6th September 2019

Quantifiers +, *, ? and {n}

Let's say we have a string like `+7(903)-123-45-67` and want to find all numbers in it. But unlike before, we are interested not in single digits, but full numbers: `7, 903, 123, 45, 67`.

A number is a sequence of 1 or more digits `\d`. To mark how many we need, we can append a *quantifier*.

Quantity {n}

The simplest quantifier is a number in curly braces: `{n}`.

A quantifier is appended to a character (or a character class, or a `[...]` set etc) and specifies how many we need.

It has a few advanced forms, let's see examples:

The exact count: `{5}`

`\d{5}` denotes exactly 5 digits, the same as `\d\d\d\d\d`.

The example below looks for a 5-digit number:

```
1  alert( "I'm 12345 years old".match(/\d{5}/) ); //  "12345"
```

We can add `\b` to exclude longer numbers: `\b\d{5}\b`.

The range: `{3,5}`, match 3-5 times

To find numbers from 3 to 5 digits we can put the limits into curly braces: `\d{3,5}`

```
1  alert( "I'm not 12, but 1234 years old".match(/\d{3,5}/) ); // "1234"
```

We can omit the upper limit.

Then a regexp `\d{3,}` looks for sequences of digits of length `3` or more:

```
1  alert( "I'm not 12, but 345678 years old".match(/\d{3,}/) ); // "34567
```

Let's return to the string `+7(903)-123-45-67`.

A number is a sequence of one or more digits in a row. So the regexp is `\d{1,}`:

```
let str = "+7(903)-123-45-67";

let numbers = str.match(/\d{1,}/g);

alert(numbers); // 7,903,123,45,67
```

Shorthands

There are shorthands for most used quantifiers:

`+`

Means “one or more”, the same as `{1,}`.

For instance, `\d+` looks for numbers:

```
let str = "+7(903)-123-45-67";

alert( str.match(/\d+/g) ); // 7,903,123,45,67
```

`?`

Means “zero or one”, the same as `{0,1}`. In other words, it makes the symbol optional.

For instance, the pattern `ou?r` looks for `o` followed by zero or one `u`, and then `r`.

So, `colou?r` finds both `color` and `colour`:

```
let str = "Should I write color or colour?";

alert( str.match(/colou?r/g) ); // color, colour
```

`*`

Means “zero or more”, the same as `{0,}`. That is, the character may repeat any times or be absent.

For example, `\d0*` looks for a digit followed by any number of zeroes (may be many or none):

```
alert( "100 10 1".match(/\d0*/g) ); // 100, 10, 1
```

Compare it with `+` (one or more):

```
alert( "100 10 1".match(/\d0+/g) ); // 100, 10
// 1 not matched, as 0+ requires at least one zero
```

More examples

Quantifiers are used very often. They serve as the main “building block” of complex regular expressions, so let’s see more examples.

Regexp for decimal fractions (a number with a floating point): `\d+\.\d+`

In action:

```
1  alert( "0 1 12.345 7890".match(/\d+\.\d+/g) ); // 12.345
```

Regexp for an "opening HTML-tag without attributes", such as `<span>` or `<p>`.

1. The simplest one: `/<[a-z]+>/i`

   ```
   1  alert( "<body> ... </body>".match(/<[a-z]+>/gi) ); // <body>
   ```

   The regexp looks for character `'<'` followed by one or more Latin letters, and then `'>'`.

2. Improved: `/<[a-z][a-z0-9]*>/i`

   According to the standard, HTML tag name may have a digit at any position except the first one, like `<h1>`.

   ```
   1  alert( "<h1>Hi!</h1>".match(/<[a-z][a-z0-9]*>/gi) ); // <h1>
   ```

Regexp "opening or closing HTML-tag without attributes": `/<\/?[a-z][a-z0-9]*>/i`

We added an optional slash `/?` near the beginning of the pattern. Had to escape it with a backslash, otherwise JavaScript would think it is the pattern end.

```
1  alert( "<h1>Hi!</h1>".match(/<\/?[a-z][a-z0-9]*>/gi) ); // <h1>, </h1>
```

To make a regexp more precise, we often need make it more complex

We can see one common rule in these examples: the more precise is the regular expression – the longer and more complex it is.

For instance, for HTML tags we could use a simpler regexp: `<\w+>`. But as HTML has stricter restrictions for a tag name, `<[a-z][a-z0-9]*>` is more reliable.

Can we use `<\w+>` or we need `<[a-z][a-z0-9]*>`?

In real life both variants are acceptable. Depends on how tolerant we can be to "extra" matches and whether it's difficult or not to remove them from the result by other means.

Tasks

How to find an ellipsis "..." ?

importance: 5

Create a regexp to find ellipsis: 3 (or more?) dots in a row.

Check it:

```
let regexp = /your regexp/g;
alert( "Hello!... How goes?....."".match(regexp) ); // ..., .....
```

solution

Regexp for HTML colors

Create a regexp to search HTML-colors written as `#ABCDEF` : first `#` and then 6 hexadecimal characters.

An example of use:

```
let regexp = /...your regexp.../

let str = "color:#121212; background-color:#AA00ef bad-colors:f#fddee #fd2 #1

alert( str.match(regexp) )  // #121212,#AA00ef
```

P.S. In this task we do not need other color formats like `#123` or `rgb(1,2,3)` etc.

solution

Previous lesson Next lesson

Share Tutorial map

Comments

- If you have suggestions what to improve - please submit a GitHub issue or a pull request instead of commenting.
- If you can't understand something in the article – please elaborate.
- To insert a few words of code, use the `<code>` tag, for several lines – use `<pre>` , for more than 10 lines – use a sandbox (plnkr, JSBin, codepen…)