Kotlin Data Types Fully Explained

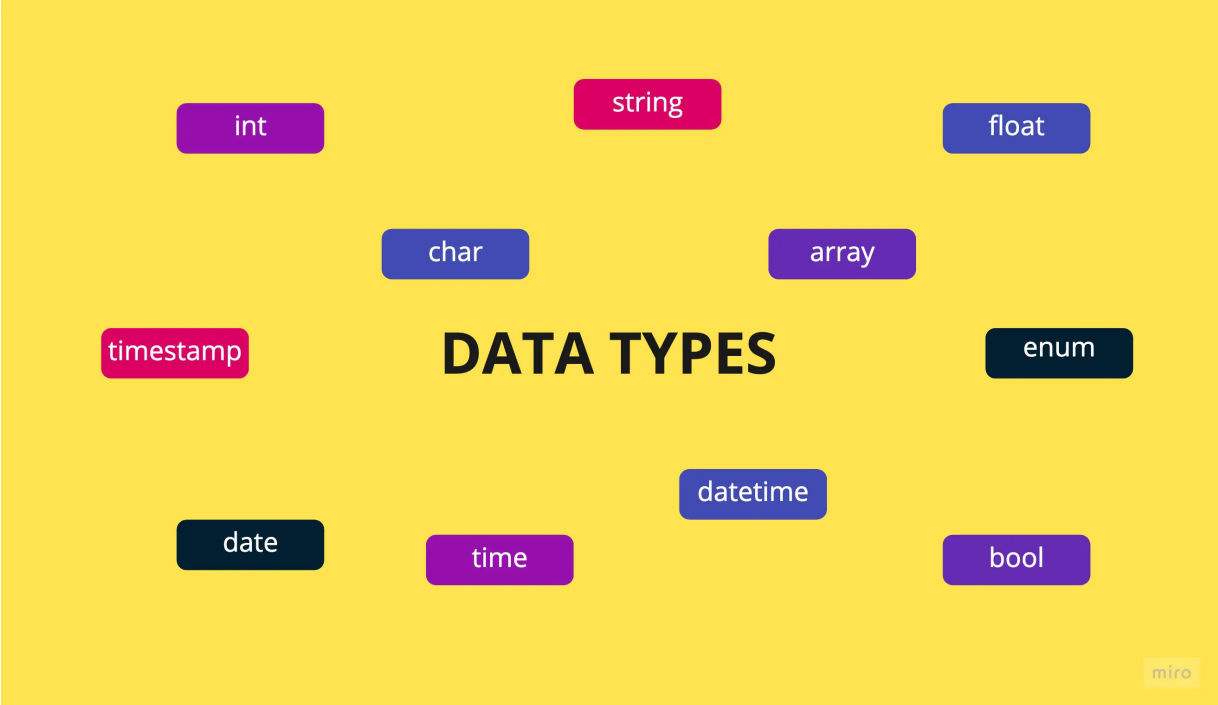
Ronnie Atuhaire

Sep 2, 2021 • 10 min read

PLAY THIS ARTICLE

0:00 / 10:02

What is a Data Type ❓

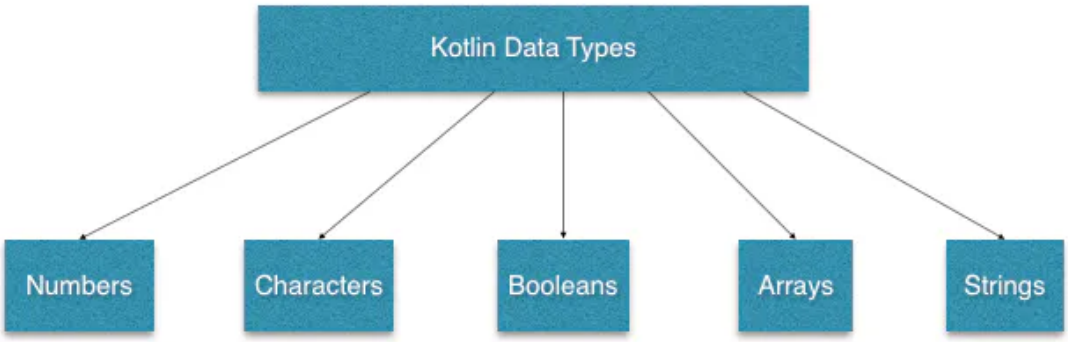


A data type is a type of data. Of course, that is rather a circular definition, and also not very helpful. Therefore, a better definition of a data type is a data storage format that can contain a specific type or range of values according to Tech Terms.

