
## Regular Expressions

In this lesson we're going to be talking about **regular expressions**.

Regular expressions as a term may sound kind of boring, but once you learn how regular expressions work, you're going to realize just how sexy they really are.

Regular expressions are good for describing patterns in a string. They use a language syntax that's common to JavaScript and a number of other languages. In fact, JavaScript inherits its particular version of regular expression interpretation patterns from Perl, which is a much older language.

## Examples of Regular Expressions

```
/it/
```

For example, this regular expression would match a string looking for "it" as a set of letters together somewhere inside that string.

## Why Bother with Regular Expressions

Recognizing and dealing with patterns can be a complex challenge for programming languages. Just to give you an example of how that might look, let's take a typical problem that you might find in a form validation issue for a browser. Let's use JavaScript to validate that a string that a user submits has a telephone number. Suppose, we want to make sure that the database recognizes this telephone number the way that we expect it to be. We're going to make sure that it's formatted with three digits, a dash, and then four digits.

## Using JavaScript Without Regular Expressions

That sounds simple enough, but let's take a look at how we would do that using JavaScript without regular expressions.

```
var phoneFormatted = function(submission) {
    var counter;
    var current;
```

```
    var submissionLength = submission.length;
    var numberAttay = ["0","1","2","3","4","5","6","7","8","9"];
    var separator = "-";
    if (typeof submission !== "string" || submissionLength != 8) {
        return false;
    }
    for(counter=0; counter < submissionLength; counter++) {
        current = submission[counter];
        if (numberArray.indexOf(current) === -1) {
            if (counter !== 3) {
                return false;
            } else if (current != separator) {
                return false;
            }
        }
    }
    return true;
};
```

## Using Regular Expressions

Now that's a lot of code isn't it? But let me show you just how easy that would be with regular expressions.

```
var phoneFormattedRegex = function(submission) {
    return /^\d{3}-\d{4}$/.test(submission);
};
```

## Why Use a Regular Expression

That's a pretty clear example of why you would want to use a regular expression. The more concise and clear you can make your code, the less likely you are to make typographical mistakes or type coercion errors. Things that can come back and bite you later.

You can make a function that's this short and that contains all the same logic. It's much more straightforward and much easier to understand, once you understand how regular expressions are formatted, and how they work.

## How Regular Expressions Work

The way that a regular expression is set up, the pattern matched by regular expression is defined by a sequence of literal characters which are exactly the same as the characters you're going to see in the string and also meta characters.

```
/.it/
```

Here's a regular expression to match any letter, any number, anything followed by an "it". In this case, the literal characters are the "i" and the "t". The meta character in this case is that dot (`.`). That dot is one of many meta characters that you'll be introduced to when you start exploring how regular expressions work. The dot means "match almost any character". It puts a placeholder in the pattern that you're going to be working with.

The regular expression when it comes to that place in the pattern is going to match almost any character except for a line break That's something that's different in different languages, in terms of how they choose to interpret regular expressions. JavaScript follows the Perl format and does not recognize line breaks and periods.

Out regexp will match a word like "kitten", because the "k" is one of any character and it's followed by an "it". It's also going to match a string like "mitten" because "m" or "k", they're both "any possible character" which is represented by that dot.

But this regular expression would not match a string such as "it" without any character before it. There has to be a character there for that dot to work properly. Similarly, this regular expression is all in lower case, and it wouldn't recognize the "KIT" as being legitimately a match for the `/.it/` with lowercase letters. There's a way to deal with that, but we'll be getting to that a little bit later.

## How to Use Regular Expressions in JavaScript

So how would you use regular expressions in JavaScript? Well first of all, the thing to know is that regular expressions in JavaScript are a type of an object, just like a string or a number. They're a variable type and you can define them in a couple of ways.

One of the ways to construct regular expressions is using the `new` keyword and creating one from the `Regexp` constructor. This is useful if you want to create a regular expression from a dynamic value that may be set at runtime, but that you won't know at the time that you're writing the code.

A more common way to define regular expressions in JavaScript is when you knowthe value up front, you just type the value as if it were a string. But instead of putting quotes around the string you put forward slashes (`/`).