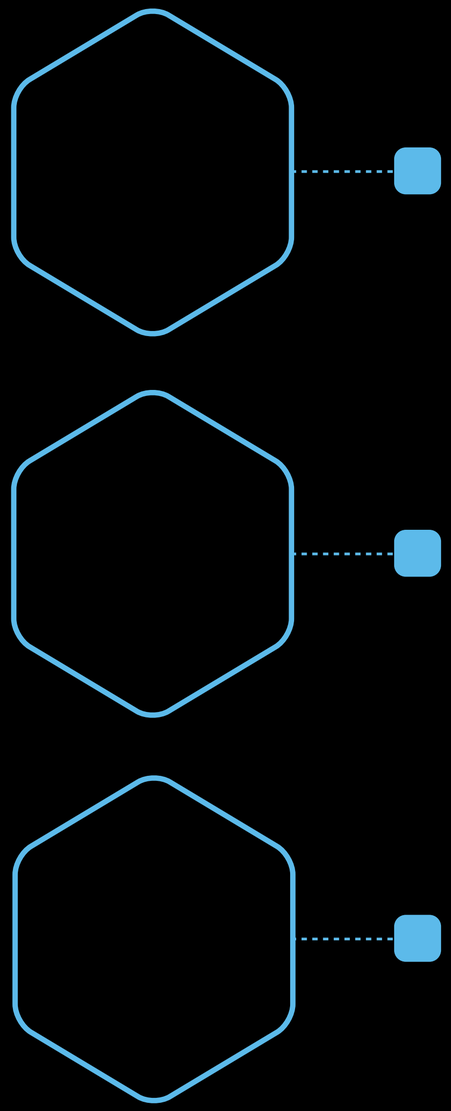
ES6 WORKSHOP

Pratik Patel @prpatel

LABO

- Install Node.js 4.2.2 https://nodejs.org/
- Install workshop from command line:
- git clone https://github.com/prpatel/es6-workshop
- Run workshop:
- cd; es6-workshop
- npm install
- npm test
- npm run test:watch
- Free online env: https://koding.com/R/prpatel

WHY MIGRATE TO ES6 _NOW_?



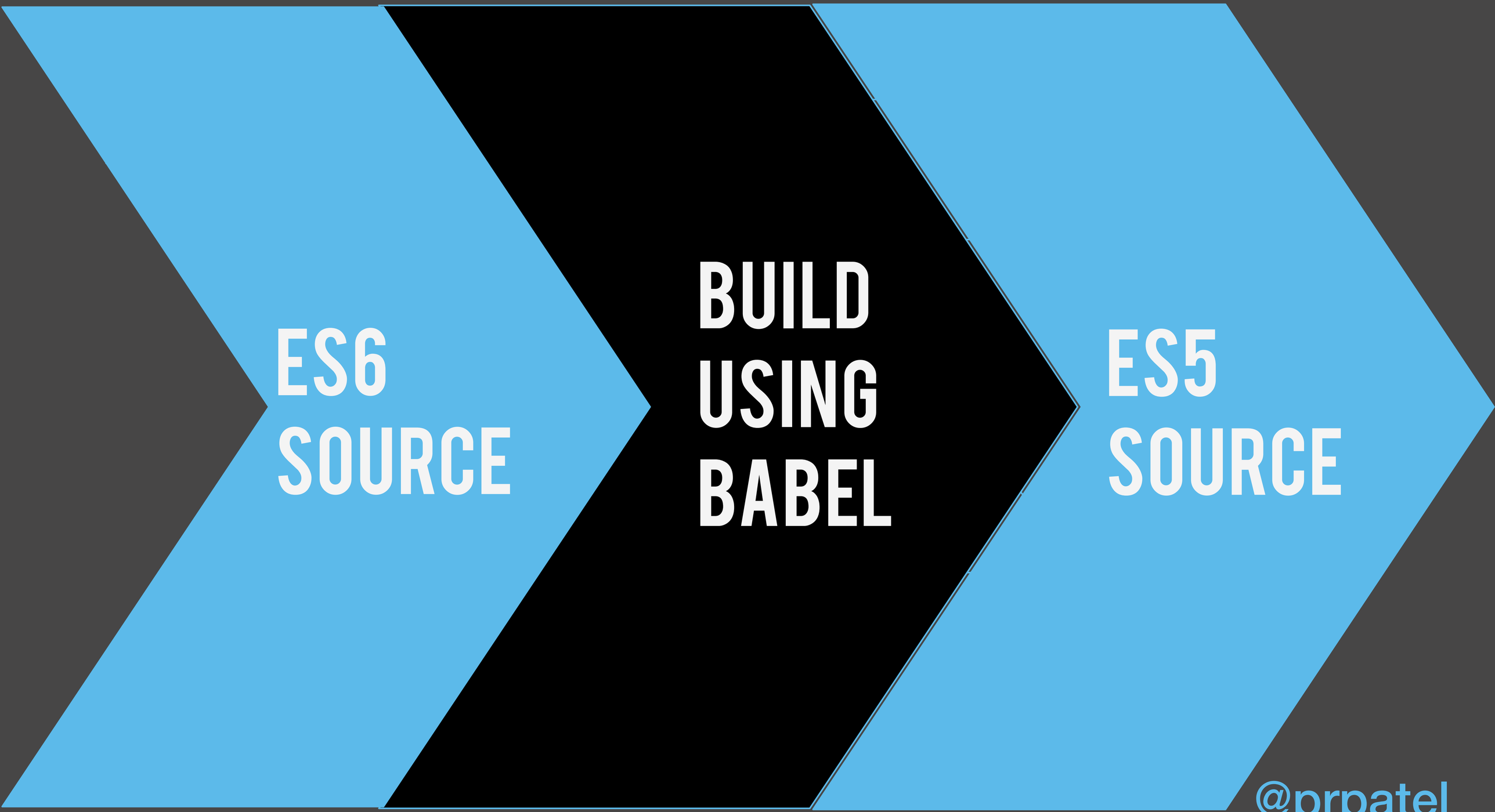
- FUTURE PROOF
- BETTER, LESS “WTF”S
- MODULES!

@prpatel

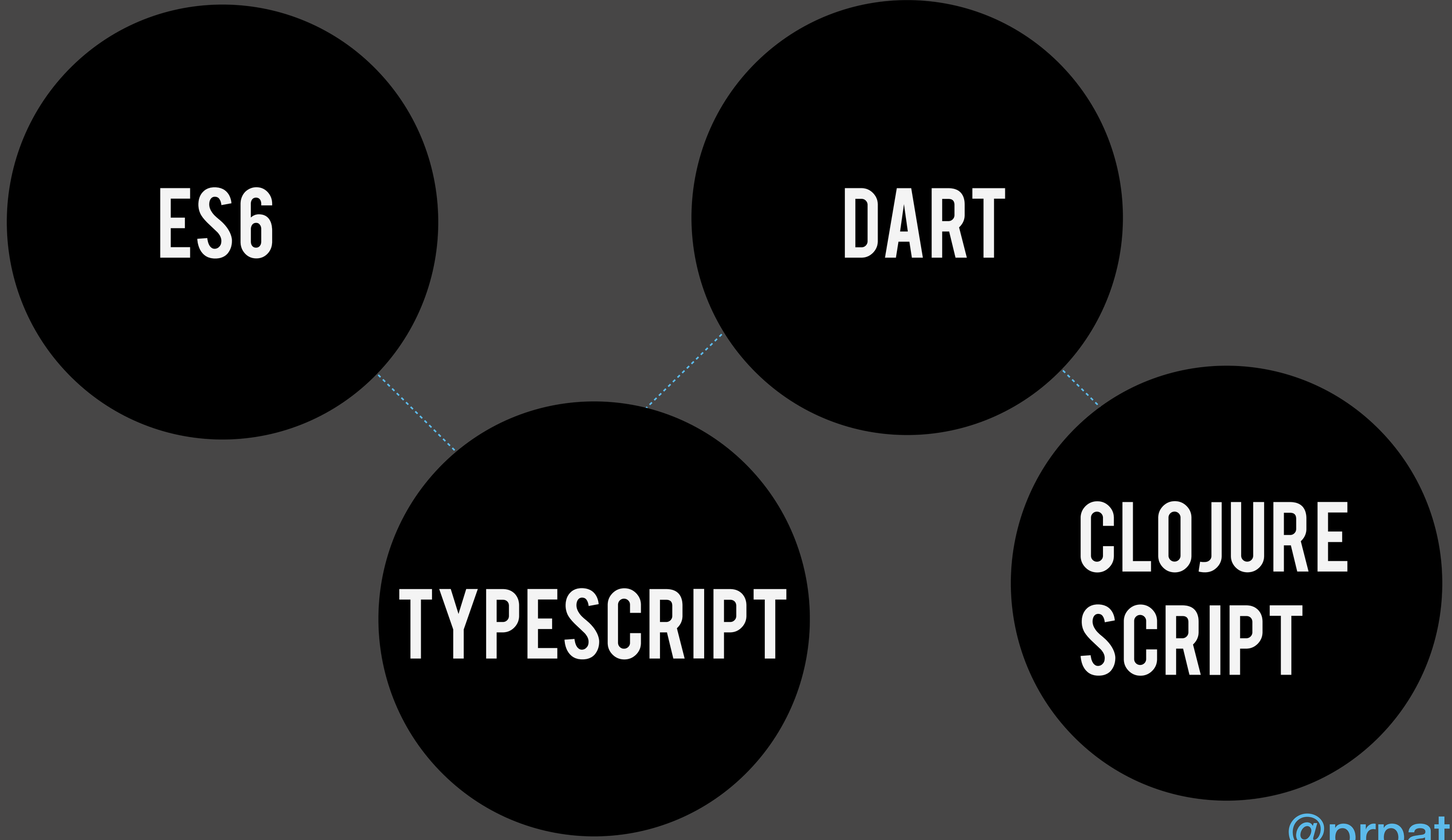
WILL ES6 WORK IN TODAY'S BROWSERS?

