

Kotlin



Kotlin for Beginners



C#Corner

Author
RaviKant Sahu

The Kotlin for Beginners Book

All rights reserved. No part of this publication may be reproduced, distributed, or transmitted in any form or by any means, including photocopying, recording, or other electronic or mechanical methods, without the prior written permission of the publisher, except in the case of brief quotations embodied in critical reviews and certain other noncommercial uses permitted by copyright law.

Although the author and publisher have made every effort to ensure that the information in this book was correct at press time, the author and publisher do not assume and hereby disclaim any liability to any party for any loss, damage, or disruption caused by errors or omissions, whether such errors or omissions result from negligence, accident, or any other cause.

The resources in this book are provided for informational purposes only and should not be used to replace the specialized training and professional judgment of a health care or mental health care professional.

Please always consult a trained professional before making any decision regarding your treatment or others. Please always consult a trained professional before making any decision regarding the treatment of yourself or others.

Author – Ravikant Sahu

Publisher – C# Corner

Editorial Team – Archie Vankar, Deepak Tewatia

Publishing Team – Praveen Kumar, Amit Kumar

Promotional & Media – Rohit Tomar, Rohit Sharma

About The Author



Ravikant Sahu is a Software Engineer working as an Android developer at MCN Solutions in Noida, India. He holds a master's degree in computer application from RDVV University Jabalpur, India.

Ravi is a technical writer and regularly contributes content on the C# Corner. As a technical writer, he has a focus on creating informative and engaging articles that provide insights into the world of technology, with a particular emphasis on programming languages like Java and Kotlin.

His book "Kotlin for Beginners: A Step-by-Step Guide to Learning Kotlin Programming" is an excellent resource for anyone who wants to learn Kotlin, a popular programming language for developing Android apps, web applications, and more.

In "Kotlin for Beginners," Ravikant shares his knowledge and expertise in a clear and concise manner, making it easy for readers to follow along and understand the concepts. Whether you're a complete beginner or have some programming experience, this book is an excellent resource for learning Kotlin and becoming a proficient developer.

Connect with me

You can connect with me on C# Corner

- [C# Corner - https://www.c-sharpcorner.com/members/ravi-sahu12](https://www.c-sharpcorner.com/members/ravi-sahu12)

Acknowledgment

Writing a book is never a solo effort, and I am grateful to have had the support and encouragement of so many people throughout this project.

First and foremost, I would like to thank the team at C# Corner for providing me with invaluable feedback and guidance throughout the writing process. Your contributions to the Kotlin community have made it a welcoming and supportive place for beginners like me.

I would also like to express my sincere thanks to my editorial team, who has provided me with invaluable feedback and guidance throughout the writing process. Your expertise and attention to detail have helped to shape this book into its final form.

To all of the individuals and organizations who have played a role in bringing this book to life, I offer my heartfelt thanks. Your contributions have been invaluable, and I could not have done this without you.

- Ravikant Sahu

Table of Contents

Introduction

Chapter 1. Get started	6
What is Kotlin	6
Why Kotlin is so popular?	7
Kotlin vs Java	9
Questions	11
Chapter 2. Setting up the development environment	12
Install the Java Development Kit (JDK)	12
Install IntelliJ IDEA	15
Chapter 3. Writing your first Kotlin Program	16
Creating a new Kotlin project	16
Writing a "Hello, World!" program	17
Running the program	18
Questions	19
Chapter 4 Basic syntax	20
Variable	20
Operators	21
Types	22
Conditions and Boolean	26
Nullability	29
Examine List, Array, and Loops	31
List	31
Array	35
Loops	39
Questions	42

Introduction

In this eBook, you will discover what Kotlin is, why it is such a well-liked programming language, how it compares to Java, and how to install the IDE to get started with your first program and explore the basics of Kotlin. Your code can be made a lot more legible and concise, which can astound you.



What you'll learn

- About Kotlin
- why Kotlin is popular
- Kotlin vs Java
- How to work with the Kotlin REPL (Read-Eval-Print Loop) interactive shell
- The basic syntax of Kotlin code
- Kotlin's data types, operators, and variables
- How Booleans and conditions work in Kotlin
- Working with nullable and non-nullable variables
- Arrays, lists, and loops in Kotlin.

What you'll do

- Install the Java Development Kit (JDK) and the IntelliJ IDEA and become familiar with some Kotlin features.
- Run programs in [IntelliJ IDEA](#) or [Kotlin](#)

Chapter 1. Get started

This E-book can help you to start your journey to learn Kotlin. If you read the chapter in order, you'll gain the greatest benefit from this E-book. You may skip some parts based on your level of familiarity. It will be an advantage if you have prior practical experience with an object-oriented language, but you can get started even if this is your first programming language to learn since we will be starting from scratch in each section. Trust me when I say that this is going to be a really fun adventure journey while exploring some fascinating topics toward learning Kotlin. In order to assist you to learn more about Kotlin than what is covered in this tutorial, we will be covering each topic with a question or exercise. So, inhale deeply, and let's get started.



Figure 1.1

What is Kotlin

Kotlin is a general-purpose, statically typed, cross-platform high-level programming language built with type inference. Kotlin was first introduced in 2011 by JetBrains and sponsored by Google, announced as one of the official languages for Android Development in 2017. It is an object-oriented language and a preferable language to Java. Kotlin mainly targets the JVM but also compiles JavaScript or native code via LLVM (low-level virtual machine). Kotlin has vast built-in support on the Best IDEs like IntelliJ IDEA, Android Studio, Eclipse, etc.

Kotlin is a modern and mature programming language aimed to simplify developers' lives. It's concise, interoperable, safe with Java and other languages, and provides many ways to reuse code between multiple platforms for productive programming. Understanding how much Kotlin has advanced and extended across platforms since July 2011 is fair.

Let's see Kotlin's timeline in detail

- JetBrains unveiled Project Kotlin (named after Kotlin island) in July 2011
- First release (Kotlin 1.0) - 15 Feb 2016
- Included in the Android Studio - Since the release of Android Studio 3.0 in October 2017,
- The first support of language by Google - Google I/O 2017

- Kotlin 1.2 was released on November 28, 2017 (Sharing code between JVM and JavaScript platforms feature was newly added to this release)
- A full-stack demo has been made with the new Kotlin/JS Gradle Plugin.
- Kotlin 1.3 was released on 29 October 2018, bringing coroutines for asynchronous programming.
- Officially announced as the preferred lang for the Android app - 7 May 2019
- Kotlin 1.4 was released in August 2020, with e.g., some slight changes to the support for Apple's platforms, i.e., to the Objective-C/Swift interoperability.
- Kotlin 1.5 was released in May 2021.
- Kotlin 1.6 was released in November 2021.
- Kotlin 1.7 was released in June 2022, including the alpha version of the new Kotlin K2 compiler.
- Kotlin 1.8 was released in December 2022, and 1.8.0 was released on January 11, 2023.



Figure 1.2

Why Kotlin is so popular?

Don't you find it intriguing that Kotlin is now widely used and supports multiple platforms, but why has it progressed so much and gone so far if not for its features? Whatever feature or competitive advantage that a language has over others is what makes it renowned, and Kotlin has a lot of those. We will discuss some of the important features here.

