
hello

# Roman Hraška

@yablko


WEBREBEL.SK
full-stack web developer
teach at learn2code.sk
HLÁSENÝCH JE UŽ
5333
REBELOV
1. KOMPLETNÝ ONLINE KURZ
WEBDESIGNU
VYSKÚŠAJ ZDARMA


doin the bizness
very import
#money

bit.ly/badassSlides

**bit.ly/badassExamples**

SO **JAVASCRIPT**

# SO JAVASCRIPT

- most popular / widely used
- most misunderstood

# SO JAVASCRIPT

- in every browser, computer, mobile
- **used to control browser, manipulate displayed content**
- but it does much more
- as we shall see

**but first let's get this straight**

javascript is

**RIDICULOUS**

# NO IT AIN'T

but things about it are

*like the name*

*like the name*

**not java**

# birth

1995 in Netscape Navigator 2
under the name **LiveScript**

but java was hot
so they **stole her name**

syntax based on **C**
but language based on **Scheme** and **Lips**

# the name is dumb, but **check these babies out**

ok

BUT KNOW **THIS**

**javascript has bad parts in it,**
**but don't judge**

*it had it hard*

**input validation**
into netspace **after days**
**adopted immediately,** globally
copied, remade, renamed, **stolen**, remade again
survived **browser wars**

# ECMA

*script*

**it has plenty or reasons to suck**

*but it doesn't*

**nifty language / beautiful things**

**YOU GUYS** *WERE SUPPOSED TO BE* **THE SAVIORS**

# java applets??

*who even*

**javascript won by default**

*while starting out,*
*unfinished, flawed*

**and was hated for it**


UNTIL RECENTLY

**frameworks** *showed us* **fun**
*we* **understood DOM's** *the evil one*

*we got to know javascript*

# node, angular, meteor

*"what isn't js used for today?"*

# SOME THINGS ARE OBVIOUS

- browser extensions
- osx, windows, google widgets & gadgets
- nosql
- adobe creative suite
- unity3d

JS

# SOME MORE INTERESTING

- **brackets.io** or **atom.io**
- tools to make desktop apps: **appjs.com**
- tools to make mobile apps: **cordova.apache.org** **phonegap.com**
- game frameworks **phaser.io** **sparklinlabs.com**

**SO**

YOU KNOW WHAT'S

**COOL?**


LASERS

# ALSO, ROBOTS

- nodebots.io
- cylonjs.com
- github.com/rwaldron/johnny-five

*so after we're all batteries*

**javascript did it**

*javascript is* ***blowing up***


*certainly* ***worth your time***


bonsaiden.github.io/JavaScript-Garden/
developer.mozilla.org/en-US/ ( NEVER w3schools )
movethewebforward.org
html5rocks.com/en
dev.opera.com/articles
developers.google.com/web/fundamentals

# WE USED TO HATE JS

- because of the DOM
- because ***it looks like*** other languages

# there are pitfalls

*pitfalls that java might not have*

# BUT

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, world!");
    }
}
```

*http://en.wikipedia.org/wiki/ListofHelloworldprogram_examples#Java*

# AND

*java*

```
Arrays.sort(values,new Comparator<Integer>(){
  public int compare(Integer val1, Integer val2) {
    return val2 - val1;
  }
});
```

*javascript*

```
values.sort(function(val1,val2){ return val2 - val1; });
```

# *KEY IDEAS*

- **load and go delivery** (programs delivered as plain text)
- **loose typing**
- **objects as general containers**
- **prototypal inheritence**
- **lambda** (functions as first class citizens)
- linkage through global variables

*now even java's getting functions!*

**NOT** *OOP?*

DUDE

# EVERYTHING

*is an object here*

*quick* **overview**

# loose typing

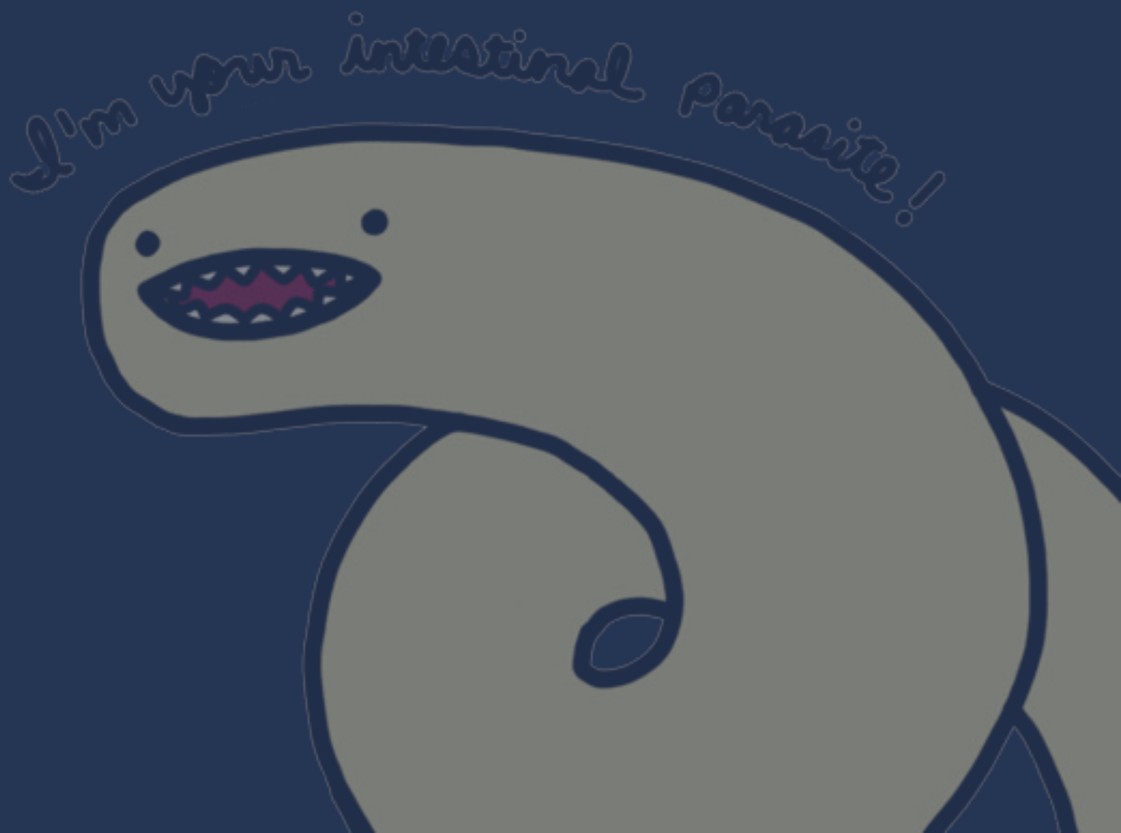
```
var name = 'balls';

if ( name ) {

}
```

# objects are sweet

```
var dude = {
    who: 'Ice King',
    wat: 'Armed with a magic crown and an icy heart, \
    the Ice King has only one goal: to secure a wife \
    by any means necessary.'
}

dude.colors = ['blue', 'white'];
```

# inheritance works differently

```
function person(name) {
    return {
        myName: function() {
            console.log(name);
        }
    }
}

function student(name) {
    var that = person(name);
    that.cry = function() {
        console.log(name + ' cries hard');
    }
    return that;
}
```



# global variables

*these just suck*

**but we will fix that**

# *JAVASCRIPT TYPES*

- **String**
- **Number**
- **Boolean**
- **Object**

also special ones like **undefined** and **null**
and also **Array**, **Math**, **Date**, **RegExp**, **Function**

# *so actually more like*

- **String**, **Number**, **Boolean**, **null**, **undefined**
- **Object**
  - Function
  - Array
  - Date
  - Math
  - RegExp

# STRINGS

# STRINGS

- 16-bit unicode
- there is no **char**

```
"balls".length // 5
"balls".charAt(1) // "a"
"balls"[1] // "a"
```

- immutable

```
"i get by, with a little help"
"i get by, with a little help".replace("by", "high")
```

ONLY ONE

**NUMBER**

TYPE

2

2

2.5

2

+

2.5

# let the cumputer count

*that's what it's there for*

```
2 + 2.5; // 4.5


5 / 3; // 1.6666666666666667


var e = 1e2;
e; // 100
```

64-bit floating point

**double**

# 64-bit floating point

# **double**

```
0.1 + 0.2
0.30000000000000004
```

**(0.1*10 + 0.2*10) / 10**

+(0.1+0.2).toFixed(10)

# a bit of fun

```
var str    = "3.14159",
    num    = +str,    //  3.14159
    str    = ""+num, //  "3.14159"
    int    = ~~str,  //  3
    float  = 1*str,  //  3.14159
    bool   = !!str;  //  !!num; // true


var a = +prompt("Gimme the first number:"),
    b = +prompt("Gimme the second one:"),
    sum = a + b;
```

# Math object

.abs(), .floor(), .max(), .min(), .random(), .sin(), .cos() ...

**you have to do**

```
Math.abs(-5); // 5
```

**cuz this won't work:(**

```
-5.abs();
```

# parseInt(str, base)

