

JavaScript RegExp

'cause basis are still important

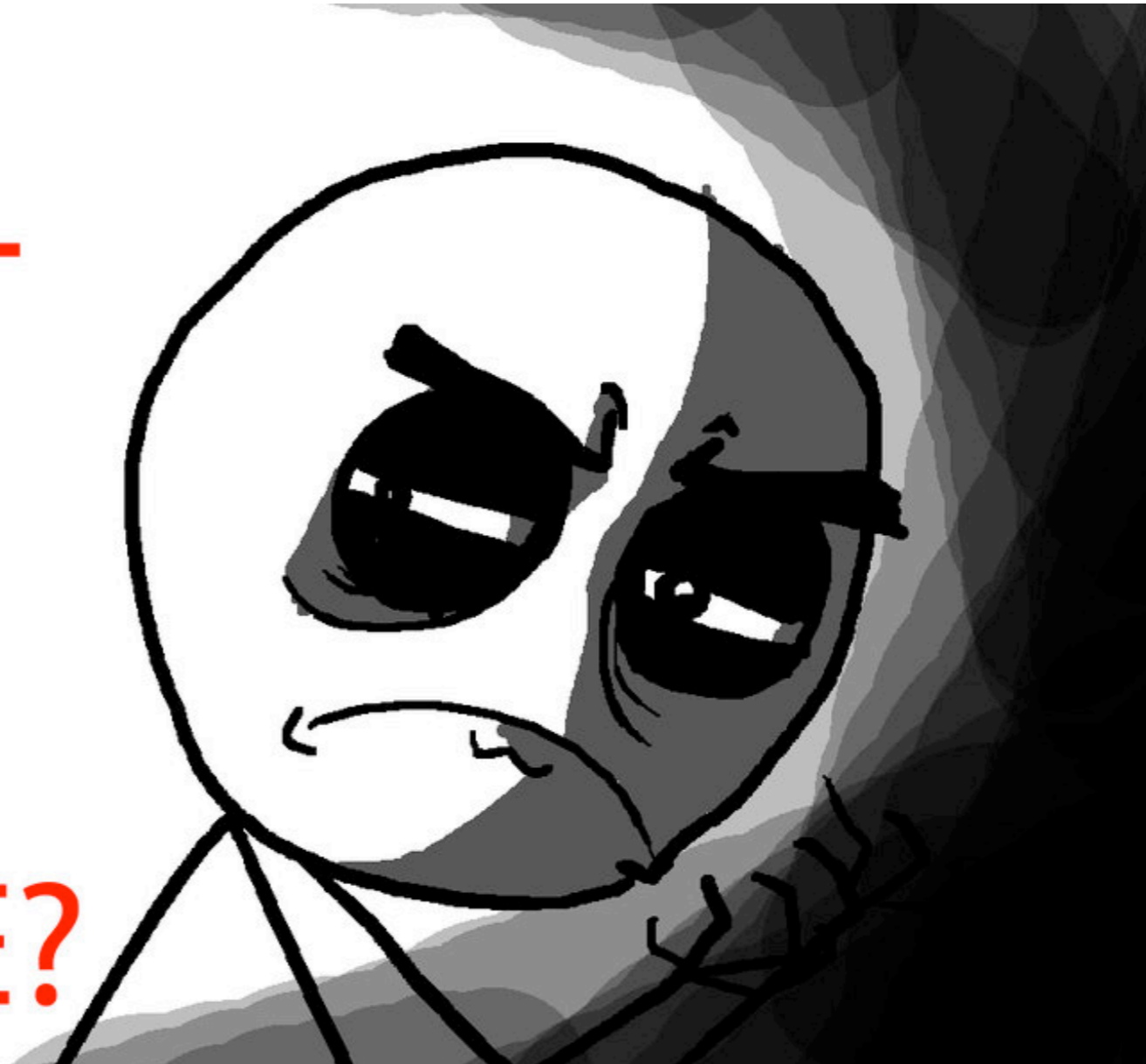
Andrea Giammarchi

Somebody Told Me This Talk Was “Too In Depth” ... so ...

- they see me explainin’ ... they’re hatin’
- I had to cut 2/3 of this talk
- ... but I did something better than that ...
- ... and definitively I didn’t take it personal ...

Somebody Told Me This Talk Was
“Too In Depth” ... so ...

YOU...
WHAT
HAVE
YOU
DONE?



so ... here the talk!