- **Statically typed** - a type of every variable and expression is known at compile time.
- **Compatibility** - Also known as Interoperable with Java. Kotlin is fully compatible with Java.
- **Concise** - Kotlin with its features like lambda & other high-order functions makes it more concise.

- **Easy** - The language is easy to learn and if you already know Java then you already have an edge in learning.
- **Safe** - With its null and type-safe feature the code is safe to use and avoids unwanted crashes.
- **Time-saving** - Kotlin is very fast in compilation with its higher performance.
- **Community Support and Tool Friendly** - Kotlin has vast community support with its robust tools.
- **Code reusability** - Kotlin is multiplatform so your code can run on any machine.
- **Build-in support** - It supports using Java libraries and framework with its Kotlin features without any changes in your project.
- **Recommended by Google** - Almost every Engineer in Google enjoy using Kotlin and most of the app (more than 70) developed by Google are built using Kotlin.
- **Codebase size & Crashes** - Kotlin reduces the codebase size by 33% and reduces crashes by 30%.



Figure 1.3

Kotlin vs Java

Kotlin was first released as a language for developing Android applications, but it has continuously expanded over time. Now that Kotlin is available everywhere, which one is better? Kotlin is superior with the numerous enhancements that we went through in the previous section of this E-book because it was established after java to overcome the problem with existing and construct a better language. So what makes Kotlin better? let's check out from below.

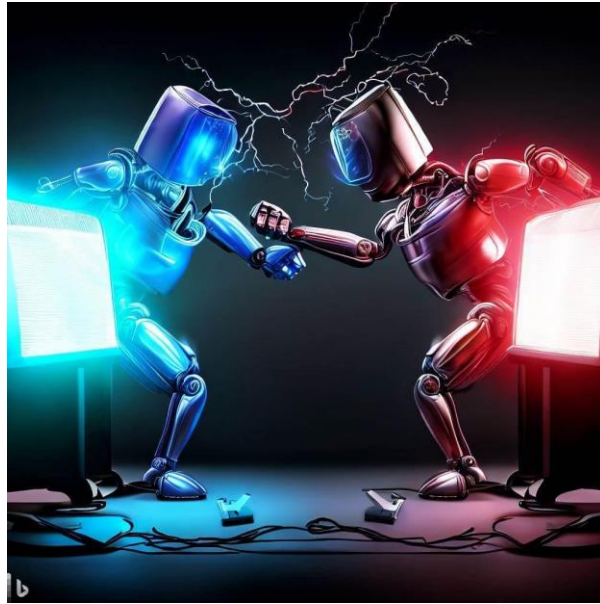


Figure 1.4

Kotlin fixes a series of issues that Java suffers from:

- Null references are controlled by the type system.
- No raw types
- Arrays in Kotlin are invariant
- Kotlin has proper function types, as opposed to Java's SAM-conversions
- Use-site variance without wildcards
- Kotlin does not have checked exceptions

What Java lacks that Kotlin possesses?

- Lambda expressions + Inline functions = performant custom control structures
- Extension functions
- Null safety
- Smart casts
- String templates
- Properties
- Primary constructors
- First-class delegation
- Type inference for variable and property types
- Singletons
- Declaration-site variance & Type projections
- Range expressions
- Operator overloading
- Companion objects
- Data classes
- Separate interfaces for read-only and mutable collections
- Coroutines

"This is important to know that Google is really pushing Kotlin to become the preferred language in Android development. And this is true nowadays. With the release of Kotlin in 2016 it has become popular because of its new feature. Kotlin is not only concise, but it is also safer, better, and null-safe (which Java developers always suffer about). If you want to use modern tools and libraries such as Jetpack Compose or Coroutines (asynchronous programming) this is not possible without Kotlin."

Congratulations 🎉 If you are reading this right now since you know a lot more about Kotlin and can add a few things to what you previously knew. Nevertheless, did you become bored with just knowing theory? So, the next step will be to get our hands dirty and prepare our system for Kotlin. But before that, below we have some questions for you.



Figure 1.5

Questions

Take a quiz on this E-book here:

Question #1: Has Kotlin implemented null safety?

Question #2: When was the first stable version of Kotlin released?

Question #3: Can Kotlin create a website for you?

Question #4: Which language program is more legible and concise, kotlin or Java?

Question #5: In Kotlin, can we define variables without specifying their types?

Chapter 2. Setting up the development environment

The Kotlin readiness of your system will be the focus of this session. So, without wasting time let's get started.

Install the Java Development Kit (JDK)



Figure 2.1

Check the JDK version of your system. If you don't have the latest JDK already installed on your computer follow and install, if not skip to the next step. The JDK must be set up to run Kotlin apps.

Type `javac -version` in a terminal window to find out, if any, which version of the JDK you have installed.

On the [Java SE Downloads page](#) (Click here to see), you may see JDK's most recent version. Install IntelliJ IDEA if you have the most recent version.

```
ravis@ravisdesktop39:~$ javac -version
javac 17.0.5
ravis@ravisdesktop39:~$
```

Figure 2.2

Step 1: Remove any earlier JDK/JRE versions from your computer.

Remove all previous JDK versions before installing the most recent:

- For Windows, select Control Panel > Add/Remove Programs.
- For Mac instructions, [see Uninstalling the JDK](#).

Step 2: Download the JDK

Download the JDK for free here: [Java SE Downloads page](#)

- Just select JDK Download.
- Choose the JDK link for your operating system under Downloads.
- The license agreement is accepted.
- On the Download button, click.

Step 3: Install the JDK

For Windows

- Launch the installer you downloaded to install the JDK and JRE (for instance, `jdk-14.0.1 windows-x64 bin.exe`). Depending on the most recent version, the JDK is typically set up in the `C:\Program Files\Java\jdk-14.0.1` directory.
- Accept the defaults and install the JDK by following the on-screen directions.

For Mac

- click on the `.dmg` file to launch the install file.
- An open box icon and the name of the `.pkg` file are displayed in a Finder window.
- Launch the installation app by double-clicking the package icon, then follow the on-screen instructions.
- To proceed, you might need to provide the administrator password.
- To save space, feel free to remove the `.dmg` file when the installation is finished.

For Linux

- Change the directory to the location where you want to install the JDK, then move the `.tar.gz` archive file to the current directory.
- Unpack the tarball and install the downloaded JDK:

```
$ tar zxvf jdk-20_linux-x64_bin.tar.gz
```

OR

```
$ tar zxvf jdk-20_linux-aarch64_bin.tar.gz
```

The Java Development Kit files are installed in a directory named `JDK-20.interim.update.patch`.

- Delete the `.tar.gz` file if you want to save disk space.

Step 4: Add the JDK installation directory to PATH (Windows only)

Windows looks for executable programs in the current directory and the folders listed in the PATH environment variable (system variable).

Search for edit environment in Windows Settings.

- From the list of matches, choose Edit environment variables for your account.
- Choose Path and click the Edit... button in the User variables portion of the Environment Variables panel.
- After any existing entries, add the path to the JDK's bin directory, for instance, `C:\Program Files\Java\jdk-14.0.1\bin`

For additional information on the Ubuntu installation guide [click here](#).

