200 XP

# Exercise - Get started with array basics

19 minutes

Arrays can be used to store multiple values of the same type in a single variable. The values stored in an array are generally related. For example, a list of student names could be stored in a string array named `students`.

Your work in the security department is focused on finding a pattern for fraudulent orders. You want your code to review past customer orders and identify markers associated with fraudulent orders. Your company hopes the markers can be used to identify potential fraudulent purchase orders in the future before they're processed. Since you don't always know in advance how many orders you need to review, you can't create individual variables to hold each Order ID. How can you create a data structure to hold multiple related values?

In this exercise, you use arrays to store and analyze a sequence of Order IDs.

## What is an array?

An array is a sequence of individual data elements accessible through a single variable name. You use a zero-based numeric index to access each element of an array. As you can see, arrays allow you to collect together similar data that shares a common purpose or characteristics in a single data structure for easier processing.

## Declaring arrays and accessing array elements

An array is a special type of variable that can hold multiple values of the same data type. The declaration syntax is slightly different because you have to specify both the data type and the size of the array.

### Prepare your coding environment

This module includes activities that guide you through the process of building and running sample code. You're encouraged to complete these activities using Visual Studio Code as your development environment. Using Visual Studio Code for these activities helps you to become

more comfortable writing and running code in a developer environment that's used by professionals worldwide.

1. Open Visual Studio Code.

   You can use the Windows Start menu (or equivalent resource for another OS) to open Visual Studio Code.

2. On the Visual Studio Code **File** menu, select **Open Folder**.

3. In the **Open Folder** dialog, navigate to the Windows Desktop folder.

   If you have a different folder location where you keep code projects, you can use that folder location instead. For this training, the important thing is to have a location that's easy to locate and remember.

4. In the **Open Folder** dialog, select **Select Folder**.

   If you see a security dialog asking if you trust the authors, select **Yes**.

5. On the Visual Studio Code **Terminal** menu, select **New Terminal**.

   Notice that a command prompt in the Terminal panel displays the folder path for the current folder. For example:

   ```dos
   C:\Users\someuser\Desktop>
   ```

   > **Note**
   >
   > If you are working on your own PC rather than in a sandbox or hosted environment and you have completed other Microsoft Learn modules in this C# series, you may have already created a project folder for code samples. If that's the case, you can skip over the next step, which is used to create a console app in the TestProject folder.

6. At the Terminal command prompt, to create a new console application in a specified folder, type `dotnet new console -o ./CsharpProjects/TestProject` and then press Enter.

   This .NET CLI command uses a .NET program template to create a new C# console application project in the specified folder location. The command creates the **CsharpProjects**

and **TestProject** folders for you, and uses *TestProject* as the name of your `.csproj` file.

7. In the EXPLORER panel, expand the **CsharpProjects** folder.

   You should see the **TestProject** folder and two files, a C# program file named **Program.cs** and a C# project file named **TestProject.csproj**.

8. In the EXPLORER panel, to view your code file in the Editor panel, select **Program.cs**.

9. Delete the existing code lines.

   You can use this C# console project to create, build, and run code samples during this module.

10. Close the Terminal panel.

## Declare a new array

1. To declare a new array of strings that can hold three elements, enter the following code:

   ```
   c#
   ```

   ```
   string[] fraudulentOrderIDs = new string[3];
   ```

2. Take a minute to examine your code.

   The `new` operator creates a new instance of an array in the computer's memory that can hold three string values. For more information about the `new` keyword, see the module "Call methods from the .NET Class Library using C#".

   Notice that the first set of square brackets `[]` merely tells the compiler that the variable named `fraudulentOrderIDs` is an array, but the second set of square brackets `[3]` indicates the number of elements that the array can hold.

   > **Note**
   >
   > This example demonstrates how to declare an array of strings, however, you can create an array of every data type including primitives like `int` and `bool` as well as more complex data types like classes. This example uses the simplicity of strings to minimize the number of new ideas you need to grasp as you're getting started.

## Assign values to elements of an array

At this point, you've declared an array of strings, but each element of the array is empty. To access an element of an array, you use a numeric zero-based index inside of square brackets. You can assign a value to an array element using the `=` as if it were a regular variable.

1. To assign Order ID values to your `fraudulentOrderIDs` array, update your code as follows:

```
c#

string[] fraudulentOrderIDs = new string[3];

fraudulentOrderIDs[0] = "A123";
fraudulentOrderIDs[1] = "B456";
fraudulentOrderIDs[2] = "C789";
```

2. Take a minute to examine your code.

   Notice that you're using the name of the array to access array elements. Each element is accessed individually by specifying zero-based index number inside the square brackets.

