Table of Contents

Introduction

Chapter 1
First Program with Kotlin

Installation of Kotlin

In order to compile and execute programs written in Kotlin, you need a compiler. To get the latest version of the Kotlin compiler, download it at the following URL: https://github.com/JetBrains/kotlin/releases/latest.

Once the .zip file has been downloaded, extract it in the folder of your choice. Then, you need to add the path to the `bin` folder it contains in the `PATH` environment variable of your system. How to do this depends on which operating system you use.

Windows

- In the Control Panel, choose *Small Icons* right after *View by* in the top-right corner
- Click on *System*
- On the left, click on *Advanced system settings…*
- In the dialog box, click on the `Environment Variables…` button
- In the bottom frame, click on the line for which the value of the Variable column is `Path`
- Click on the `Edit…` button
- Add the path to the `bin` folder in a new line
- Click `OK` three times to close the three open dialog boxes

macOS

- Open a terminal
- Type `sudo nano /etc/paths`, then hit the Enter key
- Add the path to the `bin` folder in a new line
- Type `Ctrl+O` to save, then `Ctrl+X` to quit

Linux

How to complete the Kotlin installation will vary depending on your distribution. But if you are a Linux user, chances are good that you know how to edit the `PATH` environment variable by yourself.

Hello, World!

Source Code

A **Hello, World!** is a simple program that aims to provide a first foray into a programming language. It’s a basic program that displays `Hello, World!` on your screen. The main purpose of a **Hello, World!** is to keep your first experience with a programming language as simple as possible.

Open a text editor (such as the excellent program Atom, for example), and write the following content in a new file. Then, save the file with the name `HelloWorld.kt`.

HelloWorld.kt

```
fun main(args: Array<String>) {
   println("Hello, World!")
}
```

From Source Code to a Running Program

You’ll need to take three steps to run your new Kotlin program.

Write source code

That’s what we just did in the text editor. It is all about writing a program in a language that a developer can read. What we wrote is simple code that can be read and modified easily enough by any developer.

Compiling

This step is about translating your source code into a file that your computer can understand. For some languages, such as PHP or JavaScript, you don’t need to compile source code. The system will be able to read and interpret your program directly. Therefore, it is not the best in terms of performance.

As humans, we are able to write code that a computer can understand right away. The problem with it is that it will be complex to read and to modify by a human.

HelloWorld.exe

```
4D 5A 3B 00 01 00 00 00 02 00 00 01 FF FF 02 00
00 10 00 00 00 00 00 00 1C 00 00 00 00 00 00 00
0E 1F B4 09 BA 0E 00 CD 21 B8 00 4C CD 21 48 65
6C 6C 6F 20 57 6F 72 6C 64 21 24
```

Take a look at the program above: there is almost no chance that you understand anything about it. It’s an hexadecimal transcription of a `Hello, World!` for 32-bit DOS systems. You can probably see that if we write in the *“natural language”* for the system, it becomes very hard to understand or modify this message.

So the compiler is the tool that will generate a file that is easy to understand for the system, from one or more files that are easy to understand for a human.

System and runtime environment

We just talked about code that is easy to understand for the system. It is important to note that, in this case, the “system” is the environment in which the program is running. It may be the operating system or, as is the case for Java (and also for Kotlin, which is based on Java), a dedicated runtime environment.

The advantage of a dedicated runtime environment like this is that you can write programs and share binaries regardless of the operating system on which it will be run.

For example, a `.exe` binary can be run on Windows, but if you try to run it on macOS, this will never work. But a Java or Kotlin binary file, which is a `.class` file, can be run on Windows and macOS as well as on Android.

Run

The final step is running the program. This step will produce the final result.

Compilation with Kotlin

To compile a program with Kotlin, you just need to run the following command in a terminal:

```
kotlinc HelloWorld.kt
```

This command will create a file named `HelloWorldKt.class`. It is a Kotlin binary file.

Run with Kotlin

To run a Kotlin program that has been compiled, you just need to run the following command:

```
kotlin HelloWorldKt
```

This will produce the following result:

```
Hello, World!
```

Congratulations. You just wrote a source code, compiled it, and ran your first Kotlin application.

