By: Eng. Elias Al-Showaiter

[Dart Programming Language] - Lecture [6]

[Functions]

1. Introduction

Functions are one of the most fundamental building blocks in programming, and Dart is no exception. A function is a reusable block of code that performs a specific task. It can take inputs, process them, and return a result.

Learning Objectives

1. Understand the basics of Functions.
2. Be able to apply the concepts of functions.
3. Understand how to analyze any function.

2. Context

Technical Background

Increasing the productivity and efficiency of our code is one of the main goals of any programmer, doing things in a repetitive manner is not good and is a huge waste of time. Functions solve this problem brilliantly, not only does functions help us avoid repetitive code, but also they allow us to use a code we wrote once in different situations with different results.

Functions will be our main tool in solving in problem from now until the end of this course. It is best programmer best friend.

Key Terms and Definitions

1. Functions: a reusable block of code that performs a specific task.
2. Return Type: it is very similar to a data type, but a return type will specify what is the type of the ***output*** of a function.
3. Input: a variable that is used by the function and its value is determined when we use the function.
4. Function definition: the process of writing the function, ***this will not execute the function.***
5. Function calling: the process of executing the function.

3. Main Content

1. Function creation:

From lecture 3 we learned that additional functions and classes are defined outside the main function, they are defined in the last section of the program structure.

And throughout this course we have been using the word behavior that describes a set of instructions that can change its results if we send different inputs. The similarity between the two terms is obvious, this is because behaviors and functions are the same thing.

Now let's take a look at a functions syntax.

```
ReturnType functionName (ParameterType parameterName)
 {
 // Function body
 // Return statement (if necessary)
}
```

Let's analyze this syntax, first we start by the return type, ***this is used to describe the type of the output of the function.*** Then we choose a name for our function, then we open two parenthesis (), inside of them we choose the functions input, in programming a function input is called parameter, and simply a parameter is a variable, and since it is a variable we must determine its data type, it is important to note that parameters are ***optional***, so we can write a function without any inputs. following this, we write the body of the function, which is the code that is written to achieve the task we want to do. Finally, we write the return statement which determines the output of the function. We notice also that the output of the function is also ***optional.***

Now let's look at a real example of a function

```
int add(int a, int b)
{
 return a + b;
 }
```

This is an example of a function definition, this function is called add and it has two parameters of type int called a and b, this function adds these two parameters and ***return their result.***

We need to ask ourselves a very important question here. What is the purpose of the return keyword, why do we need it???

To answer this question, we need to review two concepts we learned about earlier, first is program structure, in this we learned that functions are defined outside of main in the final section of the program. Second is variable scope, we learned that I can only access a variable in its scope. So here is the problem:

1. The function is defined outside main
2. I cannot access the variables of the function because they are not in the appropriate scope.
3. The result of function will not be used.

The return statement is used to fix this issue; ***it gives the output of the function to the main function so that I can use it in other operations.***

Another question that comes to the mind is, what if a function does not have an output, what will I return? In this case there is a special keyword that is used to indicated the absence of an output called ***void,*** if the function has a return type of void then this function will not have a return statement.

We can rewrite the previous function as follows:

```
void add(int a, int b) {
 print( a + b );
}
```

The difference between the first function and this one is, in the first one we will be able to use the result of a + b later on that code, but in the second function we will ***never*** be able to use the result of a + b.

2. Function Calling:

In the previous section we discussed function definition which is the process of writing the function, but the writing the function alone will not use it. We must call the function. It is very important to note that creating a function is done outside of the main function. However, calling a function (using it) will happen inside of main, this is natural, because we said that main is where our program starts and finishes so it is normal to use functions inside main, but since creating functions does not execute them, it is natural to create them outside of main.

Looking at the previous function (the add function with the return a + b), we can use it as follows.

```
void main() {
 int sum = add(5, 3);
 print('Sum: $sum');
}
```

The first line of main is very important to understand.
we defined a variable called sum and it is of the type int, and we made its value the result of the function that is called add. When call the function we just simply write its name, then we choose the values we want, in this case we chose 5 and 3. These values in programming are called arguments. ***Variables when defining a function are called parameters, values when we call a functions are called arguments***

For this code to work, ***two things must happen,*** the return type of the function must be integer, because the variable sum is integer. And the function must ***return*** the result.

3. Function parameters:

Function parameters can be defined in two ways, the first we call, Positional Parameters, and this way is the most common way. To understand it let's take a look at this example.

```
void main() {
greet("Ali", "Ahmed");
 }

void greet(String firstName, String lastName)
 {
print('Hello, $firstName $lastName!');
 }
```

The greet function takes two String parameters (firstName, lastName) and prints a message to them. The order in which values are passed to the function must match the order of the parameters. Meaning the value Ali will always go to the variable firstName and the value Ahmed will always go to the variable lastName. These values goes to the variables based on the position of them.

To make a parameter optional we can use two things, wither to give it an initial value or to put it inside [ ].

For example:

```
void greet(String firstName, [String? lastName])
 {
       if (lastName != null)
        {
           print('Hello, $firstName $lastName!');
        }
        else {
           print('Hello, $firstName!');
 } }
```

We used [] with the second parameter to indicate that it is not optional. If we do not send a value when calling the function, it is value will be null.

The second way is using Named Parameters, they allow you to specify the name of the argument when calling a function, improving readability.

Let's take a look how to use them:

```
void main()
{
 greet(firstName: 'Ali', lastName: 'Ahmed');
 }

void greet({String firstName = 'Alaa', String lastName = "Omar"})
 {
print('Hello, $firstName $lastName!');
}
```

We notice that we surrounded the parameters with curly brackets ({}), and are given initial values, this a good practice when using named parameters. Notice when we called the function the arguments where Ali and Ahmed. This will print Hello Ali Ahmed. But if we called the function with no arguments great(). This will print Hello Alaa Omar; ***this means that giving a parameter an initial value makes the parameter optional.***

In next courses named parameters will be very common to see.

We need to understand one more thing about named variables, how to make a parameter not optional. We can use a keyword called required to indicate that this parameter is not optional. Not that this is only used with named parameters.

```
void greet({required String firstName, String? lastName})
{
print('Hello, $firstName $lastName');
}
```

Here we have two parameters the first one is required, the second one is nullable (the data type is followed by a ?) so it is optional.

```
void main() {
 greet(firstName: 'John'); // OK
greet(); // Error: The named parameter 'firstName' is required
}
```

4. Function arrows:

Sometimes we see a different syntax for defining functions, this method uses arrows and it is just a simple shortcut. The syntax is as follows:

```
int square(int x) => x * x;

int square(int x) {
return x * x ;
}
```

These two functions are exactly the same.

5. Lambda Functions (anonymous functions):

Next Lecture.