

**e
es
e
ES2015**

es
s
6
B
+

A man is sitting on a dark couch, looking down at his hands. He is wearing a light-colored button-down shirt. In his right hand, he holds a dark glass bottle with a green label. His left hand rests on his lap. The background is a dimly lit room.

arrow
functions

arrow functions

```
var redLetterMedia = [  
  'Mike',  
  'Jay',  
  'Rich Evans'  
];  
  
redLetterMedia.forEach(  
  name => console.log( name )  
);
```

es5

```
redLetterMedia.map( function(name) {  
    return name.length;  
});
```

es6

```
redLetterMedia.map( name => name.length );
```

A close-up photograph of a person's upper torso, showing their chest and abdomen. Two thick, yellow arrows point from left to right across the center of the image. The top arrow is horizontal, and the bottom arrow is slightly lower and angled downwards to the right.

fat arrow operator

1 argument

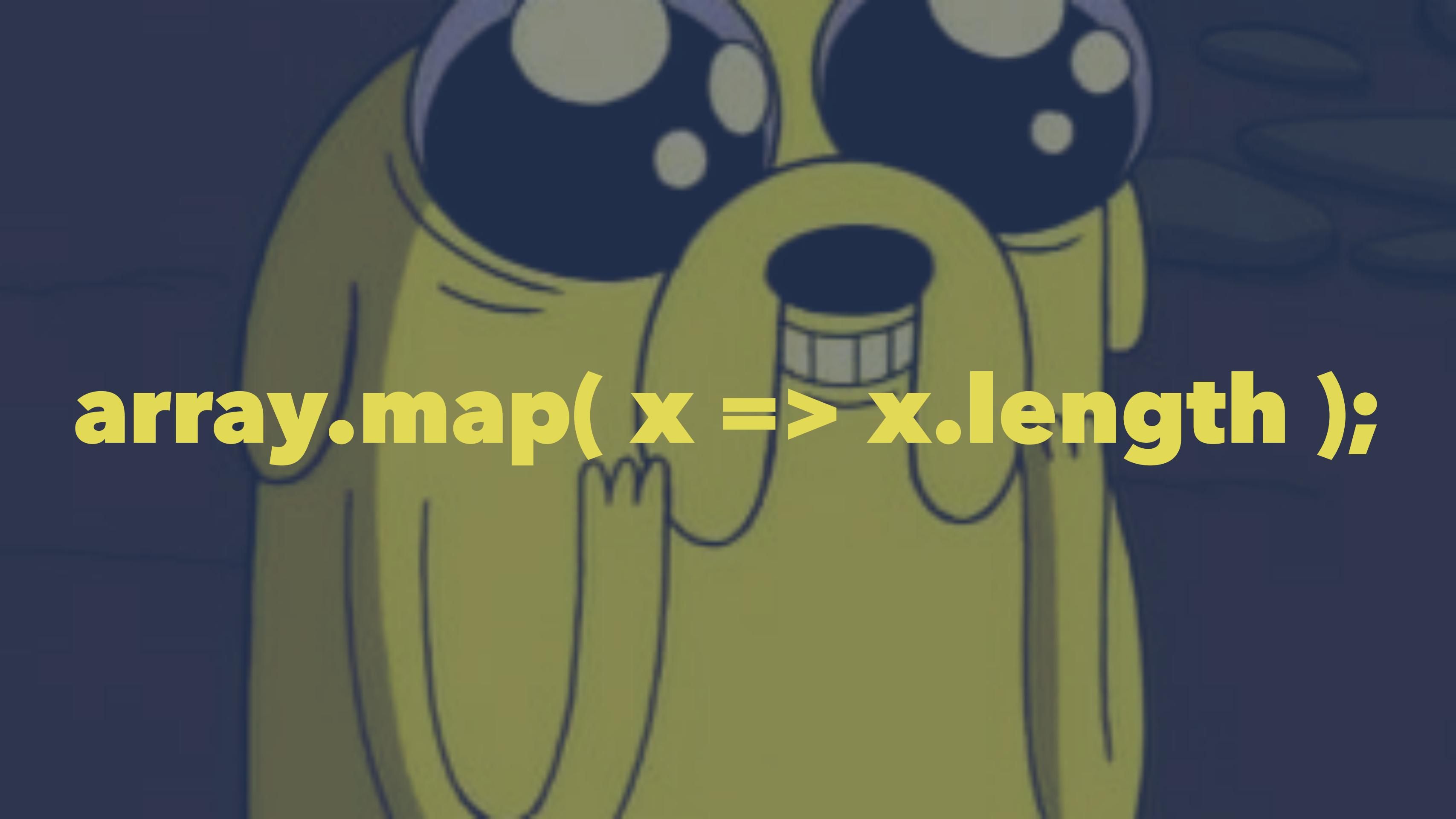
no function

no return, no woman, no cry?

