String Type

I'd like to show you how you can use regular expressions with methods of the string type. If you've worked with strings before, you've probably done replacements in strings where you used the `replace` method in order to find a substring and replace it with another substring. But did you know that you can also use regular expressions instead?

Let's start by creating a source string:

```
var source = "The kittens have mittens";
```

Now you can do replacement:

```
console.log(source.replace("it", "IT"));
```

It is going to replace only the first "it".

You can also use a regular expression instead:

```
var itMatches = /it/;
console.log(source.replace(itMatches, "IT"));
```

Being able to do that is convenient, but that doesn't necessarily get us very much. Let's see if we can take advantage of that and go one step further.

Global Modifier

Remember I told you that there were modifiers other than the case insensitive modifier that we talked about before? There's also a **global modifier** that lets you tell the string's replace method to match every instance that matches that pattern and replace it with the new string that you're passing:

```
var itMatchesGlobal = /it/g;
console.log(source.replace(itMatchesGlobal, "IT"));
```

So the g modifier says to do the replacement globally from beginning toe nd of the string and find every instance that matches.

Constructors

Of course we can also do that using the constructor for regular expressions:

```
var itAlsoMatchesGlobal = new RegExp("it", "g");
console.log(source.replace(itAlsoMatchesGlobal, "IT"));
```

So we've demonstrated that we can match globally either using the regular expression as defined directly, or as defined through the constructor.

Quantifiers

There are **quantifiers** that you can add to regular expressions. Quantifiers can help you find and replace patterns that include multiple copies of a single character or even whether a character exists at all.

For example we could use a **plus quantifier** or an **asterisk quantifier** and either match one or more of the preceding elements in a regular expression or zero or more of the preceding elements

Let's start with the plus quantifier:

```
var plusMatch = /12+3/;
console.log(plusMatch.test("123")); //true
console.log(plusMatch.test("1222223")); //true
console.log(plusMatch.test("13")); //false
```

This regexp is going to match the number `1` followed by `2` or any number of the copies of the number `2`, and then the number `3`. That plus means "take whatever comes right before the plus sign, and if there is one or more of that thing, then the regular expression matches properly". Therefore the `13` returns `false` because there is no number `2` in this string.

Asterisk Qualifiers

Let's create another regular expression:

```
var splatMatch = /12*3/;
console.log(splatMatch.test("123")); //true
console.log(splatMatch.test("1222223")); //true
console.log(splatMatch.test("13")); //true
```

Notice the difference. The splat or the asterisk (`*`) matches zero or more instances of the number `2`, therefore the last example also matches.

You can see how using quantifiers can really power up your regular expressions. The plus quantifier lets you match one or more of a preceding element in a regular expression. The asterisk quantifier lets you match zero or more of the preceding element in a regular expression.

Whitespace Character

Now let me introduce you a **first character class** in patterns for regular expressions and that's the **whitespace character**. The whitespace uses two characters together: a slash, and then an s, and together (\s) those represent a pattern called "any whitespace character". Whitespace can include a space, a tab, or even a new line character.

```
var whiteSpace = /\s+/g;
```

We've just created a regular expression that says "any place that you find one or more instances of a whitespace next to each other" (one space, two spaces, two tabs, three tabs etc).

Create a source string that we might want to clean up:

```
var source = "The     kittens        have mittens";
```

Testing code

Replace multiple spaces with one:

```
console.log(source.replace(whiteSpace, " "));
```

This is going to match one or more instances of a space character anywhere in the string and replace that with just a single space.

The ability to do global replacements, and to use quantifiers together is very convenient especially when you start introducing these character classes, such as the whitespace. Character classes exist for number of different things such as digits or letters and you'll learn a few more of them a little bit later.

The reason they work is because of the way that regular expressions treat the backslash character. Backslashes are used as an escape, much the same with that you would use them in strings to escape a quote character inside of quote. The only place where that gets tricky is when you're defining a regularexpression using a constructor.

Regular Expression Constructors

Regular expression constructors take strings as their arguments and a string could already have an escape inside of it. Therefore, in order to use a character class, or any escape character, you need to double the backslash and that can get a little bit confusing. This will not work:

```
var errorWhiteSpace = new RegExp("\s+", "g");
console.log(source.replace(errorWhiteSpace, " "));
```

And this will work correctly:

```
var alsoWhiteSpace = new RegExp("\\s+", "g");
console.log(source.replace(alsoWhiteSpace, " "));
```

because we've used the double backslash.