



LECTURE 9.2: FUNCTIONS

BY LINA HAMMAD & AHMAD BARGHASH

In this lecture, you'll learn about functions; what functions are, its syntax and how to create a user-function in Kotlin.

KOTLIN FUNCTIONS

- In programming, function is a group of related statements that perform a specific task.
- Functions are used to break a large program into smaller and modular chunks.
- Dividing a complex program into smaller components makes our program more organized, reusable, manageable and it avoids code repetition.
- For example lets say we have to write three lines of code to find the average of two numbers, if we create a function to find the average then instead of writing those three lines again and again, we can just call the function we created.

TYPES OF FUNCTION IN KOTLIN

- There are two types of functions in Kotlin:
 1. Standard Library Function
 2. User-defined functions

STANDARD LIBRARY FUNCTION

- The standard library functions are built-in functions in Kotlin that are readily available for use. For example,
 1. `print()` is a library function that prints message to the standard output stream (monitor).
 2. `sqrt()` returns square root of a number (Double value)
- So, The functions that are already present in the Kotlin standard library are called **standard library functions** or **built-in functions** or **predefined functions**.

STANDARD LIBRARY FUNCTION EXAMPLE

```
fun main(args: Array<String>) {  
    var number = 5.5  
    print("Result = ${Math.sqrt(number)}")  
}
```

The output is:

Result = 2.345207879911715

- Here, **sqrt()** is a library function which returns square root of a number (Double value).
- **print()** library function which prints a message to standard output stream.

USER-DEFINED FUNCTIONS

- A user defined function is a function which is created by user.
- For example, lets say we want a function a check even or odd number in our program then we can create a function for this task and later call the function where we need to perform the check.
- User defined function **might take** parameter(s), perform an action and **might return** the result of that action as a value.
- Kotlin functions are declared using the **fun** keyword.

HOW TO CREATE A USER-DEFINED FUNCTION IN KOTLIN?

- Before you can use (call) a function, you need to define it.
- Here's how you can define a function in Kotlin:

```
fun callMe() {  
    // function body  
}
```

- To define a function in Kotlin, **fun** keyword is used. Then comes the name of the function (**identifier**). Here, the name of the function is **callMe**.
- In the above program, the parenthesis **()** are empty. It means, this function doesn't require any argument to work.
- The codes inside curly braces **{ }** is the body of the function.


HOW TO CALL A FUNCTION?

- You have to call the function to run codes inside the body of the function. Here's how:

```
fun main() {  
    callMe() // call the function  
}
```

- This statement calls the callMe() function declared earlier.

```
fun callMe() {  
    ... ..  
    ... ..  
}  
  
fun main(args: Array<String>) {  
    ... ..  
    callMe()  
    ... ..  
}
```

A diagram consisting of a rectangular box with a line extending from the 'callMe()' call in the 'main' function to the 'fun callMe()' definition, with an arrowhead pointing to the definition, illustrating the function call.

EXAMPLE 1: SIMPLE FUNCTION PROGRAM

```
fun callMe() {  
    println("Is")  
    println("Fun")  
}  
  
fun main(args: Array<String>) {  
    callMe()  
    println("Kotlin")  
}
```

The output is:

```
Is  
Fun  
Kotlin
```

- The callMe() function in the above code doesn't accept any argument.
- Also, the function doesn't return any value.

EXAMPLE 2: SIMPLE FUNCTION PROGRAM

```
//Created the function  
fun sayHello(){  
    println("Hello")  
}  
  
fun main(args : Array<String>){  
    //Calling the function  
    sayHello()  
}
```

The output is:

Hello

USER DEFINED FUNCTION WITH ARGUMENTS AND RETURN TYPE

- Functions are also taking parameter as arguments and return value. Parameters in function are separated **using commas**.
- If a function does not returns any value than its return type is *Unit* (Similar to *void* in other languages). It is optional to specify the return type of function definition which does not returns any value.

- Syntax:

```
fun function_name(param1: data_type, param2: data_type, ...): return_type
```

EXAMPLE:ADD TWO NUMBERS USING FUNCTION

```
fun addNumbers(n1: Double, n2: Double): Int {  
    val sum = n1 + n2  
    val sumInteger = sum.toInt()  
    return sumInteger  
}  
fun main(args: Array<String>) {  
    val number1 = 12.2  
    val number2 = 3.4  
    val result: Int  
    result = addNumbers(number1, number2)  
    println("result = $result")  
}
```

The output is:

Result = 15

In the program, sumInteger is returned from addNumbers() function to the variable result in the main function.