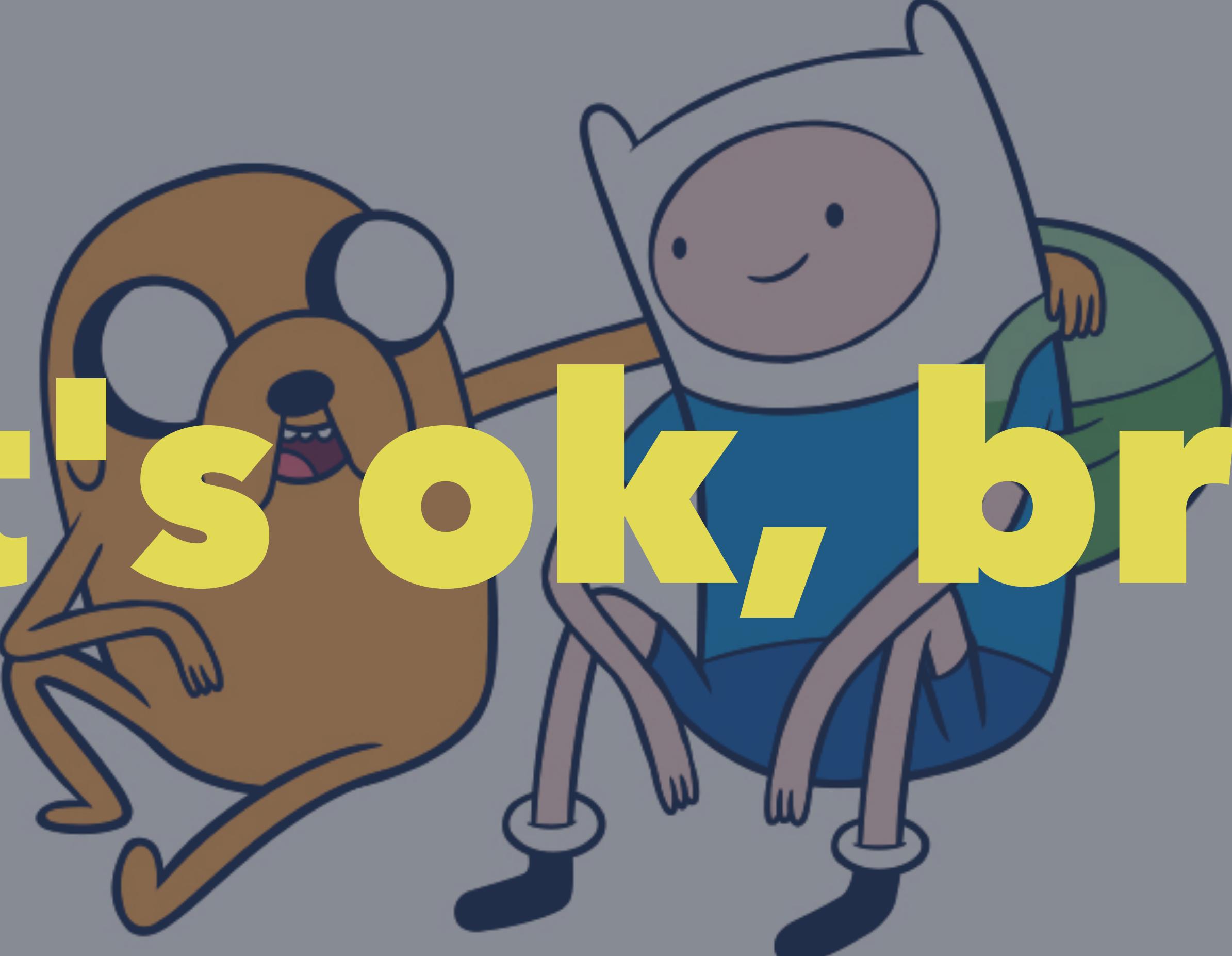
no { }

more arguments

```
redLetterMedia.map( (name, i) => {  
  let times = 100;  
  return name.length * times;  
} );
```

A person wearing a yellow raincoat and blue boots stands in a field of yellow flowers. They are holding a small blue object in their hand. The background is a dark, textured surface.

```
array.map( x => x.length );
```



it's OK, boro

functions get a **this**
parameter

it's value depends on function invocation

1.

METHOD, this = rich

```
var rich = {  
  name: 'Rich Evans',  
  disease: 'diabetes',  
  cry: function() {  
    console.log('rich cries!!')  
  }  
  
  rich.cry();
```

2.

FUNCTION, this = global object

```
function funeral() { };  
funeral();
```

window

or undefined

3.

CONSTRUCTOR, this = rich

```
var Fraud = function(name, disease) {  
    this.name = name;  
    this.disease = disease;  
}
```

```
var rich = new Fraud('Rich Evans', 'diabetes');
```

4.

CALL / APPLY

```
var redLetterMedia = [ 'Mike' , 'Jay' , 'Rich Evans' ];  
  
function kill () {  
    console.log( this );  
}  
  
kill(); // this = Window  
kill.call( redLetterMedia ); // this = redLetterMedia
```

arrow
no this

you gonna need some => homie

```
function Fraud() {  
    this.name = 'Rich Evans';  
    this.disease = 'diabetes';  
    this.health = 23;  
  
    setInterval( function fade() {  
        this.health--;  
        console.log(this.health);  
    } , 1000);  
}  
  
var rich = new Fraud();
```

```
setInterval( function fade() {  
    this.health--;  
} , 1000);
```