Step 5: Verify the JDK installation

To verify that the JDK was installed correctly, type the following commands in a terminal window:

```
java -version
```

```
javac -version
```

Windows users: If you receive an error from either command, confirm you've added the correct path for the JRE.

Install IntelliJ IDEA

IntelliJ IDEA is an IDE developed by JetBrains. You can also use the online Kotlin compiler like [Kotlin Playground](#) for learning, but IDE makes things easier in life. The IDE is a comprehensive set of tools for programmers for developing, testing, and debugging code, which can help you to solve a complex problem. So, let's not wait for further talking to start doing.

JetBrains created the IDE known as IntelliJ IDEA. Although IDEs make life easier, you may also utilize online Kotlin compilers like Kotlin Playground. The IDE is a complete collection of programming tools that programmers can use to create, test, and debug code. It can help you to solve complex problems. So, let's start acting now without waiting for more discussion.

Step 1: Download and install IntelliJ IDEA

Download [IntelliJ IDEA](#) for your operating system.

Windows:

- Run the `ideaIC.exe` file that you downloaded.
- You are Good to go follow the instructions in the installation wizard.

Mac:

- To mount the macOS disk image, double-click the `ideaIC.dmg` file that you downloaded.
- Copy IntelliJ IDEA to the Applications folder.

Linux:

- See `Install-Linux-tar.txt` in the downloaded `.tar.gz` file.

For more information on how to install and set up IntelliJ IDEA, check out [Install IntelliJ IDEA](#).

If you have a slow computer, installing the software will take some time. Nevertheless, if you have a faster computer or workstation, installing software won't take as long.

Step 2: Verify the IntelliJ IDEA installation

- Run IntelliJ IDEA now.
- When prompted, install any updates and extra material.
- Up until there are no longer any updates accessible, choose to Configure > Check for Updates.

Chapter 3. Writing your first Kotlin Program

Creating a new Kotlin project

Now, the time has come when you are ready to go with the setup. So, let's start creating a new project in Kotlin.

- Click Create New Project in the IntelliJ IDEA Welcome box.
- Choose Kotlin from the left-hand menu of the New Project box.
- In the left panel, choose Kotlin Multiplatform (Console Application) and then click Next.

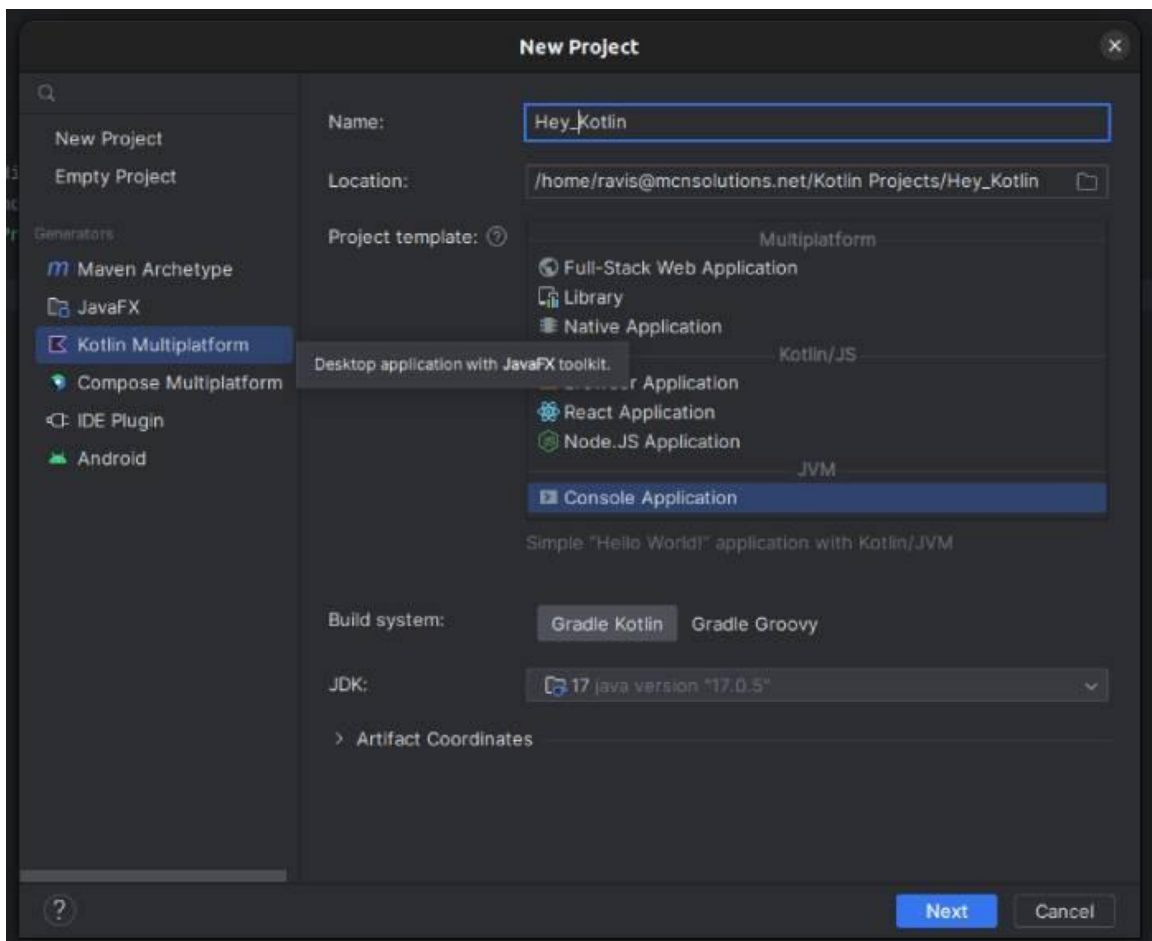


Figure 3.1

- Name your project "Hey Kotlin".
- Click Finish.

