



# Kotlin Basics

A New Way To Develop

Mustafif Khan



Copyright © 2021 Mustafif Khan

PUBLISHED BY MKPROJECTS


MKPROJ.COM

Licensed under the Creative Commons Attribution 4.0 International License (the “License”). You may not use this file except in compliance with the License. You may obtain a copy of the License at <https://creativecommons.org/licenses/by/4.0/>. Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

# Contents

<b>1</b>	<b>Getting Started</b> .....	<b>5</b>
1.1	Installation	5
1.2	Compile	6
<b>2</b>	<b>Hello World Kotlin!</b> .....	<b>7</b>
2.1	Main Function	7
2.2	Comments	7
2.3	Print Statements	8
<b>3</b>	<b>Data Types</b> .....	<b>9</b>
3.1	Mutable and Immutable Variables	10
3.1.1	Type Inference .....	10
3.2	Strings	10
3.3	Math	11
<b>4</b>	<b>Conditional Expressions</b> .....	<b>13</b>
4.1	If Statements	13
4.2	Else Statements	14
4.3	Else If Statements	14
4.4	Nested Conditionals	14
4.5	When Statements	15
4.6	Comparison Operators	15
4.7	Logic Operators	15

<b>4.8</b>	<b>The Range Operator</b>	<b>16</b>
<b>5</b>	<b>Collections .....</b>	<b>17</b>
<b>5.1</b>	<b>Lists</b>	<b>17</b>
5.1.1	Mutable and Immutable Lists .....	17
5.1.2	List Operations .....	18
<b>5.2</b>	<b>Sets</b>	<b>18</b>
5.2.1	Immutable and Mutable Sets .....	18
<b>5.3</b>	<b>Maps</b>	<b>19</b>
5.3.1	Immutable and Mutable Maps .....	19



# 1. Getting Started

Kotlin is a statically typed, general purpose programming language developed by JetBrains. Kotlin is designed to work alongside Java, and due to its type inference, its code can be a lot more concise and easier to read.

## 1.1 Installation

You can install the Kotlin Compiler manually from its Github Releases  
<https://github.com/JetBrains/kotlin/releases/tag/v1.4.32>

### MacOS Homebrew

To install on Mac OS, you may also choose to install the Kotlin compiler via Homebrew:

```
$ brew update  
$ brew install kotlin
```

### Linux Snap

To install on a Linux Distribution, you may install via Snap:

```
$ sudo snap install --classic kotlin
```

## 1.2 Compile

To Compile a Kotlin /kt program, you must first compile it using the Kotlin Compiler to produce a Java !.jar! file:

```
# We will be compile an example test.kt program
$ kotlinc hello.kt -include-runtime -d hello.jar
# -include-runtime makes the jar file self-contained and runnable
# -d is the output directory option followed by the output file
# Run kotlinc -help for more options
```

To run the program use java:

```
$ java -jar hello.jar
```

Now that you know how to compile and run the program, let's proceed to the next section!



## 2. Hello World Kotlin!

To explain this section, we will be going line by line of the different parts of our program:

```
// examples/01-hello.kt
fun main(){
    // Our program is very simple and is aimed to say hello world
    /*
    We will run each word of "Hello World Kotlin!" in separate println
    so we can show the order of execution.
    */
    println("Hello")
    println("World")
    println("Kotlin!")
}
```

### 2.1 Main Function

You can see at the beginning of our program is `fun main(){}` , the main function is the starting point of which the Kotlin Program executes the code. For an executable program, a main function is required for the Kotlin Program.

### 2.2 Comments

Comments are used for the programmer and not for the actual execution of the program. It is typically used to document and jot ideas down on the program, and is ignored by the compiler. There is two different type of comments, there is `//` which is a single line comment and `/* */` which is multi-line comments that contains comments in between `/*` and `*/`.

## 2.3 Print Statements

A print statement is a way to output values onto the console, and there is two different print statements that exist.

