# A Brief Introduction to Methods and Functions

In the following lesson, you will be given a very brief introduction on methods and functions.

**We'll cover the following**



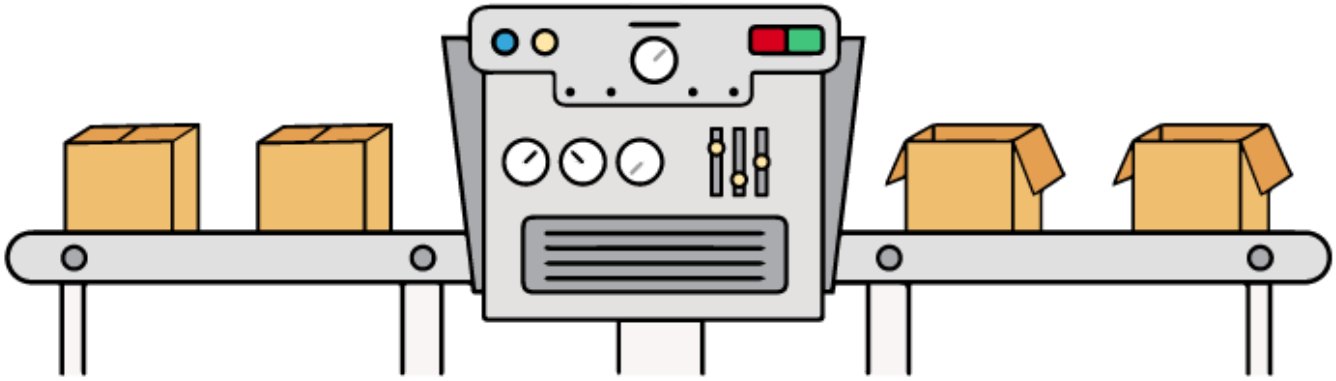
## Functions

In computer programming, a function or a method is a block of code that performs a *specific* task. That block of code is then given a name (much like a variable) that is called whenever that specific task needs to be performed. This removes the need to type the same code over and over again; all you have to do is call the function's name.

### How Do They Work?

Like mathematical functions, programming functions take in an input, known as an argument, perform some operations on that input, and then return the resulting output.

It's almost like a conveyer belt in a factory with items entering the machine from one end of the conveyer belt and coming out, completely modified, on the other end. However, as the machine remains the same throughout its life, it will modify every item that enters it the same way.



## Functions in Scala #

We can divide functions into two broad categories:

1. Built-In Functions
2. User-Defined Functions

User-defined functions are functions that users create themselves. We have a whole chapter dedicated to just these, so they aren't of any concern to us now.

Built-in functions are functions which are predefined by Scala and are part of their libraries. All we have to do to use them is call their function name.

> Built-in functions are known as methods.

The printing methods we looked at in the previous chapter are all built-in functions.

This code requires the following environment variables to execute:

| | |
|---|---|
| LANG | C.UTF-8 |

```
val printMe = "Hello World"

println(printMe)
```

In the code above `println` is a method which performs the specific task **print**. `printMe` is the argument we pass to the method and “Hello World” is the resultant output. We’ve been using functions this whole time!

## Calling a Function #

While `println` is a method, it is a very simple one which only requires passing it an argument which can be of any type. However, most methods require you to call them *on* something, let’s call that *something* an object for now. That object can be anything from `AnyVal` to `AnyRef`. For example, `objectName.method(argument)` means that the method is being called on `objectName` and `argument` are parameters passed to the method. The method will perform some action on the data stored in `objectName`.

Most methods allow you to only pass an argument of a specific data type. Let’s call one of Scalas built-in methods, `indexOf` to get a better idea of how this works.

`IndexOf` is called on a string and you pass it one argument of type `Char`. It is used to calculate the index of a character in a string.

This code requires the following environment variables to execute:

| | |
|---|---|
| LANG | C.UTF-8 |

```
val string1 = "Hello World"
val result = string1.indexOf('W')

// Driver Code
println(result)
```

In the code above, `string1` is the object we are calling the method `indexOf` on and `W` is the argument we are passing it. The output is `6` as `W` is located at the 6th index in the string `"Hello World"`.

Scala is unique in a way that it provides two approaches which can be used to call a method.

### Ordinary Method Call #

The more widely known and used method call is the same one we used in our `indexOf` example.

## `objectName.methodName(arguments)`

In this method call, the object we are calling the method on, and the method name are separated by `.` while the arguments to the method are passed in `()`.

### Method Call Using Operator Notation #

You can also call a method like you would use an operator such as `+`. The method in this would be acting as an operator.

## `objectName methodName arguments`

Let's modify the `indexOf` example in such a way that it uses the operator notation.


We get the exact same output as above.

At this point in the course, this is all you need to know about methods and functions to move forward. But don't worry, they will be covered in great detail in later chapters.

---

Now, let's move on to operators in the next lesson.