```
var dom = document.getElementsByTagName('code')[0],
    css = getComputedStyle(dom);

css.fontSize; // "14px";
dom.style.fontSize = css.fontSize + 10; // "14px10"


var newSize = parseInt(css.fontSize, 10) + 14;
dom.style.fontSize = newSize + 'px';
```

# + does **addition** and **concatenation**

```
3 + 4 + "5"; // "75"
3 + 4 + (+"5") // 12
```

# NaN

there is number called **not a number**

**because javascript**

```
parseFloat('balls'); // NaN
12 + NaN; // NaN
NaN + NaN; // false

typeof NaN; // number
```

# INFINITY

1/0
0/1

```
console.dir(Number)
```

# BOOLEAN

## **true** and **false**

but expressions eval as **truthy** or **falsey**

# falsey

- false
- null
- undefined
- The empty string "" ( '')
- The number 0
- The number NaN

everything else **truthy**

**even**

```
if ([]) { // empty array

}

if ({}) { // empty object

}
```

# goes like dis

```
                0 == false
parseInt("foo") == false // NaN is falsy
                1 == true
               -1 == true  // -1 is truthy

               "" == false // empty string is falsy
            "foo" == true  // non-empty string is truthy
          "false" == true  // ...even if it contains a falsy value

       window.foo == false // undefined is falsy
             null == false // null is falsy

               {} == true  // an (empty) object is truthy
               [] == true  // an (empty) array is truthy; PHP programmers beware!
```

# == vs === ( != vs !== )

```
3 === 3   // true
3 === '3' // false
```

# >=, <=

```
"adventure time" > "my little pony" // false
```

# && logical and

## short-circuit evaluation

**if first is truthy, result is the second**

**else result is the first**

```
true && "balls" // balls

obj && obj.member
```

# || logical or

## also called **default** operator

**if first is truthy, result is first**

**else result is second**

```
var name = firstName || "Balls";
```

```
<script>window.jQuery || document.write('<script src="js/jquery.min.js"><\/script>')</script>
```

# null

a value that isn't anything

# undefined

isn't even that

# VARIABLES

declared using keyword **var**

```
var a; // typeof a == 'undefined'
var name = "Buttpoop";

// can also be comma separated
var a,
    name = "Buttpoop",
```

# VARIABLES

declared using keyword **var**

```
var a; // typeof a == 'undefined'
var name = "Buttpoop";

// can also be comma separated
var a,
    name = "Buttpoop",
```

# SUPER IMPORTANT

if you forget var, **variable gets created as global**

# unlike elsewhere

## blocks do not have scope

but in ES6

```
if ( adventure === 'time' ) {
  let jake = 'yellow'; // The scope is inside the if-block
  var finn = 'blue';   // The scope is inside the function
}
```

```
doStuff();

function doStuff() {
    var one = 1;

    if (one >= 1 ) {
        var two = 2 *
    } else {
        var two = 2;
    }

    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();
```

# hoisting

```
doStuff();

function doStuff() {
    var one = 1;

    if (one >= 1 ) {
        var two = 2 * 1;
    } else {
        var two = 2;
    }

    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();
```

```
// so even this is bit dumb, but i'm still gonna do it
for (var i = 0; i<10; i++) {
    // super awesome code
    // will totally blow your hair back
}


// anyway, any idea why this won't work?
for (var i = 1; i <= 5; i++) {
    setTimeout(function() {
        console.log("i:", i); // ¯\_(ツ)_/¯
    }, i*1000);
}
```

*let's let **let** be the hero*

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

*let i in a for loop scopes i to **each iteration of the for loop***

```
// or like this
for (var i = 1; i <= 5; i++) {
    setTimeout(function(num) {
        console.log("i:", num);
    }, i*1000, i);
}
```

# CONTROL STRUCTURES

similar to all C family languages
**if**, **else**, **while**, **for**, **switch**... all the usual suspects

```
for (var i = 0; i < array.length; ++i) {
    var element = array[i];
    // ...
}
```

in for loops **do not forget var** in "var i = 0"

# cuz otherwise **this might happen**

```
// global scope
var items = [/* some list */];
for(var i = 0; i < 10; i++) {
    subLoop();
}

function subLoop() {
    // scope of subLoop
    for(i = 0; i < 10; i++) { // missing var statement
        // do amazing stuff!
    }
}

// outer loop terminates after first call to subLoop()
```


OBJECTS
are the BIG BOYS of javascript

# everything

is objects and objects are **collections of name-value pairs**

- Dictionaries in Python
- Hashes in Perl and Ruby
- Hash tables in C and C++
- HashMaps in Java
- Associative arrays in PHP

*name is a **string**, value is any js **value** ( including functions or other objects )*

# 2 ways to make 'em

```
var jake = new Object();
jake.colour = 'yellow';
jake.legs = 4;
jake.jobs = [ 'hero', 'father' ];
```

*vs. object literal*

```
var jake = {
    colour: 'yellow',
    legs: 4,
    jobs: [ 'hero', 'father' ]
}
```

# object literal is the base of **JSON**

```
{
    "show": "Adventure Time with Finn & Jake",
    "characters": [
        {
            "name": "Finn",
            "role": [ "hero", "adventurer" ]
        },
        {
            "name": "Jake",
            "role": [ "hero", "father" ]
        },
        {
            "name": "Marceline",
            "role": [ "vampire queen" ]
        }
    ],
    "year": 2007,
    "place": "and of Ooo"
}
```

# 2 ways to access 'em

```
jake.color = "yellow";
var color = jake.color;
```

*VS*

```
jake["color"] = "yellow";
var color = jake["color"];

// reserved words
var butt = {
    "for": "pooping",
    size: 2
}
```

# objects have a secret...

## ...link to another object

*sometimes called* ***__proto__***

```
var butt = {
    "for": "pooping",
    size: 2
}

console.dir( butt );

false.toString(); // 'false'
[1, 2, 3].toString(); // '1,2,3'
```

# arrays

ARE NOT REAL

The cake is a lie!

# because

```
typeof [1,2,3] // object
```

*but with one special property*

```
[1,2,3].length // 3
```

```
var a = [];
a[80] = 'hello';
a.length; // 81
```

# 2 ways to make 'em

```
var a = new Array(); // don't bother
var a = []; // just use this one

// iterate over them
for (var i = 0; i < a.length; i++) {
  // Do something with a[i]
}

// even better, caching length
for (var i = 0, len = a.length; i < len; i++) {
  // Do something with a[i]
}
```

# **with caution,** you could even

```
// only for arrays with no falsey values (DOM nodes)
for (var i = 0, item; item = a[i++]; ) {
  console.log(item);
}

// if you don't need IE8
a.forEach(function(item) {
    console.log(item);
});
```

*or google "forEach MDN polyfill"*

# .length be tricky

```
// using .length to TRUNCATE array
var arr = [1, 2, 3, 4, 5, 6];
arr.length = 3;
arr; // [1, 2, 3]

// or to create a sparse array
arr.length = 6;
arr.push(4);
arr; // [1, 2, 3, undefined, undefined, undefined, 4]


// also, don't use "for in" loop on arrays, we'll get to why
```

# array-like objects

```
var nodes = document.querySelectorAll('h2'); // NodeList

// piggyback on other objects' methods
Array.prototype.forEach.call(nodes, function(item) {
    console.log(item);
});

// or shorter
[].forEach.call(nodes, function(item) {
    console.log(item);
});
```

# FUNCTIONS

*the* ***baddest***

*of the* ***bad girls***

# functions are **first class objects**

- you can pass them as an argument to a function
- you can return them from a function
- can assign them to a variable
- can store them in objects and arrays
- you can dynamically create/assign properties to them

# **they do the work** of all of these

- Method
- Class
- Constructor
- Module


FUNCTIONS
YUMMY YUMMY

when you use **function statement**

```
function avg() {

}
```

what really happens is **function expression**

```
var avg = function() {

}
```

# the basics are basic

```
function add(x, y) {
    var total = x + y;
    return total;
}
```

# but

```
add(); // NaN, you can't perform addition on undefined
add(2, 3, 4); // 5, added the first two, 4 was ignored
```

# you can pass variable amount of arguments

functions get **arguments** passed as object

```
function avg() {
    var sum = 0;

    // sadly, it's an array-like object
    [].forEach.call(arguments, function(x) {
        sum += x;
    });

    return sum / arguments.length;
}

avg(1, 2, 3, 5); // 2.75
avg(1, 5); // 3
```

# functions also get a **this** parameter

**it's value depends on function invocation**

```
// method invocation, this = jake
var jake = {
    color: 'yellow',
    cry: function() {
        console.log('jake cries!!')
    }
}
jake.cry();

// function invocation, this = global object
function avg() {  };
avg();
```

# functions also get a **this** parameter

**it's value depends on function invocation**

```
// constructor invocation, this = jake
var Dog = function(size) {
    this.size = size;
}
var jake = new Dog('big'); // constructor implicitly returns this

// call / apply invocation, this = nodes
var nodes = document.querySelectorAll('h2');
[].forEach.call(nodes, function(item) {
    console.log(item);
});
```

# **call** vs **apply** invocation