VS

```
setInterval( () => {  
    this.health--;  
} , 1000);
```

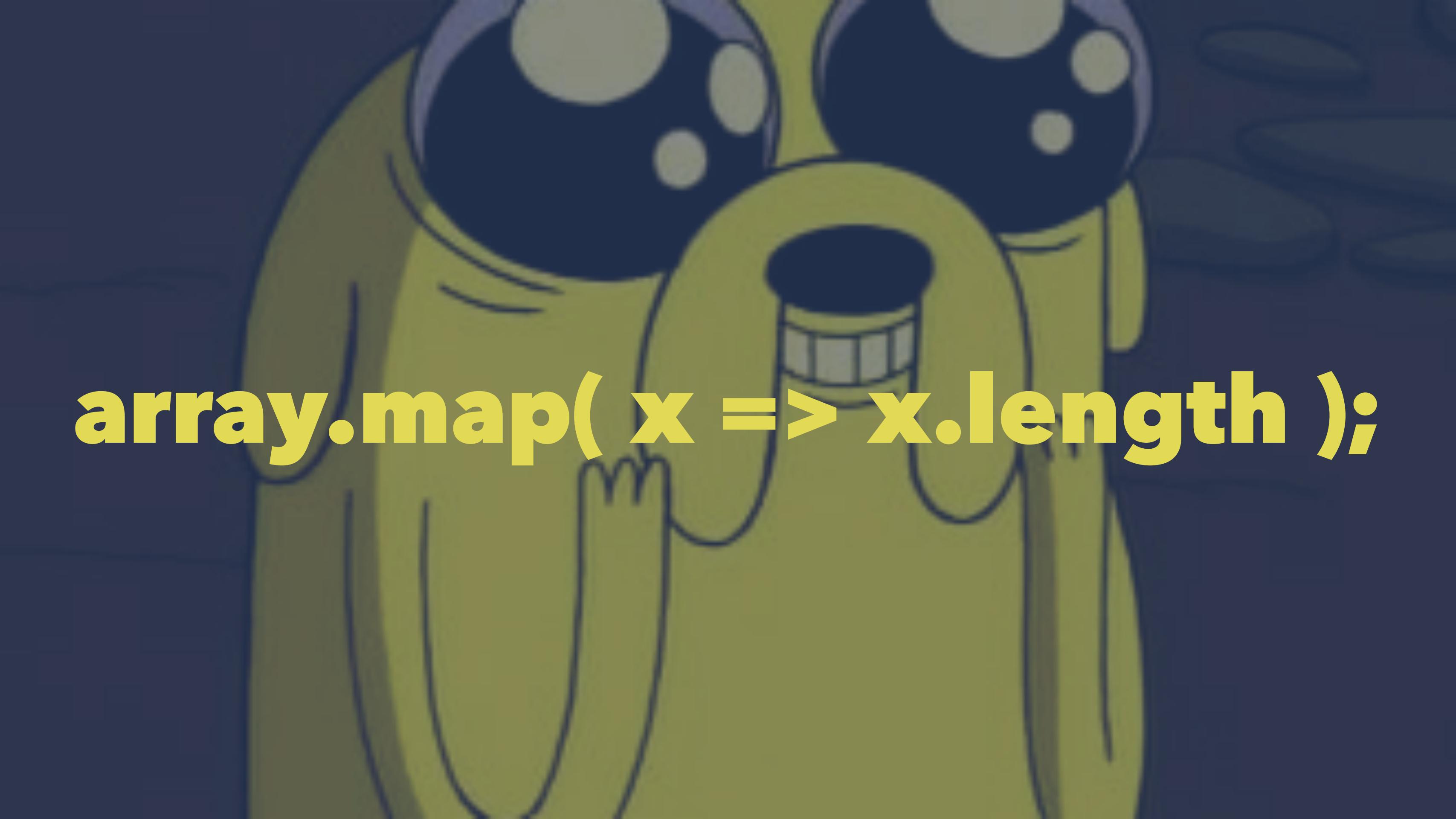
```
doStuff();
```

```
function doStuff() {  
    var one = 1;  
  
    if (one >= 1) {  
        var two = 2 * 1;  
    } else {  
        var two =  
    }  
  
    var result = one + two;  
    console.log( result );  
}
```

```
function doStuff() {  
    var one = 1, two, result;  
  
    if (one >= 1) {  
        two = 2 * 1;  
    } else {  
        two = 2;  
    }  
  
    result = one + two;  
    console.log( result );  
}  
doStuff();
```

arrow

no hoisting

A person wearing a yellow raincoat and blue boots stands in a field of yellow flowers. They are holding a small blue object in their hand. The background is a dark blue sky.

```
array.map( x => x.length );
```



block
scope

es5 only has **function (scope)**

```
function myDudes() {  
    var RLM = [  
        'Mike',  
        'Jay',  
        'Rich Evans'  
    ];  
  
    return RLM;  
}
```

```
RLM; // can't touch this
```

es6 adds { **block scope** }

```
var RLM = [ 'Mike', 'Jay' ];
```

```
{  
  let disease = 'AIDS';  
}
```

```
disease; // can't touch this
```

keyword **let**

or **const**

*implicit **block***

```
if ( bool ) {  
    let thing = 'a boob';  
    console.log(thing);  
}
```

*explicit **block***

```
if ( bool ) {  
{  
    let thing = 'a boob';  
    console.log(thing);  
}  
}
```

let me loop

```
for ( let i = 0; i < 10; i++ ) {  
    console.log( i );  
}  
  
console.log( i ); // ReferenceError
```

es6 kicking ass

```
for ( let i=1; i<=5; i++ ) {  
    setTimeout( () =>  
        console.log(i),  
        i * 1000 );  
}
```

now try that in es5

in es6 you can use a **block**

instead of this

```
(function() { // IIFE

    var name = 'My Library';
    var version = 0.1;
    var fun = function() {}

    window.yabRary = {
        name: name,
        version: version,
        fun: fun
    }

}());
```

A medium shot of a man with short brown hair, wearing a white headband and a red and white striped shirt. He is looking down at a dark-colored keyboard. The background is slightly blurred, showing what appears to be a room with some furniture and possibly a window.

const

vs let vs var

const me up, before you go go

```
const RLM = [  
  'Mike',  
  'Jay',  
  'Rich Evans'  
];
```

```
RLM.forEach(  
  name => console.log( name );  
);
```

can't re-assign a const

```
const tax = 0.5;  
tax = 0.2; // error
```

but you **can** change a const array

```
const RLM = [ 'Mike', 'Jay' ];  
RLM.pop();
```



object literal extensions

```
let name = 'Marceline';
```

es5

```
let player = {  
    name: name,  
    getName: function() {  
        return this.name;  
    }  
};
```