JavaScript Regular Expression
is such a cool thing!

JavaScript Regular Expression
is such a cool thing!



Cool story, bro!

Thanks!

... jokes a part ...

- this talk does indeed go in depth
- so I'll skip some point
- and I'll leave it online as a reference
- but following the reason I made it complete

How Many Developers Still See RegExp

How Many Developers Still See RegExp



What Are Regular Expressions About

- simplified searches over different conditions
- security friendly (if you know what you are doing)
- great help for any kind of string validation (not only forms, any sort of string)

Two Ways To Create A RegExp

- literal: recommended
- `(?:new)?RegExp`: only if necessary
- don't worry about that "new" thingy, you gonna understand it at the end of these slides

Literal RegExp

- `var re = /this is RegExp/;`
- processed at compile-time (faster and safer)

new RegExp

- `var re = new RegExp("this is RegExp");`
- `var re = RegExp("this is RegExp");`
- `var re = RegExp("this is RegExp", "gim");`
- `// ... right ... but who's gim ?`

3 Optional Flags

- **g: global multiple searches/substitutions**

- `var s = "bb";`
- `s.replace(/b/, "a"); // ab`
- `s.replace(/b/g, "a"); // aa`
- `/b/g.global; // true`

3 Optional Flags

- **g: a common mistake (ES5+)**

- `// WRONG: endless loop`
- `while (m = /b/g.exec(s)) {`
- `console.log(m);`
- `}`

3 Optional Flags

- **g: a common mistake: FIXED (ES3+)**
 - `// OK`
 - `var re = /b/g;`
 - `while (m = re.exec(s)) {`
 - `console.log(m);`
 - `}`

3 Optional Flags

- **i: ignore case/case insensitive searches**
 - `var s = "bb";`
 - `/B/.test(s); // false`
 - `/B/i.test(s); // true`
 - `/B/i.ignoreCase; // true`

3 Optional Flags

- **m: multi-line search**
 - `var s = "bb\nbb";`
 - `/^bb$/.test(s); // false`
 - `(/^bb$/m).test(s); // true`
 - `(/^bb$/m).multiline; // true`

“y” Optional Flag

- y: sticky search
- not cross browser (SpiderMonkey idea)
- similar to “g”, probably not needed

from Literal to RegExp

- `RegExp.prototype.toJSON = function () {`
- `return [`
- `this.source,`
- `(this.global ? "g" : "") +`
- `(this.ignoreCase ? "i" : "") +`
- `(this.multiline ? "m" : "")`
- `];`
- `};`
- `// and back ...`
- `RegExp.apply(null, JSON.parse(JSON.stringify(/"/g))); // /"/g`

2 Methods

- `.exec(text:String):Array|null`
- `.test(text:String):Boolean`

`.exec(text:String):Array|null`

- `var str = "bab", re = /b/g, match;`
- `match = re.exec(str); // ["b"]`
- `re.lastIndex; // "b" 1 ==> "ab" // where to start`
- `match = re.exec(str); // ["b"]`
- `re.lastIndex; // "bab" 3 ==> ""`
- `match = re.exec(str); // null`
- `re.lastIndex; // "" 0 ==> "bab" reset, reusable`

`.exec(text:String):Array|null`

- `var str = "bab", re = /b/g, match;`
- `match = re.exec(str); // ["b"]`
- `match.index; // 0 // where is the match`
- `match.input; // "bab"`
- `match = re.exec(str); // ["b"]`
- `match.index; // 2`
- `match.input.substr(match.index, match[0].length);`
- `// always === match[0]`

`.exec(text:String):Array|null`

- `// use parenthesis to capture/create sub matches`
- `var str = "bab", re = /(a)b/g, match;`
- `match = re.exec(str); // ["ab", "a"]`
- `match.length; // 2`
- `match.input; // "bab"`
- `match.index; // 1`
- `match[1]; // "a" as content of first parenthesis`

`.exec(text:String):Array|null`

- `var str = "abc", re = /(a(b))(c)/g, match;`
- `match = re.exec(str); // ["abc", "ab", "b", "c"]`
- `match.length; // 4`
- `match.index; // 0`
- `match[1]; // "ab" as content of first parenthesis`
- `match[2]; // "b" as first nested parenthesis`
- `match[3]; // "c" as content of third parenthesis`