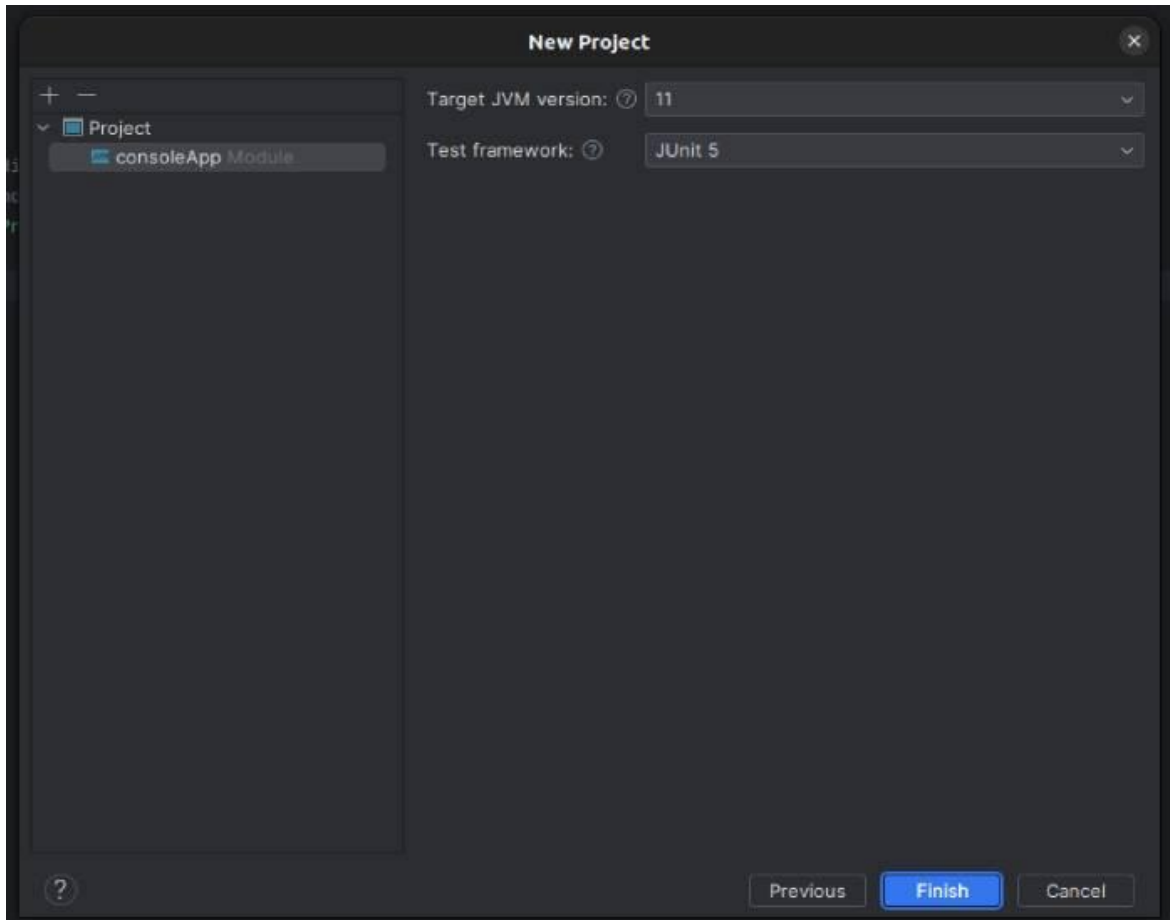


Figure 3.2

Writing a "Hello, World!" program

Do you know about REPL - A computer environment known as a Read-Eval-Print Loop, or REPL, is where user inputs are read, evaluated, and then the results are forwarded to the user. To explore the tools available in particular settings or programming languages, REPLs offer an interactive environment. Examples include the developer console found in most web browsers, IPython, the Bash shell, and the Node.js console.

Select Tools > Kotlin > Kotlin REPL to open the REPL.

It could take some time for the Kotlin menu to show up under Tools the first time you launch IntelliJ IDEA after installation.

Type or paste the code below into the REPL.

```
fun printHii() {  
    println("Hii C# Corner Learner")  
}  
printHii()
```

Running the program

Hit Control+Enter (Command+Enter on a Mac) to enter. As indicated below, you should see "Hii C# Corner Learner".

If you see your code here, the fun keyword is for function, and forwarded by its name. Similar to other programming languages, functions take arguments and code in curly braces. There is no return type of function see the structure of the function below.

```
return_type(default void) fun( arguments[] //option ){

//write your code here

}
```

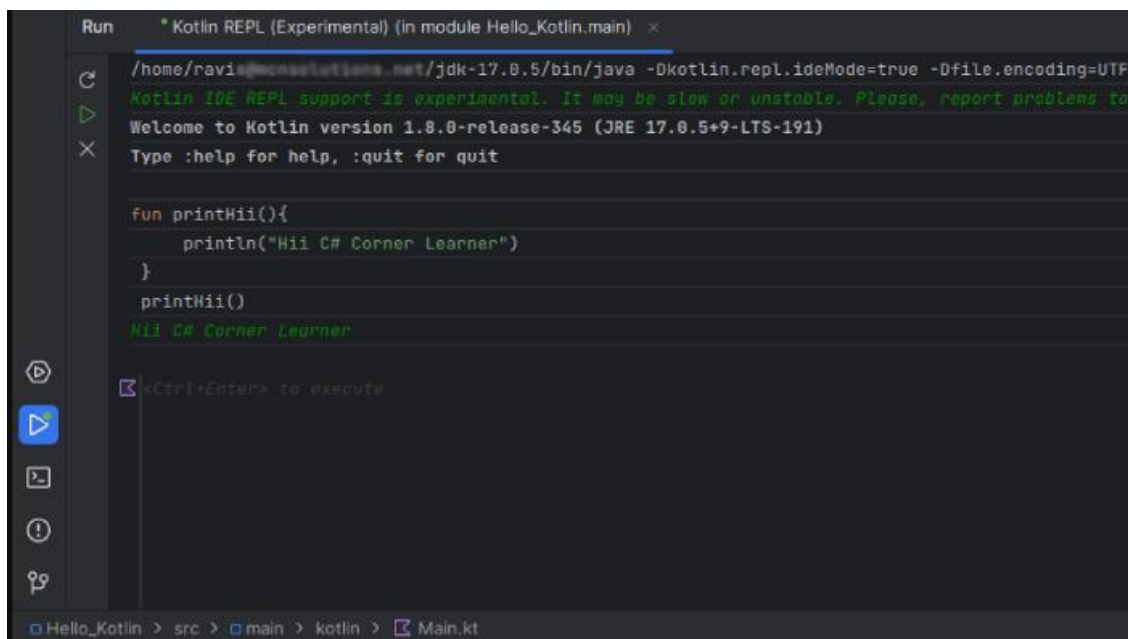


Figure 3.3

Note - there is no semicolon at the end Kotlin recognizes it based on Indentation. If you're used to putting semicolons at the end of lines, that's OK—Kotlin doesn't mind.

Congratulations! You've written your first Kotlin program. So now it's time for some questionnaires

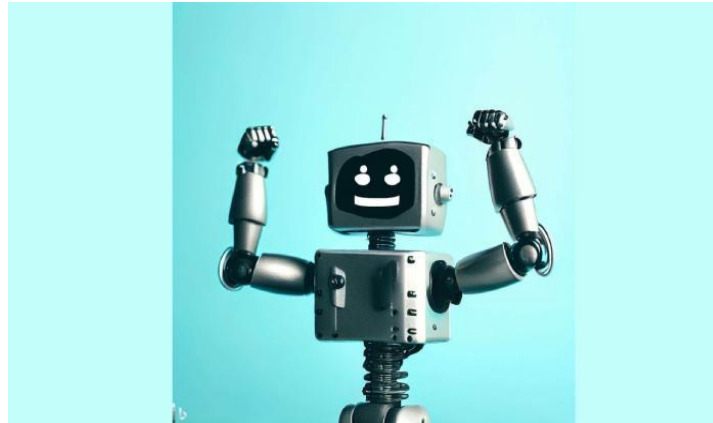


Figure 3.4

Questions

Take a quiz on this E-book here:

Question #1: How to start REPL in IntelliJ IDEA?

Question #2: Which code compiles faster Kotlin or Java?

Chapter 4 Basic syntax

In this chapter, you will discover the basic syntax of Kotlin programming language: data types, operators, variables, control structures, and nullable versus non-nullable variables. You can dive into each of the topics independently.