```
function avg() {
    var sum = 0;
    [].forEach.call(arguments, function(x) { // note ANONYMOUS function
        sum += x;
    });
    return sum / arguments.length;
}

avg.call(null, 1, 2, 3, 4, 6); // 3.2
avg.apply(null, [1, 2, 3]); // 2

// so our function works on arrays, sweet code reuse!
```

# exercise

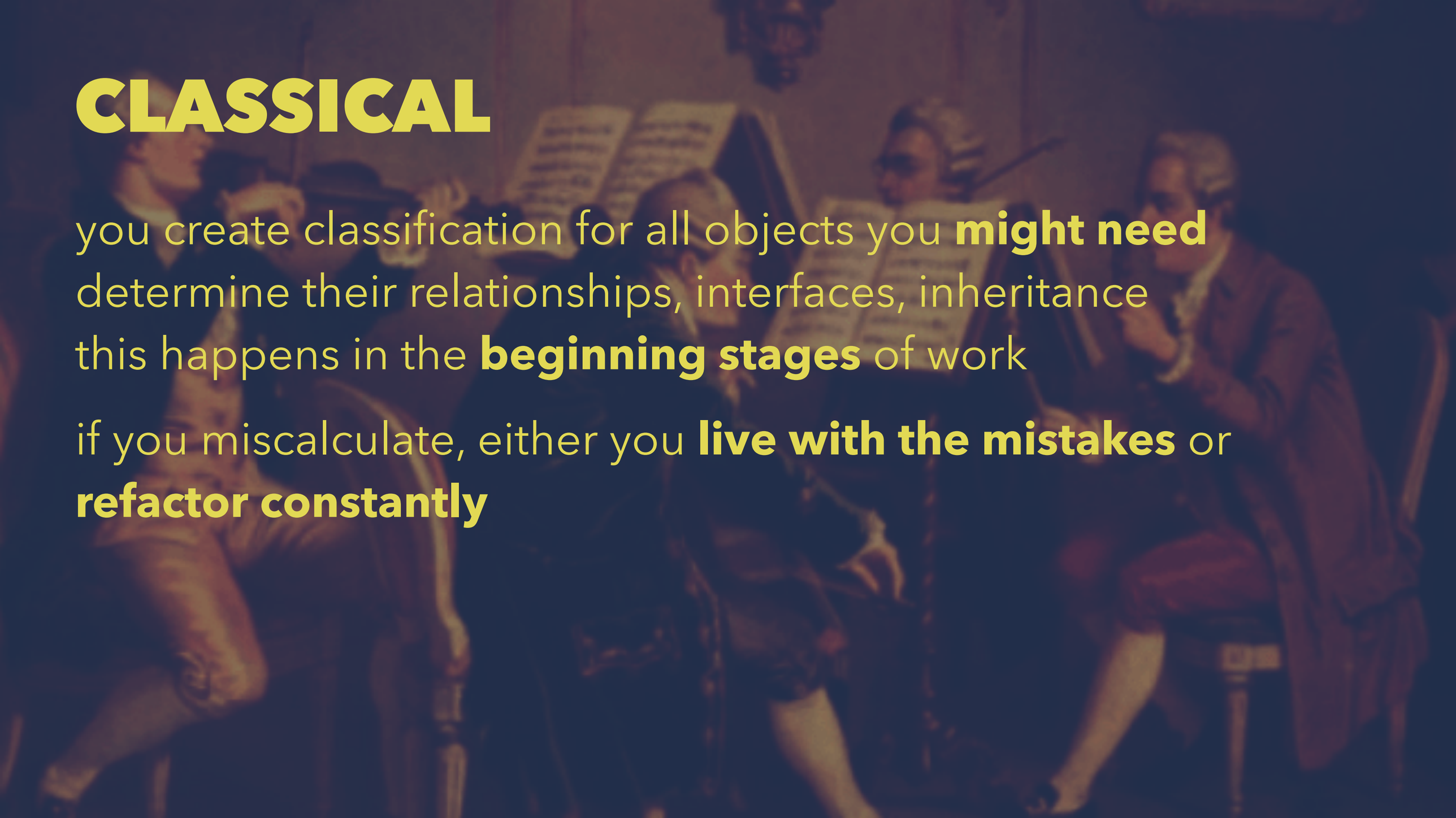
can you make your own **forEach** function?

```
// would be used like this
forEach(weapons, function(index) {
    console.log( weapons[index] );
});
```

# **optional parameters can lead to** interfaces like these in jquery

```
// so pretty ( ・_・)ノ
$('.jake').animate({ left: 500 });
$('.jake').animate({ left: 500 }, 2000, 'linear');
$('.jake').animate({ left: 500 }, function() {
    console.log('animation finished!')
});
```


# CLASSICAL *VS.* PROTOTYPAL INHERITANCE



# CLASSICAL

you create classification for all objects you **might need**
determine their relationships, interfaces, inheritance
this happens in the **beginning stages** of work

if you miscalculate, either you **live with the mistakes** or **refactor constantly**

# PROTOTYPAL

you make an object, if you like it, you make another one
you **create** and **customize**

javascript is **class free**
you use functions to create objects

**object inherit from each other**

*it's possible to augment the built in types*

# array-like objects can be weird

```
<p>The <img src="img/cat.png" alt="Cat"> in the <img src="img/hat.png" alt="Hat">.</p>
```

```
// the collection is live, we have to cycle backwards through it
var images = document.body.getElementsByTagName("img");
for (var i = images.length - 1; i >= 0; i--) {
  var image = images[i];
  if (image.alt) {
    var text = document.createTextNode(image.alt);
    image.parentNode.replaceChild(text, image);
  }
}
```

*what if we could turn them into arrays*

# might look like this

```
function objectToArray( obj ) {
    return Array.prototype.slice.call( obj );
}

// we'd use it like this
var obj = objectToArray(obj);

// but we'd rather do it the javascript way
// like this
obj.toArray();
```

# thanks, prototypal inheritance!

```
Object.prototype.toArray = function() {
    return Array.prototype.slice.call( this );
};
```

## it's easy, but that doesn't mean you should do it

```
// if you have to do it, always check
if ( typeof Object.prototype.toArray !== 'function' ) {
    Object.prototype.toArray = function() {
        return Array.prototype.slice.call( this );
    };
}
```

# so now we can do

```
var images = document.body.getElementsByTagName("img");
images.toArray().forEach(function(item) {
    console.log( item )
});

// same goes for other array-like objects
function avg() {
    var sum = 0;
    arguments.toArray().forEach(function(x) {
        sum += x;
    });
    return sum / arguments.length;
}
```

# IT COMES

*at a price*

```
var adventure = [{
        who: 'Ice King',
        wat: 'Armed with a magic crown...'
    }, {
        who: 'Marceline',
        wat: 'Marceline is a wild rocker...'
    }, {
        who: 'Peppermint Butler',
        wat: 'Peppermint Butler is an inhabitant...'
    }];

adventure.forEach(function(obj) {
    for (var key in obj) {
        console.log( key + ': ' + obj[key] ); // who, wat, toArray
    }
});
```

# when you go through properties

*the cycle runs up the inheritance chain*

```
adventure.forEach(function(obj) {
    for (var key in obj) {
        // sadly, we always have to do this
        if (obj.hasOwnProperty(key)) {
            console.log( key + ': ' + obj[key] ); // who, wat
        }
    }
});
```

# in modern browsers we can do

```
Object.defineProperty(Object.prototype, 'toArray', {
    enumerable: false, // now it won't be cycled through
    value: function() {
        return Array.prototype.slice.call( this );
    }
});
```

*google "defineProperty polyfill mdn"*

```
// the prototype notation is weird - let's say, you don't like it
Object.prototype.toArray = function() {}
```

# javascript bends to your needs

```
// let's create a way to attach new methods to all types
Function.prototype.method = function(name, func) {
    if ( !this.prototype[name] ) { // program defensively
        this.prototype[name] = func;
        return this;
    }
}

// add a toInteger method to numbers, if we don't like ~~
Number.method('toInteger', function () {
    return Math[this < 0 ? 'ceil' : 'floor'](this);
});

// sweet
(-10 / 3).toInteger(); // -3
```


inheritance

# PSEUDO-CLASSICAL *inheritance*

```
var Loader = function(selector) { // create constructor
    this.elem = $(selector)[0]; // set up properties
}

Loader.prototype.start = function() { // add inheritable methods
    this.elem.style.opacity = 1;
};

var FunkyLoader = function(selector) {
    Loader.call(this, selector); // we call the parent's constructor to set things up
    // add new properties
}

FunkyLoader.prototype = Object.create(Loader.prototype); // inherit from Loaders constructor
FunkyLoader.prototype.constructor = FunkyLoader; // the previous call "dirtied up" our new constructor

var loader = new Loader('.spinner');
var funky = new FunkyLoader('.secondary');
```

# BEHAVIOR *DELEGATION*

```
// we simply create a lean, mean object
var Loader = {
    init:  function(selector) { this.elem = $(selector)[0] },
    start: function() { this.elem.style.opacity = 1 },
    stop:  function() { this.elem.style.opacity = 0 }
};

// create a new object with parents proto and props
var FunkyLoader = Object.create( Loader );

// we simple change what we dont like, or add
FunkyLoader.init = function(selector) {
    this.elem = $(selector)[0];
    this.elem.classList.add('funky');
}

var loader = Object.create(Loader);
var funky  = Object.create(FunkyLoader);
```

# **PARASITIC** *inheritance*

```
function Loader(selector) {
    var elem = $(selector)[0];
    return {
        start: function() { elem.style.opacity = 1  },
        stop:  function() { elem.style.opacity = 0 }
    }
}

function FunkyLoader(selector) {
    var that = Loader(selector); // this is the IMPORTANT bit, inherit
    that.bounce = function() { ... } // modify
    return that;
}

var loader = Loader('.spinner');
var funky  = FunkyLoader('.secondary');
```

# those were just 3 patterns

## js is super versatile

**carldanley.com/js-module-pattern**

**addyosmani.com/resources/essentialjsdesignpatterns/book**

**shichuan.github.io/javascript-patterns**

# MODULES

## so implied globals suck

```
var name = 'Jake';
var age = 28;
var status = 'yellow';
function createHero(){
    // [...]
}
function getHeroDetails() {
    // [...]
}
// you just created 5 global variables, dummy!
```

# nah, you're no dummy, sorry about that!

