7th September 2019

Alternation (OR) |

Alternation is the term in regular expression that is actually a simple “OR”.

In a regular expression it is denoted with a vertical line character `|`.

For instance, we need to find programming languages: HTML, PHP, Java or JavaScript.

The corresponding regexp: `html|php|java(script)?`.

A usage example:

```
1 let regexp = /html|php|css|java(script)?/gi;
2
3 let str = "First HTML appeared, then CSS, then JavaScript";
4
5 alert( str.match(regexp) ); // 'HTML', 'CSS', 'JavaScript'
```

We already saw a similar thing – square brackets. They allow to choose between multiple characters, for instance `gr[ae]y` matches `gray` or `grey`.

Square brackets allow only characters or character sets. Alternation allows any expressions. A regexp `A|B|C` means one of expressions `A`, `B` or `C`.

For instance:

- `gr(a|e)y` means exactly the same as `gr[ae]y`.
- `gra|ey` means `gra` or `ey`.

To apply alternation to a chosen part of the pattern, we can enclose it in parentheses:

- `I love HTML|CSS` matches `I love HTML` or `CSS`.
- `I love (HTML|CSS)` matches `I love HTML` or `I love CSS`.

Example: regexp for time

In previous articles there was a task to build a regexp for searching time in the form `hh:mm`, for instance `12:00`. But a simple `\d\d:\d\d` is too vague. It accepts `25:99` as the time (as 99 seconds match the pattern, but that time is invalid).

How can we make a better pattern?

We can use more careful matching. First, the hours:

- If the first digit is `0` or `1`, then the next digit can be any: `[01]\d`.

- Otherwise, if the first digit is `2`, then the next must be `[0-3]`.
- (no other first digit is allowed)

We can write both variants in a regexp using alternation: `[01]\d|2[0-3]`.

Next, minutes must be from `00` to `59`. In the regular expression language that can be written as `[0-5]\d`: the first digit `0-5`, and then any digit.

If we glue minutes and seconds together, we get the pattern: `[01]\d|2[0-3]:[0-5]\d`.

We're almost done, but there's a problem. The alternation `|` now happens to be between `[01]\d` and `2[0-3]:[0-5]\d`.

That is: minutes are added to the second alternation variant, here's a clear picture:

```
1  [01]\d   |  2[0-3]:[0-5]\d
```

That pattern looks for `[01]\d` or `2[0-3]:[0-5]\d`.

But that's wrong, the alternation should only be used in the "hours" part of the regular expression, to allow `[01]\d` OR `2[0-3]`. Let's correct that by enclosing "hours" into parentheses: `([01]\d|2[0-3]):[0-5]\d`.

The final solution:

```
1  let regexp = /([01]\d|2[0-3]):[0-5]\d/g;
2
3  alert("00:00 10:10 23:59 25:99 1:2".match(regexp)); // 00:00,10:10,23:59
```

Tasks

Find programming languages

There are many programming languages, for instance Java, JavaScript, PHP, C, C++.

Create a regexp that finds them in the string `Java JavaScript PHP C++ C`:

```
1  let regexp = /your regexp/g;
2
3  alert("Java JavaScript PHP C++ C".match(regexp)); // Java JavaScript PHP C++
```

solution

Find bbtag pairs

A "bb-tag" looks like `[tag]...[/tag]`, where `tag` is one of: `b`, `url` or `quote`.

For instance:

```
[b]text[/b]
[url]http://google.com[/url]
```

BB-tags can be nested. But a tag can't be nested into itself, for instance:

```
Normal:
[url] [b]http://google.com[/b] [/url]
[quote] [b]text[/b] [/quote]

Can't happen:
[b][b]text[/b][/b]
```

Tags can contain line breaks, that's normal:

```
[quote]
  [b]text[/b]
[/quote]
```

Create a regexp to find all BB-tags with their contents.

For instance:

```
let regexp = /your regexp/flags;

let str = "..[url]http://google.com[/url]..";
alert( str.match(regexp) ); // [url]http://google.com[/url]
```

If tags are nested, then we need the outer tag (if we want we can continue the search in its content):

```
let regexp = /your regexp/flags;

let str = "..[url][b]http://google.com[/b][/url]..";
alert( str.match(regexp) ); // [url][b]http://google.com[/b][/url]
```

solution

Find quoted strings

Create a regexp to find strings in double quotes `"..."`.

The strings should support escaping, the same way as JavaScript strings do. For instance, quotes can be inserted as `\"` a newline as `\n`, and the slash itself as `\\`.

```
let str = "Just like \"here\".";
```

Please note, in particular, that an escaped quote `\"` does not end a string.

So we should search from one quote to the other ignoring escaped quotes on the way.

That's the essential part of the task, otherwise it would be trivial.

Examples of strings to match:

```
.. "test me" ..
.. "Say \"Hello\"!" ... (escaped quotes inside)
.. "\\" ..  (double slash inside)
.. "\\ \"" ..  (double slash and an escaped quote inside)
```

In JavaScript we need to double the slashes to pass them right into the string, like this:

```
let str = ' .. "test me" .. "Say \\"Hello\\"!" .. "\\\\ \\"" .. ';

// the in-memory string
alert(str); //  .. "test me" .. "Say \"Hello\"!" .. "\\ \"" ..
```

solution

Find the full tag

Write a regexp to find the tag `<style...>`. It should match the full tag: it may have no attributes `<style>` or have several of them `<style type="..." id="...">`.

…But the regexp should not match `<styler>`!

For instance:

```
let regexp = /your regexp/g;

alert( '<style> <styler> <style test="...">'.match(regexp) ); // <style>, <st
```

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…)