**Topic 1: Arrays**

Top of Form

Bottom of Form

**Content**

**Reading**

[Reading](https://dmacc.blackboard.com/webapps/blackboard/content/listContent.jsp?course_id=_102593_1&content_id=_7286836_1)

Arrays

* + Microsoft:docs.microsoft.com
    - [Array Overview](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/arrays/)
    - [Single-Dimensional Arrays](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/arrays/single-dimensional-arrays)
  + GeeksForGeeks: geeksforgeeks.com
    - [C# | Arrays](https://www.geeksforgeeks.org/c-sharp-arrays/)
  + TutorialsPoint: tutorialspoint.com
    - [C# - Arrays](https://www.tutorialspoint.com/csharp/csharp_arrays.htm)
  + W3Schools: w3schools.com
    - [C# Arrays](https://www.w3schools.com/cs/cs_arrays.asp)

**Array Basics and some gotchas!**

[Array Basics and some gotchas!](https://dmacc.blackboard.com/webapps/blackboard/content/listContent.jsp?course_id=_102593_1&content_id=_7286836_1)

https://youtu.be/byMVkswS\_4g

using System;

namespace Module6

{

class Program

{

static void Main(string[] args)

{

// Declare and Initialize an array

int[] age = new int[3];

age[0] = 2;

age[1] = 3;

age[2] = 12;

// print in order

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

{

Console.WriteLine("age[" + i +"] = " + age[i]);

}

// print in reverse order

for (int i=age.Length-1; i >= 0; i--)

{

Console.WriteLine("age[" + i + "] = " + age[i]);

}

}

}

}

**Introduction to Arrays**

[Introduction to Arrays](https://dmacc.blackboard.com/webapps/blackboard/content/listContent.jsp?course_id=_102593_1&content_id=_7286836_1)

So far, you have used variables for storing data in memory, but that only allows one value at a time. Rarely will you need to store single values. Arrays allow you to store many data values of the same type and access them via an index.

Arrays are reference type objects. Recall that for using a referenced type object, two steps are required:

* + You **declare** a reference variable
  + You **instantiate** the variable (create the object and associate it with the variables)

Below is an an example ***declaration*** of an array of integer.

int[] numbersArray;

Next, you can ***instantiate*** an array of size 7 of integers (datatype int).

numbersArray = new int[7];

Or, you can ***declare*** and ***instantiate*** an array in one line.

int[] numbersArray = new int[7];

Or, you can ***declare***, ***instantiate***, and ***initialize*** an array in one line.

int[] numbersArray = new int[] { 1, 3, 5, 2, 6, 8, 9 };

**Declare, Instantiate and Initialize**

[Declare, Instantiate and Initialize](https://dmacc.blackboard.com/webapps/blackboard/content/listContent.jsp?course_id=_102593_1&content_id=_7286836_1)

A best practice is not to hard-code the size value, but to use a constant.

Below is an example where the object is **declared** and **instantiated** in one statement. Then the array is **initialized** with data values. Here you can see a loop would be helpful for printing and sometimes initializing an array. Which loop is best?

const int SIZE = 5;

string[] namesArray = new string[SIZE];

numbersArray[0] = "Ayah";

numbersArray[1] = "Morgana";

numbersArray[2] = "Luis";

numbersArray[3] = "Troung";

numbersArray[4] = "Vasavi";

What is the output of the below code? Try it, then keep reading to find out what it does!

const int SIZE = 7;

// declare and instantiate the array

int[] numbersArray = new int[SIZE];

// initialize the array

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

numbersArray[i] = i \* 3;

}

// print the array with for

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

{

Console.Write(numberArray[i]+ "\t");

}

// print the array with foreach

foreach (int number in numbersArray)

{

Console.Write(number + "\t");

}

Console.WriteLine();

**Access Arrays**

[Access Arrays](https://dmacc.blackboard.com/webapps/blackboard/content/listContent.jsp?course_id=_102593_1&content_id=_7286836_1)

A for loop can be useful to access the elements of an array. It is important to remember the subscript (index) starts at zero, not one! Watch for off by one errors and make sure you use less-than < or less-than-or-equal-to < where appropriate. Inappropriate use may lead to an off-by-one error.

