Regular Expression Literals

In this lesson, let's see how regex work in JavaScript.

WE'LL COVER THE FOLLOWING

JavaScript supports regular expressions through the `RegExp` type and provides a simple syntax to create them, as shown in Listing 7-5 below:

Listing-07-05: Using `RegExp` in JavaScript

```
<!DOCTYPE html>
<html>
<head>
  <title>Regular expression literals</title>
  <script>
    var pattern = /[A-Z]{3}-\d{3}/gi;
    var text = "AB-123 DEF-234 ert-456 -34";
    console.log(pattern.test(text));
    pattern.lastIndex = 0;
    var matches;
    do {
```

```
      matches = pattern.exec(text);
      console.log(matches);
    }

    while (matches != null);
  </script>
</head>
<body>
  Listing 7-5: View the console output
</body>
</html>
```

> 📜**NOTE:** If you are familiar with regular expressions, you can skip this section. If you would like to get more information about using them, I suggest you to start at http://www.regular-expressions.info/quickstart.html.

How it works

The first line of the script **(line six)** of this short code defines the `pattern` variable as a regular expression.

```
var pattern = /[A-Z]{3}-\d{3}/gi;
```

The right side of the assignment statement follows the `/pattern/flags` syntax, so the JavaScript engine infers it is a regular expression.

The pattern part is `[A-Z]{3}-\d{3}`, and it matches every string that starts with three letters in the "`A`"-"`Z`" range, followed by a dash, and closed by three decimal digits. The `/gi` is the part of the flag, where `g` indicates **global mode** (*the pattern will be applied to the whole string instead of stopping after the first match is found*), and `i` indicates **case-insensitive mode** (the case of the pattern and the string are ignored when determining matches).

Invoking the `test()` method **(line eight)** on the pattern checks whether the text passed as argument matches the pattern.

```
console.log(pattern.test(text));
```

The `text` will match, because it contains two substrings, `"DEF-234"`, and `"ert-456"`, that match the definition of the regular expression.

Because this expression uses global mode, you can use the `exec()` method

Because this expression uses global mode, you can use the `exec()` method **(line 12)** to iterate through all matches. That is exactly what the `do-while` loop does **(line 11-15)**.

The `pattern` variable holds the state of the last pattern matching, so each invocation of `exec()` finds a new match if there are no more matches, `exec()` returns a `null`.

Listing 7-5 produces the following output on the console:

console

```
true
["DEF-234", index: 7,
  input: "AB-123 DEF-234 ert-456 -34"]
["ert-456", index: 15,
  input: "AB-123 DEF-234 ert-456 -34"]
null
```

JavaScript Console Output

The `index` property in output shows the index of the first matching character in the input. The code snippet contains a line that sets `pattern.lastIndex` to zero. This line is used to reset the pattern after the `test()` operation, for the next `exec()` invocation to start at the beginning of the text.

When you define the regular expression with a literal, the JavaScript engine instantiates a `RegExp` object behind the scenes.

The definition

index.js

```
var pattern = /[A-Z]{3}-\d{3}/gi;
```

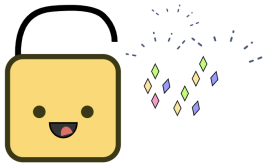
could have been written in this equivalent way:

index.js

```
// RegExp constructor
// arguments: pattern; flags
var pattern = new RegExp("[A-Z]{3}-\\d{3}", "gi");
```

As you see, the `RegExp` constructor function accepts two arguments, pattern and flags. The `“\\”` part within the string signs the single backslash character in the regular expression (using escape sequence syntax).

Achievement unlocked! 🎉



Congratulations! You’ve learned how regex work in JavaScript.

Good job! Give yourself a round of applause!

By now, you have already seen a few peculiarities of JavaScript. It’s time to dive deeper and learn the basics of the language from the *next lesson*.

See you there! :)