es6

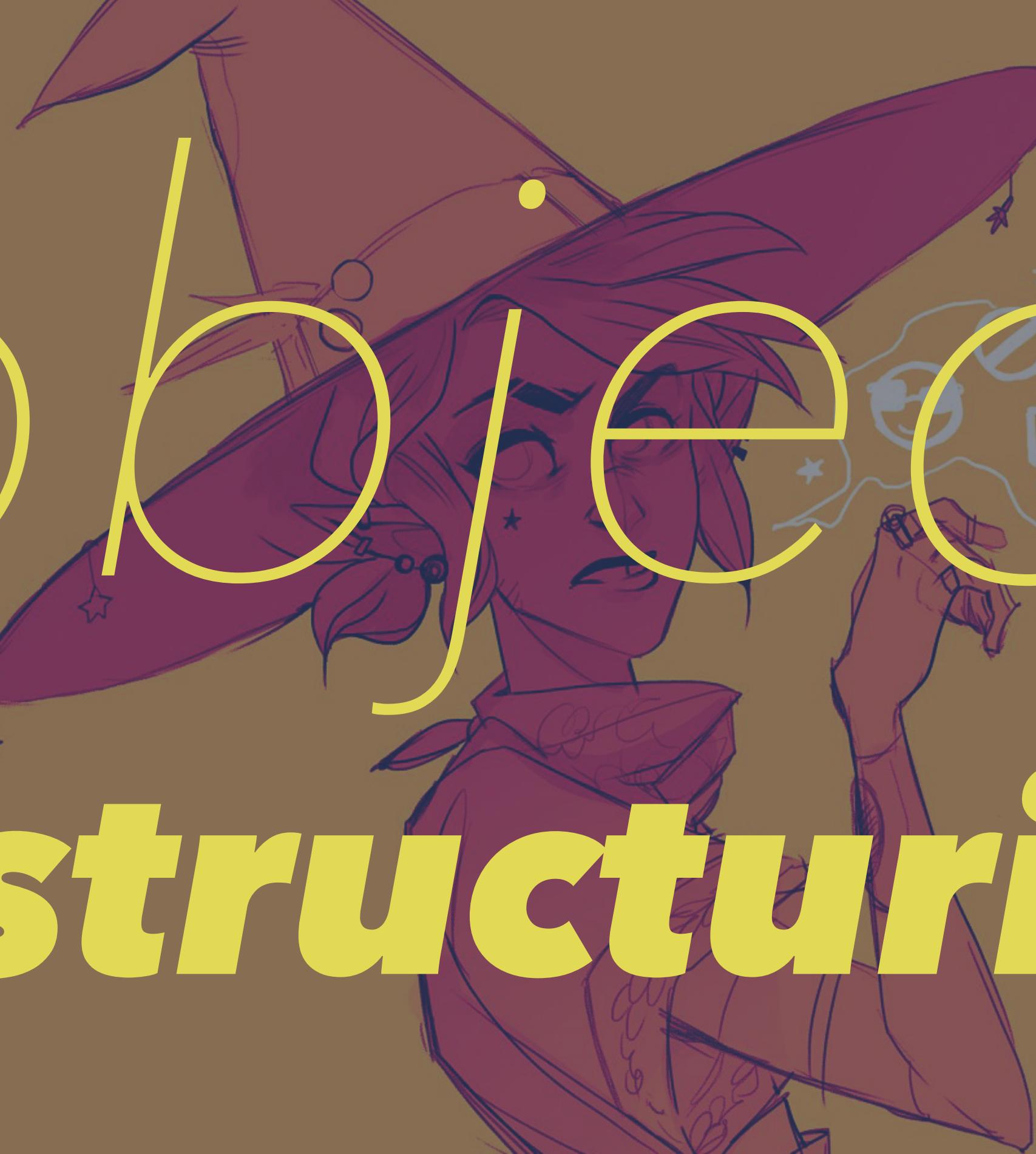
```
let player = {  
    name,  
    getName() {  
        return this.name;  
    }  
};
```

computed property names

```
let name = 'Marceline';
```

```
let player = {  
  name,  
  getName() {  
    return this.name;  
  },  
  [ 'pretty' + name ]: true // <- this bit here  
};
```

```
player.prettyMarceline; // true
```



object

destructuring

object **destructuring**

```
let hornyBoy = {  
    name: 'Taako',  
    race: 'Elf',  
    weapon: 'Umbra Staff',  
    level: 12  
}  
  
let { name, level } = hornyBoy;
```

even with **functions** and **arrays**

```
// function params  
function getData( { name, level } ) {  
  console.log( name + ' ' + level );  
}
```

```
getData( hornyBoy );
```

```
// array  
const RLM = [ 'Mike', 'Jay', 'RichEvans' ];  
let [ one, two, three ] = RLM;
```



rest

spread

...**rest** parameters

```
function doMath( ...rest ) {  
  let sum = 0;  
  rest.forEach( num => sum += num );  
  return sum;  
}
```

```
doMath( 1, 2, 3 )
```

...rest parameters have to go **last**

```
function doMath( times, ...rest ) {  
  return rest.reduce(  
    ( sum, value ) => sum + value  
  ) * times;  
}
```

the opposite of rest, **spread**

```
const adventureZone = [  
  'Magnus',  
  'Taako',  
  'Merle'  
];
```

```
function spread( one, two, three ) {  
  console.log(one, three);  
}
```

```
spread( ...adventureZone );
```



default

parameters

defualt parameters

```
function discount(price = 10, perc = 10*2, tax = getTax()) {  
    // fancy math...  
}
```

can be

- a value
- an expression
- result of a function

template

strings

KOLOWRAT

multi-line, "smart"

```
let title = 'hot piss';
```

```
`<article>
  <h1>${title}</h1>
</article>`
```

```
let className = 'active';
`<img class="${className}">`
```

