Finding Patterns in Strings

In the following lesson, you will be introduced to regular expressions and learn how to find patterns in a string in Scala.

We'll cover the following

Problem

A string is just a sequence of characters. Let's say we want to find a particular sequence of characters in a string; we want to find a pattern. That pattern can be anything from a sequence of numbers to a sequence of characters and everything in between.

Solution

To understand the solution to this problem, we need to first go over *regular expressions*.

Regular Expressions

In computing, a regular expression is defined as "a sequence of symbols and characters expressing a string or pattern to be searched for within a longer string".

For us to be able to let the compiler know which regular expression we want it to find, we need to first learn how to write a regular expression.

Writing Regular Expressions

Writing regular expressions is all about syntax. Once you know the syntax, it's quite easy.

- The `*` symbol is used to represent the repetition of the character preceding it. It basically tells us that *"the character before me can exist 0 or more times"*

```
"ab*c"
```

The regular expression above is representing the patterns *ac*, *abc*, *abbc*, *abbbc*, etc. The character *'b'* can be present from **0** times to an infinite number of times.

- The `+` symbol is also used to represent the repetition of the character preceding it. However, unlike `*`, the character must be present at least once.

```
"ab+c"
```

The regular expression above is representing the patterns *abc*, *abbc*, *abbbc*, etc. The character *'b'* can be present from **1** time to an infinite number of times.

- If you want to specify the number of times a character is being repeated, we can use curly brackets `{}` along with the number of repetitions we want in the pattern.

```
"ab{2}c"
```

The regular expression above is representing the pattern *abbc*.

We can also specify the minimum and the maximum number of times a character can be repeated.

```
"ab{2,5}c"
```

The regular expression above is representing *abbc*, *abbbc*, *abbbbc*, *abbbbbc*.

- We can also specify a set of characters using square brackets `[]`. Any

- We can also specify a set of characters using square brackets `[]`. Any character in the set will be matched to a character in a string.

```
"[abc]"
```

The regular expression above is representing the patterns *a*, *b*, *c*.

By inserting a `+` after the above expression we can have any combination of characters in the set.

```
"[abc]+"
```

The regular expression above is representing the patterns *a*, *b*, *c*, *ab*, *ac*, *ba*, *bc*, *ca*, *cb*, *abc*, *abb*, *aaa*, *bbb*, *ccc*, *acc*, *bac*, etc.

- To show multiple sets, we simply keep adding ranges one after the other.

```
"[a-bA-Zc]"
```

The regular expression above is representing patterns which include all the letters of the alphabet, regardless of if they are upper or lower case, i.e., 'a-z' and 'A-Z'.

- Sometimes we are looking for a character from a range of characters. For this, we can use `-`.

```
"[1-9]"
```

The regular expression above is representing the patterns *1*, *2*, *3*, *4*, *5*, *6*, *7*, *8*, *9*.

Like the previous example, we can also have a combination of characters from the set.

```
"[1-9]+"
```

- The `^` symbol is used to represent the characters we do not want in our pattern.

```
"[^abc]"
```

The regular expression above is representing all the letters in the alphabet except for *a*, *b*, and *c*.

Implementing Regular Expressions in Scala

When writing a regular expression in Scala we need to include the extension `.r` after the expression.

```
"ab*c".r
```

That sums up the basic syntax for regular expressions and would be sufficient enough for the scope of this course.

Let's now move onto some coding examples.

Find the First Occurrence of a Pattern

One way to solve our string problem is for the case when we only want to find the first matching occurrence of a regular expression. Scala's built-in method `findFirstIn` does just that; it finds the first occurrence of the expression and returns the expression.

Syntax

```
variableName of expression to find.findFirstIn(variableName of String)
```

OR

```
variableName of expression to find.findFirstIn(String Literal)
```

Let's look at an example where we simply want to know if the word "the" is in a sentence. For this, we don't need to find every "the", only one, specifically the first one.

This code requires the following environment variables to execute:

LANG	C.UTF-8

```
val expressionToFind = "the".r

val stringToFindExpression = "the little girl talked to the wolf"


val match1 = expressionToFind.findFirstIn(stringToFindExpression)

// Driver Code
match1.foreach(println)
```

In the above code, we are returned the expression *the*.

`findFirstIn` is great when we just want to know if an expression occurs in a string or not. But what if we want to know how many times an expression occurs in a string?

Find All the Occurrences of a Pattern

Scala's built-in method `findAllIn`, finds all the occurrences of the expression you are trying to find and returns a collection of all the matching expressions.

Syntax

```
variableName of expression to find.findAllIn(variableName of String)
```

OR

```
variableName of expression to find.findAllIn(String Literal)
```

In the example below, we want to find all the numbers from a list of numbers which contain the digits from 1 to 5.

This code requires the following environment variables to execute:

LANG	C.UTF-8

```
val expressionToFind = "[1-5]+".r

val stringToFindExpression = "12 67 93 48 51"

val match1 = expressionToFind.findAllIn(stringToFindExpression)

// Driver Code
match1.foreach(println)
```

Even though the above code works, it didn't really give us the output we were looking for. We have five individual numbers: $12, 67, 93, 48, 51$. Our code should only return 12 and 51 because they are the only two numbers which only contain digits from $1 - 5$. Since all of our numbers are two digits long, let's try to modify the regular expression in such a way that it is limited to two-digit numbers.

This code requires the following environment variables to execute:

LANG	C.UTF-8

```
val expressionToFind = "[1-5]{2}+".r

val stringToFindExpression = "12 67 93 48 51"

val match1 = expressionToFind.findAllIn(stringToFindExpression)

// Driver Code
match1.foreach(println)
```

Awesome, the code works exactly as we want it to. With this successful execution, our lesson on finding patterns in strings comes to an end.

We will continue our discussion on string patterns in the next lesson.