`.exec(text:String):Array|null`

- `// ?: to ignore capturing`
- `var str = "abc", re = /(a(?:b))(c)/g, match;`
- `match = re.exec(str); // ["abc", "ab", "c"]`
- `match.length; // 3`
- `// nested parenthesis ignored`
- `match[1]; // "ab" as content of first parenthesis`
- `match[2]; // "c" as content of second parenthesis`

Why Would You Ignore Capture

- parenthesis pollute the global RegExp function
- `/(a)/.exec("a") && RegExp.$1; // "a"`
- the match inside them may not be relevant for what we need to do
- `/search(?:es)/.test(input);`
- the match array can be smaller (memory and performances)

.test(text:String):Boolean

- `var str = "bab", re = /b/g;`
- `re.test(str); // true`
- `re.lastIndex; // "b" 1 ==> "ab"`
- `re.test(str); // true`
- `re.lastIndex; // "bab" 3 ==> ""`
- `re.test(str); // false`
- `re.lastIndex; // "" 0 ==> "bab" reset, reusable`

.test(text:String):Boolean

- `// use parenthesis to capture sub matches`
- `var str = "bab", re = /(a)b/g, match;`
- `re.test(str); // true`
- `RegExp.$1; // "a" as equivalent of match[1]`
- `RegExp["$&"]; // "ab" as equivalent of match[0]`
- `str.replace(re, "$1"); // "ba"`
- `str.replace(re, "$1-$&"); // "ba-ab"`

.test() VS .exec()

- **strictly semantic (true or false, never null)**
- **no array creation (memory and performances)**
- **inline creation with capturing parenthesis**
 - `if (/(M[rs]+)/.test(input)) {`
 - `user.gender = RegExp.$1 == "Mr" ? "m" : "f";`
 - `user.name = input.replace(RegExp["$&"], "");`
 - `}`