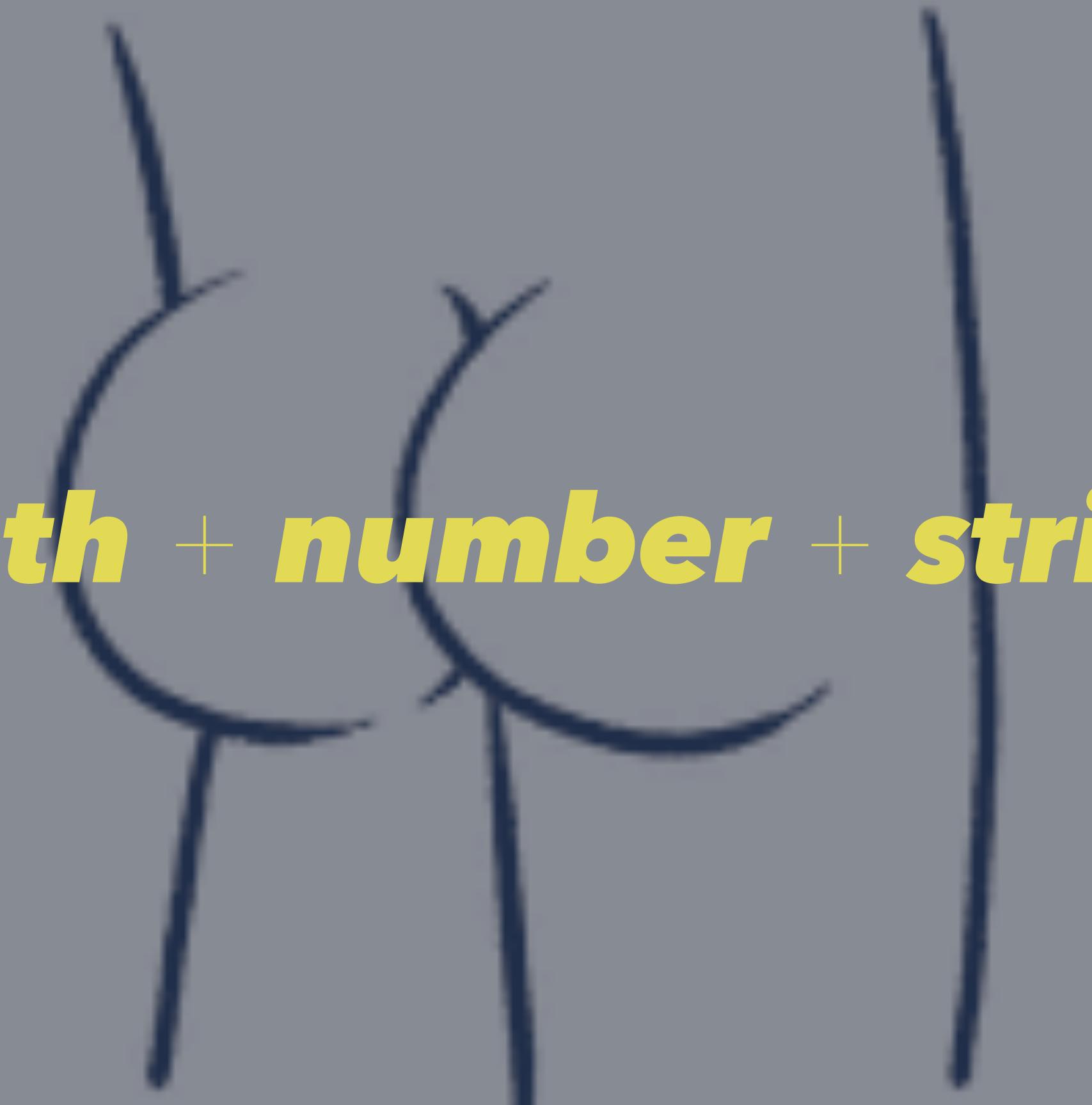
use expressions, still just a string

```
`Pieseň od ${artist} sa volá ${song} a 2 + 2 je ${ 2+2 }`
```

`

```
Pieseň od ${artist}  
sa volá ${song}  
a 2 + 2 je ${ 2+2 }
```

```
` .toUpperCase();
```



new ***math*** + ***number*** + ***string*** stuff

String APIs

```
'I suck butts.'.includes('suck')  
'I suck butts.'.startsWith('I suck')  
'I suck butts.'.endsWith('ts.')
```

```
let name = 'Vasho';  
`<li>${name}</li>`.repeat(5)
```

