Methods for Comparing Strings

In the following lesson, you will be introduced to the different methods provided by Scala for comparing strings.

We'll cover the following

- matches
- equals
- compareTo
- equalsIgnoreCase

In a previous lesson, we touched upon `String` comparison by discussing how to test the equality of strings using the equality operator `==`. In this lesson, we will look at a couple of built-in Scala methods which are specifically used for `String` comparison. Let's get right into the first method.

`matches`

By now, we've had quite a bit of practice with regular expressions. `matches` is a method which compares a string with a regular expression; it matches a regular expression pattern with an entire string.

`matches` is called on the string that you want to compare a regular expression with. It's a boolean type and returns `true` if the string matches and `false` if it doesn't. The syntax is as follows.

```
variableName.matches(regular expression)
```

OR

```
String Literal.matches(regular expression)
```

Let's look at an example where we need to check if our string matches the regular expression `"[a-zA-Z0-9]{4}"`.

false true

```
val stringMatch = "ab7"
val match1 = stringMatch.matches("[a-zA-Z0-9]{4}")

// Driver Code
println(match1)
```

When you run the first tab in the code snippet above, it should return `false` as the regular expression is specifying a pattern four characters long.

When you run the second tab in the code snippet above, it should return `true` as the string we are checking fulfills all the requirements of the regular expression.

You can try different strings to check which ones return `true` and which ones return `false`.

`equals`

`equals` works exactly like the equality operator `==` that was introduced in a previous lesson; comparing two strings. It is called on one `String` and takes the other `String` as an argument. `false` is returned when both strings are not identical and `true` is returned when both strings are identical. The syntax is as follows.

`String Literal.equals(String Literal)`

OR

`variableName.equals(variableName)`

In the example below, we are comparing `string2` with `string1`. You can insert any two strings of your choice to compare them.

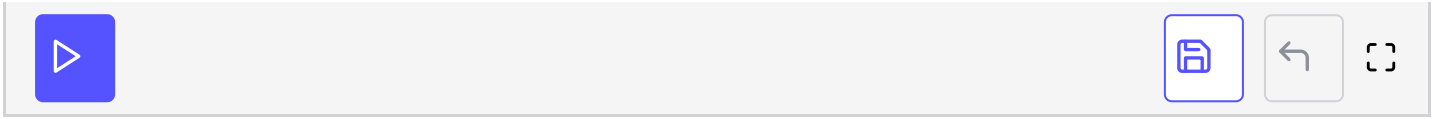
This code requires the following environment variables to execute:

LANG	C.UTF-8

```
val string1 = "Educative"
val string2 = "educative"
```

```
val comparingStrings = string1.equals(string2)


// Driver Code
println(comparingStrings)
```



When you run the code above, it should return false because `E` and `e` are different characters.

`compareTo`

`compareTo` compares two strings lexicographically, i.e., in alphabetical order. Unlike the other methods we have used above, `compareTo` returns an integer and that integer is the mathematical difference between the strings. If the strings are identical, `0` is returned. If a positive number is returned, that means that `string1` is *greater* than `string2`. If a negative number is returned, that means that `string1` is *less* than `string2.`

> `string1` is subtracted from `string2`.

A question arises; how can strings be *less* than or *greater* than other strings if they aren't numbers?

Every character, from letters to numbers to symbols, has a unique Unicode value. Using these values, we can perform mathematical operations on strings.

For example, `"a"` has a Unicode value of **U+0061** which is equivalent to a decimal value of **97**. `"1"` has a Unicode value of **U+0031** which is equivalent to a decimal value of **49**. Hence, when we subtract `"1"` from `"a"`, we are actually subtracting **49** from **97** giving us a difference of **48**.

> You don't need to memorize these values, the table of the official Unicode standard is readily available.

`compareTo` compares each character of `string1` with the character in the same position as `string2`. The syntax is as follows.

String1.compareTo(String2)

string1>string2 => positive number

string1<string2 => negative number

string1 == string2 => 0

Let’s look at two examples; one where the strings are identical and one where they aren’t.

This code requires the following environment variables to execute:

LANG	C.UTF-8

```
val string1 = "This is Educative"
val string2 = "Hello Scala"
val string3 = "Hello Scala"

val lexiCompare1 = string1.compareTo(string2) //string2-string1
val lexiCompare2 = string2.compareTo(string3) //string3-string2

// Driver Code
println(s"Comparing string1 and string2: $lexiCompare1")
println(s"Comparing string2 and string3: $lexiCompare2")
```

`equalsIgnoreCase`

The final method we are going to look at is `equalIgnoreCase.` This method works exactly like `equals` with the added feature (or lack thereof) that it ignores cases. So, for `equalsIgnoreCase`, `"a"` is equivalent to `"A"`. The syntax is identical to `equals`.

String Literal.equalsIgnoreCase(String Literal)

OR

variableName.equalsIgnoreCase(variableName)

Let’s look at the same example we saw for `equals`.

This code requires the following environment variables to execute:

LANG	C.UTF-8

```
val string1 = "Educative"
val string2 = "educative"

val comparingStrings = string1.equalsIgnoreCase(string2)

// Driver Code
println(comparingStrings)
```

This time around, by using `equalsIgnoreCase`, we are returned `true`, as the method sees `e` and `E` as identical characters. Pretty cool!

These were just a few of the many methods in Scala that you can use on strings. This lesson provided you with a taste of the most useful and commonly used methods.

And with this, our discussion on strings comes to an end. In the next lesson, you will be challenged to work with strings.