TRANSPILES ES6 TO ES5 (WHAT TODAY'S BROWSERS SUPPORT)

BABEL


ES6
SOURCE
BUILD
USING
BABEL
ES5
SOURCE

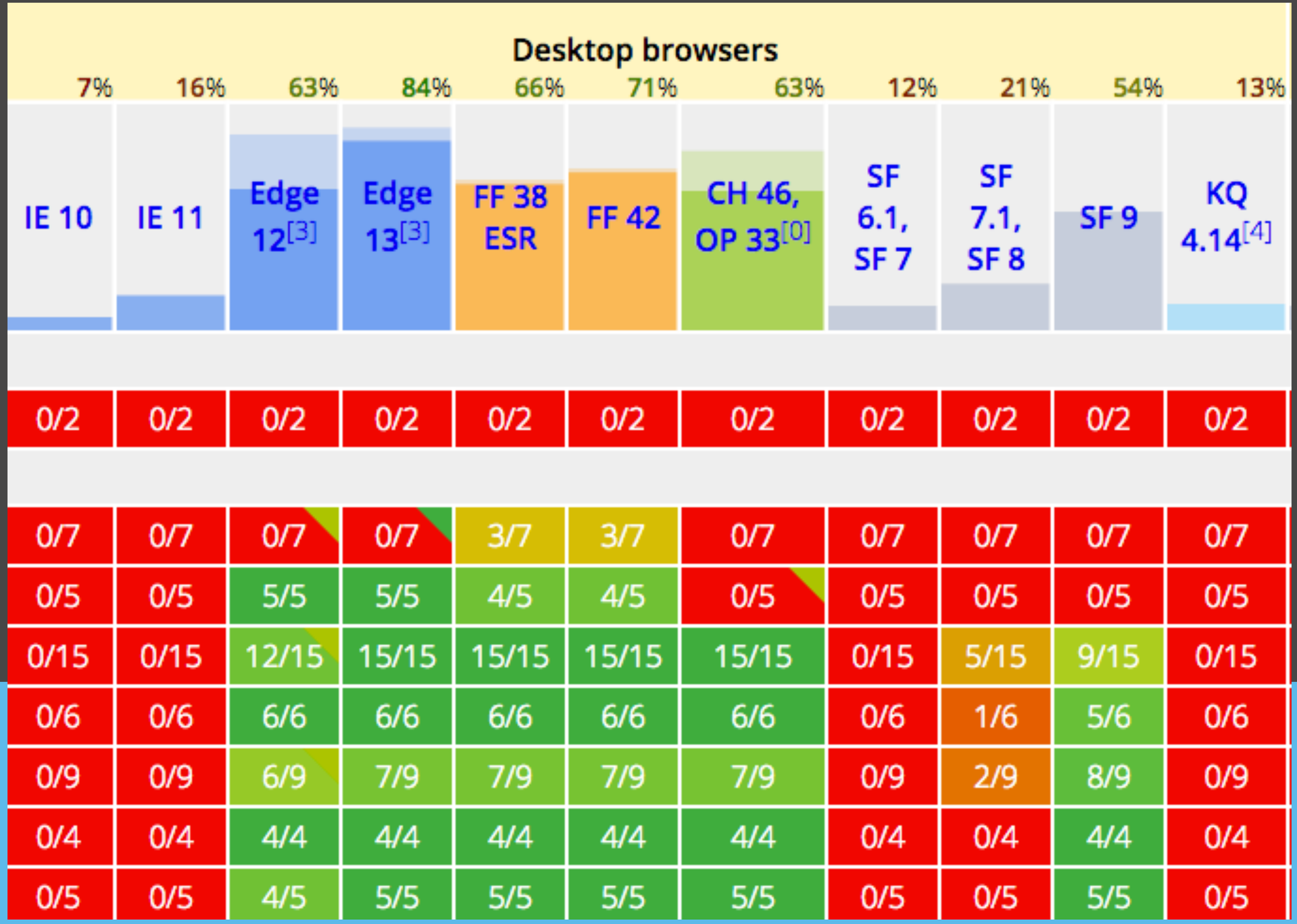
What about the other transpile to JS langs?



@prpatel

WILL BE _NATIVELY_
SUPPORTED IN BROWSERS

ES6 IS A STANDARD



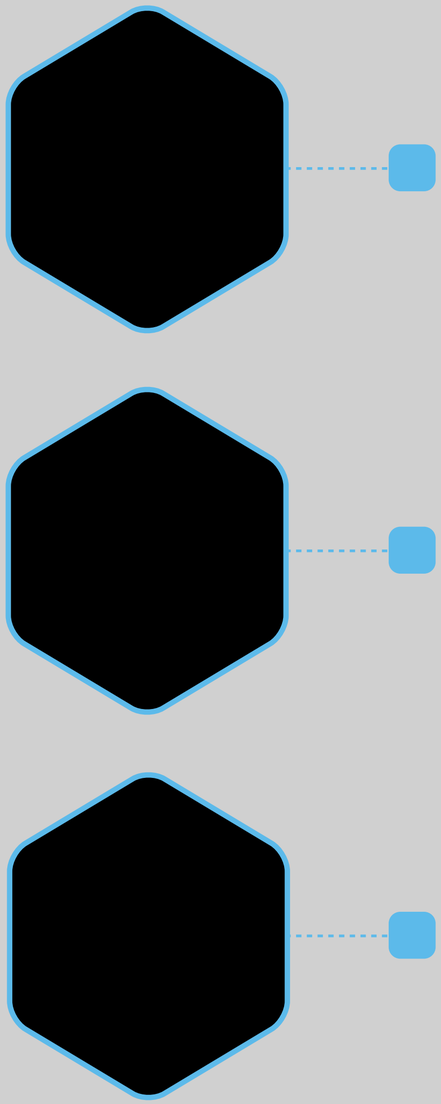
HTTPS://KANGAX.GITHUB.IO/COMPAT-TABLE/ES6/

RUN TEST

CODE

REPEAT

TIME TO GET BUSY



- "NPM RUN TEST:WATCH" FROM COMMAND LINE
- EDIT TEST/01_....JS
- WATCH CLI TO SEE RESULTS

INSTRUCTIONS

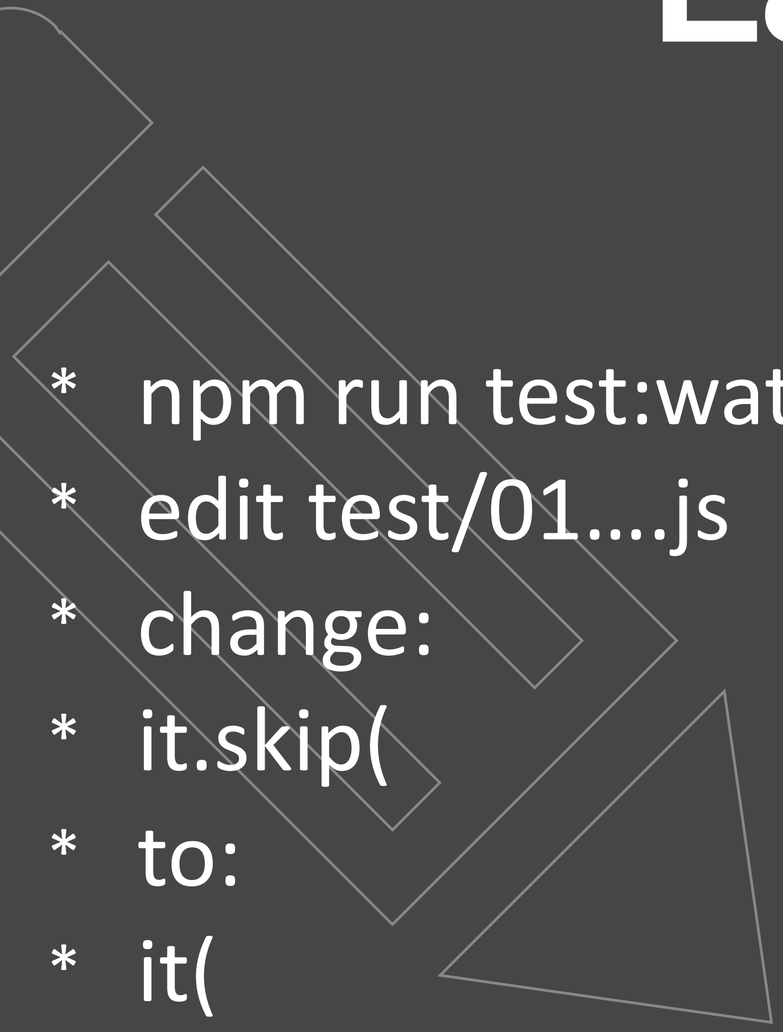
LAB 1: REAL SCOPES!

SCOPES HAVE ALWAYS BEEN AN ISSUE IN JS

let and const and var

```
{
    var a = 10;
    let b = 20;
    const tmp = a;
    a = b;
    b = tmp;
    // tmp = 30; Can't do that, will result in a SyntaxError
}
```

Lab 1: SCOPE



* npm run test:watch
* edit test/01….js
* change:
* it.skip(
* to:
* it(
* to enable the test!