   Since your array is declared as a string, the values that you assign must also be strings. In this scenario, you're assigning Order IDs to the elements of the array.

## Attempt to use an index that is out of bounds of the array

It might not seem intuitive at first, but it's important to remember that you're declaring the count of elements in the array. However, you access each element of the array starting with zero. So, to access the second item in the array, you use index `1`.

It's common for beginners to forget that arrays are zero-based and attempt to access an element of the array that doesn't exist. If you make this mistake, a runtime exception occurs informing you that you attempted to access an element that is outside the boundary of the array.

To intentionally "break" your application, attempt to access a fourth element of your array using index value of `3`.

1. At the bottom of your code file, enter the following code line:

```
c#
```

```
fraudulentOrderIDs[3] = "D000";
```

2. Ensure that your code matches this example:

```c#
string[] fraudulentOrderIDs = new string[3];

fraudulentOrderIDs[0] = "A123";
fraudulentOrderIDs[1] = "B456";
fraudulentOrderIDs[2] = "C789";
fraudulentOrderIDs[3] = "D000";
```

3. On the Visual Studio Code **File** menu, select **Save**.

4. In the EXPLORER panel, to open a Terminal at your TestProject folder location, right-click **TestProject**, and then select **Open in Integrated Terminal**.

   A Terminal panel should open, and should include a command prompt showing that the Terminal is open to your **TestProject** folder location.

5. At the Terminal command prompt, to compile your code, type `dotnet build` and then press Enter.

   You should see the following message:

```Output
Build succeeded.
    0 Warning(s)
    0 Error(s)
```

6. At the Terminal command prompt, to run your code, type `dotnet run` and then press Enter.

   When you run the app, you get the following runtime error message:

```Output
Unhandled exception. System.IndexOutOfRangeException: Index was outside the
bounds of the array.
   at Program.<Main>$(String[] args) in
C:\Users\someuser\Desktop\CsharpProjects\TestProject\Program.cs:line 6
```

Notice the following parts of the error:

- Error message: `System.IndexOutOfRangeException: Index was outside the bounds of the array.`
- Error location: `Program.cs:line 6`

7. Comment out the line that generated the runtime error.

```
c#

// fraudulentOrderIDs[3] = "D000";
```

You've seen how to assign a value to an array element. Now look at how to access a value that's being stored in an array element.

## Retrieve values from elements of an array

Accessing the value of an array element works the same way as assigning a value to an array element. You just specify the index of the element whose value you want to retrieve.

1. To write the value of each fraudulent Order ID, update your code as follows:

```
c#

string[] fraudulentOrderIDs = new string[3];

fraudulentOrderIDs[0] = "A123";
fraudulentOrderIDs[1] = "B456";
fraudulentOrderIDs[2] = "C789";
// fraudulentOrderIDs[3] = "D000";

Console.WriteLine($"First: {fraudulentOrderIDs[0]}");
Console.WriteLine($"Second: {fraudulentOrderIDs[1]}");
Console.WriteLine($"Third: {fraudulentOrderIDs[2]}");
```

2. On the Visual Studio Code **File** menu, select **Save**.

3. In the EXPLORER panel, to open a Terminal at your TestProject folder location, right-click **TestProject**, and then select **Open in Integrated Terminal**.

4. At the Terminal command prompt, type `dotnet run` and then press Enter.

   You should see the following message:

Output

```
First: A123
Second: B456
Third: C789
```

## Reassign the value of an array

The elements of an array are just like any other variable value. You can assign, retrieve, and reassign a value to each element of the array.

1. At the end of your code file, to reassign and then print the value of the first array element, enter the following code:

```c#
fraudulentOrderIDs[0] = "F000";

Console.WriteLine($"Reassign First: {fraudulentOrderIDs[0]}");
```

2. Ensure that your code matches the following example:

```c#
string[] fraudulentOrderIDs = new string[3];

fraudulentOrderIDs[0] = "A123";
fraudulentOrderIDs[1] = "B456";
fraudulentOrderIDs[2] = "C789";
// fraudulentOrderIDs[3] = "D000";

Console.WriteLine($"First: {fraudulentOrderIDs[0]}");
Console.WriteLine($"Second: {fraudulentOrderIDs[1]}");
Console.WriteLine($"Third: {fraudulentOrderIDs[2]}");

fraudulentOrderIDs[0] = "F000";

Console.WriteLine($"Reassign First: {fraudulentOrderIDs[0]}");
```

3. On the Visual Studio Code **File** menu, select **Save**.

4. In the EXPLORER panel, to open a Terminal at your TestProject folder location, right-click **TestProject**, and then select **Open in Integrated Terminal**.