Some programming languages require the programmer to define the data type of a variable before assigning it a value. Other languages can automatically assign a variable's data type when the initial data is entered into the variable.

As we saw in the previous **article**, we may not need to assign the data type since Kotlin is smart enough to identify it for us. However, there might be some special cases where you want a specific data type may be from the user or so.

The most fundamental data type in Kotlin is the Primitive data type and all others are reference types like arrays and strings.



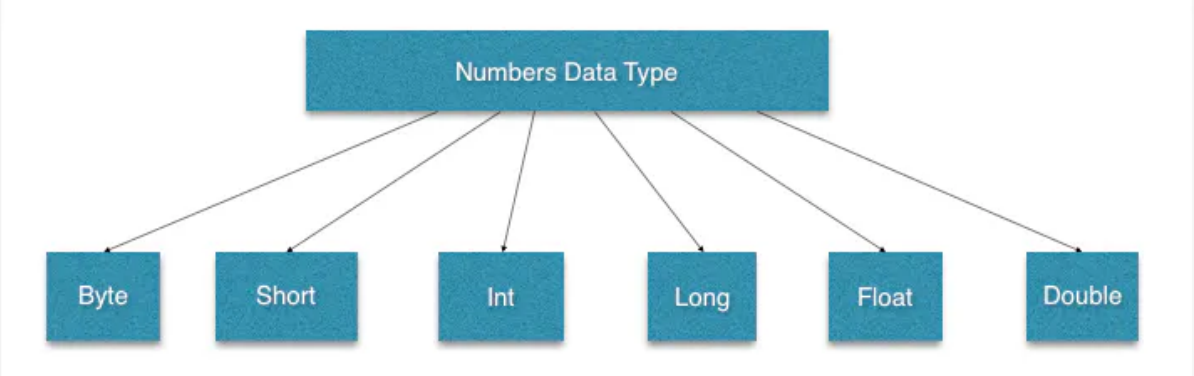
COPY

```
val myNum = 5              // Int ->representing Numbers
val myDoubleNum = 5.99     // Double
val myLetterCharac
```

```
val myBoolean = true       // Boolean
val myText = "Hello"       // String
```

The above is quite similar compared to Python language with an exception of character. Python does not have a character type. All single characters are strings with length one. We shall also learn about strings, arrays, ranges and collections in this tutorial.

- Numbers



Numbers in Kotlin are similar to Python, but not exactly the same. Number types are divided into two groups:

Integer types store whole numbers, positive or negative (such as 123 or -456), without decimals. Valid types are *Byte, Short, Int and Long*.

Floating-point types represent numbers with a fractional part, containing one or more decimals. There are two types: *Float and Double*.

Table showing the different bit widths of Numbers.

TYPE	BIT WIDTH
Byte	8
Short	16
Int	32
Long	64
Float	32
Double	64

Integer Types

Byte

The Byte data type can store whole numbers from -128 to 127. This can be used instead of Int or other integer types to save memory when you are certain that the value will be within -128 and 127:

COPY

```
val myByteNum: Byte = 90
```

Short

The Short data type can store whole numbers from -32768 to 32767:

COPY

```
val myShortNum: Short = 5000
```

Shorts and Bytes are good for memory saving.

Int

The Int data type can s -2147483648 to 2147483647:

COPY

```
val myIntNum: Int = 100000
```

Long

The Long data type can store whole numbers from -9223372036854775808 to 9223372036854775808. This is used when Int is not large enough to store the value. Optionally, you can end the value with an "L":

COPY

```
val myLongNum: Long = 15000000000L
```

Int VS Long

A whole number is an Int as long as it is up to 2147483647. If it goes beyond that, it is defined as Long:

COPY

```
val myIntNum = 2147483647  // Int
val myLongNum = 2147483648  // Long
```

Floating Point Types

Floating-point types represent numbers with a decimal, such as 9.99 or 3.14515. If you assign a floating-point value to a variable and do not declare the data type explicitly, it is inferred as a Double data type variable by the compiler.