RegExp & Replace Helpers

- `$1` to `$99` for sub-matches
- `&` as equivalent of `match[0]`
- ``` as text before the match
- `'` as text after the match
- `$$` as dollar sign
- `"ab".replace(/./g, "$`$'"); // "ba"`
- `src.replace(/jQuery(\.|\/)/g, "$$$1"); // $() $.fn`

Special Characters

- `\` to escape any special character
- `^` beginning of text or a line with multiline
- `$` ending of text or a line with multiline
- `/^\^\$\$/ .test ("^\$"); // true`
- `/^\^\$\$/ .test (" ^\$"); // false`

Special Characters

- **** also to refer a match in the RegExp
 - `/(a)b\1/.test("aba"); // true`
 - `/(a)b\1/.test("abc"); // false`
 - `/(a)b(c)d\2b\1/.test("abcdcba"); // true`
 - `/(`|")\1/.test(`var t = "some text";') // true`

Special Characters

- *** zero or more occurrences (char/group)**
- **+ one or more occurrences (char/group)**
- **? zero or one occurrence (char/group)**
- `/co+ld?/.test("cool") && /co+ld?/.test("cold")`
- `/ha\!*/.test("ha") && /ha\!*/.test("ha!!!!!!")`

Special Characters

- `{n}` exact number of occurrences (char/group)
- `{n,}` from n occurrences to any (char/group)
- `{n,m}` from n occurrences to m (char/group)
- `/Go{2,}gle/.test("Google") && /Go{2,}gle/.test("Goooooooooooooogle")`
- `/^ht{2}ps?:\:\/\/.test(url)`

Special Characters

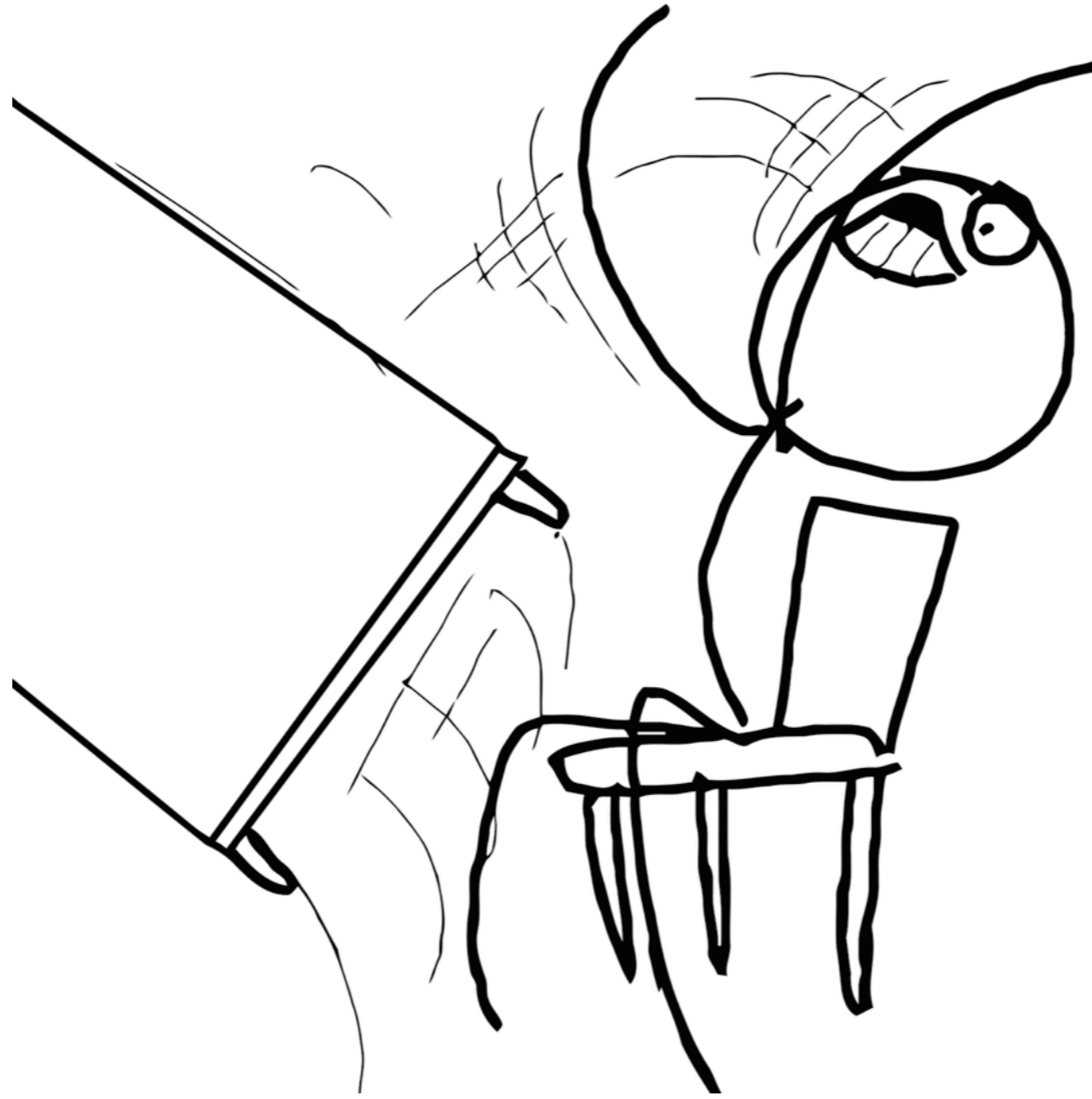
- ? also as “non greedy” operator after quantifiers such + * ? {}
- `/<p>.*</p>/ .exec("<p>a</p><p>b</p>") // result`
`["<p>a</p><p>b</p>"]`
- `/<p>.*?</p>/g.exec("<p>a</p><p>b</p>") // result`
`["<p>a</p>"]` and after `["<p>b</p>"]`

Special Characters

- ? also different meaning with parenthesis
- (?:ignore) to avoid capturing
- (?=followedBy) to look after
- (?!notFollowedBy) still looking after
- `/e(?=d)/.exec("red line").index; // 1 ["e"]`
- `/e(?!d)/.exec("red line").index; // 7 ["e"]`

No Look Behind

No Look Behind



Grouping Characters

- | used as “or” to match one or more occurrences
 - `/a|b/.test("abc"); // true`
 - `/^0049|\+49/.test(number); // international prefix`
