The expression `num => num * 5` is a lambda expression. The `=>` operator is called the "lambda operator". In this example, `num` is an input parameter to the anonymous function, and the return value of this function is `num * 5`. So when `multiplyByFive` is called with a parameter of `7`, the result is `7 * 5`, or `35`.

Parameter(s)

Notice that the `num` parameter doesn't explicitly specify a data type. The compiler always infers the data type of lambda expression parameters from context. In this case, the context is that the lambda expression is stored in a variable of type `Func<int, int>`. This means that it takes an `int` parameter and returns an `int` result.

You can also create lambda expressions with more than one parameter, as shown here:

```
Func<int, int, int> multiplyTwoNumbers = (a, b) =>
a * b;
// Returns 35
int result = multiplyTwoNumbers(7, 5);
```

We won't be using multi-parameter lambda expressions much in this course

Return value

Notice also that there is no `return` statement. Single-line lambda expressions don't need to explicitly use the `return` keyword to return a value. This same thing could also be