Lab 1: Discussion

LAB 2: TEMPLATE STRINGS

Also known as Template Literals

Template Literals

```
Basic string:
‘Hello World’
```

```
Multiline:
` line 1
line 2`
```

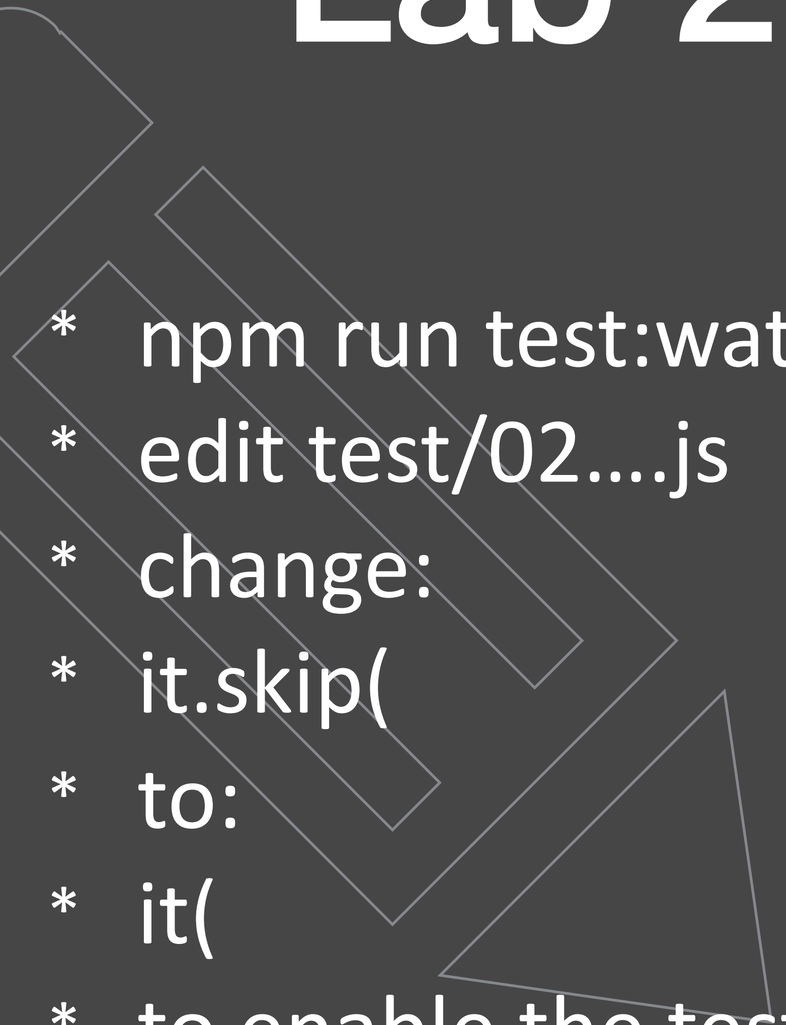
```
String interpolations:
var a = “World”;
`Hello ${a}`
```

```
Can also use a function inside ${}!
```

```
Can also tag:
const result = tagIt`Hello ${noun}!`;
```

@prpatel

Lab 2: Template String



* npm run test:watch
* edit test/02….js
* change:
* it.skip(
* to:
* it(
* to enable the test!
* Do the extra credit if you’re ahead, or skip it

@prpatel

Lab 2: Discussion

LAB 3: NEW CORE OBJECT FUNCTIONS

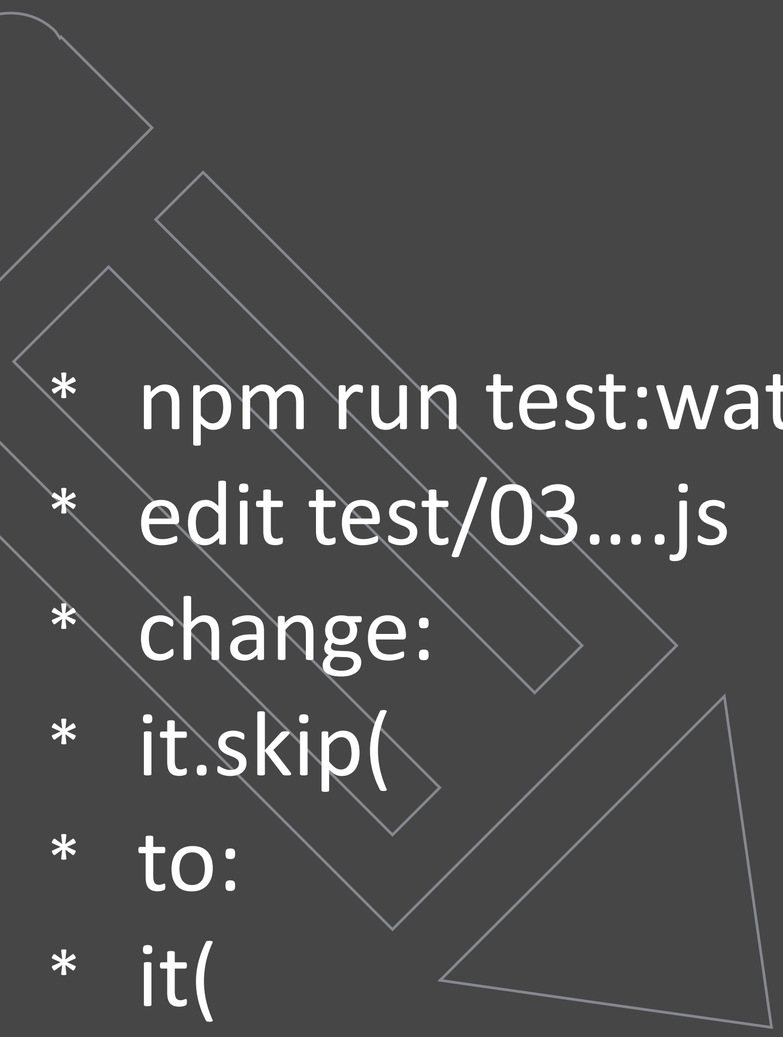
USEFUL STUFF ADDED TO Strings, Objects, Arrays

New APIs

```
Number.isInteger(Infinity) // false
Number.isNaN("NaN") // false

"abcde".includes("cd") // true
"abc".repeat(3) // "abcabcabc"

Array.from(document.querySelectorAll('*')) // Returns a real Array
Array.of(1, 2, 3) // Similar to new Array(...), but without special one-arg behavior
[0, 0, 0].fill(7, 1) // [0,7,7]
[1, 2, 3].find(x => x == 3) // 3
[1, 2, 3].findIndex(x => x == 2) // 1
[1, 2, 3, 4, 5].copyWithin(3, 0) // [1, 2, 3, 1, 2]
["a", "b", "c"].entries() // iterator [0, "a"], [1,"b"], [2,"c"]
["a", "b", "c"].keys() // iterator 0, 1, 2
["a", "b", "c"].values() // iterator "a", "b", "c"

// NOTE: NOT DEEP COPY/MERGE!
Object.assign({a:1}, {b:2}, {c:3}) => { a: 1, b: 2, c: 3 }
```

@prpatel

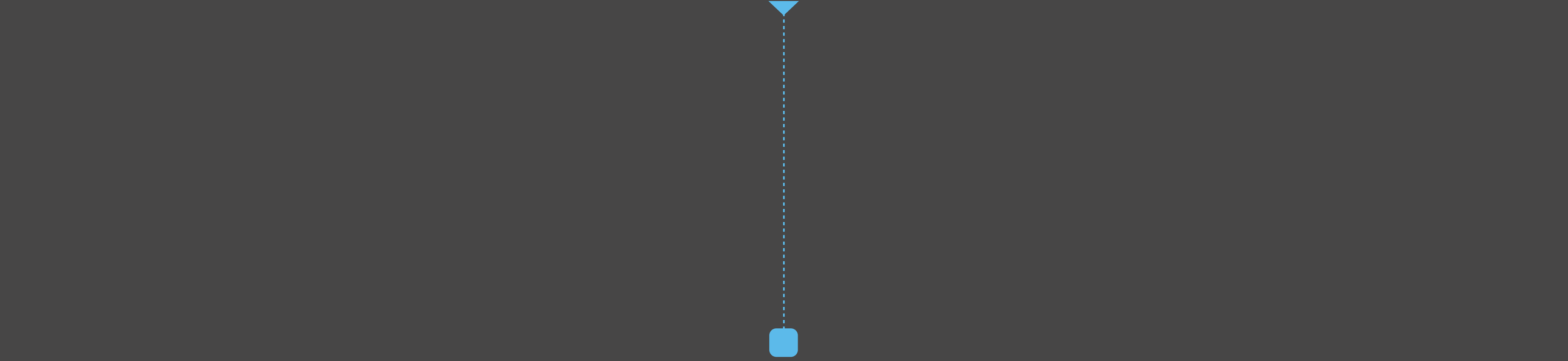
Lab 3: APIs



* npm run test:watch
* edit test/03….js
* change:
* it.skip(
* to:
* it(
* to enable the test!

Lab 3: Discussion

LAB 4:DESTRUCTURING



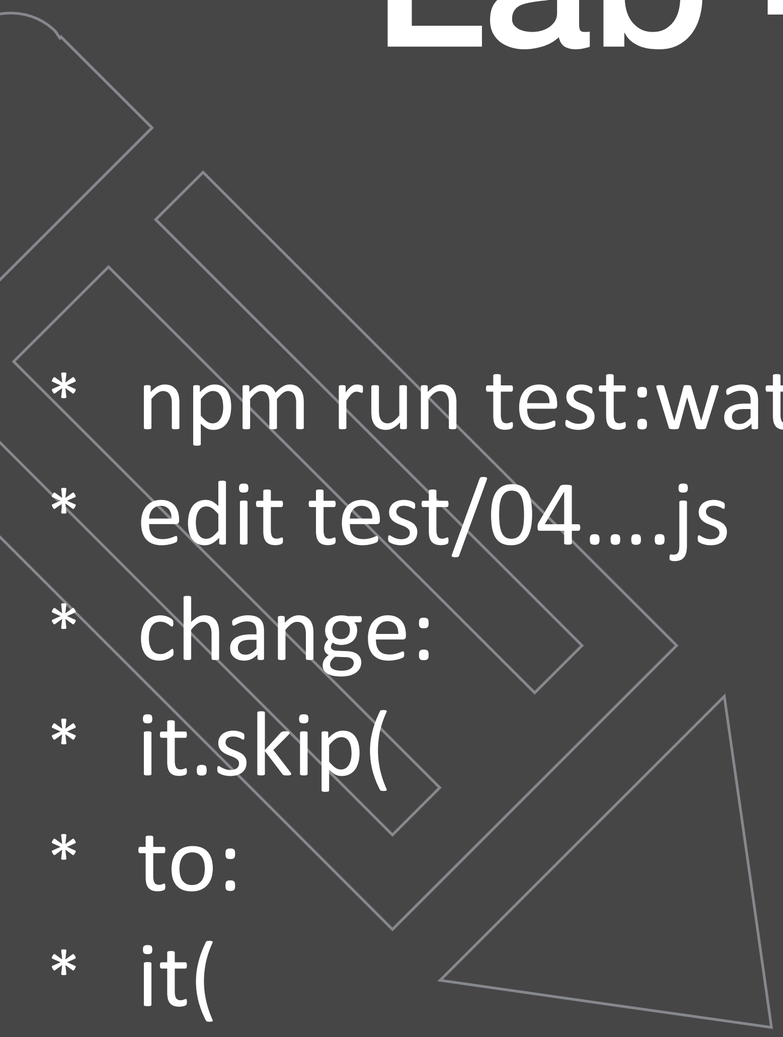
Easier to access properties of arrays and objects - similar to Clojure!