- can be used inside parenthesis too
 - `/skills: (JavaScript|Ruby)/.test(dev.skills);`
 - `/skills: (?:Perl|PHP|Python)/.test(dev.skills);`

Grouping Characters

- **[xyz] match chars x, y, or z**
 - `/[ab]/.test("abc"); // true`
 - `/[ab]/.test("a")` rather than `/(?:a|b)/.test("a")`
- **can be used to match ranges (inclusive)**
 - `/^(?:0|[1-9][0-9]*)$/ .test(number); // non octals`
 - `/[A-Za-z]/.test(name); // a to z, case insensitive`
 - `/[a-z]/i.test(name); // same as above`

Grouping Characters

- **[^xyz]** match any char but x, y, or z
 - `/[^ab]/.test("abc"); // false`
 - `/[^ab]/.test("cde"); // true`
 - `/[^a-z]/.test("123"); // true`
- **no need to escape except closing bracket**
 - `/[.]/.test("a"); // false`
 - `/[.]/.test("."); // true`
 - `/[+.[\]/-]/.test("+.[]/-");`
- **if “-” is not used as range, first or last one!**

Grouping Characters

- **the only exception is `[\b]` for backspace**
 - `/[\b]/.test(backSpace); // true`
- **do not confuse `[\b]` with word boundary**
 - `/\b/.test(backSpace); // not what we are testing`

Boundary

- similar to `[^A-Za-z0-9_]`, matches words boundaries **without capturing**
- `/end\b/.test("is this the end??"); // true`
- `/\bhi\b/i.test(greetings);`
- greetings as "hi", " hi ", "Hi!", "well, hi", etc.
- `/\bhi\b/i.exec(" Hi!"); // ["Hi"]`
- `/\Whi\W/i.exec(" Hi!"); // [" Hi!"]`
- the opposite of `\b` is `\B` as `[A-Za-z0-9_]`

Words Characters

- `\w` as range `[A-Za-z0-9_]`
- `\W` as range `[^A-Za-z0-9_]` or `[^\w]`
- **Warning:** `il8n` could be a problem, `mūs` as example could be a valid word in some language but `\w` range may not consider it
- `/^\w+$/ .test("mouse") !== /^\w+$/ .test("mūs");`
- `\b` may have same side effect
- `/\b\w+\b/ .exec("mūs"); // ["m"] rather than ["mūs"]`

Numbers

- `\d` as range `[0-9]`
- `\D` as range `[^0-9]` or `[^\d]`
 - `/^\d+$/ .test("0123"); // true`
 - `/^\D+$/ .test("abc"); // true`
 - `/^[+-]?\d*\.\d+(e\d+)?$/ .test(JSNum);`
 - `// 1. 1.2 .2 1e2 -1 +1.2e2 ... etc`