5. At the Terminal command prompt, type `dotnet run` and then press Enter.

   You should see the following message:

```
Output

First: A123
Second: B456
Third: C789
Reassign First: F000
```

## Initialize an array

You can initialize an array during declaration just like you would a regular variable. However, to initialize the elements of the array, you use a special syntax featuring curly braces.

1. Comment out the lines where you declare the `fraudulentOrderIDs` variable.

   You can use a multi-line comment (`/* ... */`) to comment out the declaration of `fraudulentOrderIDs` and the lines used to assign values to the array elements.

2. To declare the array initialize values in a single statement, enter the following code:

```
c#

string[] fraudulentOrderIDs = { "A123", "B456", "C789" };
```

3. Ensure that your code matches the following example:

```
c#

/*
string[] fraudulentOrderIDs = new string[3];

fraudulentOrderIDs[0] = "A123";
fraudulentOrderIDs[1] = "B456";
fraudulentOrderIDs[2] = "C789";
// fraudulentOrderIDs[3] = "D000";
*/

string[] fraudulentOrderIDs = { "A123", "B456", "C789" };

Console.WriteLine($"First: {fraudulentOrderIDs[0]}");
Console.WriteLine($"Second: {fraudulentOrderIDs[1]}");
```

```
Console.WriteLine($"Third: {fraudulentOrderIDs[2]}");

fraudulentOrderIDs[0] = "F000";

Console.WriteLine($"Reassign First: {fraudulentOrderIDs[0]}");
```

4. Take a minute to examine the declaration statement.

   Notice that this syntax is both compact and easy to read. When you run the application, there should be no change to the output.

5. On the Visual Studio Code **File** menu, select **Save**.

6. In the EXPLORER panel, to open a Terminal at your TestProject folder location, right-click **TestProject**, and then select **Open in Integrated Terminal**.

7. At the Terminal command prompt, type `dotnet run` and then press Enter.

   You should see the same message as before:

   Output

   ```
   First: A123
   Second: B456
   Third: C789
   Reassign First: F000
   ```

## Use the Length property of an array

Depending on how the array is created, you may not know in advance how many elements an array contains. To determine the size of an array, you can use the `Length` property.

> **Note**
>
> The `Length` property of an array is not zero-based.

1. At the end of your code file, to report the number of fraudulent orders, enter the following code:

   ```
   c#
   ```

```
Console.WriteLine($"There are {fraudulentOrderIDs.Length} fraudulent orders to
process.");
```

This code uses the array's `Length` property, an integer, to return the number of elements in your `fraudulentOrderIDs` array.

2. Ensure that your code matches this example:

c#

```
/*
string[] fraudulentOrderIDs = new string[3];

fraudulentOrderIDs[0] = "A123";
fraudulentOrderIDs[1] = "B456";
fraudulentOrderIDs[2] = "C789";
// fraudulentOrderIDs[3] = "D000";
*/

string[] fraudulentOrderIDs = { "A123", "B456", "C789" };

Console.WriteLine($"First: {fraudulentOrderIDs[0]}");
Console.WriteLine($"Second: {fraudulentOrderIDs[1]}");
Console.WriteLine($"Third: {fraudulentOrderIDs[2]}");

fraudulentOrderIDs[0] = "F000";

Console.WriteLine($"Reassign First: {fraudulentOrderIDs[0]}");

Console.WriteLine($"There are {fraudulentOrderIDs.Length} fraudulent orders to
process.");
```

3. Save the changes to your **Program.cs** file, and then run the application.

You should see the following output:

Output

```
First: A123
Second: B456
Third: C789
Reassign First: F000
There are 3 fraudulent orders to process.
```

# Recap

Here's the most important things to remember when working with arrays:

- An array is a special variable that holds a sequence of related data elements.
- You should memorize the basic format of an array variable declaration.
- Access each element of an array to set or get its values using a zero-based index inside of square brackets.
- If you attempt to access an index outside of the boundary of the array, you get a run time exception.
- The `Length` property gives you a programmatic way to determine the number of elements in an array.

# Check your knowledge

**1. What is an array? ***

- ◯ A string variable.
- ◯ A sequence of individual data elements accessible through a single variable name.
- ◯ A .NET Class Library.

**2. Which of these is a correct example of creating an array and initializing it? ***

- ◯ `string[] myarray = new string[3]; myarray = "test1"; myarray = "test2"; myarray = "test3";`.
- ◯ `string[] myarray = string[3]; myarray[0] = test1; myarray[1] = test2; myarray[2] = test3;`.
- ◯ `int[] myarray = new int[3]; myarray[0] = 1; myarray[1] = 2; myarray[2] = 3;`.

Check your answers