rows and columns starts with zero index).

**Jagged Array:**

A jagged array is an array of an array. Jagged arrays store arrays instead of any other data type value directly. A jagged array is initialized with two square brackets [][]. The first bracket specifies the size of an array and the second bracket specifies the dimension of the array which is going to be stored as values. (Remember, jagged array always store an array.)

The following jagged array stores a two single dimensional array as a value:

int[][] intJaggedArray = new int[2][];

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

intJaggedArray[1] = new int[2]{4,5};

Console.WriteLine(intJaggedArray[0][0]); // 1

Console.WriteLine(intJaggedArray[0][2]); // 3

Console.WriteLine(intJaggedArray[1][1]); // 5

The following jagged array stores a multi-dimensional array as a value. Second bracket [,] indicates multi-dimension.

int[][,] intJaggedArray = new int[3][,];

intJaggedArray[0] = new int[3, 2] { { 1, 2 }, { 3, 4 }, { 5, 6 } };

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

intJaggedArray[2] = new int[2, 2];

Console.WriteLine(intJaggedArray[0][1,1]); // 4

Console.WriteLine(intJaggedArray[1][1,0]); // 5

Console.WriteLine(intJaggedArray[1][1,1]); // 6

Note: Be careful while working with jagged arrays. It will throw an IndexOutOfRange exception if the index does not exist.

**Points to Remember:**

1. An Array stores values in a series starting with a zero-based index.
2. The size of an array must be specified while initialization.
3. An Array values can be accessed using indexer.
4. An Array can be single dimensional, multi-dimensional and jagged array.
5. The Array helper class includes utility methods for arrays.