```
var awesomeApp = (function() { // functions have scope!
    var name = 'Jake';
    var age = 28;
    var status = 'yellow';
    return {
        createHero: function() { },
        getHeroDetails: function() { }
    }
}()); // it calls itself immediately!

// more patterns: https://carldanley.com/js-module-pattern/
```

# if you need outside access

```
// the parentheses around the whole function are best practice
var awesomeApp = (function() {
    var private = function() {} // private
    return { // public
        createHero: function() {}
    }
}());
```

# if you don't

```
// self-invoking anonymous functions NEED the parentheses
(function($) {

    // all of my jQuery code is in a bubble like this

})(jQuery);
```

## a nicer way

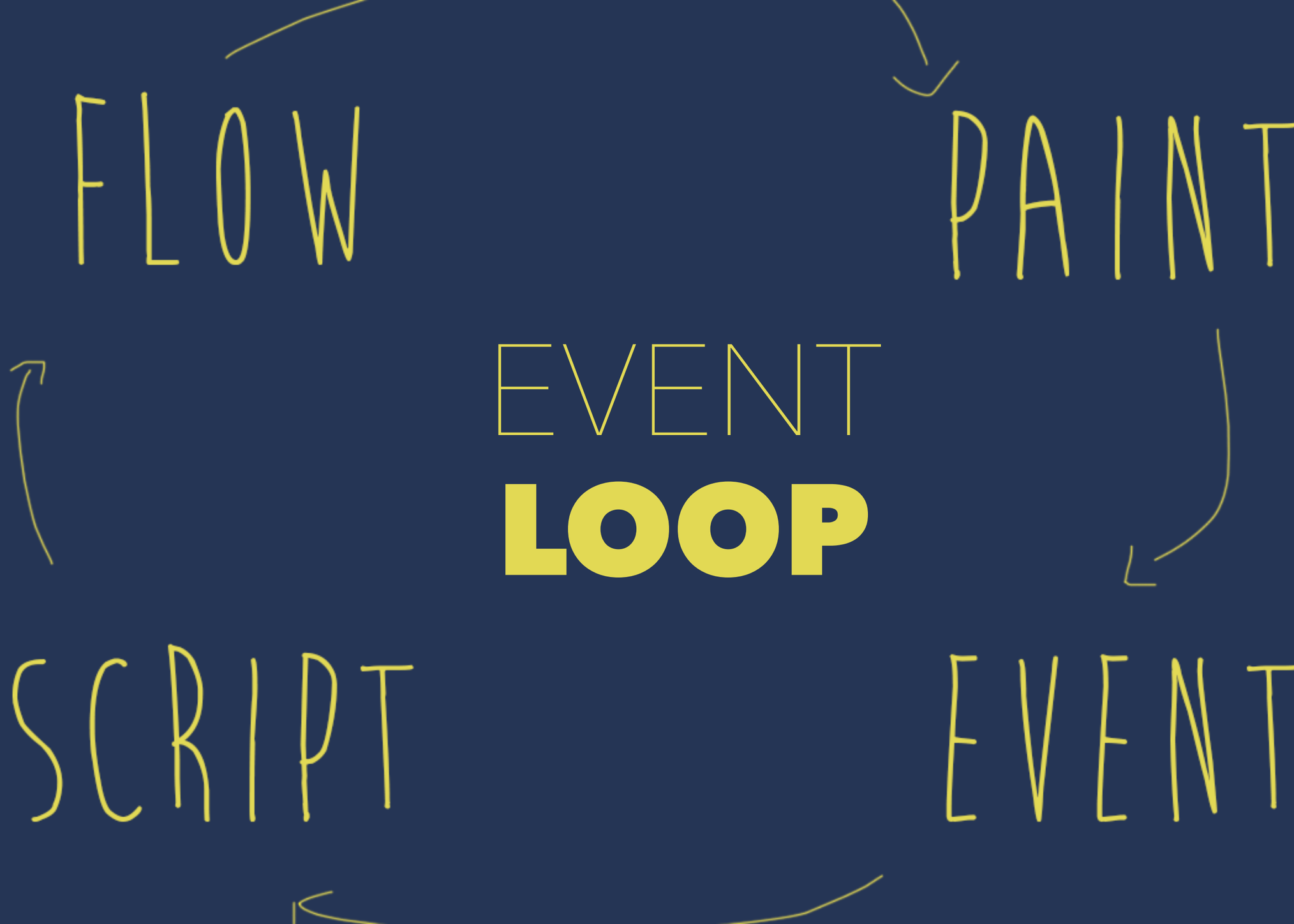
```
var awesomeApp = (function() {

    // properties
    var name = 'Jake';
    var age = 28;
    var status = 'yellow';

    // methods
    function createMember() { ... }
    function getMemberDetails() { ... }

    // give the outside some access
    return {
        create: createMember,
        get: getMemberDetails
    }

}());
```

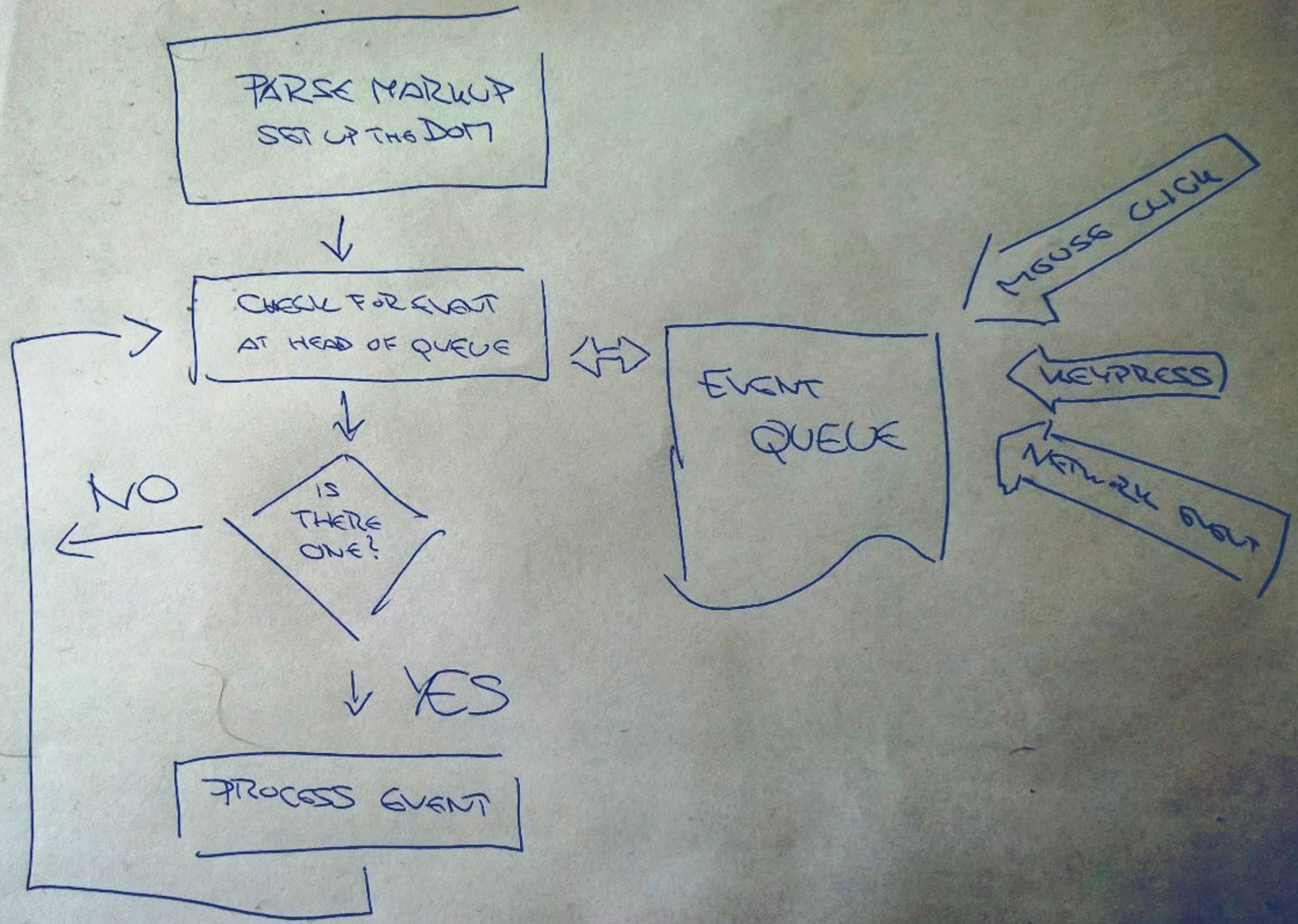
# exercise

can you add next/prev image functionality to the **lightbox** from /03 modules?


CLOSURES
a function has access to all variables of the function it's contained within
it sees the variables of its parent
even after
THE PARENT DIEEEEEEEEES

