A regular expression literal is specified in the form /pattern/modifiers, where "pattern" is the regular expression itself and the optional "modifiers" section specifies various options.

* The pattern portion above starts with an ^ indicating the beginning of a string.
* The \d indicates a digit followed by {2} meaning 2 consecutive digits.
* The $ indicates end of a string.

So, this pattern will attempt to find exactly 2 consecutive digits from the beginning to the end of a string.

1. var regex = /^\d{2}$/;
2. alert(regex.test("29")); // => true
4. alert(regex.test("3g8")); // => false

You can add 3 possible modifiers to the regular expression:

* case-sensitivity (i),
* global searches (g),
* multiline mode (m).

You can combine these by stringing them together, like so: /pattern/gim.

For example matching the string "Facebook" is as simple as /Facebook/, but matching emails, XML, or HTML tags can be rather complex.

For example, an email pattern: /^[a-zA-Z0-9.\_-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}$/.

Regular expressions can be used for parsing, substitution, and format checking. They are often used in form validation, for example to validate an email or social security number. JavaScript's String type has 4 built-in methods: **search(), replace(), split(), and match()**.

When searching in a given string, you can specify the pattern you're looking for. Below we search for the first instance of a lower-case character [a-z] followed by one or more digits [0-9]+.

1. var str = "James Bond007";
2. var regex = /[a-z][0-9]+/;
3. alert(str.search(regex)); // => 9. the position of 'd'

The next expression parses the string, removes all the digits [\d] from it, and returns the remaining string. Note that the g is a global modifier which means find all matches and don't stop after the first match is found.

1. var str = "James Bond007";
2. var regex = /[\d]/g;
3. alert(str.replace (regex, "")); // => James Bond

The method match() tests whether a string matches a regular expression or extracts specific parts of a string. If nothing is found it returns null, else it returns an array of all matches.

1. var regExp = /\d+/g;
2. var str = "Mary: 36, Tim: 38";
3. alert(str.match(regExp)); // => [36,38]

The method **split()** splits a string into an array of strings based on the regular expression. It uses the regular expression as a delimiter to split the string. The delimiters themselves are not included in the resulting array.

1. var regex = /[**,:.**]/; // delimiters are **comma, colon, and period**
2. var str = 'Tim:20,Henry:30.Linda:35';
3. alert(str.split(regex)); // => [Tim,20,Henry,30,Linda,35]

In JavaScript, there are actually two ways to create a regular expressions: using a regular expression literal, which we have discussed before and using a RegExp() constructor. In fact, regular expression literals implicitly make a call to the RegExp constructor. You can also call RegExp() yourself to create objects that represent regular expressions. When defining a pattern using RegExp it must be enclosed by quotes and any special character must be escaped with a backslash to retain their meaning. In the example below we see that this adds considerable complexity to the pattern, which is why they are not recommended.

1. var regex = new RegExp("^\\s\*(\\+|-)?\\d+\\s\*$");
2. var str = "-1";
3. alert(regex.test(str)); // => true

The test() method on RegExp checks whether the string argument contains a match of the pattern specified by the regular expression. Here we are testing if the string is an integer, with an optional sign (+/-) character. The test() returns true.

Another method on RegExp called **exec()** checks whether the string argument contains one or more occurrences of the pattern specified by the regular expression. If no match is found, it returns null. If a match is found, it returns an array whose first element is the string that matches the entire pattern. Suppose you have a web page and you wish to extract valid email addresses from it:

1. var page = "annie@js.org, jim@email.com, some-invalid-email-address";
2. var regex = /[0-9a-zA-Z]+@[0-9a-zA-Z]+[\.]{1}[0-9a-zA-Z]+[\.]?[0-9a-zA-Z]+/g;
3. do {
4. var address = regex.exec(page);
5. alert(address[0]); // => annie@js.org, then jim@email.com
6. } while (address.index < page.length);