What is Kotlin?

Kotlin is a modern, trending programming language that was released in 2016 by JetBrains.

It has become very popular since it is compatible with [Java](https://www.w3schools.com/java/default.asp) (one of the most popular programming languages out there), which means that Java code (and libraries) can be used in Kotlin programs.

Kotlin is used for:

* Mobile applications (specially Android apps)
* Web development
* Server side applications
* Data science
* And much, much more!

Why Use Kotlin?

* Kotlin is fully compatible with Java
* Kotlin works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc.)
* Kotlin is concise and safe
* Kotlin is easy to learn, especially if you already know Java
* Kotlin is free to use
* Big community/support

## Kotlin Syntax

In the previous chapter, we created a Kotlin file called Main.kt, and we used the following code to print "Hello World" to the screen:

**Example**

fun main() {

println("Hello World")

}

### Example explained

The fun keyword is used to declare a function. A function is a block of code designed to perform a particular task. In the example above, it declares the main() function.

The main() function is something you will see in every Kotlin program. This function is used to **execute** code. Any code inside the main() function's curly brackets {} will be **executed**.

For example, the println() function is inside the main() function, meaning that this will be executed. The println() function is used to output/print text, and in our example it will output "Hello World".

**Good To Know:** In Kotlin, code statements do not have to end with a semicolon (;)

# Kotlin Variable

Variable refers to a memory location. It is used to store data. The data of variable can be changed and reused depending on condition or on information passed to the program.

## Variable Declaration

Kotlin variable is declared using keyword **var** and **val**.

1. var language ="Java"
2. val salary = 30000

The difference between var and val is specified later on this page.

Here, variable language is String type and variable salary is Int type. We don't require specifying the type of variable explicitly. Kotlin complier knows this by initilizer expression ("Java" is a String and 30000 is an Int value). This is called type inference in programming.

We can also explicitly specify the type of variable while declaring it.

1. var language: String ="Java"
2. val salary: Int = 30000

It is not necessary to initialize variable at the time of its declaration. Variable can be initialized later on when the program is executed.

1. var language: String
2. ... ... ...
3. language = "Java"
4. val salary: Int
5. ... ... ...
6. salary = 30000

Difference between var and val

* **var** (Mutable variable): We can change the value of variable declared using **var** keyword later in the program.
* **val** (Immutable variable): We cannot change the value of variable which is declared using **val** keyword.

**Example**

1. var salary = 30000
2. salary = 40000 //execute

Here, the value of variable salary can be changed (from 30000 to 40000) because variable salary is declared using **var** keyword.

1. val language = "Java"
2. language = "Kotlin" //Error

Here, we cannot re-assign the variable language from "Java" to "Kotlin" because the variable is declared using **val** keyword.

# Kotlin Data Type

**Data type** (basic type) refers to type and size of data associated with variables and functions. Data type is used for declaration of memory location of variable which determines the features of data.

In Kotlin, everything is an object, which means we can call member function and properties on any variable.

Kotlin built in data type are categorized as following different categories:

* Number
* Character
* Boolean
* Array
* String

## Number Types

Number types of data are those which hold only number type data variables. It is further categorized into different Integer and Floating point.

|  |  |  |
| --- | --- | --- |
| **Data Type** | **Bit Width (Size)** | **Data Range** |
| Byte | 8 bit | -128 to 127 |
| Short | 16 bit | -32768 to 32767 |
| Int | 32 bit | -2,147,483,648 to 2,147,483,647 |
| Long | 64 bit | -9,223,372,036,854,775,808 to +9,223,372,036,854,775,807 |
| Float | 32 bit | 1.40129846432481707e-45 to 3.40282346638528860e+38 |
| Double | 64 bit | 4.94065645841246544e-324 to 1.79769313486231570e+308 |

## Character (Char) Data Type

Characters are represented using the keyword **Char**. Char types are declared using single quotes ('').

|  |  |  |
| --- | --- | --- |
| **Data Type** | **Bit Width (Size)** | **Data Range** |
| Char | 4 bit | -128 to 127 |

**Example**

1. val value1 = 'A'
2. //or
3. val  value2: Char
4. value2= 'A'

## Boolean Data Types

Boolean data is represented using the type **Boolean**. It contains values either true or false.

|  |  |  |
| --- | --- | --- |
| **Data Type** | **Bit Width (Size)** | **Data Value** |
| Boolean | 1 bit | true or false |

**Example**

1. val flag = **true**

## Array

Arrays in Kotlin are represented by the Array class. Arrays are created using library function arrayOf() and Array() constructor. Array has get (), set() function, size property as well as some other useful member functions.

### Creating Array using library function arrayOf()

The arrayOf() function creates array of wrapper types. The item value are passed inside arrayOf() function like arrayOf(1,2,3) which creates an array[1,2,3].

The elements of array are accessed through their index values (array[index]). Array index are start from zero.

1. val id = arrayOf(1,2,3,4,5)
2. val firstId = id[0]
3. val lasted = id[id.size-1]

### Creating Array using Array() constructor

Creating array using Array() constructor takes two arguments in Array() constructor:

1. First argument as a size of array, and
2. Second argument as the function, which is used to initialize and return the value of array element given its index.
3. val asc = Array(5, { i -> i \* 2 }) //asc[0,2,4,6,8]

## String

String in Kotlin is represented by String class. String is immutable, which means we cannot change the elements in String.

**String declaration:**

1. val text ="Hello, JavaTpoint"

### Types of String

String are categorize into two types. These are:

1. **Escaped String:**Escape String is declared within double quote (" ") and may contain escape characters like '\n', '\t', '\b' etc.

1. val text1 ="Hello, JavaTpoint"
2. //or
3. val text2 ="Hello, JavaTpoint\n"
4. //or
5. val text3 ="Hello, \nJavaTpoint"

2. **Raw String:**Row String is declared within triple quote (""" """). It provides facility to declare String in new lines and contain multiple lines. Row String cannot contain any escape character.

1. val text1 ="""
2. Welcome
3. To
4. JavaTpoint
5. """

# Kotlin Type Conversion

Type conversion is a process in which one data type variable is converted into another data type. In Kotlin, implicit conversion of smaller data type into larger data type is not supported (as it supports in java). For example Int cannot be assigned into Long or Double.

## In Java

1. **int** value1 = 10;
2. **long** value2 = value1;  //Valid code

## In Kotlin

1. var value1 = 10
2. val value2: Long = value1  //Compile error, type mismatch

However in Kotlin, conversion is done by explicit in which smaller data type is converted into larger data type and vice-versa. This is done by using helper function.

1. var value1 = 10
2. val value2: Long = value1.toLong()

The list of helper functions used for numeric conversion in Kotlin is given below:

* toByte()
* toShort()
* toInt()
* toLong()
* toFloat()
* toDouble()
* toChar()

### Kotlin Type Conversion Example

Let see an example to convert from Int to Long.

1. **fun** main(args : Array<String>) {
2. var value1 = 100
3. val value2: Long =value1.toLong()
4. println(value2)
5. }

We can also converse from larger data type to smaller data type.

1. **fun** main(args : Array<String>) {
2. var value1: Long = 200
3. val value2: Int =value1.toInt()
4. println(value2)
5. }

What you'll **do**

* Create a new Android project and a default app called HelloWorld.
* Create an emulator (a virtual device) so you can run your app on your computer.
* Run the HelloWorld app on virtual and physical devices.
* Explore the project layout.
* Explore the AndroidManifest.xml file.