DESTRUCTURING

```
var foo = 123;
    var bar = 456;

    // swapping of the value
    [bar, foo] = [foo, bar];

var pt = {x: 123, y: 444};
    var {x, y} = pt;
    console.log(x, y); // 123 444

//nested too!

var pt =  { data:
      {x: 123, y: 444}
    };
let {data: {x, y}} = pt;
// x and y are 123, 444
```

@prpatel

Lab 4: Destructuring



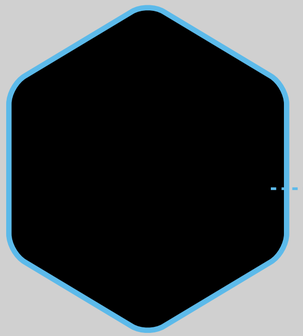
* npm run test:watch
* edit test/04….js
* change:
* it.skip(
* to:
* it(
* to enable the test!

Lab 4: Discussion

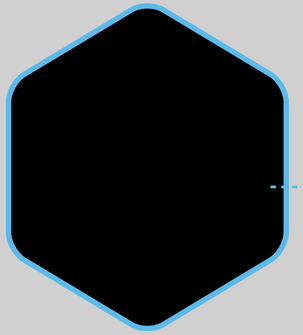
LAB 5: MODULES

Modular JavaScript has been around with tools

@prpatel

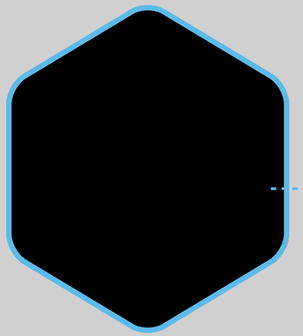


BROWSERIFY, AMD, COMMONJS

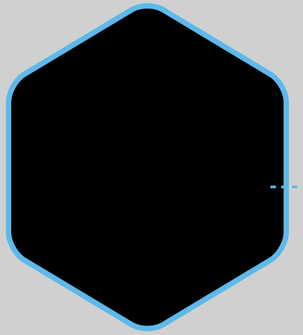


USED TO USE IIFE TO DO THE SAME THING IN PAST

MODULAR JS



NODE HAS SUPPORTED MODULES FOR A WHILE



NPM PACKAGING ALSO FOR THE BROWSER

MODULAR JS

Modules - the past

```
(function () {
    var $ = this.jQuery;

    this.myExample = function () {};
}());
--
common.js:
    exports.message = 'Hello Babel';
    var message = require('./Message').message;
    console.log(message); // Hello Babel
```

Modules - the future

```
export const message = 'Hello Babel’;
_
import {message} from ‘./Message';
___________
export function sum(x, y) {
  return x + y;
}
export var pi = 3.141593;
_________
import * as math from "lib/math";
alert("2π = " + math.sum(math.pi, math.pi));

import {sum, pi} from "lib/math";
alert("2π = " + sum(pi, pi));

import {sum as Summer} from "lib/math";
```

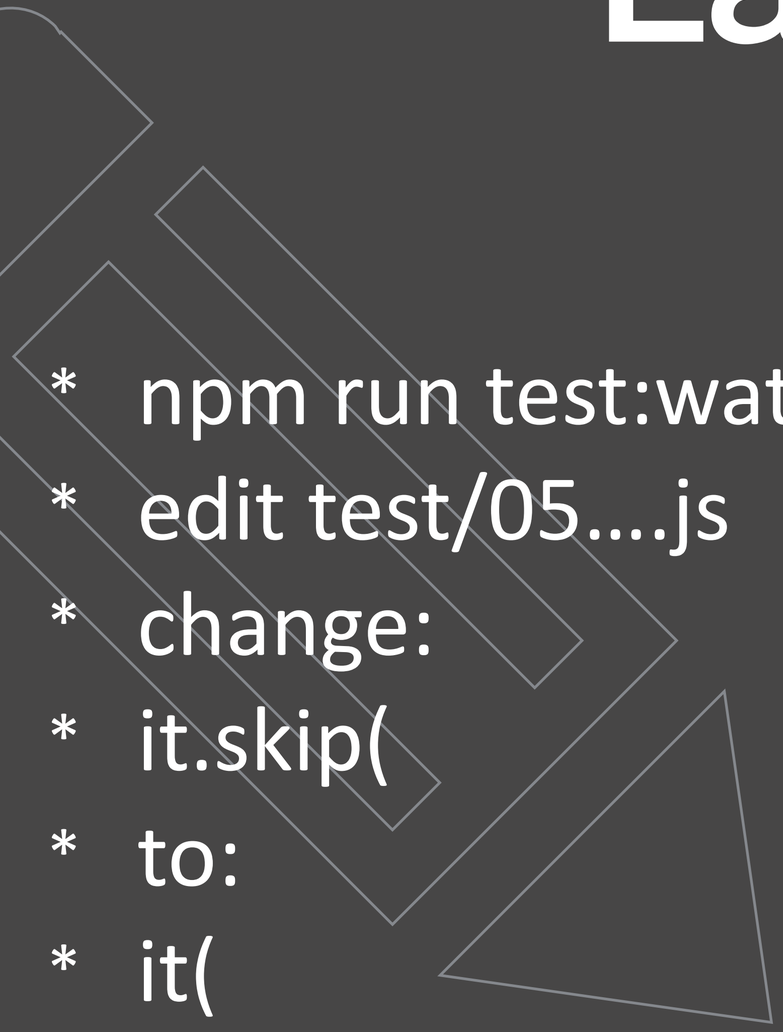
@prpatel

Modules - the future

```
const greeting = 'Hello';
    const name = 'Babel';
    const version = 'v5.0';
    export default {
      greeting: greeting,
      name: name,
      version: version
    };

--

import Message from './Message';
```

@prpatel

Lab 5: Modules

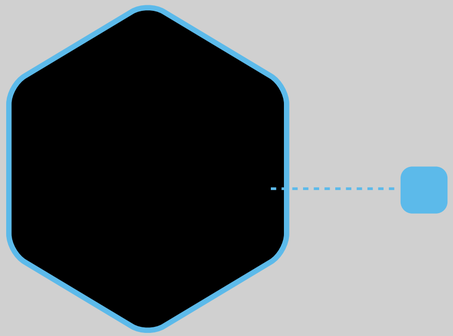


* npm run test:watch
* edit test/05….js
* change:
* it.skip(
* to:
* it(
* to enable the test!

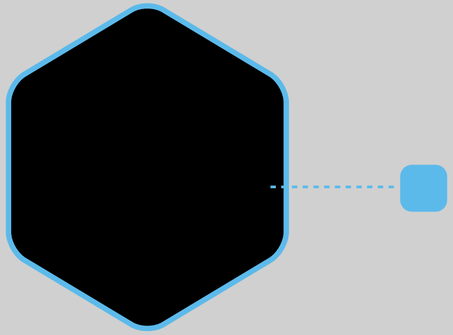
Lab 5: Discussion

@prpatel

LAB 6: OBJECT LITERALS



SET PROTOTYPE AT CONSTRUCTION



SHORTHAND FOR ASSIGNMENTS

OBJECT LITERALS

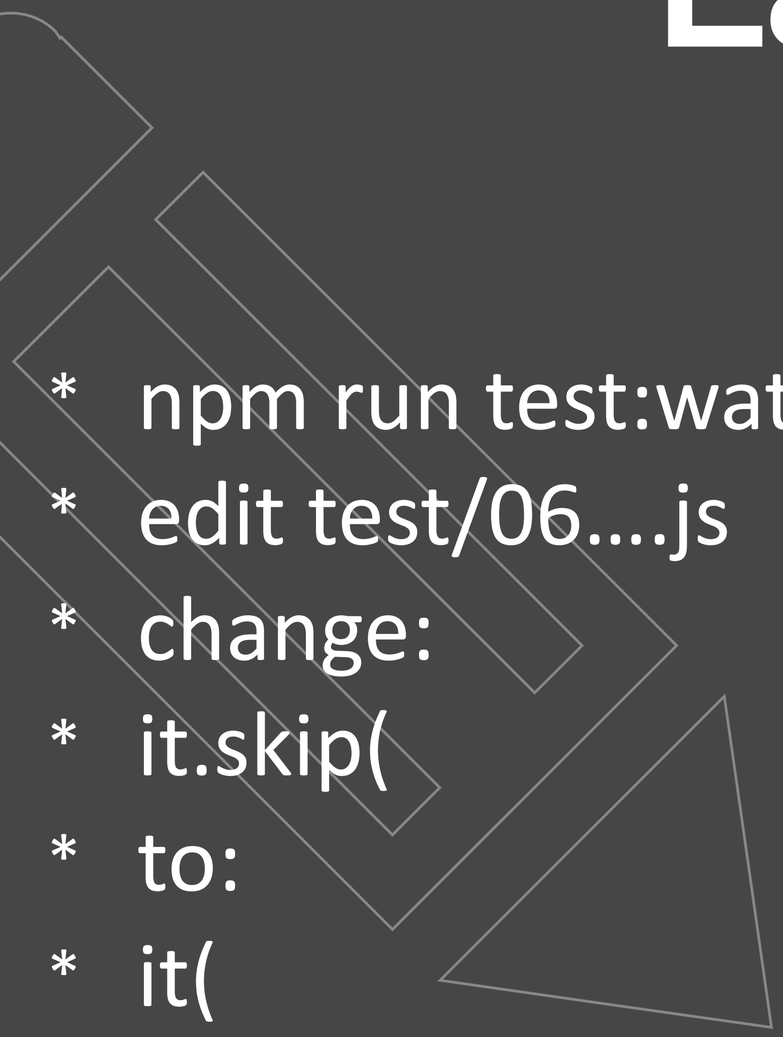
- DEFINED METHODS
- SUPER CALLS
- COMPUTED PROP NAMES

OBJECT LITERALS

Obj Literals

```
var obj = {
    // __proto__
    __proto__: theProtoObj,
    // Shorthand for ‘handler: handler’
    handler,
    // Methods
    toString() {
     // Super calls
     return "d " + super.toString();
    },
    // Computed (dynamic) property names
    [ 'prop_' + (() => 42)() ]: 42
};:
```