- **Warning:** `il8n` could be a problem due thousand/decimal separator

Spaces

- `\n` as new line (linefeed)
- `\r` as carriage return
- `\f` as form-feed
- `\v` as vertical tab
- `\t` as horizontal tab

Spaces

- **\s as all previous spaces, plus others**

- `[\f\n\r\t\v\u00A0\u1680\u180e\u2000\u2001\u2002\u2003\u2004\u2005\u2006\u2007\u2008\u2009\u200a\u2028\u2029\u202f\u205f\u3000]`

- **\S as none of previous spaces, neither others**

- `[^\f\n\r\t\v\u00A0\u1680\u180e\u2000\u2001\u2002\u2003\u2004\u2005\u2006\u2007\u2008\u2009\u200a\u2028\u2029\u202f\u205f\u3000]`

Spaces and JSON

- **JSON.stringify(obj) is mostly broken**
- `alert(JSON.stringify("\u2028\u2029"));`

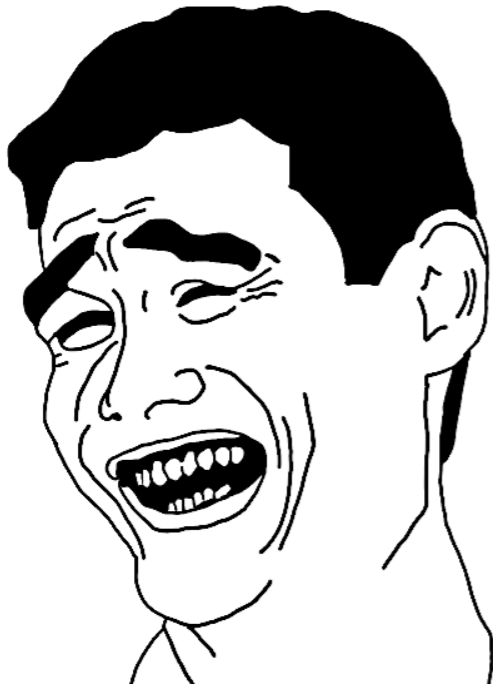
Spaces and JSON

- **JSON.stringify(obj) is mostly broken**
- `alert(JSON.stringify("\u2028\u2029"));`



Spaces and JSON

- **JSON.stringify(obj) is mostly broken**
- `alert(JSON.stringify("\u2028\u2029"));`



Spaces and JSON

- **JSON.stringify(obj) is mostly broken**
- `alert(JSON.stringify("\u2028\u2029"));`



Spaces and JSON

- `// fix it via RegExp and feature detection`
- `if (JSON.stringify("\u2028") == '"\u2028"') {`
- `JSON.stringify = function (stringify) {`
- `function place(m) {`
- `return "\\u202" + (m == "\u2028" ? "8" : "9");`
- `}`
- `var re = /\u2028|\u2029/g;`
- `return function fixed(data) {`
- `return stringify(data).replace(re, place);`
- `};`
- `} (JSON.stringify);`
- `}`

More Special Characters

- `\x00` to `\xff` for extended ASCII characters
- `\u0000` to `\uffff` for unicode characters
- `/\x00/.test(someString)`
- `/\u0000/.test("\x00"); // true`
- **both available for ranges**
- `/[\u0000-\u0002]/.test("\u0001"); // true`
- `/^[\\x00-\\xFF]*$/ .test(ASCIIOnly); // true`

More Special Characters

- **\cX for control characters**

- `/\cM/.test(control_M); // true`

- `/\cA/.test("\x01") // true`

- **\0 as NUL character**

- `/\0/.test("\x00"); // true`

More Special Characters

- **\cX** for control characters