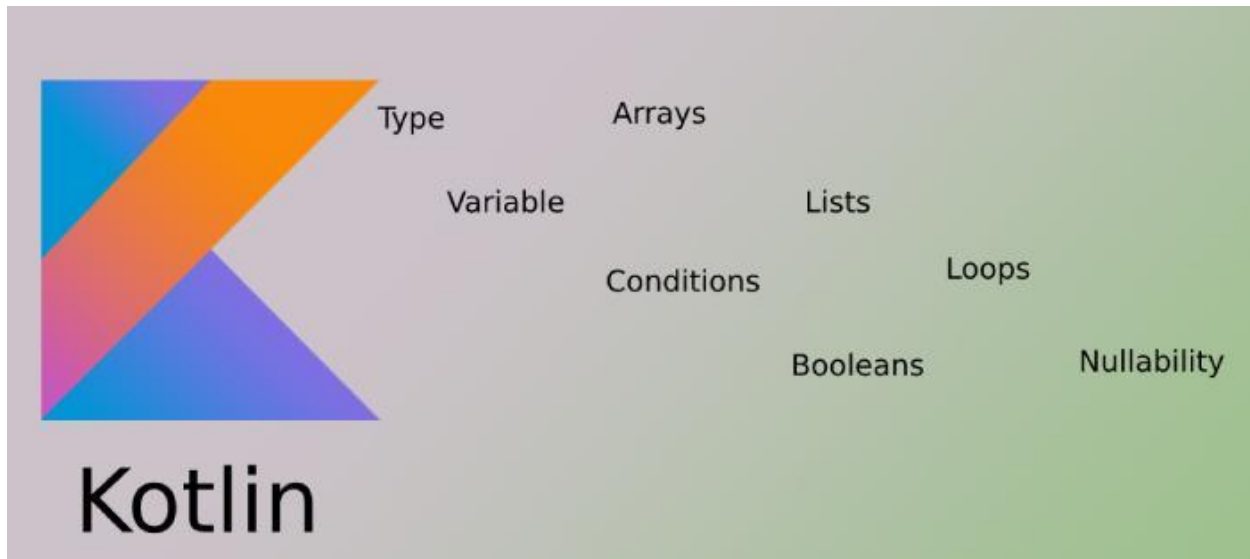


Figure 4.1

All languages differ from one another, but they also have many similar elements and features. The role of a variable is straightforward: assign a value and give it a name to be used later. To improve the efficiency and readability of the code, Kotlin takes this straightforward idea and adds a few extra rules.

Variable

There are two types of variables:

val (unchangeable) - values of this variable can be assigned only once.

var (changeable) - values of this can be reassigned or changed to them.

Kotlin variables have a simple syntax like below. The first word is used to declare which type of variable it is, the second is the name of the variable, and after :(colon) is the type of var and is optional assignment takes place after the equals (=) mark.

```
// Using var
var welcome:String = "Hello learners"
// Using val
val welcome_again:String = "Hello again learners"
```

Note - As Kotlin does not by default support null values, you cannot declare a variable using only a declaration and without immediately assigning it a value. Due to Kotlin's concept of emphasizing readability, you are forced to explicitly assign the value (even null) when declaring the variable's initial state. But if you want to use a variable whose value is assigned by runtime then you can use a keyword `lateinit` before the variable declaration.


```
lateinit var welcome_after: Int
```

Operators

To learn about Operators and types in Kotlin you can use Kotlin REPL very wisely. This will give you the internal store type which Kotlin runs about the data. As we discussed in the previous chapter you can use Kotlin REPL on IntelliJ IDEA by selecting *Tools->Kotlin->Kotlin REPL*.

Use Kotlin operator + (plus), - (minus), * (multiply), / (division). Kotlin supports Int, Long, Double, and Float types of numbers.

To evaluate the expression press Ctrl + Enter.



```
Run * Kotlin REPL (Experimental) (in module Hello_Kotlin.main) x
/home/ravis@mcnsolutions.net/jdk-17.0.5/bin/java -Dkotlin.repl.ideMode=
Kotlin IDE REPL support is experimental. It may be slow or unstable. P
Welcome to Kotlin version 1.8.0-release-345 (JRE 17.0.5+9-LTS-191)
Type :help for help, :quit for quit

1+2
res0: kotlin.Int = 3

<Ctrl+Enter> to execute
```

Figure 4.2

You can use some of these to compare the results when using integers and doubles. Also, you can call some of these techniques at the numbers listed below.

```
2+4
res0: kotlin.Int = 6
5/2
res1: kotlin.Int = 2
5/2.0
res2: kotlin.Double = 2.5
2.times(3)
res3: kotlin.Int = 6
3.5.plus(4)
res4: kotlin.Double = 7.5
2.4.div(2)
res5: kotlin.Double = 1.2
```

You cannot assign a short value to a long variable or a byte to an int because Kotlin does not implicitly convert between number types. This is because the conversion of this type can occur errors in programs although you can assign values of different types by casting. see the below examples

```
val i: Int = 25
val i1 :Double = (Int)i
// In Java you can do like that but this will not work in Kotlin
instead use this.
val i1 :Double = i.toDouble()
```

Kotlin lets you insert underscores where it makes sense to you to make long numeric constants easier to read. Try typing various numerical constants.

```
val mobile_no = 789_123_6166
val card_details = 9999_9999_9999L
//this (underscore) will not affect your numbers, it exactly be like
mobile no - 7891236166
card details - 9999999999
```

Types

Kotlin is strongly typed, Kotlin compiler can infer the type of variables, so you don't need to pre-declare the type of it (duck-typing). We can define the variable and specify the type explicitly Following the concept everything in Kotlin is an object, this makes sense that we can call its member function and properties on any of the variables. Numbers, characters, and Booleans may be represented as primitive values at the runtime, but they look like an ordinary class.

The basic types are

- Number and their unsigned counterparts
- Boolean
- Character
- String
- Array

Number and their unsigned counterparts

There are several built-in types for representing numbers in Kotlin. There are four types of integers, each having a distinct size and thus a different value range:

1. Integer types

Type	Size (bits)	Min value	Max value
Byte	8	-128	127
Short	16	-32768	32767
Int	32	-2,147,483,648 (-231)	2,147,483,647 (231 - 1)
Long	64	-9,223,372,036,854,775,808 (-263)	9,223,372,036,854,775,807 (263 - 1)

2. Floating-point types:

Type	Size (bits)	Significant bits	Exponent bits	Decimal digits
Float	32	24	8	6-7
Double	64	53	11	15-16

3. Literal constants for numbers:

```
val oneMillion = 1_000_000
val creditCardNumber = 1234_5678_9012_3456L
val socialSecurityNumber = 999_99_9999L
val hexBytes = 0xFF_EC_DE_5E
val bytes = 0b11010010_01101001_10010100_10010010
```

4. Unsigned integer types:

Unsigned numbers are mostly used to express positive values by making use of the entire bit range of an integer. For unsigned integer numbers, Kotlin offers the following kinds in addition to integer types:

- UByte: an unsigned 8-bit integer, ranging from 0 to 255
- UShort: an unsigned 16-bit integer, ranging from 0 to 65535
- UInt: an unsigned 32-bit integer, ranging from 0 to $2^{32} - 1$
- ULong: an unsigned 64-bit integer, ranging from 0 to $2^{64} - 1$

Similarly, you can use other unsigned numbers, unsigned arrays, and unsigned integers literals.

