Shorter Lambda Expressions

Time to put on our detective caps: using deductive reasoning, we can make our lambda expression even shorter. Here's what we have to start:

```
bool hasEvenNumbers = Array.Exists(numbers, (int num) => num % 2 == 0);
```

The type of `num` is `int`. It's great to be explicit like this to avoid errors, but some developers wouldn't include `int`. To them, it's obvious! Here's their reasoning:

> The modulo operator (`%`) is only used with numbers, so `num` must be a number
>
> The result of the operation `num % 2` is compared to the integer `0`. We can only compare similar types, so num must also be an integer!

Therefore, we can remove `int` without causing any errors:

```
bool hasEvenNumbers = Array.Exists(numbers, (num) => num % 2 == 0);
```

When there is just one parameter in a lambda expression, we don't need the parentheses around the parameter either:

```
bool hasEvenNumbers = Array.Exists(numbers, num => num % 2 == 0);
```

We just learned two new shortcuts "within" the lambda expression shortcut. Though we don't need to use them all the time, we do need to recognize them in other developers' code:

> We can remove the parameter type if can be inferred
>
> We can remove the parentheses if there is one parameter

☑Instructions

1.
Apply the first shortcut to the lambda expression (remove the parameter type).

Hint

Here's an example of the shortcut. This expression:

```
(int num) => num % 2 == 0
```

becomes:

```
(num) => num % 2 == 0
```

2.
Apply the second shortcut to the lambda expression (remove the parentheses).

Hint

Here's an example of the shortcut. This expression:

```
(num) => num % 2 == 0
```

becomes:

```
num => num % 2 == 0
```