Float

The Float data type can store fractional numbers from 3.4e−038 to 3.4e+038. Note that y h an "F":

COPY

COPY

```
val myFloatNum: Float = 5.75F
```

Double

The Double data type can store fractional numbers from 1.7e−308 to 1.7e+038:

COPY

```
val myDoubleNum: Double = 19.99
```

Float or Double?

The precision of a floating-point value indicates how many digits the value can have after the decimal point. The precision of Float is only six or seven decimal digits, while Double variables have a precision of about 15 digits. Therefore it is safer to use Double for most calculations.

Scientific Numbers

A floating-point number can also be a scientific number with an "e" or "E" to indicate the power of 10: Try running this >>>

COPY

```
fun main() {
  val myNum1: Float = 535E3F // Expected Output -> 535000.0
  val myNum2: Double = 612E4 // Expected Output -> 6120000.0
  println(myNum1)
  println(myNum2)
}
```

If you want to declare a variable that can take any value it is assigned, you should declare its type as Number. In this way, you can assign any value (Integer, float or double) to it.

COPY

```
fun main() {

    var numb: Number = 12.2
    println("$numb")

    numb = 12.4F
    // Float smart cast from Number
    println("$numb")

    numb = 12
    // Int smart cast from Number
    println("$numb")

    numb = 120L
    // Long smart cast from Number
    println("$numb")
}
```

We can call this Number Casting 😜 and the $ helps place the variable in the quotes.

Notes on Numbers

📍 Numbers are boxed when a nullable reference is needed. Identity is not preserved by the boxing operation.

📍 Due to different representations of Number types, smaller types are not sub-types of bigger type

📍 Every number type versions:

COPY

```
toByte(): Byte
toShort(): Short
toInt(): Int
toLong(): Long
toFloat(): Float
toDouble(): Double
toChar(): Char
```

📍 Kotlin supports the standard sets of arithmetical operations over numbers.
📍 Complete list of bitwise operations (available for Int and Long only) that you can perform in Numbers and we shall tackle this when doing operators.

Literal constants
A literal constant, or simply a literal, is a value, such as a number, character, or string that may be assigned to a variable or symbolic constant, used as an operand in arithmetic or logical operation, or as a parameter to a function.

There are the following kinds of literal constants for integral values:

- [ ] Decimals: `123`
- [ ] Longs are tagged by a capital `L`: `123L`
- [ ] Hexadecimals: `0x0F`
- [ ] Binaries: `0b00001011`

Unsigned integers
In addition to integer types, Kotlin provides the following types for unsigned integer numbers:

→ `UByte` : an unsigned [redacted] to 255
→ `UShort` : an unsigned 16-bit integer, ranges from 0 to 65535

→ `UInt` : an unsigned 32-bit integer, ranges from 0 to 2^32 - 1

→ `ULong` : an unsigned 64-bit integer, ranges from 0 to 2^64 - 1

Unsigned types support most of the operations of their signed counterparts.

Booleans

Just like Python, the Boolean data type and can only take the values true or false: Note that in Python the first T & F letters are capitalised.

COPY

```
val isKotlinFun: Boolean = true
val isPythonBad: Boolean = false
```

Boolean values are mostly used for conditional testing, which you will learn more about in a later chapter.

Built-in operations on boolean include →

COPY

```
|| // -> lazy disjunction
&& // -> lazy conjunction
!  // -> negation
```

Characters

The Char data type is used to store a single character. A char value must be surrounded by single quotes, like 'A' or 'c':

COPY

```
val myChar = 'H' //
val myLogo = "H" //
```

```
val myBlog = "Hashnode" // String
val myExtract = blog[0] // Char -> H
```

Python does not have a character or char type. All single characters are strings with length one.

🔹 Strings

The String data type is used to store a sequence of characters (text).

