Lesson Introduction

Let's start talking about some of the methods that regular expressions offer to make themselves convenient for you. First of all, I'm going to show you how a basic match test would work.

Test Method

The `test` method always returns a boolean. You should use it from a regular expression instance by passing a string that you want to test against. That regular expression instance then uses its own pattern to match against that string, and tell you whether there's a match or not.

```
var itMatches = /it/;
var source = "The kittens have mittens";
var result = itMatches.test(source);
console.log(result);
```

We will get `true` as a result. That's because the result is the matching of the letters "it" inside the string "the kittens have mittens". Of course, "The kittens have mittens" has two sets of "it's", after the "k", and after the "m". But it only needs to match one in order for the regular expression to return `true`.

Testing Against Variable Values

You can also test against variable values:

```
var something = "cats";
var somethingMatches = new RegExp(something);
var source = "The kittens have mittens";

var result = somethingMatches.test(source);
console.log(result);

var source2 = "The cats have hats";
var result2 = somethingMatches.test(source2);
console.log(result2);
```

If you have a variable that you want to set to a regular expression, you can construct your regular expression dynamically using a regular expression constructor. Using `new RegExp` we create a new regular expression based around the value of the variable `something`.

The ability to match a regular expression using a variable value is very important because you don't always know in advance what your variables are going to contain, and you don't always know in advance what you're going to have to test against. That provides some flexibility, but regular expressions can actually support more flexibility than that.

Modifiers

Regular expressions also support **modifiers**, and one of the most convenient modifiers is the `i`, for "case insensitive". This allows regular expressions to take the string and match it against the pattern, ignoring the case.

```
var itMatches = /it/;
var itMatchesInsensitive = /it/i;

var source = "THE KITTENS HAVE MITTENS";

console.log(itMatches.test(source)); //false
console.log(itMatchesInsensitive.test(source)); //true
```

You can also use the constructor for regular expressions to do the exact same thing, by passing it as a separate argument:

```
var itAlsoMatchesInsensitive = new RegExp("it","i");
console.log(itAlsoMatchesInsensitive.test(source)); //true
```

That second argument gets passed as the modifier to that regular expression.