- `/\cM/.test(command_M); // true`

- **\0** as NUL character

- `/\0/.test("\x00"); // true`

Yeah! We've got the NUL character

More Special Characters

- **. as any character (except some space!)**
 - `././.test("whatever"); // true`
 - `././.test("\n"); // false`
- **remember the NUL character ?**
 - `/[^\0]+/.test("\n"); // true`

When Is RegExp Constructor Useful

- access `.test()` results
- runtime creation (i.e. remove a class from a generic DOM node)
- `replace(RegExp("(?:^|)" + className + "(?: |$)", "g"))`
- markdown from js|k

String.prototype And RegExp

- `str.match(re)` similar to `re.exec(str)`
- `str.search(re)` as enriched `str.indexOf(s)`
- `str.split(re:RegExp|s:String)`
- `str.split("a")` similar to `str.split(/a/)`

String.prototype And RegExp

- **str.replace(re, f:Function|s:String)**

- `"abcd".replace(/(b)c(d)/, function (`
- `found, // "bcd" aka match[0]`
- `m1, // "b" aka match[1]`
- `m2, // "c" aka match[2]`
- `index, // 1 aka match.index`
- `input // "abcd" aka match.input`
- `) {`
- `return "whatever";`
- `}); // "awhatever"`

String.prototype And RegExp

- **str.replace(re, f:Function|s:String)**

- `"abcd".replace(/bcd/, function (`
- `found, // "bcd" aka match[0]`
- `index, // 1 aka match.index`
- `input // "abcd" aka match.input`
- `) {`
- `return "whatever";`
- `}); // "awhatever"`

String.prototype And RegExp

- **str.replace(re, f:Function|s:String)**
 - `"abcd".replace(/(b)c(d)/, "$2c$1"); // "adcb"`
- the content in the string is like accessing through the RegExp properties

Some Handy RegExp

- **markup boundaries**

- `var re = /(<\s*\b([a-z]+)\b.*?>)(.*)<\/\2\s*>/ig`
- `document.body.innerHTML.replace(re, cb);`
- `// cb("<p>a<p>b</p>c</p>", "p", "a<p>b</p>c", ...)`

Some Handy RegExp

- **double or single quoted strings**

- `/ ("|') (?: (?! \\) \2.)* ? \1 /g`

- **lat and long on maps input field**

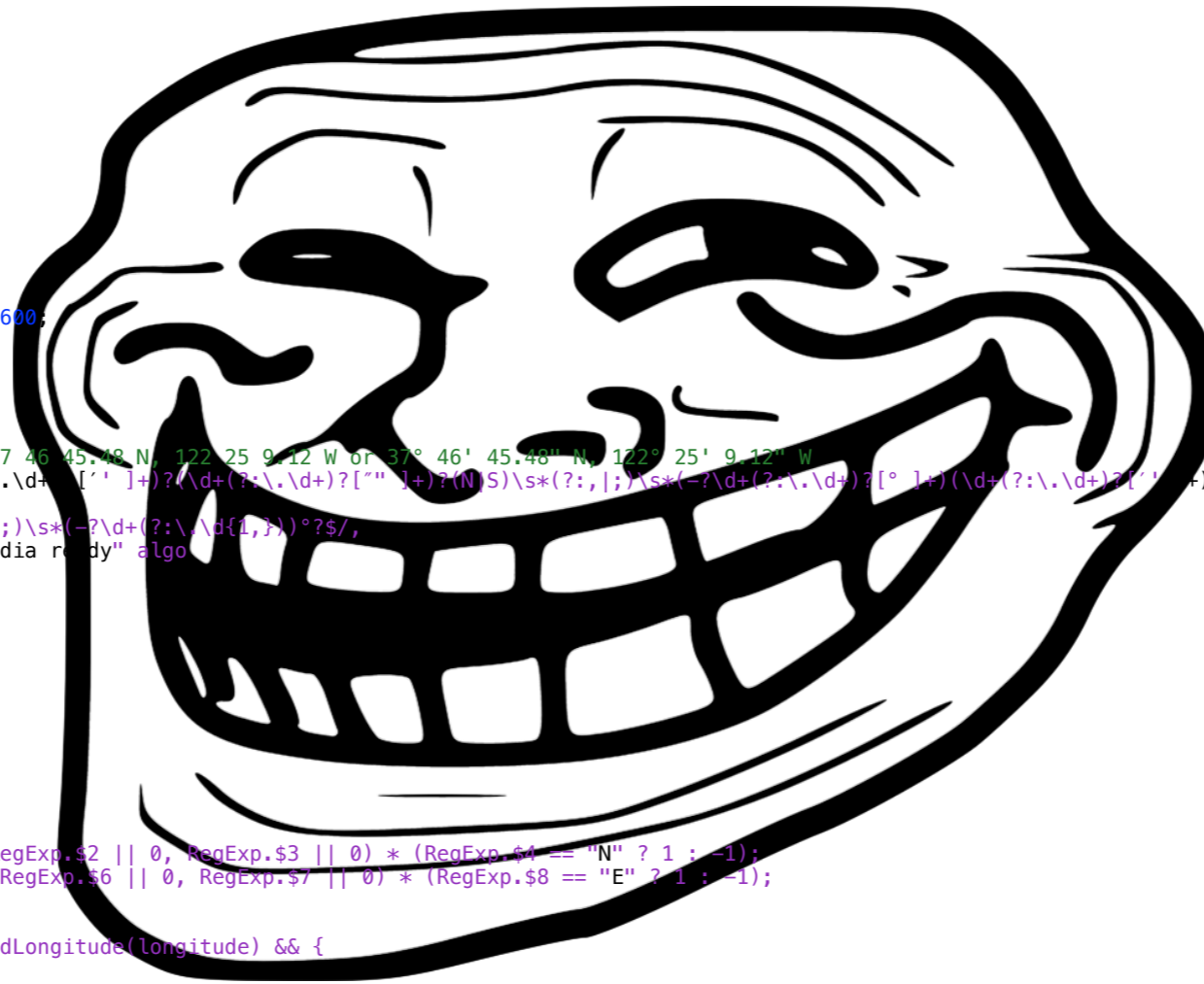
- `/ ^ (-? \d+ (?: \. \d{1,})) [°] ? [, ;] \s* (-? \d+ (?: \. \d{1,})) ° ? /`

WGS84 Map Coords RegExp