- `println()` : This will create a new line after outputting
- `print()` : This will only output (**not create a new line**)

Now let's Compile and Run:

```
# If using this repo, cd examples
$ kotlinc 01-hello.kt -include-runtime -d 01-hello.jar
$ ls
01-hello.jar  01-hello.kt
$ java -jar 01-hello.jar
Hello
World
Kotlin!
```



### 3. Data Types

We will begin this section off by dissecting this example program that contains most topics that will be discussed:

```
fun main(){
    /*
        Our goal is to be able to have variables that
        calculate various geometric areas
    */

    // An immutable variable for the value of pi
    val pi = 3.14159

    var radius = 10.0
    var area = pi * radius // Circle

    //Since we will be using the area statement a lot
    //it's good to make it a variables

    var stm = "The area of this "
    var shape = "circle"

    println(stm + shape + " is $area") // String Concetation + Template

    // Now let's say we want to find the area and volume of a square

    var length = 56.21
    area = Math.pow(length, 2.00) // Will square the length
    var volume = Math.pow(length, 3.00) // Will cube the length
```

```

    // Now to prepare this statement
    // However I want each beginning letter of each word capitalized

    shape = "square"

    stm = stm + shape + " is $area" +
    "\nThe volume of this " + shape + " is $volume"

    println(stm.capitalize())
}

```

### 3.1 Mutable and Immutable Variables

Variables are used in a program to store data, and in Kotlin there is two different ways to declare a variable, one being `var` which is for mutable variables, and the other `val` being for immutable. Now what is the difference between mutable and immutable, well the difference is between being able to be reassigned. A mutable variable is a variable that can be reassigned any amount of times in a program, while a immutable cannot be reassigned a value once it is given one.

You can see this in the program in the use case of `val pi = 3.14159` where we don't want `pi` to ever change. You can see a few of our variables were reassigned, such as `shape`, `area`, etc.

#### 3.1.1 Type Inference

If you're coming from Java to Kotlin, you must've been being so confused by the no semi colons, or the fact that when you declare a variable, you don't need to provide a type with it. This is what Kotlin means with type inference, where the Kotlin compiler at compiling time can infer what data type each variable is.

A nice way to look at Kotlin's type inference is to compare Kotlin and Java:

Java	Kotlin
<code>double num_double = 67.89;</code>	<code>var num_double = 67.89</code>

In Java a data type must be declared explicitly while in Kotlin, the type is implicitly inferred.

### 3.2 Strings

Strings are essentially an array of characters, and in Java is not considered to be a primitive type, however, Strings do have useful builtin properties that are good to take advantage of.

#### String Concatenation

String Concatenation is to combine Strings together with the `+` operator, and as you can see, it was useful for formatting our statement with the various variables we had with us.

Ex.

```

stm = stm + shape + " is $area" +
"\nThe volume of this " + shape + " is $volume"

```

#### String Templates

String Templates, which is also used in our example is useful to have variables in a String with `$variable` inside the String to cleanly write the strings without needing to concatenate a lot.

### String Builtin Properties

In the ending of the program, we use a builtin String function called `.capitalize()` which served our purpose to capitalize each word in a String. There is also `.length()` which returns the number of characters in a String. It is a good idea to explore the various String properties that Kotlin offers.

### Character Escape Sequences

Character escape sequences consist of a backslash and character and are used to format text.

- `\n` Inserts a new line
- `\t` Inserts a tab
- `\r` Inserts a carriage return
- `\'` Inserts a single quote
- `\"` Inserts a double quote
- `\\` Inserts a backslash
- `\$` Inserts a dollar symbol

## 3.3 Math

### Arithmetic Operators

Kotlin includes the following arithmetic operators:

- `+` addition
- `-` subtraction
- `*` multiplication
- `/` division
- `%` modulus

### Order of Operations

The order of operations for compound arithmetic expressions is as follows:

1. Parentheses
2. Multiplication
3. Division
4. Modulus
5. Addition
6. Subtraction

When an expression contains operations such as multiplication and division or addition and subtraction side by side, the compiler will evaluate the expression in a left to right order.

