**Data Types**

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 variables which determines the features of data.

In Kotlin, everything is an object, which means we can call member functions and properties on any variable. Kotlin built in data type are categorized as following different categories:

* Number
* Character
* Boolean
* Array
* String

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 (Bytes) | Min Value | Max Value |
| Byte | 1 | -128 | 127 |
| Short | 2 | -32768 | 32767 |
| Int | 4 | -2147483648 | 2147483647 |
| Long | 8 | -9223703854775808 | 9223703854775807 |
| Float | 4 | 1.40129846432481707e-45 | 3.40282346638528860e+38 |
| Double | 8 | 4.94065645841246544e-324 | 1.79769313486231570e+308 |

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.

fun main() {

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*("Int Value is " + a);

*println*("Double Value is " + d);

*println*("Float Value is " + f);

*println*("Long Value is " + l);

*println*("Short Value is " + s);

*println*("Byte Value is " + b);

}

Int Value is 10000

Double Value is 100.0

Float Value is 100.0

Long Value is 1000000004

Short Value is 10

Byte Value is 1

Characters

The Char data type can store a single character which must be wrapped in single quotes. Characters are represented by the type Char. They can not be treated directly as numbers.

|  |  |  |  |
| --- | --- | --- | --- |
| Type | Size (Bytes) | Min Value | Max Value |
| Char | 1 | -128 | 127 |

fun main() {

val value1 = 'A'

*//or*

val value2: Char

value2 = 'A'

}

Character literals go in single quotes: '1'. Special characters can be escaped using a backslash. The following escape sequences are supported: \t, \b, \n, \r, \', \", \\ and \$. To encode any other character, use the Unicode escape sequence syntax: '\uFF00'.

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.

|  |  |  |  |
| --- | --- | --- | --- |
| Type | Size (Bits) | Min Value | Max Value |
| Boolean | 1 | flase | true |

fun main() {

val flag = true

}

Strings

Strings are represented by the type String. Strings are immutable. Elements of a string are characters that can be accessed by the indexing operation: s[i].

Kotlin, like any good programming language, can work directly with characters and

strings. It does so through the data types Char and String, respectively. In this section, you’ll learn about these data types and how to work with them.

Char data type is designed to hold only single characters. The String data type, on the

on the other hand, stores multiple characters, which must be wrapped in double quotes.

val stringDog: String = "Dog"

It’s as simple as that! The right-hand side of this expression is what’s known as a string

literal; it’s the Kotlin syntax for representing a string.

Of course, type inference applies here as well. If you remove the type in the above

declaration, then Kotlin does the right thing and makes the stringDog a String

constant:

val stringDog = "Dog" *// Inferred to be of type String*

Concatenation

You can do much more than create simple strings. Sometimes you need to manipulate a

string, and one common way to do so is to combine it with another string.

In Kotlin, you do this in a rather simple way: by using the addition operator. Just as you can add numbers, you can add strings:

var message = "Hello" + " my name is "

val name = "Dick"

message += name *// "Hello my name is Dick"*

You need to declare message as a variable rather than a constant because you want to

modify it. You can add string literals together, as in the first line, and you can add string

variables or constants together, as in the last line.

It’s also possible to add characters directly to a string. This is similar to how you can

easily work with numbers if one is an Int and the other is a Double.

To add a character to a string, you do this:

val exclamationMark: Char = '!'

message += exclamationMark *// "Hello my name is Dick!"*

No need to explicitly convert the Character to a String before you add it to the message; Kotlin takes care of that for you!

String templates

You can also build up a string by using string templates, which use a special Kotlin

syntax that lets you build a string in a way that’s easy to read:

message = "Hello my name is $name!" *// "Hello my name is Dick!"*

This is much more readable than the example from the previous section. It’s an

extension of the string literal syntax, whereby you replace certain parts of the string with other values. Simply prepend the value you want to insert with a $ symbol.

This syntax works in the same way to build a string from other data types, such as

numbers:

val oneThird = 1.0 / 3.0

val oneThirdLongString = "One third is $oneThird as a decimal."

Here, you use a Double in the template. At the end of this code, your oneThirdLongString constant will contain the following:

One third is 0.3333333333333333 as a decimal.

Of course, it would actually take infinite characters to represent one third as a decimal,

because it’s a repeating decimal. Using string templates with a Double gives you no way to control the precision of the resulting string.

This is an unfortunate consequence of using string templates: they’re simple to use, but

offers no ability to customize the output.

You can also put expressions inside a string template, by following the $ symbol with a pair of braces that contain the expression:

val oneThirdLongString = "One third is ${1.0 / 3.0} as a decimal."

The result is just the same as before.

Multi-line strings

Kotlin has a neat way to express strings that contain multiple lines. This can be rather useful when you need to put a very long string in your code. You do it like so:

val bigString = """

|You can have a string

|that contains multiple

|lines

|by

|doing this.

""".*trimIndent*()

*println*(bigString)

The three double-quotes signify that this is a multi-line string. Handily, the first and

final new lines do not become part of the string. This makes it more flexible as you don’t have to have the three double-quotes on the same line as the string.

In the case above, it will print the following:

You can have a string

that contains multiple

lines

by

doing this.

Notice |, also known as the "pipe character", at the start of each line as well as the call

to trimMargin(). This prevents the string from having leading spaces, allowing you to

format your code with pretty indentation without affecting the output.

Array

Arrays in Kotlin are represented by the Array class. Arrays are created using the 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 an array of wrapper types. The item values are passed inside arrayOf() function like arrayOf(1,2,3) which creates an array[1,2,3].

Array that holds multiple different data types.

var arr = *arrayOf*(10, "BeginnersBook", 10.99, 'A')

Array that can only hold integers

var arr = *arrayOf*<Int>(1, 22, 55)

Array that can only hold strings

var arr2 = *arrayOf*<String>("ab", "bc", "cd")