Built-In Methods

In C#, there are several built-in methods we can use with arrays. The full list can be found in the Microsoft documentation of the Array class, under methods.

SORT

The built-in method `Array.Sort()` (documentation), as its name suggests, sorts an array. This method is a quick way to further organize array data into a logical sequence:

```
int[] plantHeights = { 3, 6, 4, 1 };

// plantHeights will be { 1, 3, 4, 6 }
Array.Sort(plantHeights);
```

`Sort()` takes an array as a parameter and edits the array so its values are sorted. If it is an array of integer values, it will sort them into ascending values (lowest to highest). If it's an array of string values, they would be sorted alphabetically.

INDEX OF

The `Array` method `Array.IndexOf()` (documentation) takes a value and returns its index. `IndexOf()` works best when you have a specific value and need to know where it's located in the array (or if it even exists!).

```
int[] plantHeights = { 3, 6, 4, 1, 6, 8 };

 // returns 1
Array.IndexOf(plantHeights, 6);
```

`IndexOf()` typically takes two parameters: the first is the array and the second is the value whose index we're locating. `IndexOf()` also has several overloads that allow you to search for a specific range of the array. If the value appears more than once in an array, it returns only the first occurrence within the specified range. If it cannot find

the value, it returns the lower bound of the array, minus 1 (since most arrays start at 0, it's usually -1).

FIND

The `Array` method `Array.Find()` (documentation) searches a one-dimensional array for a specific value or set of values that match a certain condition and returns the first occurrence in the array.

```
int[] plantHeights = { 3, 6, 4, 1, 6, 8 };

// Find the first occurence of a plant height that is greater than 5 inches
int firstHeight = Array.Find(plantHeights, height => height > 5);
```

`Find()` takes two parameters: the first is the array and the second is a *predicate* that defines what we're looking for. A predicate is a method that takes one input and outputs a boolean. Unlike `IndexOf()`, `Find()` returns the actual values that match the condition, instead of their index.

It's customary to use a lambda function for the predicate to determine if the value meets the necessary criteria. If you need a refresher on lambda functions, check out our C# methods lesson.

For more information on using built-in methods, check out our lessons on C#: Methods.

☑Instructions

1.

Using an Array method, find the position for the first 3-star rated song and save it to a variable. Print a message to the console, like "Song number X is rated three stars".

Hint

Use `IndexOf()` to find the first occurrence of a value in an array:

```
string[] players = { "Emily", "Kyle", "Todd", "Rachel", "Grayson" };

// This will return 2
int toddPosition = Array.IndexOf(players, "Todd");
```

Since arrays are zero-indexed, you can add 1 to the result to simulate one-indexed ordering:

```
int toddPosition = Array.IndexOf(players, "Todd");

// This will print "Todd is 3 on the list"
Console.WriteLine($"Todd is {toddPosition + 1} on the list");
```

2.
Find the first song that has more than 10 characters in its title. Save it to a variable and print a message to the console, such as "The first song that has more than 10 characters in the title is X."

Hint

Use `Find()` to locate certain values in an array, based off a specified condition:

```
string[] players = { "Emily", "Kyle", "Todd", "Rachel", "Grayson" };

// This will return "Rachel"
string longName = Array.Find(players, name => name.Length > 5);
```

3.

Organize the playlist alphabetically. To check that it worked, print the first and last song titles to the console.

Hint

Use `Sort()` to organize an array. For strings, it organizes it alphabetically:

```
string[] players = { "Emily", "Kyle", "Todd", "Rachel", "Grayson" };

// This will return { "Emily", "Grayson", "Kyle", "Rachel", "Todd" };
Array.Sort(players);
```