More Information about Source Code

Let's go back to our source code. This time, we will focus on understanding exactly what it does.

HelloWorld.kt

```
fun main(args: Array<String>) {
   println("Hello, World!")
}
```

Functions and Instructions

The `fun` keyword let Kotlin know that we will define a function. When talking about programming, a function is a group with one or more instructions to be run in a specific order.

An instruction is just a line calling an action to be performed by the computer. This action may be a system action (create a file, display text on screen, etc.), or a call to a function.

Right after the `fun` keyword comes the name of the function, which is `main` in the current file. Because this function is named `main` , Kotlin knows that this is the function it has to call when we run the `HelloWorldKt` program. Actually, we can write many functions in a single program. If we do so, Kotlin needs to know which one should be called first.

Between parentheses, there are arguments of our function. Here, we only have one argument: `args: Array<String>` . The arguments are the data that the function will interact with. In the current function, those data are useless. So we will not go deep into the arguments of this `main` function, but they are required here. If we wrote our `main` function like this: `fun main()` , there would have been a bug while running—even though writing it like this is absolutely correct and is a valid `main` function. It has more to do with a bug with the specific `main` function, which is the first one called when running a program.

Then, we have an opening bracket and a closing one at the end of the file. Those brackets are here to delimit the instructions for a function. So all the instructions between those brackets are instructions for the main function.

Println

Then, we call the `println` function. The main purpose of this function is to print text on a new line. The argument of this function is the string `"Hello, World!"` So we see how useful arguments are here. In this case, the argument is the data that will be printed by `println` .

Conclusion

Now we've seen how to write a source code, compile it, and run the binaries with Kotlin. We also know what functions, instructions, and arguments are.

In case you're feeling a little fuzzy on these terms: in the beginning of this book, there will be a lot of definitions that you may find abstract and that you may find difficult to understand. This is absolutely normal and should not discourage you.

Yes, those terms need definitions when we use them the first time, but you don't need to learn them by heart to continue reading (and understanding) the book.

It is absolutely not required to know the definitions of those terms by heart. In fact, I had to go to Wikipedia to get a better idea of how to define those terms. Once you have developed some programs, you will find that you know what a function is, although you may not be able to define what it is precisely. You will just have a clear idea about what it is and what it is not (and how to apply that knowledge), and that is the important part.

Exercises

Each chapter will end with one or more exercises, and each of these will include a word to describe how difficult it is: easy, medium, or difficult. It is not about how complex it is (because complexity is specific to each person), but is more of a scale to help you know what to expect:

- **Easy**: Everything you need to complete the exercise is in the chapter. You just have to apply the theory to this specific task.
- **Medium**: You can complete this exercise with the content of the chapter, but you will need to experiment, try things, and get things wrong in order to find the way to do it. This is because everything you have to do is not necessarily in this chapter.
- **Difficult**: You will not be able to complete the exercise with the content of the chapter alone. You will have to check on the Internet or use other sources to help you complete the task.

Of course, I could write an exhaustive book with every piece of information you need to complete these exercises. But the aim is to help you learn to face these problems. Once more, the main purpose of this book is to enable you to become an entry-level developer. And, a developer will spend all his days trying, helping coworkers, getting help from coworkers, and finding solutions to problems he encounters on the Internet. Since you have to get used to this process as soon as possible, let's do it … right now:

Exercise 1 - Medium

Go back to the `HelloWorld.kt` program, and edit it so that:

- It consists of two functions, `main` and `displayHelloWorld`.
    - `displayHelloWorld`, without any argument, will be a function that displays `Hello, World!`

    - `main` will only call `displayHelloWorld`.

Exercice 2 - Difficile

We saw that when compiling a program, a file named `HelloWorldKt.class` is created in the current folder. Find a way to compile it so `HelloWorld.kt` will be placed in a folder called `exercise`, and use only `kotlinc` command. In other words, don't create the `exercise` folder any way but by using `kotlinc` command.