Below the Length property is used to print the data values of the array.

for (int index = 0; index < temperatures.Length; index++)

{

Console.WriteLine(temperatures[index].ToString());

}

Note that there are no parentheses () after Length. Length is a property, not a method. There are parentheses () after ToSring(), as it is a method.

Using the Length property is a best practice. Using a Magic Number (hard-coding a number), may lead to an Array IndexOutOfRange Exception if you change the array size changes but you do not update the all code accessing the loop. With the Length property, you need not worry about updated array references when you change its size.

Make sure to use < and <= appropriately, as this may also lead to an Array IndexOutOfRangeException.

The code below has an off-by-one error and will produce IndexOutOfRangeException.

for (int index = 0; index <= temperatures.Length; index++) // ERROR

{

Console.WriteLine(temperatures[index].ToString());

}

**foreach**

[foreach](https://dmacc.blackboard.com/webapps/blackboard/content/listContent.jsp?course_id=_102593_1&content_id=_7286836_1)

Some programming languages offer a foreach loop or a loop that behaves like a foreach loop (an enhanced for loop). C# does has a foreach. Here is an example:

string[] names = { "Morgan", "Wyatt", "Natalie", "Nilah" };

foreach (string name in names) {

Console.WriteLine(name);

}

**Arrays in methods**

[Arrays in methods](https://dmacc.blackboard.com/webapps/blackboard/content/listContent.jsp?course_id=_102593_1&content_id=_7286836_1)

Now you can even pass the arrays to method or have a return value of an array for a method:

Read the page on [Passing an array](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/arrays/passing-arrays-as-arguments)

Example:

You can return an array

Example:

public static string[] GetNames()

{

// get user input

return new[] { "Ayah", "Wyatt", "Natalie", "Nilah" };

}

You can also have local arrays in methods that are not accessible once you leave the method:

Example:

public static double GetAverage()

{

const int SIZE = 5;

double average;

int[ ] scores = new int[SIZE];

// get user scores with for loop--left for you to try

// compute average with foreach--left for you to try

return average;

}

You are probably wondering if you can return an array like an data type in a method. You can!

Example:

private static int[] DoubleArray(int[] arrayToDouble)

{

int[] returnArray = new int[arrayToDouble.Length]; // Notice use of Length property

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

{

returnArray[i] = arrayToDouble[i] \* 2;

}

return returnArray;

}

static void Main(string[] args)

{

// Declare and Initialize an array

int[] size = new int[5] { 2, 3, 1, 16, 25 };

// using foreach to print array

Console.WriteLine("\nPrinting the array:");

foreach (int item in size)

{

Console.Write(item + "\t");

}

Console.WriteLine("\nAfter doubling:");

int[] newSize = DoubleArray(size);

foreach (int item in newSize)

{

Console.Write(item + "\t");

}

Console.WriteLine();

}

Notice the above examples were pass by value, you passed a copy of the array. You are probably wondering if you can pass by ref to change the array. You can!

Example:

private static void DoubleArray(ref int[] arrayToChange)

{

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

{

arrayToChange[i] = arrayToChange[i] \* 2;

}

}

static void Main(string[] args)

{

// Declare and Initialize an array

int[] size = new int[5] { 2, 3, 1, 16, 25 };

// using foreach to print array

Console.WriteLine("\nPrinting the array:");

foreach (int item in size)

{

Console.Write(item + "\t");

}

Console.WriteLine("\nAfter doubling:");

DoubleArray(ref size);

foreach (int item in size)

{

Console.Write(item + "\t");

}

Console.WriteLine();

}