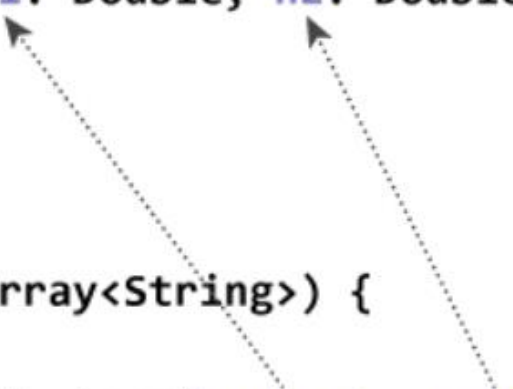
HOW FUNCTIONS WITH ARGUMENTS AND RETURN VALUE WORK?

- Here, two arguments number1 and number2 of type Double are passed to the addNumbers() function during function call. These arguments are called actual arguments.

```
result = addNumbers(number1, number2)
```

- The parameters n1 and n2 accepts the passed arguments (in the function definition). These arguments are called formal arguments (or parameters).

```
fun addNumbers(n1: Double, n2: Double): Int {  
    ... ..  
    ... ..  
}  
  
fun main(args: Array<String>) {  
    ... ..  
    result = addNumbers(number1, number2)  
    ... ..  
}
```



The diagram consists of two dotted arrows. One arrow originates from the variable 'number1' in the function call 'addNumbers(number1, number2)' within the 'main' function and points to the parameter 'n1' in the function definition 'addNumbers(n1: Double, n2: Double)'. The second arrow originates from the variable 'number2' in the same function call and points to the parameter 'n2' in the function definition.

HOW FUNCTIONS WITH ARGUMENTS AND RETURN VALUE WORK?

- In Kotlin, arguments are separated using commas. Also, the type of the formal argument must be explicitly typed.
- Note that, the data type of actual and formal arguments should match, i.e., the data type of first actual argument should match the type of first formal argument. Similarly, the type of second actual argument must match the type of second formal argument and so on.

- Here,

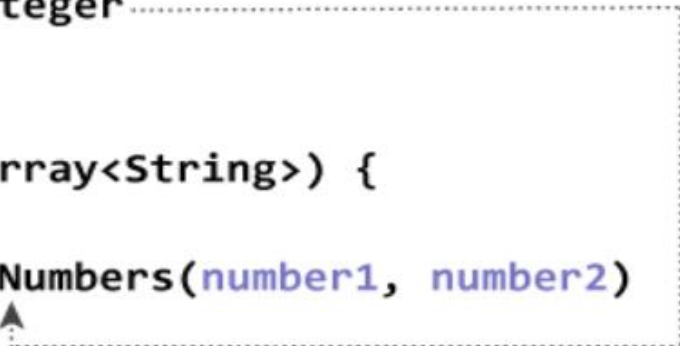
```
return sumInteger
```

is the return statement. This code terminates the `addNumbers()` function, and control of the program jumps to the `main()` function.

- In the program, `sumInteger` is returned from `addNumbers()` function. This value is assigned to the variable `result`.

HOW FUNCTIONS WITH ARGUMENTS AND RETURN VALUE WORK?

```
fun addNumbers(n1: Double, n2: Double): Int {  
    ... ..  
    return sumInteger  
}  
  
fun main(args: Array<String>) {  
    ... ..  
    result = addNumbers(number1, number2)  
    ... ..  
}
```



- Notice,
 - both sumInteger and result are of type Int.
 - the return type of the function is specified in the function definition.
- If the function doesn't return any value, its return type is Unit. It is optional to specify the return type in the function definition if the return type is Unit.

OPTION 1 OF FUNCTION BODY

```
fun main(args: Array<String>) {  
    println(getName("Kotlin", "Functions"))  
}  
  
fun getName(firstName: String, lastName: String): String {  
    var name = "$firstName $lastName"  
    return name  
}
```

The output is:

Kotlin Functions

OPTION 2 OF FUNCTION BODY

```
fun main(args: Array<String>) {  
    println(getName("Kotlin", "Functions"))  
}  
  
fun getName(firstName: String, lastName: String): String {  
    return "$firstName $lastName"  
}
```

The output is:

Kotlin Functions

OPTION 3 OF FUNCTION BODY

```
fun main(args: Array<String>) {  
    println(getName("Kotlin", "Functions"))  
}
```

```
fun getName(firstName: String, lastName: String): String = "$firstName $lastName"
```

The output is:

Kotlin Functions

OPTION TO RETURN VALUE

```
fun main(args: Array<String>){  
    sum()  
    print("code after sum")  
}  
  
fun sum(){  
    var num1 =5  
    var num2 = 6  
    println("sum = "+(num1+num2))  
}
```

The output is:

sum = 11

code after sum