Defining a Function

In this lesson, we will start our discussion on functions and learn how to define our very own function.

We'll cover the following

Writing Your First Function

In a previous lesson, you were introduced to built-in functions; also known as methods. In this chapter, we will cover user-defined functions and learn how to write our very own functions. Let's get straight to it and look at an example of a user-defined function.

```
def sum(x: Double, y: Double): Double ={
  x+y
}
```

The above function takes two integers and returns their sum. Let's look at what is exactly happening in the code above.

Syntax

- `def` is the keyword used for defining a function the same way `val` and `var` are used for defining variables.
- `sum` is the name of the function which you get to choose yourself. Make sure the name is meaningful to the functionality of the function.
- `sum` is followed by `()` in which you define the function parameters separated by commas. Parameters specify the type of input to be given to the function.
- `(x: Double, y: Double)` is telling us that our function takes two parameters

- `(x: Double, y: Double)` is telling us that our function takes two parameters. The first one is named `x` and must be of type `Double`. The second one is named `y` and must also be of type `Double`.
- After defining the parameters, we define the return type of the function which in our case is `Double` and is done by inserting `: Double` after the `()`.

> Inserting the return type is not required as Scala can use type inference to infer the return type of the function.

- After defining the function's return type, we insert a `=`. Whatever comes after `=` is the body of the function and defines what the function will do. The body of the function is wrapped in curly brackets `{}`. You can choose not to use the curly brackets if the function's body only consists of a single expression.

```
def functionName(parameters): returnType ={
    function body}
```

Calling a Function

You call a user-defined function the same way you call a built-in function; by calling its name followed by the input in `()`. Let's call the `sum` function we defined above. We will store the return value of the function in a variable `result`.

This code requires the following environment variables to execute:

LANG   C.UTF-8

```
def sum(x: Int, y: Int): Int ={
  x+y
}

val total = sum(2,3)

// Driver Code
println(total)
```

In the code above, we are passing **2** and **3** to the function `sum` which will add them together and return their sum. This sum will then be stored in the variable `result` whose value we are printing on line **3**.

Calling Functions within Functions

Sometimes we come across a situation where the functionality of an already existing function is required in a new function. Instead of rewriting code, we can simply call the old function in the body of the new one we are writing. This will be made clear with an example.

Let's write a function which gives the square of a given number.

This code requires the following environment variables to execute:

LANG	C.UTF-8

```
def square(x: Double) ={
  x * x
}

// Driver Code
val result = square(5)
print(result)
```

Now, we want to write a function which takes the sum of the squares of two numbers. Let's try doing this using the `square` function we defined above.

```
def squareSum(x: Double, y: Double) ={
  square(x) + square(y)
}
```

In the code above, we are calling the function `square` in the function `SquareSum`. Let's call `SquareSum` and see what happens.

This code requires the following environment variables to execute:

LANG	C.UTF-8

```
def square(x: Double) = {
  x * x
}
def squareSum(x: Double, y: Double) = {
  square(x) + square(y)
}

val result = squareSum(2,5)
```

```
// Driver Code
println(result)
```

Now that you’ve learned how to write your own function, try writing a function yourself in the next lesson.