### Augmented Assignment Operators

An augmented assignment operator includes a single arithmetic and assignment operator used to calculate and reassign a value in one step.

```
var a = 10
var b = 7

a += b // 10 + 7
a -= b // 17 - 7
a *= b // 10 * 7
a /= b // 70 / 7
a %= b // 10 % 7
```

### Increment and Decrement Operators

Increment and decrement operators provide a shorthand syntax for adding or subtracting 1 from a value. An increment operator consists of two consecutive plus symbols, ++, meanwhile a decrement operator consists of two consecutive minus symbols, --.

```
var num = 78
```

```
num++ // 79
```

```
num-- // 78
```

*The Increment Operator is commonly used in loops (Chapter 6)*

### The Math Library

The Math library, inherited from Java, contains various mathematical functions that can be used within a Kotlin program.

```
Math.pow(2.0, 3.0) // 8.0
```

```
Math.min(6, 9) // 6
```

```
Math.max(10, 12) // 12
```

```
Math.round(13.7) // 14
```

## 4. Conditional Expressions

### 4.1 If Statements

An if statement is used to execute a section of code if an expression results to true. A great way to look at this is an example:

```
var a = 5

if (a == 5) {
    println("True")
}
```

Since a is indeed equal to 5, then the program will execute `println("True")` however, let's say we had...

```
var a = 5

if (a == 6) {
    println("True")
}
```

Since a doesn't equal to 6, the expression results to false, so the program doesn't print anything.

## 4.2 Else Statements

An else statement is used to execute a section of code if an expression results to false. Let's fix our previous example a bit to include an else statement:

```
var a = 5

if (a == 6) {
    println("True")
} else {
    println("False")
}
```

Since a doesn't equal to 6, it follows the else statement, and executes `println("False")`.

## 4.3 Else If Statements

An else if expression allows for more conditions to be evaluated within an if/else expression.

**Note:** You can use multiple else if expressions as long as they appear after the if expression and before the else expression.

```
var num = 8

if (num > 8){
    println(num)
} else if (num == 8){
    println(num++)
} else {
    println(num--)
}
// Prints 9
```

## 4.4 Nested Conditionals

A nested conditional is a conditional statement within another conditional statement. Essentially an if statement inside an if statement.

```
var foo = true
var bar = true

if (foo){
    println("FOO!")
    if (bar){
        println("BAR!")
    }
} else {
    println("No Foo or Bar for you")
}
```

```
/*
Prints:
FOO!
BAR!
*/
```

## 4.5 When Statements

After a while you may get sick of a huge tree of `else if` statements, so any better option? The `when` statement is useful to execute code depending on the value of the expression.

```
var grade = "F"

when (grade){
    "A" -> println("Excellent")
    "B" -> println("Okay")
    "C" -> println("Bad")
    "D" -> println("Terrible")
    else -> println("FAIL")
}
// Prints FAIL
```

## 4.6 Comparison Operators

Comparison operators are symbols that are used to compare two values in order to return a result of `true` or `false`.

The comparison operators include the following:

- `<` less than
- `>` greater than
- `<=` less than or equal to
- `>=` greater than or equal to

```
var Khloe = 19
var Zane = 18
var Nimu = 20

Khloe > Zane //true
Zane < Nimu // true
Khloe >= Nimu // false
Nimu <= Zane // false
```

## 4.7 Logic Operators

Logical operators are symbols used to evaluate the relationship between two or more Boolean expressions in order to return a `true` or `false` value.

The logic operators include the following:

- `!` NOT
- `&&` AND
- `||` OR

**"!" NOT Operator**

The NOT operator evaluates a boolean expression and return it's opposite value (true -> false).

```
var exp = true
println(!exp)
// Prints false
```

**"&&" AND Operator**

The AND operator only evaluates to true if both expressions it's evaluating results to true.

```
if (4 < 5 && 6 > 3){ //Both expressions are true
    println("True")
} else {
    println("False")
}
// Prints true
```

**"||" OR Operator**

The OR operator only evaluates to true if one of the expressions result to true.

```
if (1.2 < 9.8 || 5 > 9){ // One results to true
    println("True")
} else {
    println("False")
}
//Prints True
```

**Order of Evaluation**

The order of evaluation when using multiple logical operators in a single Boolean expression is:

1. Expressions placed in parentheses
2. NOT(!) Operator
3. AND(&&) Operator
4. OR(||) Operator

Ex.

```
!true || (true && false) // false
```

**4.8 The Range Operator**

The range operator (.. ) is used to create a succession of number or character values.

```
var grades = 90

when (grades){
    90..101 -> println("A+")
    80..90 -> println("A")
    .
    .
    .
    else -> println("F")
}
// Prints A+
```



## 5. Collections

In Kotlin, there is various type of collections that exist for different case usages. In this section, we will be covering Lists, Sets and Maps.