```
function yellowFade( element ) {
    var level = 1, // step fn has access to this, even after i'm long dead
        fps = 20;

    function step() {
        setTimeout(function() {
            var h = level.toString(16);
            element.style.backgroundColor = '#FFFF' + h + h;

            if ( level < 15 ) {
                level += 1;
                requestAnimationFrame(step);
            }
        }, 1000 / fps);
    }

    requestAnimationFrame(step);
}
```


FLOW
PAINT
EVENT
LOOP
SCRIPT
EVENT


PARSE MARKUP
SET UP THE DOM
CHECK FOR EVENT
AT HEAD OF QUEUE
IS
THERE
ONE?
NO
YES
PROCESS EVENT
EVENT
QUEUE
MOUSE CLICK
KEYPRESS
NETWORK EVENT

# many beautiful types of events

**browser events** *...page finishes loading*
**network events** *...response to ajax request*
**user events** *...mouse clicks, keypresses*
**timer events** *...timer expires, interval fires*

events are **unpredictable,** so we handle them **asynchronously,** js doens't wait, we're **non-blocking**

**on a** single thread

# CALLBACKS

a function that waits for the right time to strike, **like a fucking ninja**

```
$.ajax({
  url: "script.php",
  method: "POST",
  data: { id : menuId }
}).done(function( msg ) {
  $( "#log" ).html( msg );
}).fail(function( jqXHR, textStatus ) {
  alert( "Request failed: " + textStatus );
});
```


DOM

*so what does the browser do, when you open a website?*

**1/ LOADS THE RESOURCE**

*dns lookup, tcp handshake, downloads the html*

**2/ PARSES THE HTML**

*tag soup into html objects*

*so what does the browser do, when you open a website?*

**3/ CREATES DOM TREE**

*tree of elements based on relationships, API, CCSOM*

**4/ CREATES RENDER TREE**

*combines DOM and CSSOM for a rough list of elements to be shown*

*so what does the browser do, when you open a website?*

## 5/ LAYOUT OF THE RENDER TREE

*layout or flow, calculates size & positions of boxes*

## 6/ PAINT

*takes all info and turns it into actual pixels*

# steps **4, 5, 6 get repeated** whenever new info comes in

when image gets loaded
javascript affects css layout
browser window gets resized
user scrolls
...

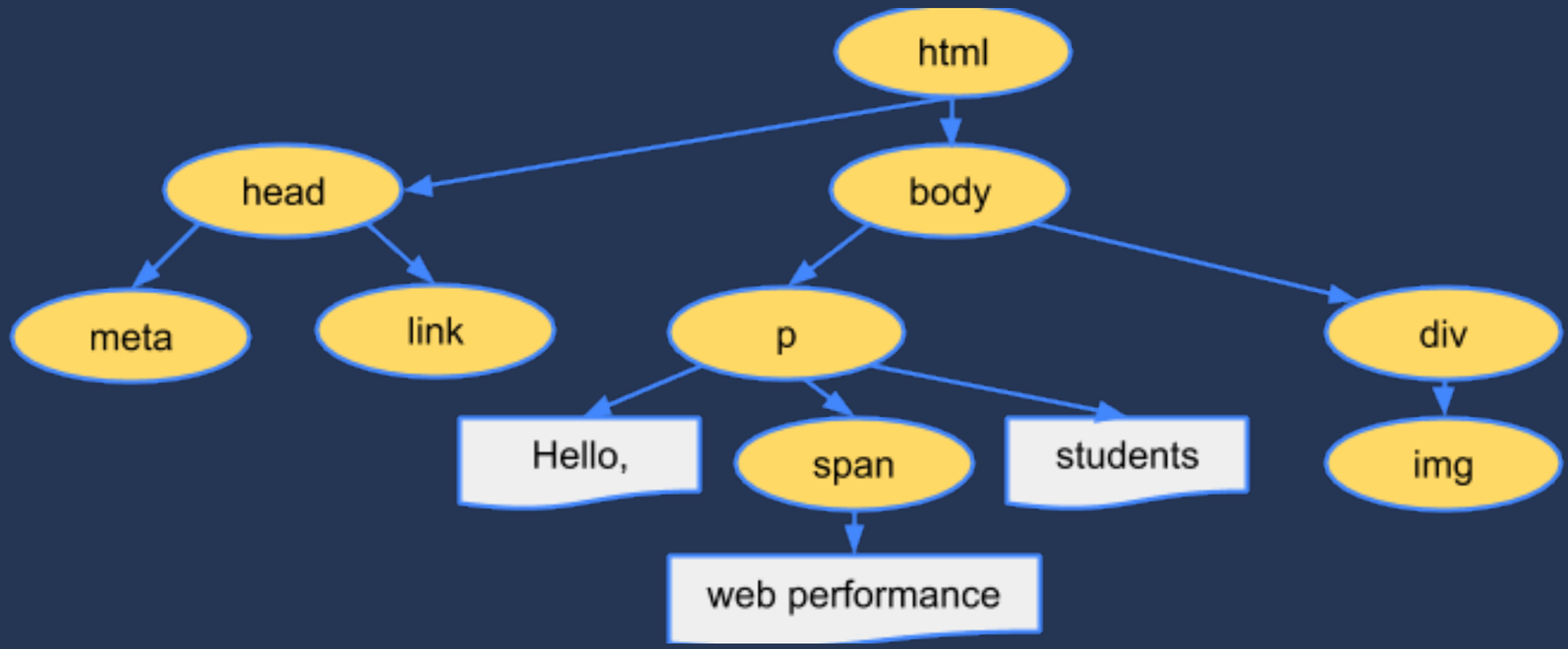
**reflow** when layout recalculates **repaint** for visual changes

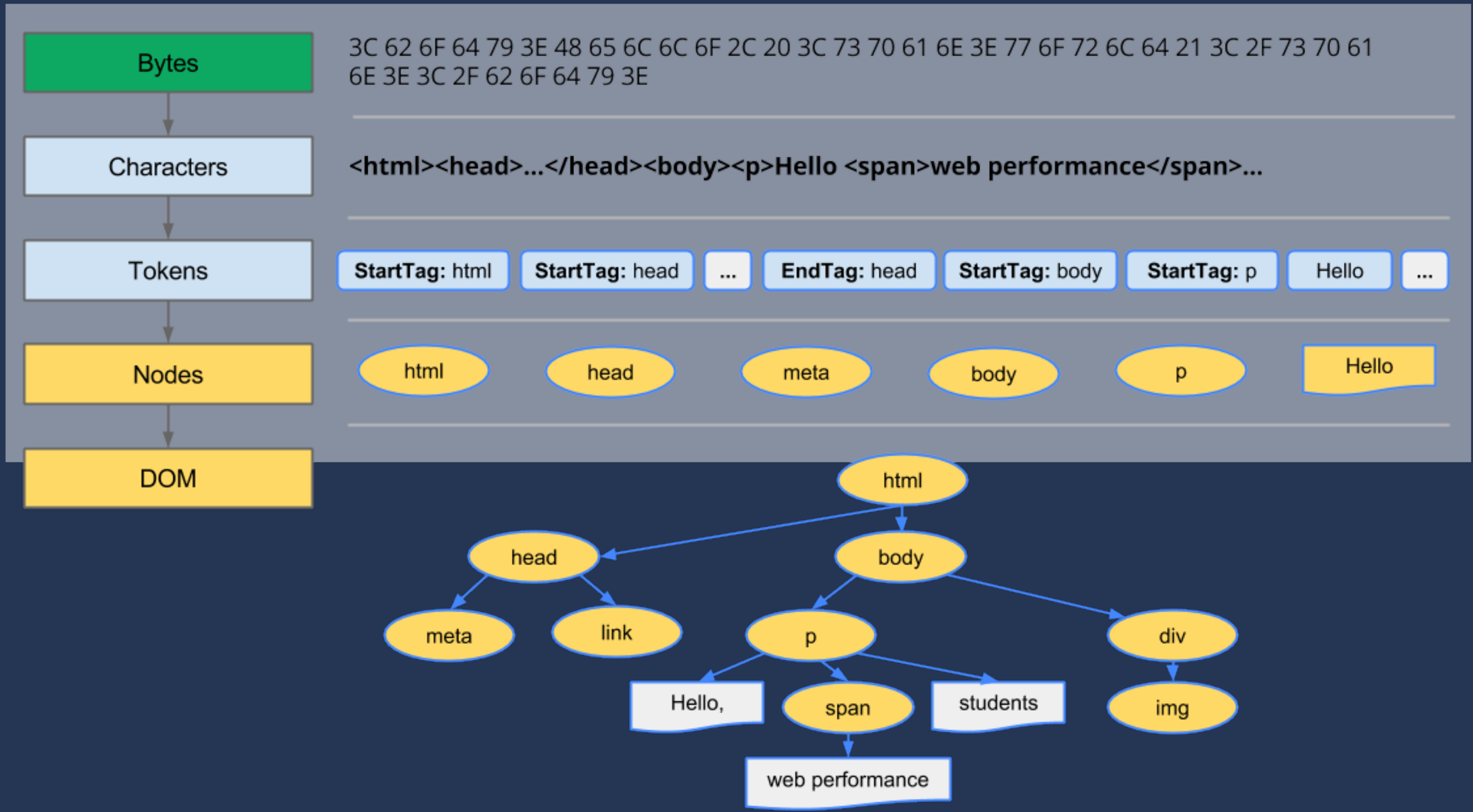
# so the browser turns html code like this

