Var

Every LINQ query returns either a single value or an object of type `IEnumerable<T>`. For now, all you need to know about that second type is that:

- It works with `foreach` loops, just like arrays and lists
- You can check its length with `Count()`

Since the single value type and/or the parameter type `T` is not always known, it's common to store a query's returned value in a variable of type `var`.

`var` is just an implicitly typed variable — we let the C# compiler determine the actual type for us. Here's one example:

```
string[] names = { "Tiana", "Dwayne", "Helena" };
var shortNames = names.Where(n => n.Length < 4);
```

In this case `shortNames` is actually of type `IEnumerable<string>`, but we don't need to worry ourselves about that as long as we have `var`!

☑Instructions

1.

Let's practice using `var` with LINQ.

Create a variable of type `var` named `shortHeroes` and set it equal to this LINQ query:

```
from h in heroes
where h.Length < 8
select h;
```

Hint

Here's an example that sets the variable `eBirds` to a LINQ query:

```
var eBirds = from bird in birds
  where bird.StartsWith("e")
  select bird;
```

2.
Use a `foreach` loop to print out each element in `shortHeroes`.
3.
Create another variable of type `var` named `longHeroes` and set it equal to this LINQ query:

```
heroes.Where(n => n.Length > 8);
```

Hint

Here's an example that sets the variable `eBirds` to a LINQ query:

```
var eBirds = birds.Where(bird => bird.StartsWith("e"));
```

4.
Use `Count()` to print the number of elements in `longHeroes`.

Hint

`longHeroes` is of type `IEnumerable<T>`, not `List<T>`, so we use the method `Count()`, not the property `Count`.