Number / Math APIs

`Number.EPSILON`

`Number.isFinite()`

`Number.isInteger()`

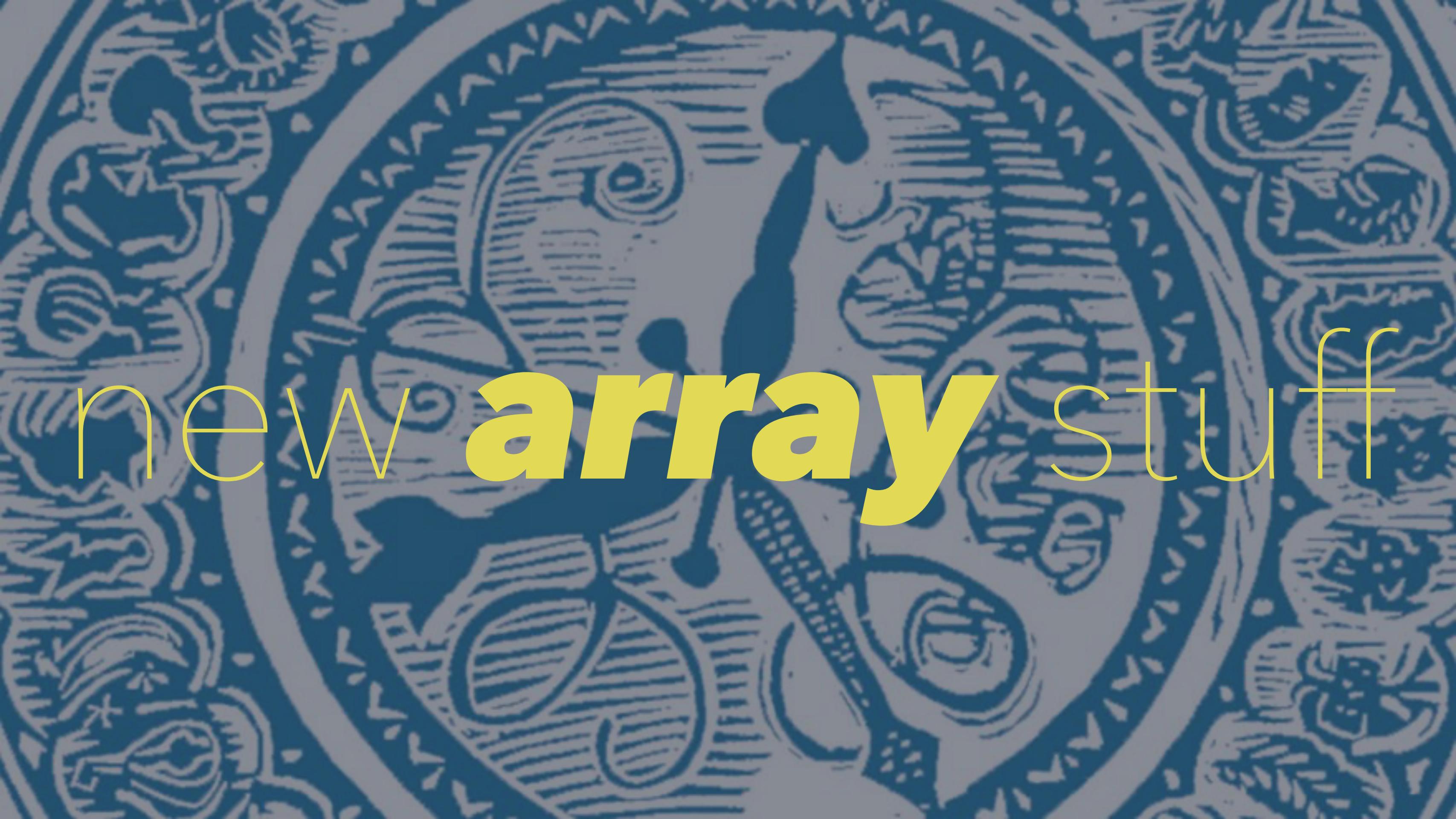
`Number.isNaN()`

`Math.cosh()`

`Math.sinh()`

`Math.hypot()`

...



new array stuff

Array APIs

```
// Jay in array?  
const RLM = [ 'Mike', 'Jay', 'Rich Evans' ];
```

```
RLM.includes('Jay');
```

```
// find 1st with name longer than 4
```

```
RLM.find( name => name.length > 4 )
```

```
// find his location
```

```
RLM.findIndex( name => name.length > 4 )
```

Array APIs

```
// check MDN for these
Array.from(), Array.of()
[].fill(), [].copyWithin()

// does ANYONE have more than 4 letters?
RLM.some( name => name.length > 4 )

// does EVERYONE?
RLM.every( name => name.length > 4 )
```

Array APIs

```
// any admins in da house?  
const users = [  
  { name: 'Marceline' },  
  { name: 'Lyra', administrator: true },  
  { name: 'Matilda' }  
];  
  
users.some( user => 'administrator' in user );
```

```
// iterator? wat??  
users.values()  
users.keys()  
users.entries()
```

A group of nine people, including adults and children, from the television series 'The Wonder Years'. They are posed together in a living room setting. In the foreground, two boys are seated at a piano. Behind them, a woman sits on a couch, and several men and women stand behind her. All are smiling and looking towards the camera.

iterators
for...of

Iterator

```
const RLM = [ 'Mike', 'Jay', 'Rich Evans' ];  
let iterator = RLM.entries();
```

```
// gimme the next one  
iterator.next();
```

```
// and so on  
iterator.next().value;
```

```
// are we done?  
iterator.next().done;
```

for...of loop

```
// loop through iterator
for ( let val of RLM.entries() ) {
    console.log( val );
}
```

```
// it loops through everything
(function(){
    for ( let arg of arguments ) console.log( arg );
}(1, 2, 3));
```

A photograph of three young women with long dark hair, all wearing short, light-colored skirts and dark tops. They are posed in a line, facing the camera with their backs slightly to the left. The woman on the left has her hands on her hips. The woman in the middle is leaning forward with her hands on her hips. The woman on the right is also leaning forward. They are all smiling. The background is a plain, light color.

generator
functions

a function **you can step through**

```
// * = generator
function *kittenMe() {
    yield 'Liz McClarnon';
    yield 'Kerry Katona';
    yield 'Heidi Range';
}
```

```
// returns an iterator object
let iter = kittenMe();
```

```
// .next() starts the dance
iter.next();
iter.next().value;
```

id me, please

```
function *idGenerator() {  
    let id = 0;  
  
    // this is fine (in generators)  
    while ( true ) {  
        yield ++id;  
    }  
}  
  
let gen = idGenerator();  
gen.next();  
gen.next();
```

A close-up portrait of a man with dark hair and a beard. He is wearing a light-colored, textured jacket over a dark shirt. His right hand is raised to his face, with his fingers resting near his eye, suggesting a thoughtful or melancholic mood. The background is a solid, dark navy blue.

Promise

promise

a value we **don't have yet** but will **in the future**

```
let data = fetch(  
  'https://itunes.apple.com/search?term=kolowrat'  
);
```

```
data.then( res => res.json() )  
  .then( out => console.log(out) );
```