Lab 6: Literals

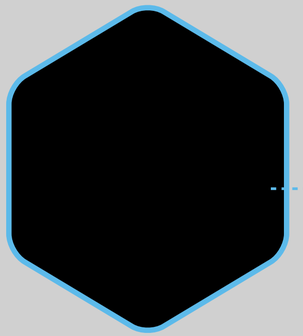


* npm run test:watch
* edit test/06….js
* change:
* it.skip(
* to:
* it(
* to enable the test!

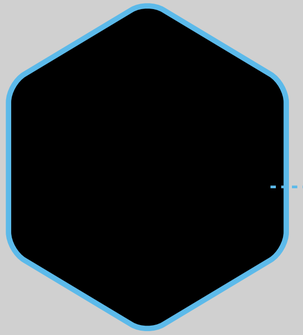
Lab 6: Discussion

@prpatel

LAB 7: PARAMETERS



- DEFAULT VALUES!



- BETTER VARARGS!

PARAMETERS

Params

```
function f(x, y=12) {
  // y is 12 if not passed (or passed as undefined)
  return x + y;
}
f(3) == 15
```

“arguments” keyword replaced with “args”

@prpatel

... rest

```
var sum = function(...args){
      return args.reduce( (sum, n) => sum + n );
    };
```

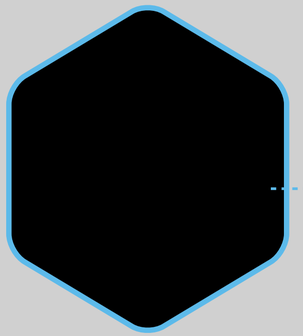
Lab 7: Parameters

* npm run test:watch
* edit test/07….js
* change:
* it.skip(
* to:
* it(
* to enable the test!

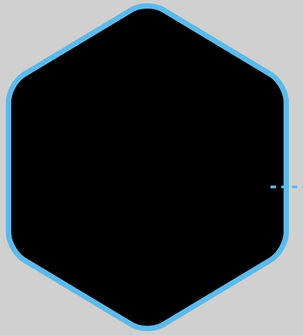
Lab 7: Discussion

@prpatel

LAB 8: SPREAD



TURN AN ARRAY INTO CONSECUTIVE ARGUMENTS



BETTER VARARGS!

SPREAD

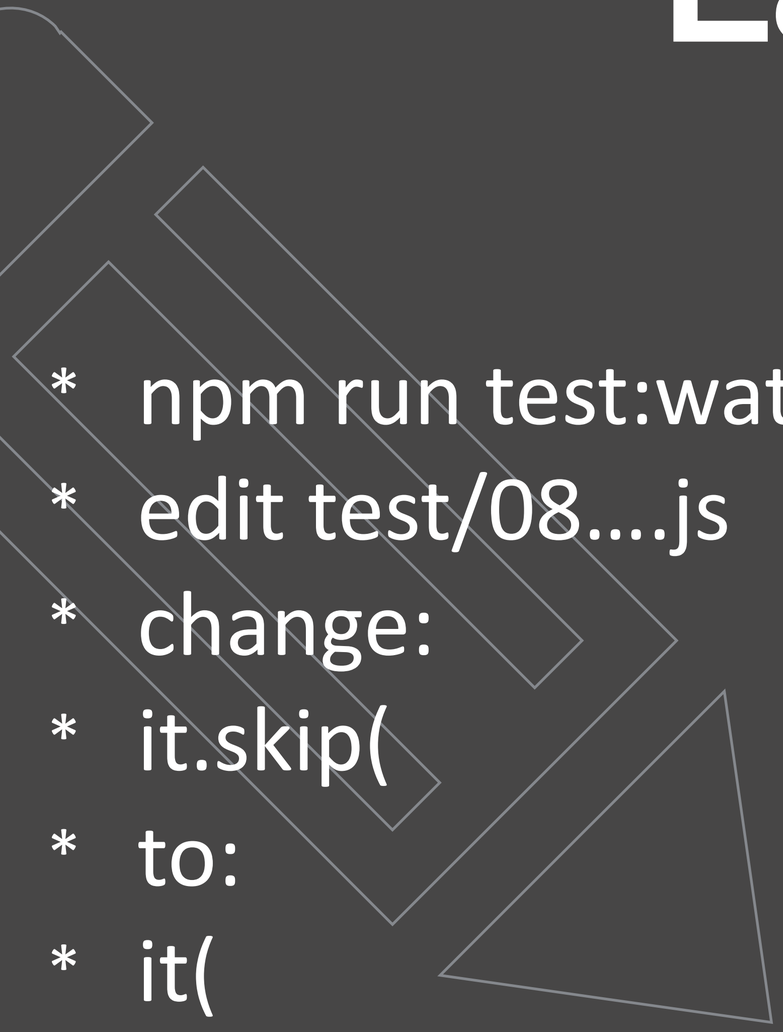
spread operator: pass array to a function and unwrap it

@prpatel

... spread

```
var sum = function(...args){
  return args.reduce( (sum, n) => sum + n );
};

var array = [1, 2, 3, 4];
// This is like calling `sum(1, 2, 3, 4)`
console.log(sum(...array)); // 10
```

Lab 8: Spread



* npm run test:watch
* edit test/08….js
* change:
* it.skip(
* to:
* it(
* to enable the test!

Lab 8: Discussion

LAB 9: ARROW FUNCTIONS

Function declaration - syntax sugar!

@prpatel

SWEET

```
setTimeout(function(){
  console.log('Test');
}, 100);
```

```
setTimeout(()=>{console.log('Test');}, 100);
```

```
var square = function(x) { return x * x; };
```

```
var square = (x) => {
  return x * x;
};

var square2 = x => x * x;
```

@prpatel

- BINDS `THIS` TO THE EVAL SCOPE, NOT THE RUNTIME SCOPE
- SHORTER!

ARROW FUNCTIONS

ARROW Functions

```
var sum = function(...args){
  return args.reduce( (sum, n) => sum + n );
};

var array = [1, 2, 3, 4];
// This is like calling `sum(1, 2, 3, 4)`
console.log(sum(...array)); // 10
```

@prpatel

Lab 9: ARROW Functions

* npm run test:watch
* edit test/09….js
* change:
* it.skip(
* to:
* it(
* to enable the test!

@prpatel

Lab 9: Discussion

@prpatel

LAB 10: CLASSES & OOP

Yes, real classes are coming to JavaScipt!

@prpatel

OOP IN JS?

@prpatel

ES5 Classes

```
var Person = function(name) {
    this.name = name;
};

Person.prototype.getName = function() {
    return this.name;
};

Person.prototype.setName = function(name) {
    this.name = name;
};

var alice = new Person("alice");
alice.getName(); // alice
```

@prpatel

ES6 Classes

```
class Person {
  constructor(name) {
    this.name = name;
  }
  getName() {
    return this.name;
  }
  setName(name) {
    this.name = name;
  }
}

var alice = new Person("alice");
alice.getName(); // alice
```

@prpatel

ES6 Extends

```
class Monster extends Character {
      constructor(x, y, name) {
        super(x, y);
        this.name = name;
      }
    }
```

Lab 10: CLASSES & OOP

* npm run test:watch
* edit test/10….js
* change:
* it.skip(
* to:
* it(
* to enable the test!

@prpatel

Lab 10: Discussion

LAB 11: SET

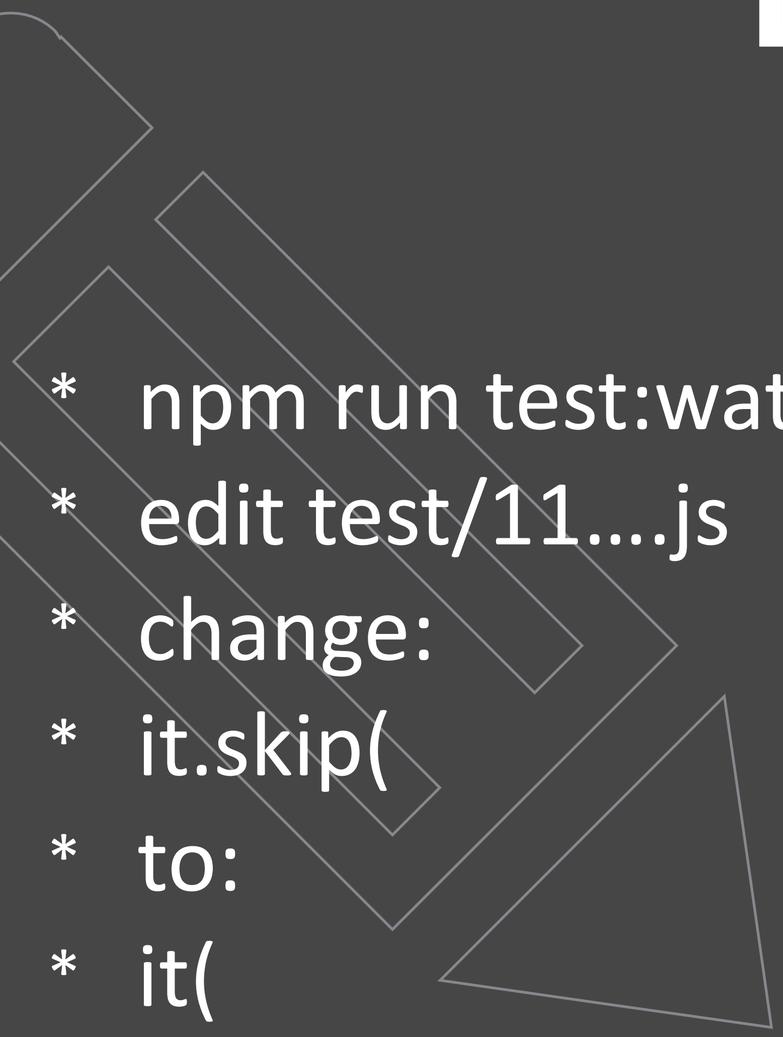
Wow! A new data structure in JavaScript: SET