In Python, strings are arrays of bytes representing Unicode characters. We can write Python strings (str) with either single or double quotes. It does not really matter however, in Kotlin, string values must be surrounded by double quotes:

- Python String Type: `str`
- Kotlin String Type: `String`

Kotlin String Examples

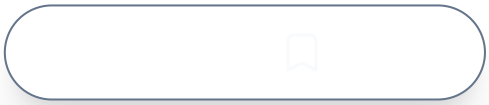
COPY

```
val myLogo = "H" // String
val myBlog = "Hashnode" // String
val myOtherBlog = 'Ronnie Hashnode' // -> Error and Not a string
```

You can also escape strings in Kotlin the same way as Python. Try this:

COPY

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



We also have raw strings in Kotlin which are exactly output in the format presented. Try:

COPY

```
val rawString = """
Kotlin For
Python
Developers
"""
```

- Data Collections

According to **Techopedia**, a collection is a class used to represent a set of similar data type items as a single unit. These unit classes are used for grouping and managing related objects.

A collection has an underlying data structure that is used for efficient data manipulation and storage.

Arrays & Lists

Arrays are used to store multiple values in a single variable, instead of declaring separate va s in Kotlin have a

constant length, so one normally uses lists, which are similar to the ones in Python.

In Kotlin, we use `arrayOf()` to construct an array/list with a similar collection.

COPY

```
// Kotlin Arrays
fun main() {
var numList = arrayOf(1,2,3,4,5,6,7,8,9,10);
    for(number in numList) {
        println(number)
    }
}
```

You can also specify the data type like this:

COPY

```
val numOne: IntArray = intArrayOf(2, 3, 4)
val longNums = arrayOf<Long>(21,22,23,24)
// Use a for loop like in the above to print out various items
```

Coming from Python, you might be used to creating lists that contain elements of different types like this:

COPY

```
python_list = ['a' ,1 ,4 ,4.89] # This is a python list
```

The above is discouraged in Kotlin, except when dealing with **polymorphic** types. H a list of different items you can use `listOf()` in Kotlin.

COPY

```
val mixedData = listOf("a", 2, 3.14) // List<Any>
```

Sets

A set is a generic unordered collection of elements that allows no duplicates. Kotlin distinguishes between read-only and mutable sets. Read-only sets are created with `setOf()` and mutables set with `mutableSetOf()`

COPY

```
val words = setOf("pen", "cup", "dog", "spectacles")
```

Mutable Set

A generic unordered collection of elements that does not support duplicate elements, and supports adding and removing elements.

Try running this.

COPY

```
val set = mutableSetOf<Int>()
println("set.isEmpty() is ${set.isEmpty()}") // true

set.add(1)
set.add(2)
set.add(1)

println(set) // Expected Output -> [1, 2]
```

The `is` operator helps us in type-checking

Kotlin Maps == Python Dicts

A collection that holds pairs of objects (keys and values) and supports efficiently retrieving the value corresponding to each key. Python dictionaries are the equivalent of Kotlin Maps. We use `mapOf()` in Kotlin to construct one with `to` keyword.