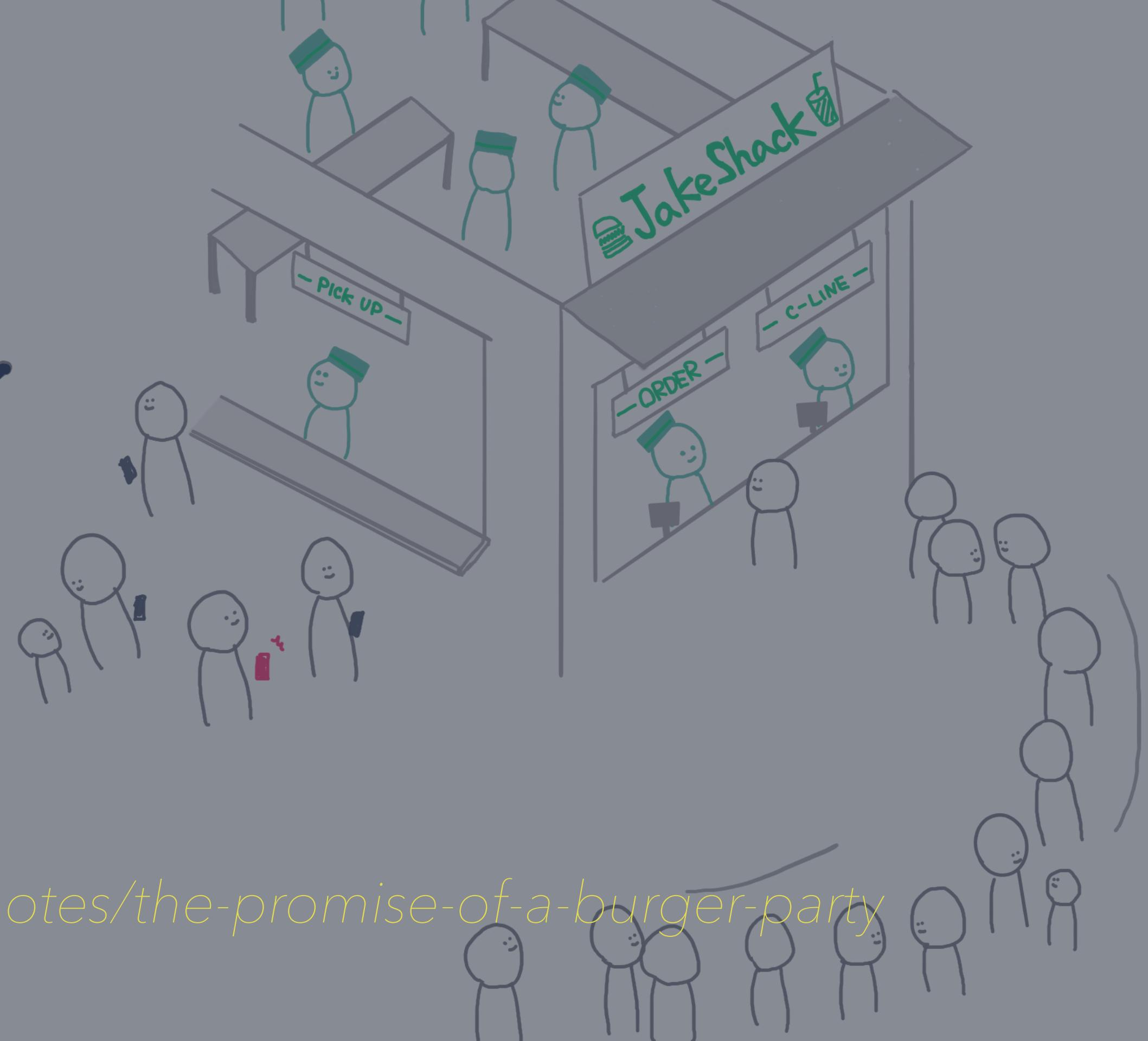
when we get the answer **then()** we do stuff

The Promise of a **BURGER PARTY**

written by @kosamari

A quest to understanding
JavaScript Promise

<https://kosamari.com/notes/the-promise-of-a-burger-party>



promise template

```
var promise = new Promise(function(resolve, reject) {  
  // do a thing, possibly async, then...  
  
  if /* everything is ok */ {  
    resolve("Stuff worked!");  
  }  
  else {  
    reject(Error("It broke"));  
  }  
});
```

how you **use it**

```
// many libraries are built on promises  
// this is how you USE them
```

```
promise.then(function(result) {  
  console.log(result); // "Stuff worked!"  
});
```

```
promise.catch(function(err) {  
  console.log(err); // Error: "It broke"  
});
```

let's imagine **get()** fetches json

```
// these do the same thing
let promise = get('search?term=kolowrat');
promise.then( res => 'yay!' )
    .catch( err => 'oh:( ' );
```

```
// or
promise.then(
    res => 'yay!', err => 'oh:( '
);
```

```
// or even
get('search?term=kolowrat')
    .then( res => 'yay!' )
    .catch( err => 'oh:( ' )
```

Promise.all() / Promise.race()

```
// starts timer, returns promise
function start(id, time) {
  return new Promise( (res, rej) => setTimeout( () => res(`#${id} done`), time ) );
}

// start 3 timers
let timers = [ start(1, 4000), start(2, 1000), start(3, 2500) ];

// when ALL finish
Promise.all( timers ).then( res => 'all done' )
  .catch( err => 'damn:' );

// when FIRST finishes
Promise.race( timers ).then( res => '1st found' )
  .catch( err => 'damn:' );
```

```
get('/comments')
  .then( JSON.parse ).then( res => get('/posts/' + res[0].postId) )
  .then( JSON.parse ).then( res => get('/users/' + res.userId) )
  .then( JSON.parse ).then( console.log )
  .catch( err => console.error(err));
```

```
func1(function(val1) {
  func2(val1, function(val2) {
    func3(val2, function(val3) {
      func4(val3, function(val4) {
        func5(val4, function(val5) {
          // Do something with val5
        })
      })
    })
  })
})
```





asyncawait

write **async code** like it was **sync**

goodbye, callback hell (you piece of shit)

```
(async function() {  
  try {  
    let comments = await get('/comments');  
    let post = await get('/posts/' + comments[0].postId);  
    let user = await get('/users/' + post.userId);  
  
    console.log( user.name );  
  }  
  catch (err) {  
    console.log('nope', err);  
  }  
}())
```

sequential

```
// takes 6,5 seconds
let timer1 = await start(1, 4000);
let timer2 = await start(2, 2500);
```

VS. ***parallel***

```
let timer1 = start(1, 4000);
let timer2 = start(2, 2500);
```

```
// takes 4 seconds
await timer1; await timer2;
```

An aerial photograph of Reykjavik, Iceland, showing a dense grid of buildings and green spaces. In the top left corner, there is a decorative graphic element featuring a banner with the word "REYKJAVÍK" written on it in a stylized font. A small white airplane is flying over the banner. The city is built along a coastline with several docks and harbors where boats and ships are visible.

