Metacharacters

It is important to know that regular expressions can establish where in a string a particular pattern appears. For example, it's possible to use a regular expression to specify that you want the specific pattern that you're looking for only to appear at the very beginning of a string.

In order to specify that, you can use the caret (^) meta character and you use that at the beginning of a regular expression pattern, to say that this pattern has to match the very beginning of a string.

```
var source = "The kittens have mittens";
var itMatchesFirst = /^it/;
console.log(itMatchesFirst.test(source)); //false
```

This regular expression will match letters "it" together at the very beginning of a string, but not anywhere else. Therefore the example returns `false`.

Caret Metacharacter

You can use meta characters with various flags:

```
var theMatchesFirst = /^the/i;
console.log(theMatchesFirst.test(source)); //true
```

Dollar Sign Metacharacter

In addition, there's another meta character that lets you specify that the pattern has to exist at the end of the string. It looks like a dollar sign ($). You put that at the end of your regular expression and it specifies that the pattern must appear at the very end of the string that's being tested:

```
var source = "The kittens have mittens";
var ensMatchesLast = /ens$/i;
console.log(ensMatchesLast.test(source)); //true
var secondSource = "The Kitten Ensemble";
console.log(ensMatchesLast.test(secondSource)); //false
```

Creating tsMatches Variable

Now let's apply those two meta characters together:

```
var tsMatches = /^t.+s$/i;
console.log(tsMatches.test(source)); //true
console.log(tsMatches.test(secondSource)); //false
```

This regular expression looks for strings that starts with a "t", and then has any number of any characters (spaces, letters, numbers etc). Then the string should end with an "s". The regular expressions is also case insensitive.

So you can see how using both the caret and the dollar sign is very convenient, if you want to match exactly the complete pattern in your string.

In addition, you can also use regular expressions to define sets of characters. What you can do is test for the presence of any one of a set of characters anywhere within your pattern, just by enclosing those characters within square brackets (`[ ]`):

```
var source = "The kittens have mittens.";
var tteMatches = /t[te]/;
console.log(tteMatches.test(source)); //true
```

This regular expression will be looking for any pattern that starts with a "t", and then is followed by either a "t" or an "e".

The thing to remember about these square brackets, is that if the letter doesn't appear in the right place, there will be no match:

```
var secondSource = "The cats have hats";
console.log(tteMatches.test(secondSource)); //false
```

Although there are "t"'s, and there are "e"'s, there is no "t" followed by either "t" or "e" anywhere in the second string. For sets of characters in square brackets to match, the letters have to be in the proper sequence. But if either letter appears in the same relative space that you would expect itto, based on the pattern that we've established, there will be a match:

```
var thirdSource = "The cats hate hats";
console.log(tteMatches.test(thirdSource)); //true
```

This returns `true` because there is a "te" now.

Using square brackets can be very powerful and allow you to create very versatile patterns that can match in precise ways, exactly the type of string that you're looking for.

Pipe Metacharacter

There are other regular expression meta characters that a let us optionally replace or ignore certain characters depending on what their context is. One of these is the "or" meta character represented by a pipe (|). It will match one of a set of subpatterns inside of a pattern, as long as it's the only one that exists there:

```
var source = "The cats hate hats";
var animalsMatches = /dogs|cats|horses|chickens/;
console.log(animalsMatches.test(source)); //true
```

This regexp will match any string that contains either dogs, or cats, or horses, or chickens.

In addition, you can make characters optional by using the question mark metacharacter:

```
var secondSource = "The cat hates hats";
console.log(animalsMatches.test(secondSource)); //false

var animalMatches = /dogs?|cats?|horses?|chickens?/;
console.log(animalMatches.test(secondSource)); //true
```

The ? makes the preceding character (in our case that's "s") optional. The first example fails, because the string has "cat", not "cats". However, when we make the "s" optional, the match is found.