This regular expression matches only the first occurrence of the word `"cat"` in a string, and it is case-sensitive. To make the regular expression match all occurrences of `"cat"`, you can add the second argument to the constructor:

```
var reCat = new RegExp("cat", "g");
```

In this line, the second argument `"g"` is short for *global*, meaning that the entire string is searching for occurrences of `"cat"` instead of stopping after the first occurrence. If you want to make the pattern case-insensitive, you can add the character `"i"` to the second argument (`"i"` is short for *insensitive*, as in *case-insensitive*):

```
var reCat = new RegExp("cat", "gi");
```

Some regular expression literals use Perl-style syntax:

```
var reCat = /cat/gi;
```

Regular expression literals begin with a forward slash, followed by the string pattern, followed by another forward slash. If you want to specify additional processing instructions, such as `"g"` and `"i"`, these come after the second forward slash (as in the previous example).

Using a RegExp object

After creating a `RegExp` object, you apply it to a string. You can use several methods of both `RegExp` and `String`.

The first thing you do with a regular expression is determine if a string matches the specified pattern. For this simple case, the `RegExp` object has a method called `test()`, which simply returns `true` if the given string (the only argument) matches the pattern and `false` if not:

```
var sToMatch = "cat";
var reCat = /cat/;
alert(reCat.test(sToMatch));   //outputs "true"
```

In this example, the alert outputs `"true"` because the pattern matches the string. Even if the pattern only occurs once in the string, it is considered a match, and `test()` returns `true`. But what if you want access to the occurrences of the pattern? For this use case, you can use the `exec()` method.

The `RegExp exec()` method, which takes a string as an argument, returns an `Array`. The first item in the array is the first match; the others are back references (which will be discussed shortly).

```
var sToMatch = "a bat, a Cat, a fAt baT, a faT cat";
var reAt = /at/;
var arrMatches = reAt.exec(sToMatch);
```

Here, `arrMatches` contains only one item: the first instance of `"at"` (which is in the word `"bat"`). But of what use is returning an array with only one item? Suppose you want to get all of the occurrences of a pattern. This is where a different method can be used.

The `String` object has a method called `match()`, which returns an array of all matches within the string. The method is called on the `string` object and the `RegExp` object is passed in as an argument:

```
var sToMatch = "a bat, a Cat, a fAt baT, a faT cat";
var reAt = /at/gi;
var arrMatches = sToMatch.match(reAt);
```

This code fills the aMatches with all of the instances of "at" in the order in which they appear in `sToMatch`. Here are the contents of the aMatches array after this code executes:

Index	Value	From
0	"at"	"bat"
1	"at"	"Cat"
2	"At"	"fAt"
3	"aT"	"baT"
4	"aT"	"faT"
5	"at"	"cat"

Another string method called `search()` acts the same way as `indexOf()`, but uses a `RegExp` object instead of a substring. The `search()` method returns the index of the first occurrence in the string:

```
var sToMatch = "a bat, a Cat, a fAt baT, a faT cat";
var reAt = /at/gi;
alert(sToMatch.search(reAt));   //outputs "3"
```

In this example, the alert will display `"3"`, because the first occurrence of `"at"` is at position 3 in the string. Specifying the regular expression as global (with the `g`) has no effect when using `search()`.

Extended string methods

Two String methods, discussed earlier in the book, also accept regular expressions as parameters. The first is the `replace()` method, which replaces all occurrences of a substring (the first argument) with a different string (the second argument). For example:

```
var sToChange = "The sky is red. ";
alert(sToChange.replace("red", "blue"));    //outputs "The sky is blue."
```

Here, the substring `"red"` is replaced with the string `"blue"`, making the output `"The sky is blue."` It is possible to pass in a regular expression as the first argument as well:

```
var sToChange = "The sky is red.";
var reRed = /red/;
alert(sToChange.replace(reRed, "blue"));    //outputs "The sky is blue. "
```