Learn to manage data collections using the generic list type

Create lists

- 28 minutes remaining

Run the following code in the interactive window. Select the **Enter focus mode** button. Then, type the following code block in the interactive window (replace `<name>` with your name) and select **Run**:

C#Copy

```
var names = new List<string> { "<name>", "Ana", "Felipe" };
foreach (var name in names)
{
  Console.WriteLine($"Hello {name.ToUpper()}!");
}
```

If you are running this on your environment, you should follow the instructions for the local version instead.

You've created a list of strings, added three names to that list, and printed out the names in all CAPS. You're using concepts that you've learned in earlier tutorials to loop through the list.

The code to display names makes use of the string interpolation feature. When you precede a `string` with the `$` character, you can embed C# code in the string declaration. The actual string replaces that C# code with the value it generates. In this example, it replaces the `{name.ToUpper()}` with each name, converted to capital letters, because you called the String.ToUpper method.

Let's keep exploring.

var names = new List<string> { "<name>", "Ana", "Felipe" };

foreach (var name in names)

{

Console.WriteLine($"Hello {name.ToUpper()}!");

}

```
Hello <NAME>!
Hello ANA!
Hello FELIPE!
```

Modify list contents

- 26 minutes remaining

The collection you created uses the List<T> type. This type stores sequences of elements. You specify the type of the elements between the angle brackets.

One important aspect of this List<T> type is that it can grow or shrink, enabling you to add or remove elements. You can see the results by modifying the contents after you've displayed its contents. Add the following code below the code you've already written (below the loop that prints the contents):

C#Copy

```
Console.WriteLine();
names.Add("Maria");
names.Add("Bill");
names.Remove("Ana");
foreach (var name in names)
{
  Console.WriteLine($"Hello {name.ToUpper()}!");
}
```

You've added two more names to the end of the list. You've also removed one as well. The output from this block of code shows the initial contents, then prints a blank line and the new contents.

The List<T> enables you to reference individual items by **index** as well. You access items using the `[` and `]` tokens. Add the following code below what you've already written and try it:

C#Copy

```
Console.WriteLine($"My name is {names[0]}.");
Console.WriteLine($"I've added {names[2]} and {names[3]} to the list.");
```

You're not allowed to access past the end of the list. You can check how long the list is using the Count property. Add the following code to try it:

C#Copy

```
Console.WriteLine($"The list has {names.Count} people in it");
```

Select **Run** again to see the results. In C#, indices start at 0, so the largest valid index is one less than the number of items in the list.

```
var names = new List<string> { "<name>", "Ana", "Felipe" };
foreach (var name in names)
{
  Console.WriteLine($"Hello {name.ToUpper()}!");
}
Console.WriteLine();
names.Add("Maria");
names.Add("Bill");
names.Remove("Ana");
foreach (var name in names)
{
  Console.WriteLine($"Hello {name.ToUpper()}!");
}
Console.WriteLine($"My name is {names[0]}.");
Console.WriteLine($"I've added {names[2]} and {names[3]} to the list.");
Console.WriteLine($"The list has {names.Count} people in it");
```

```
Hello <NAME>!
Hello ANA!
Hello FELIPE!

Hello <NAME>!
Hello FELIPE!
Hello MARIA!
Hello BILL!
My name is <name>.
I've added Maria and Bill to the list.
The list has 4 people in it
```

Search and sort lists

- 23 minutes remaining

Our samples use relatively small lists, but your applications may often create lists with many more elements, sometimes numbering in the thousands. To find elements

in these larger collections, you need to search the list for different items.
The IndexOf method searches for an item and returns the index of the item. If the item isn't in the list, `IndexOf` returns `-1`. Try it to see how it works. Add the following code below what you've written so far:

C#Copy
```
var index = names.IndexOf("Felipe");
if (index != -1)
  Console.WriteLine($"The name {names[index]} is at index {index}");

var notFound = names.IndexOf("Not Found");
  Console.WriteLine($"When an item is not found, IndexOf returns {notFound}");
```

You may not know if an item is in the list, so you should always check the index returned by IndexOf. If it is `-1`, the item was not found.