@prpatel

Set

```
var s = new Set();
s.add("hello").add("goodbye").add("hello");
s.size === 2;
s.has("hello") === true;
```

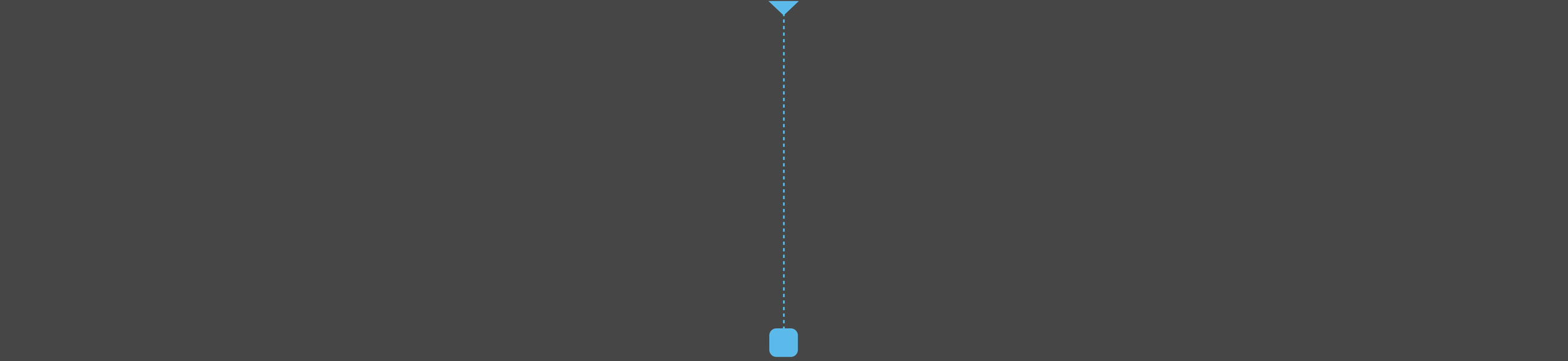
Lab 11: SET



* npm run test:watch
* edit test/11….js
* change:
* it.skip(
* to:
* it(
* to enable the test!

Lab 11: Discussion

LAB 12: MAP



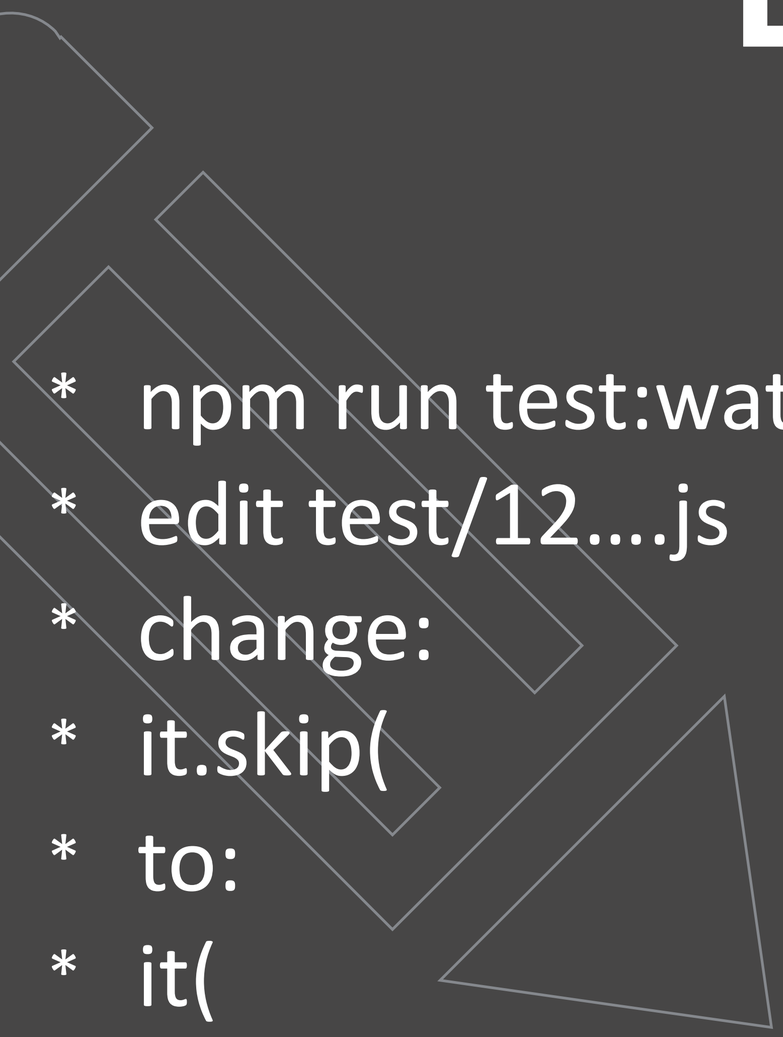
HOLY S**T! Another new data structure in JavaScript: map

@prpatel

map

```
var m = new Map();
m.set("hello", 42);
m.set(s, 34);
m.get(s) == 34;
```

Lab 12: map



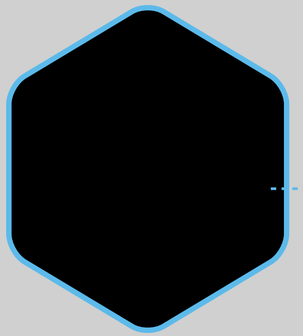
* npm run test:watch
* edit test/12….js
* change:
* it.skip(
* to:
* it(
* to enable the test!

Lab 12: Discussion

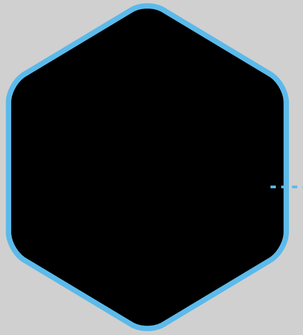
LAB 13: WEAKMAP & WEAKSET

HOLY S**T! Moe new data structures in JavaScript!

@prpatel



EXTEND OBJC FROM THE OUTSIDE WITHOUT INTERFERING WITH GC



EXTEND A SEALED OBJ, OR EXTERNAL OBJ

WEAK MAP & SET

EXTRA CREDIT

@prpatel

WEAK

```
// Weak Maps
var wm = new WeakMap();
wm.set(s, { extra: 42 });
wm.size === undefined

// Weak Sets
var ws = new WeakSet();
ws.add({ data: 42 });
// Because the added object has no other references, it will not be held
in the set
```

Lab 13: WEAK

* npm run test:watch
* edit test/13….js
* change:
* it.skip(
* to:
* it(
* to enable the test!

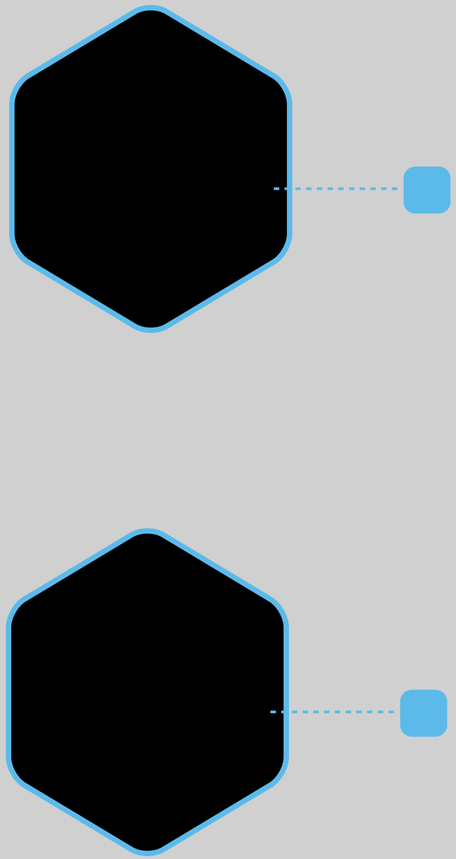
@prpatel

Lab 13: Discussion

LAB 14: PROMISES

Promises, now included in the language!