```
<html>
 <head>
   <link href="style.css" rel="stylesheet">
   <title>Critical Path</title>
 </head>
 <body>
   <p>Hello <span>web performance</span> students!</p>
   <div><img src="awesome-photo.jpg"></div>
 </body>
</html>
```

into this here **DOM tree**

*https://developers.google.com/web/fundamentals/performance/critical-rendering-path/constructing-the-object-model?hl=en*




Bytes
3C 62 6F 64 79 3E 48 65 6C 6C 6F 2C 20 3C 73 70 61 6E 3E 77 6F 72 6C 64 21 3C 2F 73 70 61 6E 3E 3C 2F 62 6F 64 79 3E
Characters
<html><head>...</head><body><p>Hello web performance...
Tokens
StartTag: html
StartTag: head
...
EndTag: head
StartTag: body
StartTag: p
Hello
...
Nodes
html
head
meta
body
p
Hello
DOM
html
head
body
meta
link
p
div
Hello,
span
students
img
web performance


FLOW
PAINT
SCRIPT
EVENT
GOES LIKE THIS

# DOM PERFORMANCE

gotta limit **reflows** and **repaints**

# DOM PERFORMANCE

HTML, CSS, JS are **render blocking**
browser has to dl / execute them, before creating render tree

SO

put CSS **as close to top**
put JS **as close to bottom**
minimize, compress, cache

# DOM PERFORMANCE

**if your .js doesn't touch the DOM or CSSOM**

```
<script src="script.js" async></script>
```

**or you can defer till all else is done**

```
<script src="script.js" defer></script>
```

# **some of what causes** reflows **and** repaints

- Adding, removing, updating DOM nodes

- Hiding a DOM node with display: none (reflow and repaint)
  or visibility: hidden (repaint only, because no geometry changes)

- Moving, animating a DOM node on the page

- Adding a stylesheet, tweaking style properties (size, margin, padding, border, font...)

- Retrieving measurements ( offsetWidth, offsetHeight, ... )

- User action such as resizing the window,
  changing the font size, or even scrolling

// can cause BIG performance hits

# some tips, how not to reflow or repaint

- animate position: absolute/fixed; (css = better)
- change class, not css properties one by one
- consider hiding elements
- cache values

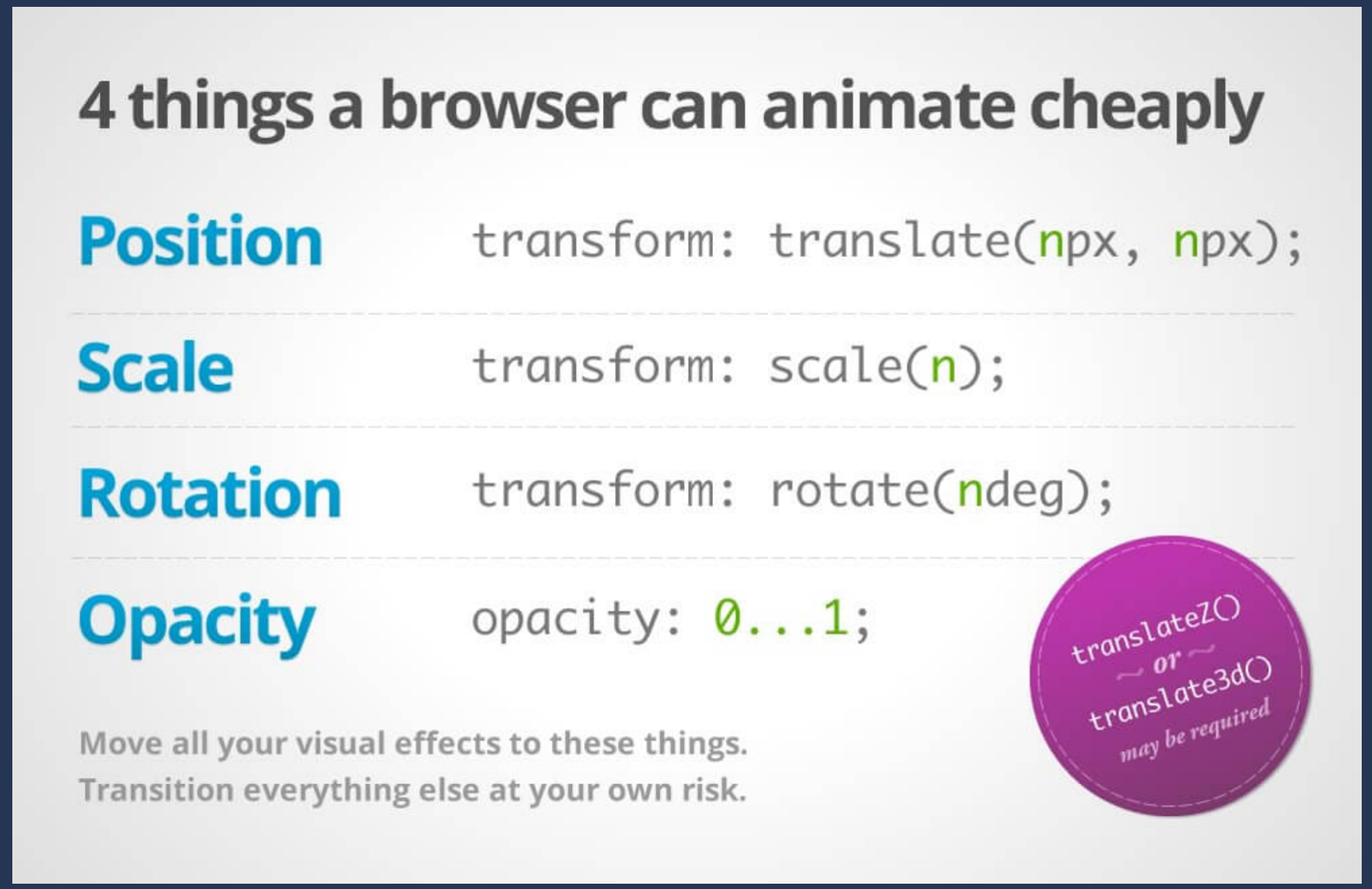
```
for (var i = 0; i < paragraphs.length; i++) {
    paragraphs[i].style.width = box.offsetWidth + 'px';
}
// vs.
var width = box.offsetWidth;
for (var i = 0; i < paragraphs.length; i++) {
    paragraphs[i].style.width = width + 'px';
}
```

# more tips, how not to reflow or repaint

- clump READ/WRITE together

```
// Suboptimal, causes layout twice.
var newWidth = aDiv.offsetWidth + 10; // Read
aDiv.style.width = newWidth + 'px'; // Write
var newHeight = aDiv.offsetHeight + 10; // Read
aDiv.style.height = newHeight + 'px'; // Write

// Better, only one layout.
var newWidth = aDiv.offsetWidth + 10; // Read
var newHeight = aDiv.offsetHeight + 10; // Read
aDiv.style.width = newWidth + 'px'; // Write
aDiv.style.height = newHeight + 'px'; // Write