### 5.1 Lists

#### 5.1.1 Mutable and Immutable Lists

- An immutable list represents a group of elements with read-only operations.
  - It can be declared with the term `listOf`, followed by a pair of parentheses containing elements that are separated by commas.
  - Ex. `var fruits = listOf("Apples", "Bananas", "Oranges")`
- A mutable list represents a group of ordered elements with read and write operations.
  - It can be declared with the term, `mutableListOf` followed by a pair of parentheses containing elements that are separated by commas.
  - Ex. `var companies = mutableListOf("Google", "Tesla", "Apple")`

#### Accessing Lists Elements

In order to retrieve an element from a list, we can reference its numerical position or index using square bracket notation `[ ]`.

**Note:** The beginning index of a list starts at 0.

#### The Size Property

The `size` property returns the size of a collection of the number of elements that exists.

```
var grades = listOf("A", "B", "C", "D", "F")
println(grades.size)
// Prints 5
```

### 5.1.2 List Operations

The list collection supports various operations in the form of built-in functions that can be performed on its elements.

Some functions perform read and write operations, whereas others perform read-only operations.

The functions that perform read and write operations can only be used on mutable lists while read-only operations can be performed on both mutable and immutable lists.

```
var programming_languages = mutableListOf("Kotlin", "Java", "C++", "Rust")

if (programming_languages.contains("Python") /*read only*/){
    println("Python is pretty cool")
} else {
    programming_languages.add("Python") //write
}
```

## 5.2 Sets

### 5.2.1 Immutable and Mutable Sets

- An immutable set represents a collection of unique elements in an unordered format whose elements cannot be changed throughout a program.
  - It is declared with the term, `setOf`, followed by a pair of parentheses, ( ) holding unique values.
  - `var origin_factions = setOf("Mythicals", "Sorcerers", "Kindgom")`
- A mutable set represents a collection of ordered elements that possess both read and write functionalities.
  - It is declared with the term, `mutableSetOf`, followed by a pair of parantheses, ( ) holding unique values.
  - `var mkproj = mutableSetOf("Phaktionz", "Books", "UniConv", "Moka")`

#### Accessing Set Elements

Elements in a set can be accessed using the `elementAt()` or `elementAtOrNull()` functions.

- The `elementAt()` function gets appended onto a set name and returns the element at the specified position within the parentheses.
- The `elementAtOrNull()` function is a safer variation of the `elementAt()` function
  - Returns null if the position is out of bounds as opposed to throwing an error.

```
var example = setOf("Foo", "Bar", "Baz")

println(example.elementAt(1)) // Prints Bar
println(example.elementAt(5)) // Returns an error
println(example.elementAtOrNull(5)) // Prints null
```

## 5.3 Maps

### 5.3.1 Immutable and Mutable Maps

- An immutable Map represents a collection of entries that cannot be altered throughout a program.
  - It is declared with the term, `mapOf`, followed by a pair of parentheses.
    - \* Within the parentheses, each key should be linked to its corresponding value with the `to` keyword, and each entry should be separated by a comma.
    - \* `var student = mapOf("Johnny" to 95, "Billy" to 65, "Kimmy" to 85)`
- A mutable map represents a collection of entries that possess read and write functionalities. Entries can be added, removed, or updated in a mutable map.
  - A mutable map can be declared with the term, `mutableMapOf`, followed by a pair of parentheses holding key-value pairs.
  - `var prices = mutableMapOf("Gum" to 1.50, "Kitkat" to 0.88, "Spinach" to 3.97)`