@prpatel



ASYNCHRONOUS PROGRAMMING

VALUE THAT MAY BE MADE AVAILABLE IN THE FUTURE

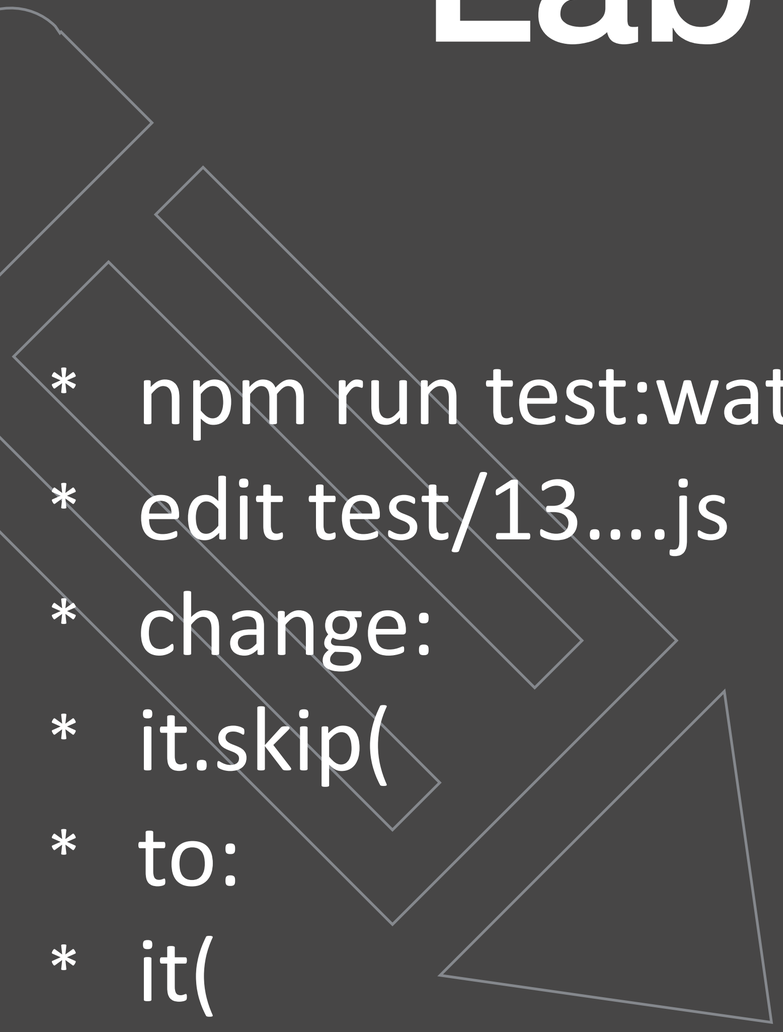
PROMISES

PROMISES

```
function timeout(duration = 0) {
    return new Promise((resolve, reject) => {
        setTimeout(resolve, duration);
    })
}

var p = timeout(1000).then(() => {
    return timeout(2000);
}).then(() => {
    throw new Error("hmm");
}).catch(err => {
    return Promise.all([timeout(100), timeout(200)]);
})
```

@prpatel

Lab 14: PROMISES



* npm run test:watch
* edit test/13….js
* change:
* it.skip(
* to:
* it(
* to enable the test!

@prpatel

Lab 14: Discussion

LAB 15: GENERATORS

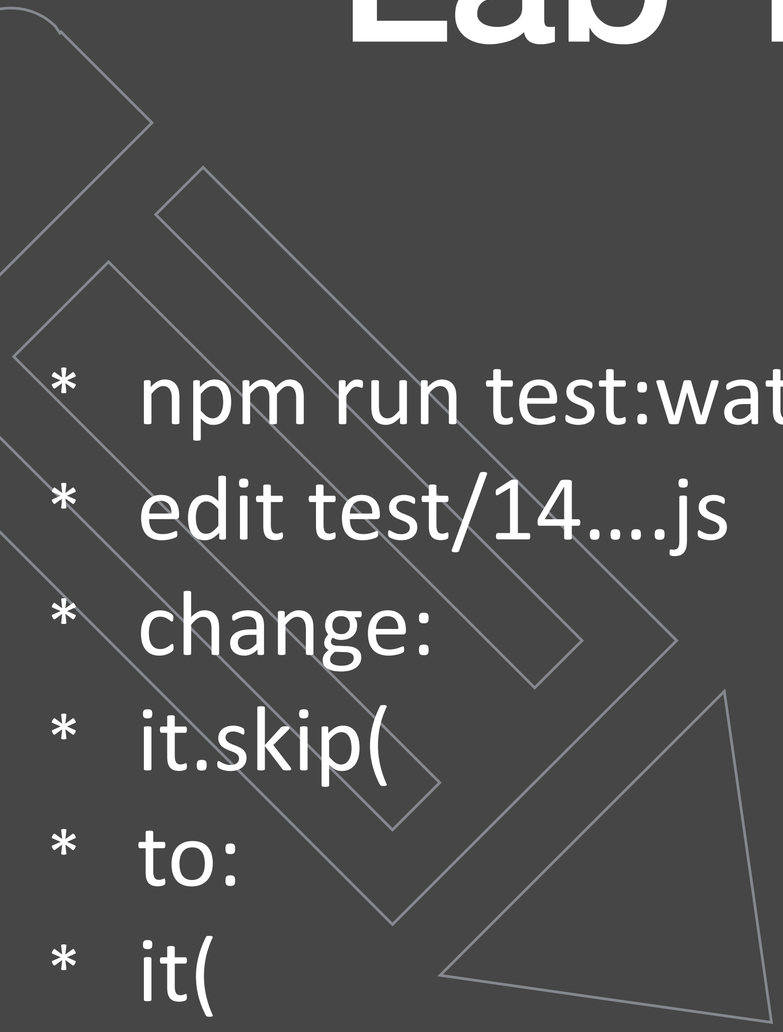
Generators simplify iterator-authoring using function* and yield

@prpatel

EXTRA CREDIT

@prpatel

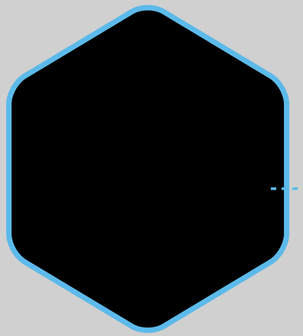
Lab 14: GENERATOR



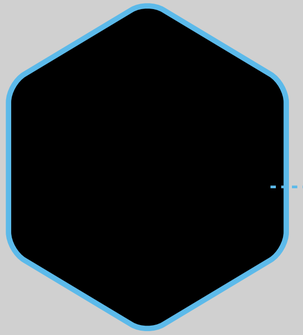
* npm run test:watch
* edit test/14….js
* change:
* it.skip(
* to:
* it(
* to enable the test!

LAB 16: SYMBOLS

Symbols are a new primitive type



SYMBOLS ENABLE ACCESS CONTROL FOR OBJECT STATE



SYMBOLS ARE UNIQUE

SYMBOLS

SYMBOLS

```
var MyClass = (function() {
  // module scoped symbol
  var key = Symbol("key");
  function MyClass(privateData) {
    this[key] = privateData;
  }
  MyClass.prototype = {
    doStuff: function() {
      ... this[key] ...
    }
  };
  return MyClass;
})();
var c = new MyClass("hello")
c["key"] === undefined
```

@prpatel

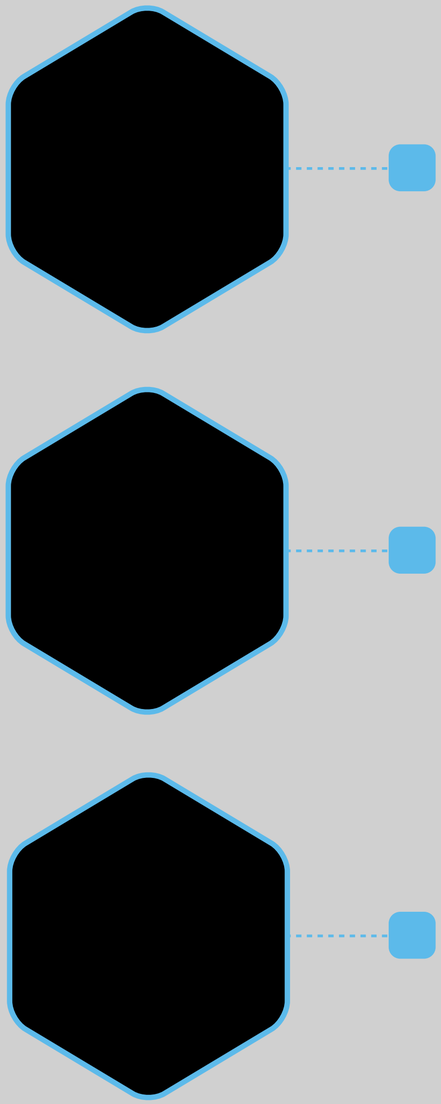
EXTRA CREDIT

@prpatel

Lab 16: SYMBOLS

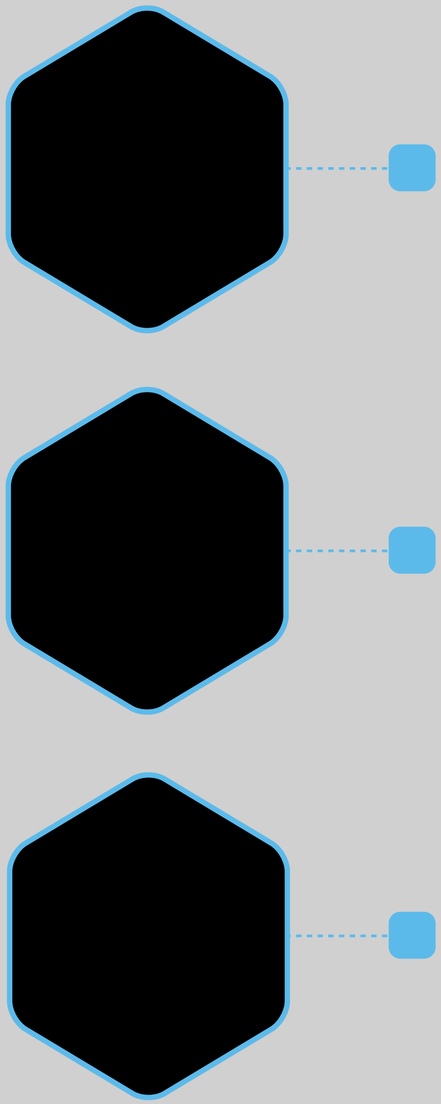
* npm run test:watch
* edit test/13….js
* change:
* it.skip(
* to:
* it(
* to enable the test!

