10 Arrays, Lists, and Collections

May 12, 2022

1 Arrays, Lists, and Collections

Arrays, lists, and collections can be pretty useful. Try looking at a list: > Run the following code. Does it print out what you expected?

[1]:
```
using System;
using System.Collections.Generic;

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

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

1.1 So what does that code mean?

- **System.collections.Generic**: This is a namespace that has lists in it. If you don't tell the code that you're using it, you have to write "Systems.Collections.Generic.List" every time you want to use a list. This saves some typing!
- **var**: It's what you put when you have a variable, but don't know/care what the variable type is.
- **List<string>**: This means that you're making a list of strings. In place of `string`, you can put in `int`, `double`, or any other variable.
- **foreach**: This is another for loop! It goes through every item in a list.
- **name in names**: This is a style that a lot of people prefer. `names` is the whole list that contains plural names. `name` is an individual item in `names`.

1.2 Alternative method

The previous code is a bit more clear to read for human than the code below, but the code below has some more recognizable code, based off of what we've learned. These are really just two different styles of writing the same code. Feel free to use whatever makes the most sense to you! > Run the following code. > > Can you identify similar parts of code between the two different methods? > > Which method do you prefer?

[2]:
```
using System;
using System.Collections.Generic;

var names = new List<string> { "<name>", "Ana", "Felipe" };
for (int i = 0;i < names.Count; i++)
{
    Console.WriteLine($"Hello {names[i].ToUpper()}");
}
```

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

1.3 Add

You can add names to lists pretty easily. Lists have the method `Add()`, which tacks on a new item to the end of the list. > Run the code. > > Then try adding your own name instead.

[3]:
```
var names = new List<string> { "<name>", "Ana", "Felipe" };

names.Add("Sophia");

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

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

1.4 Remove

You can also remove names. Try that out:

> Run the code.
>
> Then choose which name you want removed.
>
> What happens when you try to remove something that isn't there?

[4]:
```
var names = new List<string> { "<name>", "Ana", "Felipe" };

names.Remove("<name>");

for (int i = 0;i < names.Count; i++)
{
    Console.WriteLine($"Hello {names[i].ToUpper()}");
}
```

```
Hello ANA
Hello FELIPE
```

1.5 Printing a specific item

What if you don't want to print out all of your friends? What if you just want to print out one friend? That's where brackets come in. > Run the code. > > Try printing a different spot in the list. > > Do you need a 0 or 1 to print the first item in a list?

[5]:
```
var names = new List<string> { "<name>", "Sophia", "Felipe" };
Console.WriteLine(names[1]);
```

```
Sophia
```

Don't forget that lists are "0" based. The first spot is the "0th" spot.

[ ]:
```
```