Boolean

Boolean objects are represented by the type Boolean and can have true or false values. Boolean? Also exists, and It also has a value of null.

Built-in Boolean operations include:

- `||` – disjunction (logical OR)
- `&&` – conjunction (logical AND)
- `!` – negation (logical NOT)

See the below example -

```
println(true || false) // || print when any value is true
println(true && false) // && print when both value is true
println(!true) // ! print when the opposite value is true
```

Character

To represent a single character a Char variable is used. Char is represented by a single quote literal e.g., 'a'. Starting from an escaping backslash, special characters are used. There is support for the following escape sequences:

- `\t` – tab
- `\b` – backspace
- `\n` – new line (LF)
- `\r` – carriage return (CR)
- `\'` – single quotation mark
- `\"` – double quotation mark
- `\\` – backslash
- `\$` – dollar sign

to encode any Unicode or special character you can use the `'/unicode_value'` in code.

```
val myChar: Char = 'a'
println(myChar)
println('\n') // Prints an extra newline character
println('\uFF00')
```

String

Kotlin String is like the Java programming language string, you can define a string between two double quotes (") to use and a single quote (') for single char, a string template can be made by the '\$' variable which will be replaced by the text.

```
val str = "hello 123"
```

The string is like an array of characters so you can access any char of the string using `string_name[index]` and you can iterate through it.

- Escaped strings - contain escape characters.

```
val s = "Hello, world!\n"
```

- Raw strings - Raw strings can include blank lines and any kind of text. It has no escaping, is separated by a triple quote (""), and is free to contain newlines and any other characters:

```
val text = """
|Tell me and I forget.
|Teach me and I remember.
|Involve me and I learn.
|(Benjamin Franklin)
""".trimMargin()
```

- String template - When you wish to give a string a real-time value, a string template comes into play. An expression that begins with a dollar symbol (\$) is used to calculate a changeable value, and concatenation is used to calculate a string's value

```
val limit : Int = 5
println( "you can't use more than ${limit} value ")
//You can't use more than 5 value
```

Arrays

The Array class in Kotlin features `get()` and `set()` methods as well as a size property. It also supports an iterator to interpolate values.

Use the `arrayOf()` function to use an array.

Kotlin can also be used to generate an array of a specific type, such as an array of int, byte, or short, without incurring the complexity of boxing. They share the same properties and methods as the Array class.

```
val my_array = arrayOf(1, "two", 3)
val x: IntArray = intArrayOf(1, 2, 3)

my_array.forEach { println(it) }
x.forEach { println(it) }
```

Conditions and Boolean

Kotlin condition checking has an operator like another language these are equal to, greater than, and so on (<, ==, >, !=, <=, >=).

If expression

In Kotlin, the if condition returns a value. Kotlin also doesn't have a ternary operator because if the condition can full fill all cases. See the below example-

```
var max = a
    if (a < b) max = b

// With else
    if (a > b) {
        max = a
    } else {
        max = b
    }

// As expression
max = if (a > b) a else b

// You can also use `else if` in expressions:

    val maxLimit = 1
    val maxOrLimit = if (maxLimit > a) maxLimit
else
    if (a > b) a else b
```

Although if expression can be used to give values to variables, else must always be used in that situation.

```
val max = if (a > b) {
    print("Choose a")
    a
} else {
    print("Choose b")
    b
}
```

When expression

Kotlin has a nicer way to write if / else if / else block which is "when", When defining a condition statement with multiple options, it is similar to a switch statement in another language. Developers can take various actions depending on the value of an expression thanks to the when expression.



Figure 4.3

The when expression has the following syntax:

```
when (expression) {  
    value1 -> action1  
    value2 -> action2  
    value3 -> action3  
    else -> defaultAction  
}
```

Here, the expression is assessed, and depending on its result, the appropriate action is carried out. The else block is invoked if none of the provided values match the expression.

The when expression can be applied to a variety of expression kinds, including characters, numbers, strings, enums, and even more complicated objects. Ranges, which can be used to match values inside a defined range, are also supported.

When compared to a switch statement, the when expression has the benefit of being able to return a value that can be included into other expressions. For instance:

```
val result = when (expression) {  
    value1 -> "Result 1"  
    value2 -> "Result 2"  
    else -> "Default result"  
}
```

In this instance, the result variable is given a value by the when expression. The when expression is a strong and adaptable construct overall. There are many additional things we may do in the when statement, some of which are listed here:

```
when (x) {  
    1 -> print(" 1 is correct ")  
    2,4 -> print (" 2 or 4 is there ")  
    in 5..10 -> print(" is between 5 to 10 ")  
    is String -> print("is a string")  
    x.isOdd() -> print("x is odd")  
    y.isEven() -> print("y is even")  
    s.toInt() -> print("s encodes x")  
    else -> false  
}
```

Nullability

The most common issue for any developer in earlier languages like Java was null. Because of nulls, programmers have committed a lot of errors, have problems in their code, and frequently experience crashes. These bugs have been overcome by the non-nullable variables.

What is Nullability?

In Kotlin, by default, any variable cannot be null except we explicitly say our compiler to make it nullable. That can be done using the question mark operator '?', see below

```
var a: String = "abc" // Regular initialization means non-null by default
a = null // compilation error
You can declare a variable as a string that can contain nulls by writing String?:
var b: String? = "abc" // can be set to null
b = null // ok
print(b)
```

you can allow complex data also to have null.

What are '?', '?.', '?:', '!!' operators ?

The question mark operator is used to declare variable as nullable. Now if we have declared something as nullable simply means before performing any operation on nullable, we have to perform checks. Let's see how we can perform checks on nullable.



Figure 4.4

1. Explicitly check -

The simple check for nullability using the if-else condition.

```
val l = if (b != null) b.length else -1
```

2. Safe calls or '?' operator -

The safe call operator ?. is your second choice for getting at a property on a nullable variable.

```
val a = "Kotlin"
val b: String? = null
println(b?.length)
println(a?.length) // Unnecessary safe call
```

Safe calls simply skip the operation if the object is null.

3. Elvis or '?:' operator -

The '?' the operator is sometimes called the "Elvis operator," because it's like a smiley on its side with a pompadour hairstyle, the way Elvis Presley styled his hair. Read more about the Elvis operator in the Kotlin documentation.

Kotlin is all about making things shorter and easier to read so as the Elvis operator does, use this operator in place of checking nullability with if-else like below

```
val l: Int = if (b != null) b.length else -1
with Elvis operator
val l = b?.length ?: -1
```

4. Double bang or "!!" operator -

The Double bang operator or '!!' is used if you sometimes want to throw a null pointer exception. The not-null assertion operator (!!) converts any value to a non-null type and throws an exception if the value is null. See the example below

```
val l = b!!.length
```

This option is for NPE aficionados.