Lab 16: Discussion



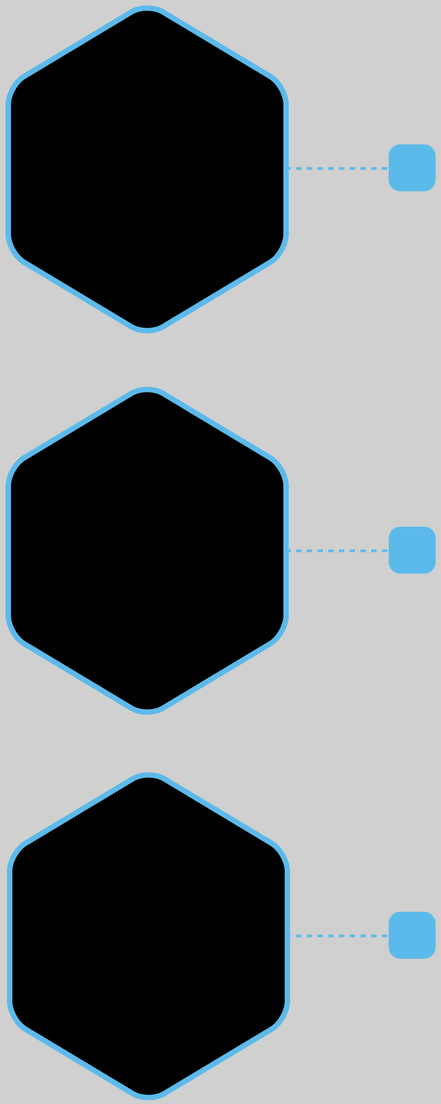
- COMPUTED PROPERTY KEYS
- UNICODE
- PROXIES

STUFF WE DIDN'T COVER



- BINARY AND OCTAL LITERALS
- REFLECT API
- TAIL CALLS

STUFF WE DIDN'T COVER

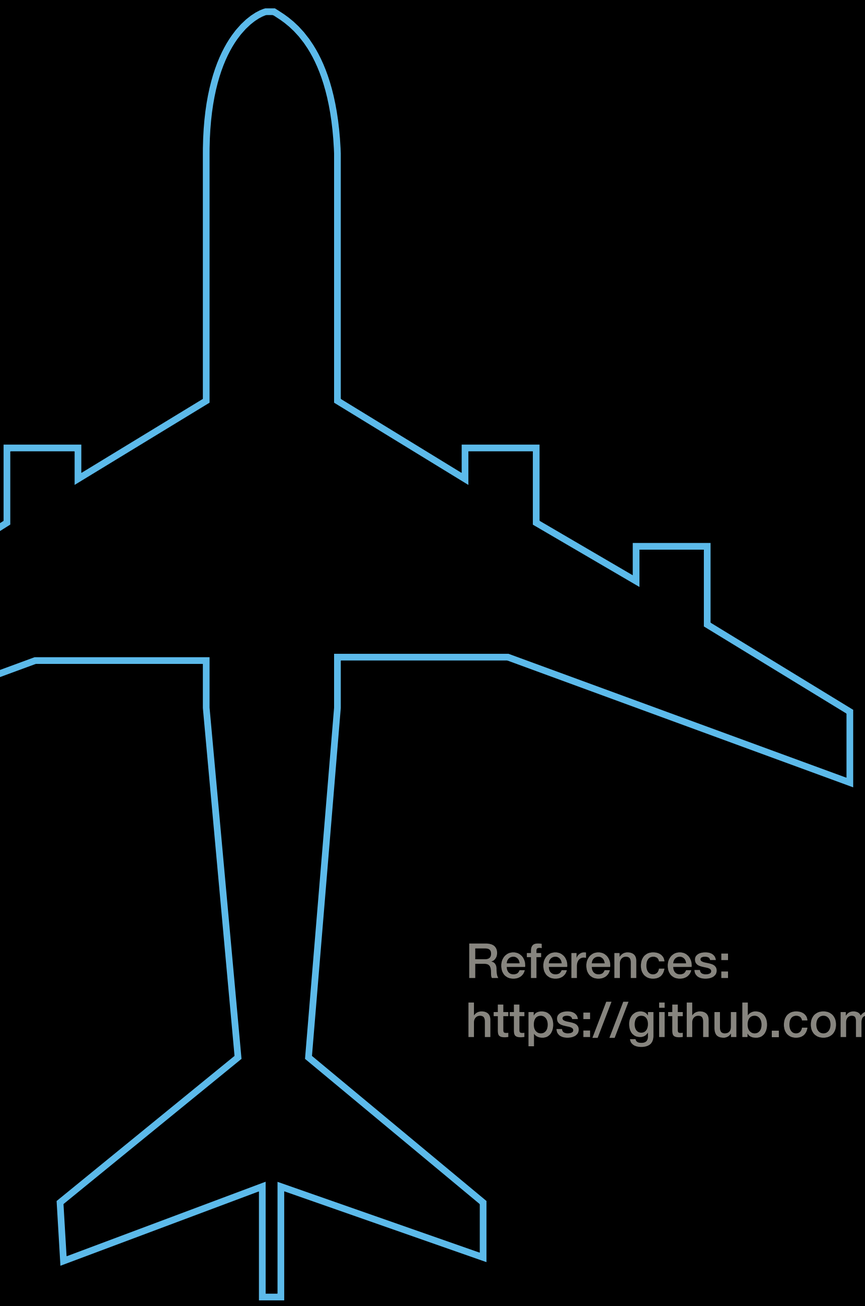


- ITERATORS + FOR..OF
- SUBCLASSABLE BUILT-INS
- ALL THE STUFF IN ES2016!

STUFF WE DIDN'T COVER

References

https://github.com/lukehoban/es6features
https://github.com/kentcdodds/es6-workshop



THANK YOU

References:
https://github.com/davezuko/react-redux-starter-kit

@prpatel

@prpatel

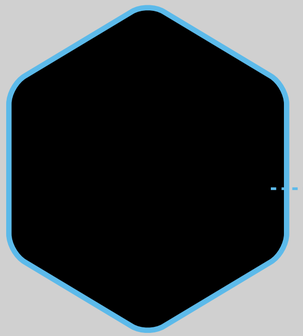
LAB 7: COMPUTED ROPERTIES

Use expressions as property names for defining/ accesing literals

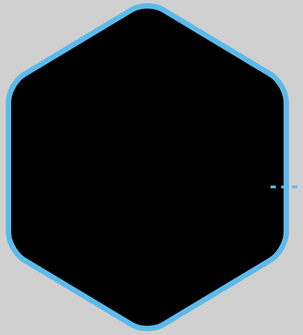
@prpatel

Props

```
// ES5
var prop = "foo";
    var obj = {};
    obj[prop] = "bar";

// ES6

    var prop = "foo";
    var obj = {
      [prop]: "bar"
    };
```



EXPRESS DYNAMIC PROPERTIES WITHOUT USING TEMPORARY VARIABLES



EVEN FUNCTION CALLS!

PROPS

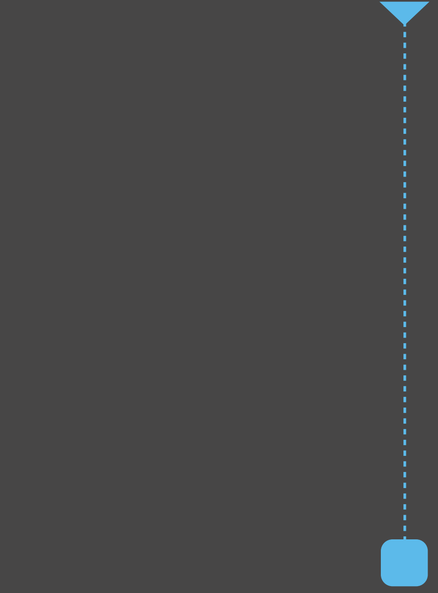
Lab 7: COMPUTED PROPS

* create lab7.js
* tower-of-babel, exercise 7
* Develop: babel-node lab7.js
* Run: tower-of-babel verify lab7.js
* HINT:
  Try to solve without any temporary variables (inline everything)

Lab 7: Discussion

@prpatel

LAB 8: ITERABLE – FOR OF

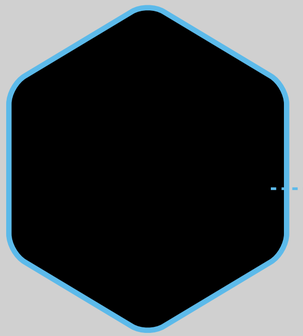


New Iterable type, can use “for of” with this new type

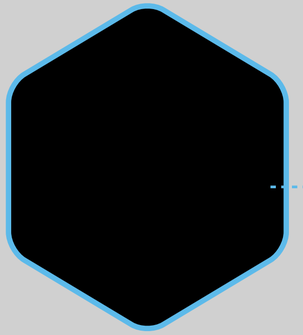
@prpatel

for of

```
var res = [];
    // The next line contains the for-of syntax.
    for (let element of [1, 2, 3]) {
      res.push(element * element);
    }
    console.log(res); // [1, 4, 9]
```



“FOR OF” CAN BE USED WITH ANY UTTERABLE



ITERABLE IS: SYMBOL.ITERATOR

ITERABLE

```
// calculating the fibonacci sequence to the 1000st number
var fibonacci = {
  // Make a method that has the Symbol.iterator function.
  [Symbol.iterator]() {
    let pre = 0, cur = 1;
    // The resulting iterator object has to have a next method:
    return {
      next() {
        // The result of next has to be an object with the property
`done` that states whether or not the iterator is done.
        [pre, cur] = [cur, pre + cur];
        if (pre < 1000)  return { done: false, value: pre };
        return { done: true };
      }
    }
  }
}
for (var n of fibonacci) {
  console.log(n);
```

@prpatel

Lab 8: ITERABLE

* create lab8.js
* tower-of-babel, exercise 8
* Develop: babel-node lab8.js
* Run: tower-of-babel verify lab8.js

Lab 8: Discussion