// is this really important? well, consider scrolling
```

# **animate CSS instead of JS** position, scale, rotation, opacity




http://www.html5rocks.com/en/tutorials/speed/high-performance-animations/

# layers and hardware acceleration

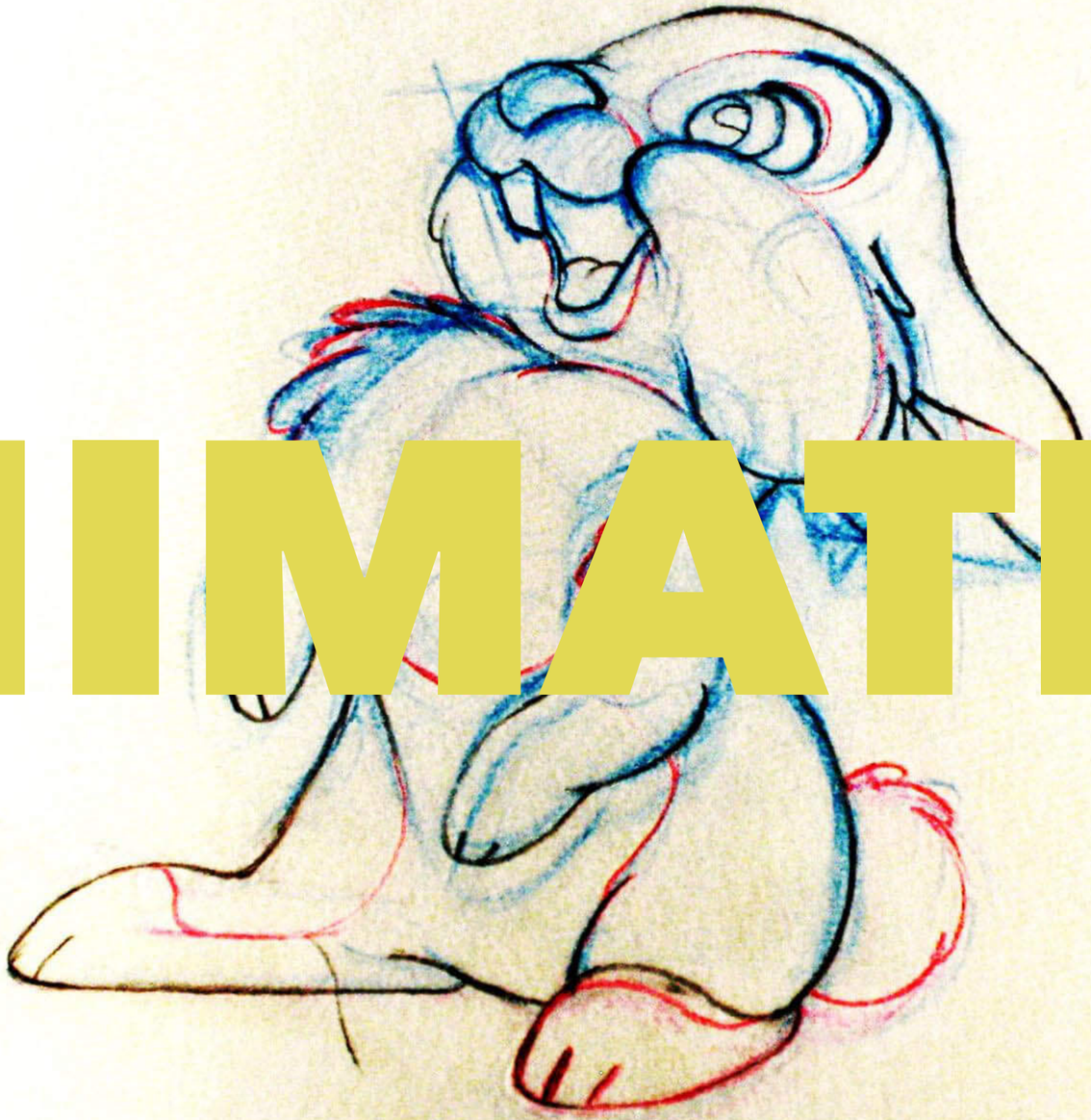
```
/* the old way */
.myAnimation {
    animation: someAnimation 1s;
    transform: translate3d(0, 0, 0); /* force hardware acceleration */
}

/* the new way (no IE, no Safari) */
.element {
    transition: transform 1s ease-out;
}
.element:hover {
    will-change: transform; /* the magic happens here */
}
.element:active {
    transform: rotateY(180deg);
}

/* use SPARINGLY */
```

# exercise

what could you do, to speed up the **parallax** in /10 jquery/04 parallax?


and
DON'T TOUCH
THE DOM

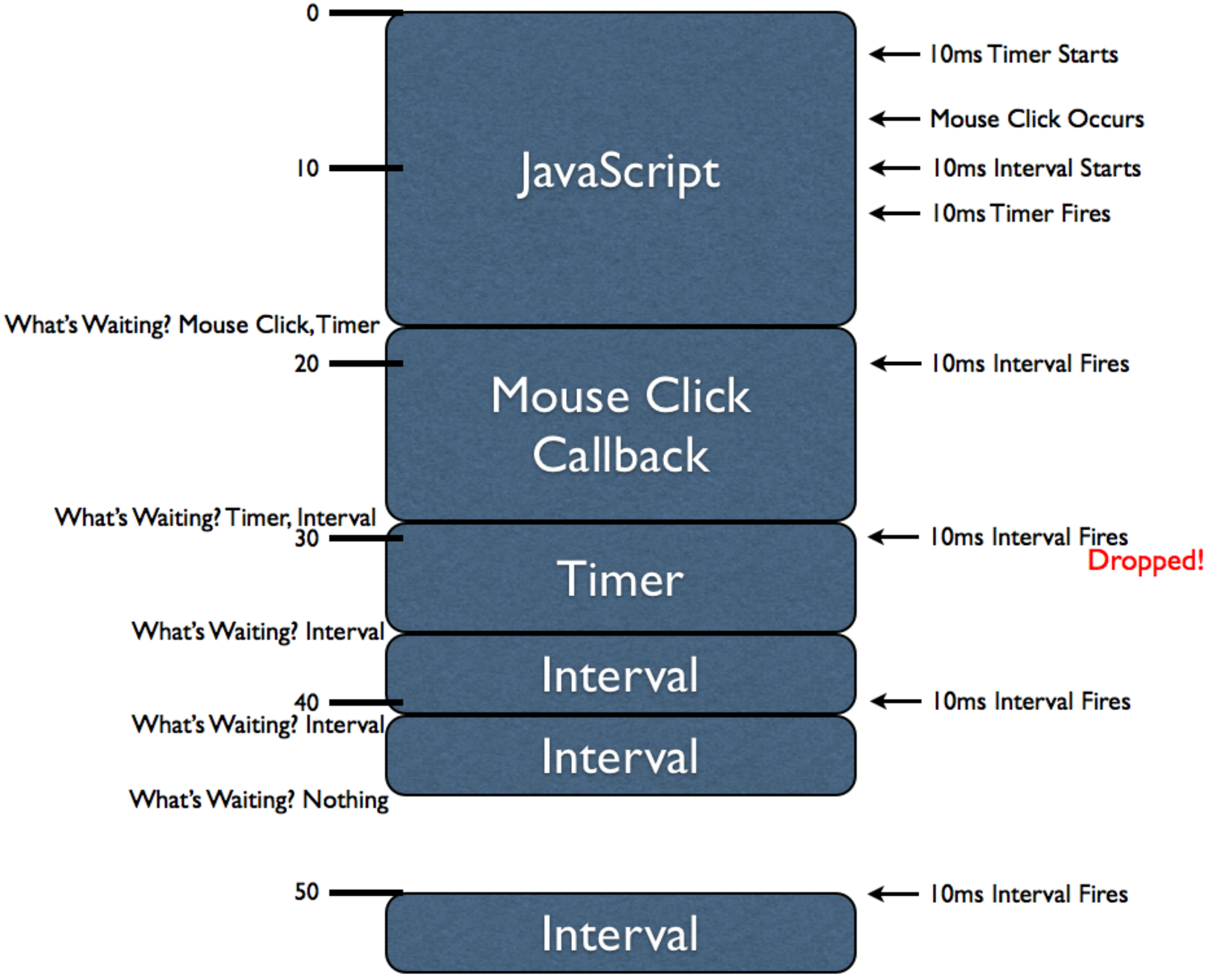
# ANIMATION



# **setTimeout(), setInterval()** craziness

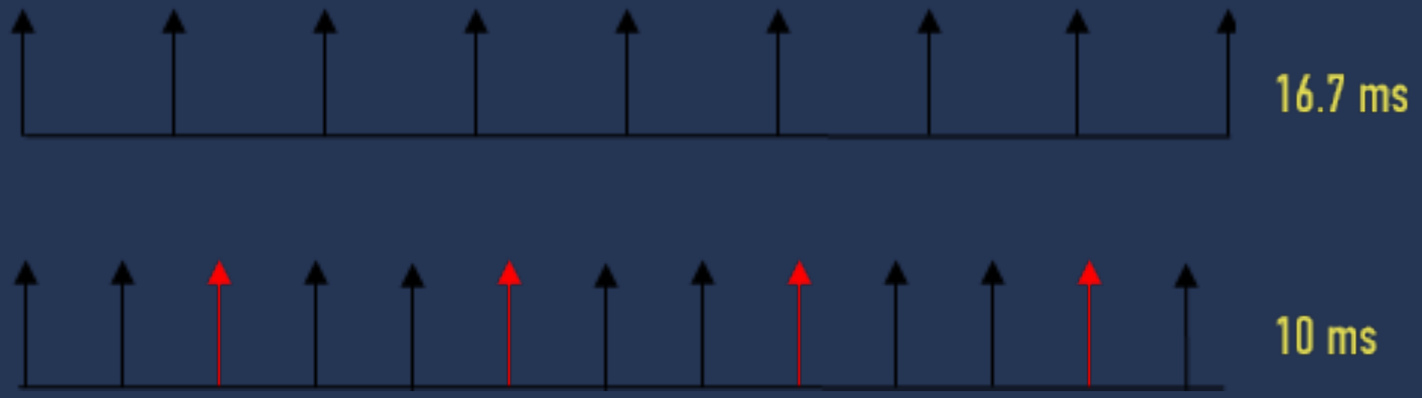
**single-threaded** with events coming in
the delay **will not be** exact
setInterval calls **might stack**