Note - The double-bang operator is typically a bad idea, but it can be used if you use this with the legacy Java code.

Examine List, Array, and Loops

List

A list is a grouping of objects of a similar type. Lists in Kotlin can either be read-only (List) or changeable (MutableList) (List). A list in Kotlin is declared by using the `listOf` function. In Kotlin, you don't have to remake a list to modify or add an item to the list rather than you can simply make a mutable list using the `mutableListOf()` function. Use the `listOf()` and `mutableListOf()` functions from the standard library to create lists, depending on whether they are read-only or mutable.

```
//list
    val fruits = listOf("orange", "peer", "raspberry")           //1

//mutable list
    val myList = mutableListOf(1,2,3)                           //2
    fun addItem(item: Int)                                     //3
{
    myList.add(item)
}
    fun getItems(): List<Int>                                   //4
{
    return myList
}
    fun main() {
        println(fruits)
        addItem(4)                                             //5
        println("List size: ${getItems().size}")              //6
        getItems().forEach
    {
        i -> println("item $i")                                //7
    }
    // getSysSudoers().add(5) <- Error!                         //8
}
```

1. Creates a read-only view of the list.
2. Creates a mutable list.
3. Adds a new item to the Mutable List.
4. A function that returns an immutable List.
5. Updates the Mutable List. All related read-only views are updated as well since they point to the same object.
6. Retrieves the size of the read-only list.
7. Iterates the list and prints its elements.
8. Attempting to write to the read-only view causes a compilation error.

To access and manipulate list elements, you can also use a variety of available methods in the List. Typical techniques include:

Element index of list:-

A list has an index for each item. The index of the first entry is 0, and the index of the last element is `list.size - 1`, where size is the list's size.

```
fun main() {  
    val numbers = listOf(0,1,2,3,4,5,6,7,8,9)  
  
    val first_index = numbers.get(0)  
    println(first_index)  
  
    val middle_index = numbers[5]  
    println(middle_index)  
  
    val index1 = numbers.indexOf(1)  
    println("The first index of number is $index1")  
  
    val index2 = numbers.lastIndexOf(1)  
    println("The last index of number is $index2")  
  
    val index3 = numbers.lastIndex  
    println("The last index of the list is $index3")  
  
    println(numbers.first())  
  
    println(numbers.last())  
}
```

Output -

```
0  
5  
The first index of number is 1  
The last index of number is 1  
The last index of the list is 9
```

Sorting the elements in list :-

Let's see an example how we can sort a list in ascending or descending order.

```
fun main() {  
    val list = listOf(99, 55, 66, 11, 22, 44, 33, 88, 77 )  
  
    val list_in_asc = list.sorted()  
    println(list_in_asc)  
  
    val list_in_desc = list.sortedDescending()  
    println(list_in_desc)  
}
```

Output -

```
[11, 22, 33, 44, 55, 66, 77, 88, 99]  
[99, 88, 77, 66, 55, 44, 33, 22, 11]
```

Checking existence of element in list :-

In Kotlin there is `contains()` and `containsAll()` methods of list that can find the element existence over the list using its inner iteration, let's checkout these methods using example -

```
fun main() {  
    val list = listOf("ravi", "gaurav", "abhishek", "pravesh")  
    val res = list.contains("ravi")  
  
    if (res)  
        println("The list contains ravi")  
    else  
        println("The list does not contain ravi")  
  
    val result = list.containsAll(listOf("ravi", "gaurav"))  
  
    if (result)  
        println("The list contains ravi and gourav")  
    else  
        println("The list does not contain ravi and gourav")  
}
```

Output-

```
The list contains ravi  
The list contains ravi and gourav
```

Iterating List with different methods:-

Iterating list simply refers to go through each elements of list one by one. With Kotlin, there are various ways to accomplish this.

```
fun main() {  
    val latest_phones = listOf("OnePlus Nord CE2 Lite ", "Samsung  
Galaxy M33 ", "OnePlust 11R ", "Apple Iphone 14 " )  
  
    // method 1 that we seen previously  
    latest_phones.forEach { it -> print("$it, ") }  
    println()  
  
    // method 2  
    for (phones in latest_phones) {  
        print("$phones, ")  
    }  
    println()  
  
    // method 3  
    for (i in 0 until latest_phones.size) {  
        print("${latest_phones[i]} ")  
    }  
    println()  
  
    // method 4  
    latest_phones.forEachIndexed({ i, j -> println("phone[$i] = $j")})  
  
    // method 5  
    val it: ListIterator<String> = latest_phones.listIterator()  
    while (it.hasNext()) {  
        val i = it.next()  
        print("$i ")  
    }  
    println()  
}
```

Output -

```
OnePlus Nord CE2 Lite , Samsung Galaxy M33 , OnePlust 11R , Apple  
Iphone 14 ,  
OnePlus Nord CE2 Lite , Samsung Galaxy M33 , OnePlust 11R , Apple  
Iphone 14 ,  
OnePlus Nord CE2 Lite  Samsung Galaxy M33  OnePlust 11R  Apple Iphone  
14  
phone[0] = OnePlus Nord CE2 Lite  
phone[1] = Samsung Galaxy M33  
phone[2] = OnePlust 11R  
phone[3] = Apple Iphone 14  
OnePlus Nord CE2 Lite  Samsung Galaxy M33  OnePlust 11R  Apple Iphone  
14
```

Considering all things, lists in Kotlin offer a versatile and potent approach to store and manage groupings of objects.

Array

Array are basic data types of any language, Kotlin also has arrays. There is no way to modify the array in Kotlin, you simply must create a new copy after performing operations. The guidelines for using val and var with arrays are the same as those for lists.

The Array class in Kotlin is used to represent arrays. This class have get() and set() functions that turn into [] along with property like size. Let's take a look of the class how it's written on documentation:

```
class Array<T> private constructor() {
    val size: Int
    operator fun get(index: Int): T
    operator fun set(index: Int, value: T): Unit

    operator fun iterator(): Iterator<T>
    // ...
}
```

syntax -

```
val myArray = arrayOf(1, 2, 3)    //implicit type declaration
val myArray = arrayOf<String>(1, 2, 3) //explicit type declaration
```

Creating Array –

Use the arrayOf function to declare a string array. To print it out, use the java.util.Arrays.toString() array function.

```
val programming_lang = arrayOf("kotlin", "java", "python")
println(Arrays.toString(programming_lang))
```

Output -

```
["kotlin", "java", "python"]
```

Array constructor -

Since Array is a class in Kotlin, we can also use the Array constructor to create an array. The constructor requests two inputs: size and a function that accepts the index and return the value of the index.

```
import java.util.Arrays

fun main() {
    val num = Array(5, {i-> i*i})
    println(Arrays.toString(num))
}
```

Output -

```
[0, 1, 4, 9, 16]
```

You can mix types in an array declared with `arrayOf` since the elements don't have a type assigned to them. Create an array of several types.

```
val mix = arrayOf("mango", 2)
println(Arrays.toString(mix))
```

Although Kotlin also has classes that can represent arrays of primitive types without boxing overhead. Although these classes don't share any inheritance with the `Array` class, they do share a number of its methods and properties. Also, each of them has a matching factory function some of them are :

1. `intArrayOf()`
2. `byteArrayOf()`
3. `charArrayOf()`
4. `shortArrayOf()`
5. `longArrayOf()`

For example -

Use `intArrayOf` to declare an array of integers (). For arrays of other kinds, there are comparable builders or instantiation functions.

```
val numbers = intArrayOf(1,2,3)
println(Arrays.toString(numbers))
```

Output -

```
[1,2,3]
```

Accessing and modifying arrays –

Now after creating a array in kotlin let's see how we can access the array and how to modify them. As we have seen earlier the `Array` class in kotlin contains methods like `get()` and `set()`.

```
val hero = superheroes.get(0)
//or
val hero = superheroes[0]
//now the first element of superhero has been assigned in hero

superheroes.set(0, "Super Man")
//or
val superheroes[0] = "Super Man"
//now the first element of superhero is Super Man
```

Arrays traversing –

The ability to traverse an array programmatically and manipulate each element separately is a crucial feature of an array. Kotlin has a few efficient methods for traversing arrays.

