# Anonymous Functions

In this lesson, you will be introduced to anonymous functions and learn their syntax.

**We'll cover the following**

Remember when we learned about literals at the beginning of this course? They are defined as fixed values appearing directly in the source code. Literals don't need to be named, they can simply be used directly.

In the previous lessons, we needed to create multiple small functions for our summation functions to execute. But we only required the functionality of those functions. Naming them was an extra unnecessary step in that scenario.

What we need are something similar to literals. Functions that do not need to be named as their functionality is only required for a single instance.

Scala provides a solution known as anonymous functions.

Anonymous functions are function literals which don't require a name. They can be passed as-is to a higher-order function.

## Syntax #

Let's look at the syntax for an anonymous function.

```
parameters of anonymous function => function body
```

All we have to do is write the parameter of the anonymous function followed by `=>` which is further followed by the function body of the anonymous function.

So, if we wanted to write an anonymous function which returns the cube of a number, it would look like this:

```
(x: Int) => x * x * x
```

OR

```
x => x * x * x
```

Here, `(x:Int)` is the parameter of the function and `x * x * x` is the function's body. If the compiler can infer the type of the parameter, you can omit the data type. Hence, `(x:Int)` can simply be written as `x`.

Multiple parameters are separated by a comma `,`.

## Syntactic Sugar #

An anonymous function `(x: T) => E` can always be expressed as `{def f(x: T) = E; f}` where `f` is an arbitrary name that has not yet been used in the program.

Hence, it can be said that anonymous functions are like **syntactic sugar**. They sometimes make writing functions easier, but they're not essential in the sense that they do not add anything to the fundamentally expressive power of the language.

In the next lesson, we will rewrite our summation functions using anonymous functions.