```
// better than setInterval
function draw(){
    setTimeout(draw, 16);
    // code for this frame
}
draw();
```


0
JavaScript
10ms Timer Starts
Mouse Click Occurs
10
10ms Interval Starts
10ms Timer Fires
What's Waiting? Mouse Click, Timer
20
Mouse Click Callback
10ms Interval Fires
What's Waiting? Timer, Interval
30
Timer
10ms Interval Fires
Dropped!
What's Waiting? Interval
Interval
40
10ms Interval Fires
What's Waiting? Interval
Interval
What's Waiting? Nothing
50
Interval
10ms Interval Fires

# timers can

- drop frames
- waste CPU, extra power usage
- animate elements that are not visible
- keep running in background or minimized tabs



# you wanna use
# **requestAnimationFrame**

```
function draw() {
    requestAnimationFrame(draw);
    // code for this frame
}
draw();

// also check http://jsfiddle.net/haw7mvk8/
```

*runs code when screen refreshes*

***so at 60Hz, that's ~16.7ms***


debounce

# limit how often some events *( mousemove scroll )* fire

```
input.addEventListener('keyup', function() { // only fire ajax after user stopped typing
    clearTimeout(timeout);
    timeout = setTimeout(function() {
        fauxAjax();
    }, 500);
});

var scheduled = false;
addEventListener('scroll', function() { // don't draw on every scroll, but only every 70ms
    if ( !scheduled ) {
        scheduled = true;
        setTimeout(function() {
            scheduled = false;
            // draw...
        }, 70);
    }
});
```

# exercise

can you make a functions that shows the
**mouse coordinates** on move,
but only every 250ms?

# use event delegation

```
// get the list
var list = document.querySelector('ol');

// attach a click handler on list
list.addEventListener('click', function(event) {

    // only do shit if we clicked an li
    // == DELEGATION
    if ( event.target && event.target.tagName.toLowerCase() === 'li' ) {
        yellowFade( event.target );
    }

}, false);
```

# DOM APIs, DOM INTERACTION

# CSS BEST PRACTICES

# general tips

- css reset, normalize.css
- organize css files, code
- keep dom nodes to a minimum
- keeps SELECTORS & SPECIFICITY to a minimum
- avoid IDs, use Classes
- use :focus as well as :hover
- user border/outline for debugging
- transform rather than animation
- DRY


DevTools
workspaces

```
/* general */
html { box-sizing: border-box; }
*, *:before, *:after { box-sizing: inherit; }

/* clearfix */
.group:before,
.group:after { content: " "; display: table; }
.group:after { clear: both; }

/* responsive images */
img {
    width: 100%;
    max-width: 100%;
    vertical-align: middle;
}

/* <meta name="viewport" content="width=device-width, initial-scale=1"> */
```

```css
/* dry */
.btn {
    font-size: 1.0625em;
    font-family: 'Montserrat', sans-serif;
    display: inline-block;
    padding: 1.25em 1.5em; /* 20/16 */ /* 24/16 */
    border-radius: 0.375em; /* 6/16 */
    background-color: rgba(84, 87, 102, 1);
}

.btn-red {
    background-color: rgba(253, 104, 91, 1);
}

.btn-red:hover,
.btn-red:focus {
    background-color: rgba(253, 104, 91, 0.9);
}
```

```
/* a background that covers the whole screen */
body {
    background: url('../img/city.jpg') center center no-repeat;
    background-attachment: fixed;
    background-size: cover;
}

/* for a small screen, load smaller image */
@media ( max-width: 600px ) {
    body {
        background-image: url('../img/city600.jpg');
    }
}

/* styling inputs */
input:not([type="checkbox"]):not([type="radio"]):not([type="file"]) { }
```


optimize for
CH-CH-CH CHANGES

# *Andy Hume:* CSS for Grownups

```
/* instead of */
.post h1 {
    font-size: 1.2em;
    text-trasform: uppcercase;
}

/* do */
.post .post-title {
    font-size: 1.2em;
    text-trasform: uppcercase;
}

/* or */
.post-h {  }
```

# **headings** are pretty important

```
/* doing this */
h1 { font-size: 3em; }
h2 { font-size: 2.3em; }
h3 { font-size: 2.1em; }
h4 { font-size: 1.8em; }
h5 { font-size: 1.3em; }

/* will force you */
h3 { font-size: 2.1em; }
.sidebar h3 { font-size: 1.8em; } /* size of h4 */
```

# headings

```
/* so instead you might do classes */
.h1 { font-size: 3em; }
.h2 { font-size: 2.3em; }
.h3 { font-size: 2.1em; }
.h4 { font-size: 1.8em; }
.h5 { font-size: 1.3em; }

/* or even better */
.h-headline    { font-size: 3em; }
.h-subheadline { font-size: 2.3em; }
.h-byline      { font-size: 2.1em; }
.h-related     { font-size: 1.8em; }
.h-promo       { font-size: 1.3em; }
```

# say you have **products**

```
/* instead of */
ul.product-list li {

}

/* do just */
.product-item {

}

/* you might even */
.product-item { }
.product-item-h { }
```

*never make **assumptions** about HTML*

# be prepared to **reuse modules**

```
/* instead of */
.box {
    background: black;
}

/* focusing on the placement */
.sidebar .box {
    background: white;
}

/* focus on what changes */
.box-light {
    background: white;
}
```

# same thing goes for **sub-pages**

```
/* this */
.product-page .box {

}

/* leads to this */
.product-page .box,
.about-page .box,
.contact-page .box,
.home-page. box {

}
```

*just create one class,* ***independent*** *of HTML*

# decouple **icons** from html and css

```
<!-- istead of -->
<p class="icon icon-folder">
    text
</p>

<!-- it's becoming common to -->
<p>
    <i class="icon icon-folder"></i> text
</p>
```

*http://fortawesome.github.io/Font-Awesome/icon/folder/*


jQuery

**lab.abhinayrathore.com/jquery-standards**

**github.com/stevekwan/best-practices/blob/master/javascript/best-practices.md**

*learn.jquery.com*

## if in <head>

```
$( document ).ready(function() {
    console.log( "ready!" );
});

// or shorthand
$(function() {
    console.log( "ready!" );
});
```

## if before </body>

```
(function($) {
    // ..code
})(jQuery);
```

# make use of .siblings()

```
// cache elements
var list = $('.jokes');

// hide them
list.find('dd').slideUp();

// show one, but hide all its siblings
list.find('dt').on('click', function(event)
{
    $(this).next().slideToggle()
           .siblings('dd').slideUp();

    event.preventDefault();
});
```

# grab the right keycode

```
$(document).on('keydown', function(event) {

    // one of these will exist
    var keyCode = event.keyCode || event.which,
        arrow = {left: 37, up: 38, right: 39, down: 40 };

    switch(event.which) {
        case arrow.left:
            Slider.prev();
            break;
        case arrow.right:
            Slider.next();
            break;
    }

});
```

# use ajax promises

```
var fb = [
    'http://graph.facebook.com/learn2codesk',
    'http://graph.facebook.com/spaceunicorn',
    'http://graph.facebook.com/bedroomssk'
]

function getData( index ) {
    return $.ajax({
        url: fb[index],
        type: 'GET',
        dataType: 'json'
    }).promise();
}

$.when( getData(0), getData(1), getData(2) ).then(function() {
    console.log( 'all finished!' );
});
```

# exercise

can you change the **ajax gallery** in
/10 jquery/03 ajax gallery
so it starts loading the next page, when
you scroll to the bottom?

# sometimes js object is more useful than jq object

```
// after click on <a href="#video"> scroll to element #video
$('#nav').find('a').on('click', function(event) {
    event.preventDefault();

    // due to compatibility we have to animate both
    $('html,body').animate({
        // the standard js object carries this.hash
        // but the jquery object doesn't
        scrollTop: $(this.hash).offset().top
    });
});
```

# you can animate numbers and other properties

```
$({ num: 0 }).animate({ num: 250 }, {
    duration: 9000,
    easing: 'easeInExpo',
    step: function(now) {
        if ( Math.floor(now) > val ) {
            draw();
            val = Math.floor(now);
        }
    }
});
```

# protect your jq plugins

```
// since an external script could overwrite these "contants"
;(function($, window, document, undefined) {

    $.fn.pluginName = function( options ) {

        // allow for customization
        var settings = $.extend({
                default: 'value'
            }, options);

        // allow for chaining
        return this.each( function() {
            // plugin code
        });

    }

})(jQuery, window, document);
```


NOW YOU'RE A PRO

# thanks