The items in your list can be sorted as well. The Sort method sorts all the items in the list in their normal order (alphabetically for strings). Add this code and run again:

C#Copy
```
names.Sort();
foreach (var name in names)
{
  Console.WriteLine($"Hello {name.ToUpper()}!");
}
```

```
var names = new List<string> { "<name>", "Ana", "Felipe" };
foreach (var name in names)
{
  Console.WriteLine($"Hello {name.ToUpper()}!");
}
Console.WriteLine();
names.Add("Maria");
names.Add("Bill");
names.Remove("Ana");
foreach (var name in names)
{
  Console.WriteLine($"Hello {name.ToUpper()}!");
}
Console.WriteLine($"My name is {names[0]}.");
Console.WriteLine($"I've added {names[2]} and {names[3]} to the list.");
Console.WriteLine($"The list has {names.Count} people in it");
var index = names.IndexOf("Felipe");
if (index != -1)
  Console.WriteLine($"The name {names[index]} is at index {index}");
```

```
var notFound = names.IndexOf("Not Found");
  Console.WriteLine($"When an item is not found, IndexOf returns {notFound}");
  names.Sort();
foreach (var name in names)
{
  Console.WriteLine($"Hello {name.ToUpper()}!");
}
```

```
Hello <NAME>!
Hello ANA!
Hello FELIPE!

Hello <NAME>!
Hello FELIPE!
Hello MARIA!
Hello BILL!
My name is <name>.
I've added Maria and Bill to the list.
The list has 4 people in it
The name Felipe is at index 1
When an item is not found, IndexOf returns -1
Hello <NAME>!
Hello BILL!
Hello FELIPE!
Hello MARIA!
```

Lists of other types

- 18 minutes remaining

You've been using the `string` type in lists so far. Let's make a List<T> using a different type. Let's build a set of numbers. Delete the code you wrote so far, and replace it with the following code:

C#Copy

```
var fibonacciNumbers = new List<int> {1, 1};
```

That creates a list of integers, and sets the first two integers to the value 1.
The *Fibonacci Sequence*, a sequence of numbers, starts with two 1s. Each next Fibonacci number is found by taking the sum of the previous two numbers. Add this code:

C#Copy

```
var previous = fibonacciNumbers[fibonacciNumbers.Count - 1];
```

```
var previous2 = fibonacciNumbers[fibonacciNumbers.Count - 2];

fibonacciNumbers.Add(previous + previous2);

foreach(var item in fibonacciNumbers)
    Console.WriteLine(item);
```

```
var names = new List<string> { "<name>", "Ana", "Felipe" };
foreach (var name in names)
{
  Console.WriteLine($"Hello {name.ToUpper()}!");
}
Console.WriteLine();
names.Add("Maria");
names.Add("Bill");
names.Remove("Ana");
foreach (var name in names)
{
  Console.WriteLine($"Hello {name.ToUpper()}!");
}
Console.WriteLine($"My name is {names[0]}.");
Console.WriteLine($"I've added {names[2]} and {names[3]} to the list.");
Console.WriteLine($"The list has {names.Count} people in it");
var index = names.IndexOf("Felipe");
if (index != -1)
  Console.WriteLine($"The name {names[index]} is at index {index}");

var notFound = names.IndexOf("Not Found");
  Console.WriteLine($"When an item is not found, IndexOf returns {notFound}");
  names.Sort();
foreach (var name in names)
{
  Console.WriteLine($"Hello {name.ToUpper()}!");
}
var fibonacciNumbers = new List<int> {1, 1};
var previous = fibonacciNumbers[fibonacciNumbers.Count - 1];
var previous2 = fibonacciNumbers[fibonacciNumbers.Count - 2];

fibonacciNumbers.Add(previous + previous2);
```

```
foreach(var item in fibonacciNumbers)
    Console.WriteLine(item);
```

```
Hello <NAME>!
Hello ANA!
Hello FELIPE!

Hello <NAME>!
Hello FELIPE!
Hello MARIA!
Hello BILL!
My name is <name>.
I've added Maria and Bill to the list.
The list has 4 people in it
The name Felipe is at index 1
When an item is not found, IndexOf returns -1
Hello <NAME>!
Hello BILL!
Hello FELIPE!
Hello MARIA!
1
1
2
```

Complete challenge

- 3 minutes remaining

Did you come up with something like this?

C#Copy

```
var fibonacciNumbers = new List<int> {1, 1};

while (fibonacciNumbers.Count < 20)
{
    var previous = fibonacciNumbers[fibonacciNumbers.Count - 1];
    var previous2 = fibonacciNumbers[fibonacciNumbers.Count - 2];

    fibonacciNumbers.Add(previous + previous2);
}
foreach(var item in fibonacciNumbers)
    Console.WriteLine(item);
```

With each iteration of the loop, you're taking the last two integers in the list, summing them, and adding that value to the list. The loop repeats until you've added 20 items to the list.

Previous

```
var fibonacciNumbers = new List<int> {1, 1};

while (fibonacciNumbers.Count < 20)
{
    var previous = fibonacciNumbers[fibonacciNumbers.Count - 1];
    var previous2 = fibonacciNumbers[fibonacciNumbers.Count - 2];

    fibonacciNumbers.Add(previous + previous2);
}
foreach(var item in fibonacciNumbers)
    Console.WriteLine(item);
```

```
1
1
2
3
5
8
13
21
34
55
89
144
233
377
610
987
1597
2584
4181
6765
```