Using a for-loop to traverse an array is the most straightforward and often used idiom.

Syntax of traverse using for-loop:

```
for(i in array.indices){  
    println(num[i])  
}
```

example -

```
fun main() {  
    val my_array = arrayOf<Int>(1, 2, 3, 4, 5)  
    for (i in my_array.indices)  
    {  
        println(my_array[i])  
    }  
}
```

Output -

```
1  
2  
3  
4  
5
```

Syntax of traverse using for-range :

```
for(i in first_index..last_index){  
    println(i)  
}
```

example -

```
fun main() {  
    val my_array = arrayOf<Int>(1, 2, 3, 4, 5)  
    for (i in 0..my_array.size-1)  
    {  
        println(my_array[i])  
    }  
}
```

Output -

```
1  
2  
3  
4  
5
```

So here is the mostly used less tedious way to iterate array : Foreach loop.

Syntax of traverse using for-each :

```
myarray.forEach( {index -> //statement } )
```

example -

```
fun main() {  
  
    val my_array = arrayOf<Int>(1, 2, 3, 4, 5)  
    my_array.forEach( { index -> println(index)} )  
}
```

Output -

```
1  
2  
3  
4  
5
```

The ability to start arrays using code rather than setting them to 0 is a good feature of Kotlin.
Consider the following illustration:

```
val array = Array (5) { (it * 2) }  
println(Arrays.toString(array))
```

Output -

```
[0,2,4,6,8]
```

Loops

After creating a list and array we can iterate it through loops, looping in kotlin are 3 types for, while, and do... while although More complex looping structures are also supported by Kotlin, including the forEach loop, which allows you to iterate over a collection using a lambda expression, and the repeat loop, which allows you to run a block of code a predetermined number of times.

Loops in



Figure 4.5

Like in most languages, For in Kotlin operates in the same way. Now let's see how for works -

```
val SuperHeros = listOf("Batman", "Thor", "Superman", "IronMan", "Hulk")
for ( hero in SuperHeros) {                                     // 1
    println("i am $hero! ready for the mission")
}
```

output -

```
i am Batman! ready for the mission
i am Thor! ready for the mission
i am Superman! ready for the mission
i am IronMan! ready for the mission
i am Hulk! ready for the mission
```


You can also loop through both indexes and elements in Kotlin like below,

```
for ( (index,hero) in superHeros.withIndex())
{
    // 1
    println("i am $hero! ready for the mission")
}
```

Other properties we can use while looping -
`for (i in 1..5) print(i)`
⇒ 12345

`for (i in 5 downTo 1) print(i)`
⇒ 54321

`for (i in 3..6 step 2) print(i)`
⇒ 35

`for (i in 'd'..'g') print (i)`
⇒ defg

Now let's see how while and do.. while works in Kotlin

```
fun main() {
    var apples = 0
    while (apples < 10) {
        apples++
    }
    println("$apples apples in the tree\n")
    do {
        apples--
    } while (apples > 10)
    println("$apples apples in the tree\n")
}
```

Output -

```
10 apples in the tree
9 apples in the tree
```

It's time to explore how Kotlin's Foreach & repeat functions works in kotlin.

The `forEach` loop is a higher-order function in Kotlin that may be used to repeatedly iterate over a group of elements and take a certain action on each element. It is a replacement for the `for`-loop and is frequently used with lambda expressions to make programming simpler.

```
SuperHeros.forEach { hero ->
    println(element)
}
```

Output –

```
Batman
Thor
Superman
IronMan
Hulk
```

The `repeat` loop in Kotlin is a particular kind of loop that runs a block of code a predetermined number of times. Similar to a `for` loop, but without the need to explicitly specify a range or collection, it iterates across a set of values.

```
repeat(2) {
    println("i plus an apple from tree")
}
```

Output –

```
i plug an apple from tree
i plug an apple from tree
```

Questions



Figure 4.6

Take a quiz on this chapter here:

Question #1: which variable is changeable val or var?

Question #2: In Kotlin, can we assign null to any custom class object?

Question #3: what will be the output of the following code? `for (i in 1..5 step 2) println(i)`

Question #4: what will be the output of the following code? `for (i in s..z step 2) println(i)`

Question #5: `for (i in 'd' .. 'm' step 2) print (i)`

Summary

This E-book explained Kotlin from scratch. Since its first release, it has gained popularity in recent years, especially for Android app development. Kotlin is fully interoperable with Java and can be used for a wide variety of tasks, including server-side development, web development, desktop applications, and more. Its syntax is concise and intuitive, making it easy to learn and use, especially for Java developers. Additionally, Kotlin's advanced type system can catch certain kinds of errors at compile-time, making it a safer language than Java. We also learned about types, operators, variables, Booleans, conditions, nullability, arrays, and loops in this E-book. Basic Kotlin learning concepts are similar to learning other languages. The distinctive features include the inability of Kotlin to cast or implicitly covert types. Kotlin has an advantage in finding the type at runtime for variables like `val` and `var` and not nullability.

So why should you learn Kotlin? Here are the points it's easy to learn, it's versatile, it's safe, it's actively developed and supported, and It's in demand. Overall, Kotlin is a great language to learn, whether you're a seasoned developer or someone who's just getting started with programming. So why not give it a try? You might just discover a new favorite language!

Note – To know the solutions of the question in the E-book [visit here](#).



OUR MISSION

Free Education is Our Basic Need! Our mission is to empower millions of developers worldwide by providing the latest unbiased news, advice, and tools for learning, sharing, and career growth. We're passionate about nurturing the next young generation and help them not only to become great programmers, but also exceptional human beings.

ABOUT US

CSharp Inc, headquartered in Philadelphia, PA, is an online global community of software developers. C# Corner served 29.4 million visitors in year 2022. We publish the latest news and articles on cutting-edge software development topics. Developers share their knowledge and connect via content, forums, and chapters. Thousands of members benefit from our monthly events, webinars, and conferences. All conferences are managed under Global Tech Conferences, a CSharp Inc sister company. We also provide tools for career growth such as career advice, resume writing, training, certifications, books and white-papers, and videos. We also connect developers with their potential employers via our Job board. Visit [C# Corner](#)

MORE BOOKS

