Kotlin - Basic Types

In this chapter, we will learn about the basic data types available in Kotlin programming language.

Numbers

The representation of numbers in Kotlin is pretty similar to Java, however, Kotlin does not allow internal conversion of different data types. Following table lists different variable lengths for different numbers.

Type	Size
Double	64
Float	32
Long	64
Int	32
Short	16
Byte	8

In the following example, we will see how Kotlin works with different data types. Please enter the following set of code in our coding ground.

Live Demo

```
fun main(args: Array<String>) {
   val a: Int = 10000
   val d: Double = 100.00
   val f: Float = 100.00f
   val l: Long = 1000000004
   val s: Short = 10
   val b: Byte = 1

   println("Your Int Value is "+a);
   println("Your Double  Value is "+d);
   println("Your Float Value is "+f);
   println("Your Long Value is "+l);
   println("Your Short Value is "+s);
```

```
   println("Your Byte Value is "+b);
}
```

When you run the above piece of code in the coding ground, it will generate the following output in the web console.

```
Your Int Value is 10000
Your Double  Value is 100.0
Your Float Value is 100.0
Your Long Value is 1000000004
Your Short Value is 10
Your Byte Value is 1
```

Characters

Kotlin represents character using **char**. Character should be declared in a single quote like **'c'**. Please enter the following code in our coding ground and see how Kotlin interprets the character variable. Character variable cannot be declared like number variables. Kotlin variable can be declared in two ways - one using **"var"** and another using **"val"**.

Live Demo

```
fun main(args: Array<String>) {
   val letter: Char    // defining a variable
   letter = 'A'        // Assigning a value to it
   println("$letter")
}
```

The above piece of code will yield the following output in the browser output window.

```
A
```

Boolean

Boolean is very simple like other programming languages. We have only two values for Boolean – either true or false. In the following example, we will see how Kotlin interprets Boolean.

Live Demo

```
fun main(args: Array<String>) {
   val letter: Boolean   // defining a variable
   letter = true         // Assinging a value to it
   println("Your character value is "+"$letter")
}
```

The above piece of code will yield the following output in the browser.

```
Your character value is true
```

Strings

Strings are character arrays. Like Java, they are immutable in nature. We have two kinds of string available in Kotlin - one is called **raw String** and another is called **escaped String**. In the following example, we will make use of these strings.

Live Demo

```
fun main(args: Array<String>) {
   var rawString :String  = "I am Raw String!"
   val escapedString : String  = "I am escaped String!\n"

   println("Hello!"+escapedString)
   println("Hey!!"+rawString)
}
```

The above example of escaped String allows to provide extra line space after the first print statement. Following will be the output in the browser.

```
Hello!I am escaped String!

Hey!!I am Raw String!
```

Arrays

Arrays are a collection of homogeneous data. Like Java, Kotlin supports arrays of different data types. In the following example, we will make use of different arrays.

Live Demo

```
fun main(args: Array<String>) {
   val numbers: IntArray = intArrayOf(1, 2, 3, 4, 5)
   println("Hey!! I am array Example"+numbers[2])
}
```

The above piece of code yields the following output. The indexing of the array is similar to other programming languages. Here, we are searching for a second index, whose value is “3”.

```
Hey!! I am array Example3
```

Collections

Collection is a very important part of the data structure, which makes the software development easy for engineers. Kotlin has two types of collection - one is **immutable collection** (which means lists, maps and sets that cannot be editable) and another is **mutable collection** (this type of collection is editable). It is very important to keep in mind the type of collection used in your application, as Kotlin system does not represent any specific difference in them.

Live Demo

```
fun main(args: Array<String>) {
   val numbers: MutableList<Int> = mutableListOf(1, 2, 3) //mutable List
   val readOnlyView: List<Int> = numbers                  // immutable list
   println("my mutable list--"+numbers)        // prints "[1, 2, 3]"
   numbers.add(4)
   println("my mutable list after addition --"+numbers)        // prints "[1, 2, 3,
   println(readOnlyView)
   readOnlyView.clear()    // ⇒ does not compile
// gives error
}
```

The above piece of code will yield the following output in the browser. It gives an error when we try to clear the mutable list of collection.

```
main.kt:9:18: error: unresolved reference: clear
   readOnlyView.clear()    // -> does not compile
                ^
```

In collection, Kotlin provides some useful methods such as **first(), last(), filter()**, etc. All these methods are self-descriptive and easy to implement. Moreover, Kotlin follows the same structure such as Java while implementing collection. You are free to implement any collection of your choice such as Map and Set.

In the following example, we have implemented Map and Set using different built-in methods.

Live Demo

```
fun main(args: Array<String>) {
   val items = listOf(1, 2, 3, 4)
   println("First Element of our list----"+items.first())
   println("Last Element of our list----"+items.last())
   println("Even Numbers of our List----"+items.
      filter { it % 2 = = 0 })   // returns [2, 4]

   val readWriteMap = hashMapOf("foo" to 1, "bar" to 2)
   println(readWriteMap["foo"])  // prints "1"

   val strings = hashSetOf("a", "b", "c", "c")
```

```
   println("My Set Values are"+strings)
}
```

The above piece of code yields the following output in the browser.

```
First Element of our list----1
Last Element of our list----4
Even Numbers of our List----[2, 4]
1
My Set Values are[a, b, c]
```

Ranges

Ranges is another unique characteristic of Kotlin. Like Haskell, it provides an operator that helps you iterate through a range. Internally, it is implemented using **rangeTo()** and its operator form is **(..)**.

In the following example, we will see how Kotlin interprets this range operator.

Live Demo

```
fun main(args: Array<String>) {
   val i:Int  = 2
   for (j in 1..4)
   print(j) // prints "1234"

   if (i in 1..10) { // equivalent of 1 < = i && i < = 10
      println("we found your number --"+i)
   }
}
```

The above piece of code yields the following output in the browser.

```
1234we found your number --2
```