map & set

SET

collection of **unique** values

```
let tags = new Set([
  'petite',
  'ebony',
  'ebony',
  'scat'
])
```

```
set.forEach( val => console.log(val) ) // petite, ebony, scat
```

```
for ( let val of set ) {
  console.log(val) // petite, ebony, scat
}
```

SET

collection of **unique** values

```
let set = new Set()  
set.add('one').add('two').add('two') // one, two
```

```
set.size          // 2  
set.delete('two') // one  
set.has('two')   // false
```

```
// remove duplicates from array  
var arr = [...new Set([1, 3, 2, 3, 3])]
```

MAP

key-value pairs where **key** can be **anything**

```
let logo = document.querySelector('#logo')
```

```
// the key can even be an HTML node
var meta = new Map()
meta.set(logo, {
  animated: false,
  nice: true
})
```

MAP (dictionary)

```
for ( let item of meta.keys() ) { // meta.values(), meta.entries()
  console.log( item )
}
```

```
meta.size
meta.has(logo)
meta.delete(logo)
meta.clear()
```

WeakMap

```
// can be garbage collected
let weak = new WeakMap().set(logo, { 'nice': true });
```

A photograph of a person from the chest up. They are wearing round-rimmed glasses and a light-colored shirt with a diagonal striped pattern in shades of pink, purple, and yellow. The person is looking down intently at a small, dark object held in their hands. The background is dark and out of focus.

Symbo

Symbol

for creating **unique names** for **object properties**

```
let one = Symbol('hey')
let two = Symbol('hey')
one === two // false
```

```
// these won't overwrite each other (good for libraries)
{
  name: 'vasho',
  [Symbol('name')]: 'jquery',
  [Symbol('name')]: 'beyoncé',
  [Symbol('name')]: 'wat'
```



getters

setters

Getter

calls a **function** when you access a **property**

```
let legend = {  
    first: 'Vašo',  
    last: 'Patejdl',  
  
    // you can format things before output (like time & date)  
    get fullName() {  
        return this.first + ' ' + this.last;  
    }  
}  
  
legend.fullName // Vašo Patejdl
```

Setter

calls a **function** when you try to **set a property**

```
let legend = {  
    first: 'Vašo',  
    last: 'Patejdl',  
    oldness: 31,  
  
    // good for VALIDATION  
    set age( val ) {  
        if ( ! Number.isInteger(val) ) return 'error';  
        this.oldness = val;  
    }  
}  
  
legend.age = 'balls' // error  
legend.age = 69 // 69
```



Kory

Proxy

add **validation** or **behaviour** to objects

```
let legend = {  
    first: 'vašo',  
    last: 'patejdl',  
    age: 31  
}  
  
// set a TRAP for when a property gets set  
legend = new Proxy(legend, {  
    set: function( target, name, val ) {  
        if ( name === 'age' ) {  
            if ( ! Number.isInteger(val) ) return;  
        }  
  
        target[name] = val;  
    }  
});
```

other
stuff



exponentiation

```
2 ** 6 // Math.pow(2, 6)
```

.padStart() .padEnd()

console.log('0.00'.padStart(20))	0.00
console.log('10,000.00'.padStart(20))	10,000.00
console.log('250,000.00'.padStart(20))	250,000.00

better unicode

```
// es5
let text = '🐴👤❤️';
text.length; // 5? wat

for (let i = 0; i < text.length; i++) {
  console.log(text[i]); // ❓❓❓❓❤️
}
```

```
// es6+
for (let char of '🐴👤❤️') {
  console.log(char) // 🐾👤❤️
```

A photograph of a classroom interior. The room has a high ceiling with a grid of wooden beams. Large windows are visible in the background, letting in natural light. Rows of wooden desks and chairs are arranged facing forward. In the background, there are shelves with various items and a small stage area with a red curtain.

class

class syntax

```
class Human {  
    // properties  
    constructor( first, last, age ) {  
        this.first = first;  
        this.last = last;  
        this.age = age;  
    }  
  
    // getters setters  
    get fullName() {  
        return this.first + ' ' + this.last;  
    }  
  
    // methods  
    who() {  
        return `${this.fullName} is ${this.age} years old.`;  
    }  
}
```

create **new** objects

```
// use the NEW keyword
let people = [
  new Human( 'Mária', 'Trošková', 15 ),
  new Human( 'Tomáš', 'Maštalír', 33 ),
  new Human( 'Tim', 'Heidecker', 40 )
];
```

```
// iterate over them
people.map( person => person.who() )
```

inheritance (extends Class)

```
class Legend extends Human {  
    constructor( first, last, age ) {  
        super( first, last, age ); // call constructor of parent  
        this.hits = [ 'Voňavky dievčat', 'Kamarátka nádej', ... ]; // add new props  
    }  
  
    // add new methods  
    writeHit() {  
        let hit = _.random(this.hits);  
        return super.who() + `\\n Just wrote ${hit}.`; // call .who() of the parent  
    }  
}  
  
let legend = new Legend( 'Vašo' , 'Pat' , 33 );  
legend.who();
```

rm odie es

Modules

because we wanna forget these

AMD, UMD, CommonJS,
RequireJS, Browserify...

```
// and this
var ModulePattern = (function () {

    return {
        publicMethod: function () { // code }
    };
})();
```

import / export

lets us split js into **multiple files** each has its **own scope**

```
// math.js
export let bestNumber = 69;
export function doMath(...rest) {
  return rest.reduce((sum, val) => sum + val);
}
```

```
// app.js
import { bestNumber, doMath } from "./math.js";
import * as math from "./math.js";
```

import / export

```
// rename stuff (works with export too)
```

```
import {  
    bestNumber as nice,  
    doMath as sum  
} from "./math.js";
```

```
console.log( nice, sum(1,2,3) );
```

```
// DEFUALT export, skip the { } in import
```

```
export default class Mario { ... }
```

```
import Mario from "./math.js";
```

great job!

up to @ es6 now