```
var WGS84ToObject = (function (RegExp) {  
    // (C) Andrea Giammarchi - WTFPL License ( http://sam.zoy.org/wtfpl/ )  
    // 338 bytes once minified and gzipped  
  
    // paranoid approach not implemented yet  
    function isValidLatitude(latitude) {  
        // return Math.abs(longitude) < 181  
        return latitude != null;  
    }  
  
    function isValidLongitude(longitude) {  
        // return Math.abs(longitude) < 87  
        return longitude != null;  
    }  
  
    function deg2latlon(deg, min, sec) {  
        return f(deg) + (f(min) * 60 + f(sec)) / 3600;  
    }  
  
    var  
        // shortcuts  
        f = parseFloat,  
        // 37° 46' 45.48" N, 122° 25' 9.12" W or 37 46 45.48 N, 122 25 9.12 W or 37° 46' 45.48" N, 122° 25' 9.12" W  
        degrees = /^(~?\d+(?:\.\d+)?[° ]+)(\d+(?:\.\d+)?[' ]+)?(\d+(?:\.\d+)?[" ]+)?(N|S)\s*(?:,|;)\s*(~?\d+(?:\.\d+)?[° ]+)(\d+(?:\.\d+)?[' ]+)?(\d+(?:\.\d+)?[" ]+)?(W|E)$/i,  
        // 50.345, 10.123 or 50.345°, 10.123°  
        decimals = /^(~?\d+(?:\.\d{1,})[° ]?(?:,|;)\s*(~?\d+(?:\.\d{1,})[° ]?)?$/i,  
        // simple trim, no need for the "enciclopedia ready" algo  
        trim = /\s+|\s+$/g  
    ;  
  
    return function WGS84ToObject(string) {  
        var latitude, longitude;  
        string = string.replace(trim, "");  
        switch(true) {  
            case decimals.test(string):  
                latitude = f(RegExp.$1);  
                longitude = f(RegExp.$2);  
                break;  
            case degrees.test(string):  
                latitude = deg2latlon(RegExp.$1, RegExp.$2 || 0, RegExp.$3 || 0) * (RegExp.$4 == "N" ? 1 : -1);  
                longitude = deg2latlon(RegExp.$5, RegExp.$6 || 0, RegExp.$7 || 0) * (RegExp.$8 == "E" ? 1 : -1);  
                break;  
        }  
        return isValidLatitude(latitude) && isValidLongitude(longitude) && {  
            latitude: latitude,  
            longitude: longitude  
        };  
    };  
})(RegExp);
```

WGS84 Map Coords RegExp

```
var WGS84ToObject = (function (RegExp) {  
    // (C) Andrea Giammarchi - WTFPL License ( http://sam.zoy.org/wtfpl/ )  
    // 338 bytes once minified and gzipped  
  
    // paranoid approach not implemented yet  
    function isValidLatitude(latitude) {  
        // return Math.abs(longitude) < 181  
        return latitude != null;  
    }  
  
    function isValidLongitude(longitude) {  
        // return Math.abs(longitude) < 87  
        return longitude != null;  
    }  
  
    function deg2latlon(deg, min, sec) {  
        return f(deg) + (f(min) * 60 + f(sec)) / 3600;  
    }  
  
    var  
        // shortcuts  
        f = parseFloat,  
        // 37° 46' 45.48" N, 122° 25' 9.12" W or 37 46 45.48 N, 122 25 9.12 W or 37° 46' 45.48" N, 122° 25' 9.12" W  
        degrees = /^(~?\d+(?:\.\d+)?[° ]+)(\d+(?:\.\d+)?[' ]+)?(\d+(?:\.\d+)?[" ]+)?(N|S)\s*(?:,|;)\s*(~?\d+(?:\.\d+)?[° ]+)(\d+(?:\.\d+)?[' ]+)?(\d+(?:\.\d+)?[" ]+)?(W|E)$/i,  
        // 50.345, 10.123 or 50.345°, 10.123°  
        decimals = /^(~?\d+(?:\.\d{1,})[° ]?(?:,|;)\s*(~?\d+(?:\.\d{1,})[° ]?)?$/i,  
        // simple trim, no need for the "enciclopedia ready" algo  
        trim = /^s+|s+$g  
    ;  
  
    return function WGS84ToObject(string) {  
        var latitude, longitude;  
        string = string.replace(trim, "");  
        switch(true) {  
            case decimals.test(string):  
                latitude = f(RegExp.$1);  
                longitude = f(RegExp.$2);  
                break;  
            case degrees.test(string):  
                latitude = deg2latlon(RegExp.$1, RegExp.$2 || 0, RegExp.$3 || 0) * (RegExp.$4 == "N" ? 1 : -1);  
                longitude = deg2latlon(RegExp.$5, RegExp.$6 || 0, RegExp.$7 || 0) * (RegExp.$8 == "E" ? 1 : -1);  
                break;  
        }  
        return isValidLatitude(latitude) && isValidLongitude(longitude) && {  
            latitude: latitude,  
            longitude: longitude  
        };  
    };  
})(RegExp);
```



JavaScript RegExp

... and that's pretty much it ...

Andrea Giammarchi

JavaScript RegExp

... and that's pretty much it ...

Andrea Giammarchi

- [MDN RegExp](#)
- [ragefac.es](#)
- [“cool story bro”](#)