Python Dict

COPY

```
my_python_dict = {“firstName”: “John”, “age”: 31, “married”:true}
```

Kotlin Map

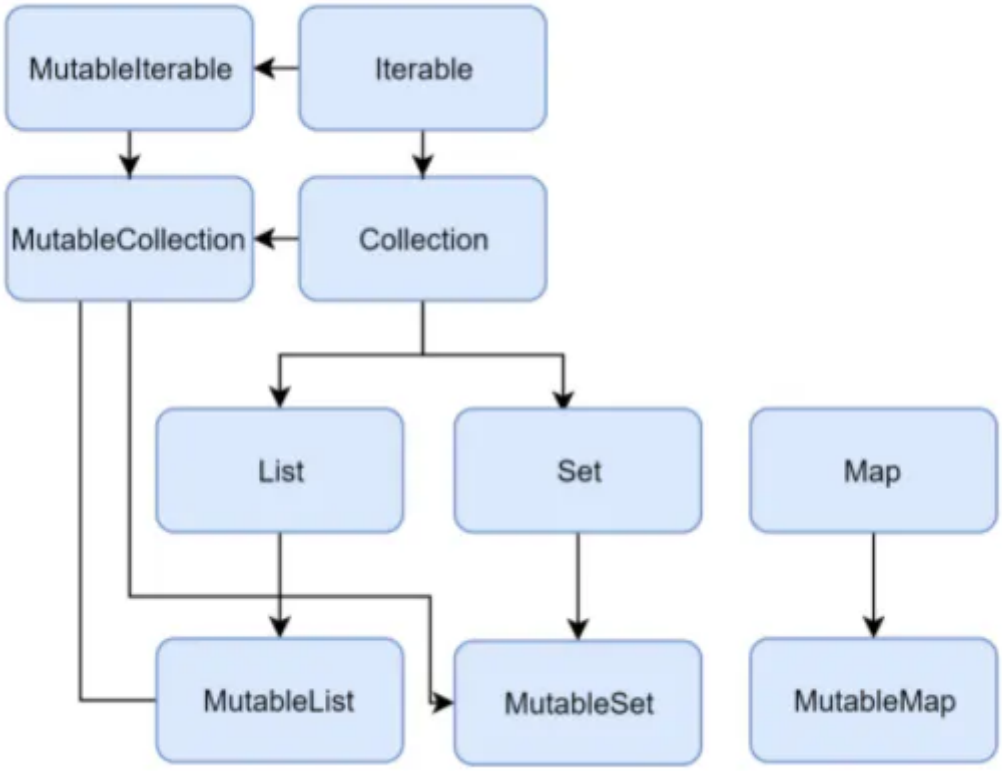
COPY

```
// equivalent to python dict above
val myKotlinMap = mapOf(
    "firstName" to "John",
    "age" to 31,
    "married" to true
)
```

Remember Kotlin provides implementations for basic collection types in two modes:

🎈 A read-only interface that provides operations for accessing collection elements.

🎈 A mutable interface that extends the corresponding read-only interface with write operations: adding, removing, and updating its elements.



Example:

COPY

```
val numbers = mutableListOf("one", "two", "three", "four")
numbers.add("five")   // this is OK
//numbers = mutableListOf("six", "seven")      // compilation error
```

Explanation

Altering a mutable collection doesn't require it to be a `var`: write operations modify the same mutable collection object, so the reference doesn't change. Although, if you try to reassign a `val` collection, you'll get a compilation error.

Read more about data collections at **Kotlin** Official Docs >>

Summary 📋

In Kotlin, everything is an object. Rather than using wrapper classes, Kotlin promotes primitive datatypes to objects. To get better performance, the Kotlin compiler maps basic types to JVM primitives.

JVM - Java Virtual Machine is a virtual machine that enables a computer to run Java programs as well as programs written in other languages.

Conclusion

Congrats if you finished this long tutorial! You are now good to go in Kotlin basics.

We discussed different data types of Kotlin programming language. we also went through different examples of Kotlin basic data types. We saw how to use Kotlin number, character, string, array or boolean data type.

If you happen to have enjoyed this article to, consider subscribing for email alerts and feel free to follow me here for related content.

Let's connect on **Twitter** with ❤️

Ronnie Atuhaire 😎

Subscribe to my newsletter

Read articles from **Ronnie Atuhaire's Blog** 🤓 directly inside your inbox. Subscribe to the newsletter, and don't miss out.

Enter your email address **SUBSCRIBE**

Kotlin | kotlin beginner | Python | 2Articles1Week

Written by

Ronnie Atuhaire

My name is Ronnie & a fellow geek like you😎. I am passionately tech curious.

I blog, tweet, write, code, vlog, discuss and eat anything about tech.

😍 Feel Free To Connect 😍

Follow

ARTICLE SERIES

Kotlin for Python Developers

1

Introduction to Kotlin

📌 Learning a new language is quite easy especially if you already know another one, all you got

2

Installation & Hello World!

Hello there 👋, welcome back! We shall continue from our last post (Introduction to Kotlin.) I want t...

Show all 12 posts

15

Kotlin Enum Classes

Hey, welcome again to more about Kotlin Enum Classes and if you have ...

16

Kotlin Data Classes

Hey, welcome again to my blog! Today, we shall learn more about Kotlin Data Classes and if you have ...

DIGITAL MARKETER | VLOGGER | AI | PYTHONISTA | GGMU | GRAPHICS | FOODIE