ES6 PARTY

CASSIDY WILLIAMS

SOFTWARE ENGINEER & DEVELOPER EVANGELIST AT CLARIFAI

ES6

ES-WHAATTT

ES-WHAATTT

- ES STANDS FOR ECMASCRIPT
- CURRENT JAVASCRIPT IS ACTUALLY ECMASCRIPT 5.1
- ES6 IS ALSO NICKNAMED ES2015

THE CHALLENGE

ES6 ISN'T EVERYWHERE.

WHICH SUCKS.

THE SOLUTION IS TRANSPILING

(TRANSFORMATION + COMPILING)

TRANSPILING

- TRANSFORM INTO EQUIVALENT MATCHES THAT WORK IN ES5 ENVIRONMENTS
- THIS IS USUALLY DONE DURING YOUR BUILD PROCESS

THE OTHER SOLUTION IS POLYFILLING

AKA SHIMMING

"POLYFILLS ARE A PATTERN FOR DEFINING EQUIVALENT BEHAVIOR FROM A NEWER ENVIRONMENT INTO AN OLDER ENVIRONMENT, WHEN POSSIBLE."

- KYLE SIMPSON, YOUDONTKNOWJS.COM

JS WILL EVOLVE CONSTANTLY

JS WILL EVOLVE CONSTANTLY

- THE BEST STRATEGY IS TO KEEP A TRANSPILER IN YOUR BUILD PROCESS, AND ADD SHIMS AS THINGS CHANGE.
- IF YOU DON'T DO THIS, YOU WILL FALL BEHIND.

WHO'S COMING TO PARTY?

- BLOCK SCOPING
- DEFAULT VALUES
- DESTRUCTURING
- LITERALS (OBJECT + TEMPLATE)
- ARROW FUNCTIONS =>

BLOCK SCOPING

BLOCK SCOPING ISN'T NEW.

"I'M STILL I'M STILL JENNY FROM THE BLOCK... SCOPE"
- J-LO

BLOCK SCOPING ISN'T NEW.

```
var x = 10;

(function sample() {
    var x = 20;
    console.log(x);    // 20
})();

console.log(x);        // 10
```

BLOCK SCOPING

IN ES6, WE CAN CREATE DECLARATIONS IN ANY BLOCK, NOT JUST IN FUNCTIONS.

WE INVITED `let` TO HELP

```
var x = 10;

{                           // All we need is {...} for a scope
    let x = 20;
    console.log(x);         // 20
}

console.log(x);             // 10
```

YOU CAN MAKE SCOPES OUT OF ANYTHING

```
let x = 2;

if (x > 1) {
    let y = x * 3;
    console.log(y);         // 6

    for (let i = x; i < y; i++) {
        let j = i + 15;
        console.log(j);
    }                       // 17 18 19 20

    let z = x + y;
    console.log(z);         // 8
}
```

let VS. var

```
{
    console.log(x);      // undefined
    console.log(y);      // ReferenceError

    var x;
    let y;
}
```

WE BROUGHT ALONG `const` TO PARTY, TOO

```
{
    const x = 10;
    console.log(x);         // 10

    x = 7;                  // TypeError
}
```

FUNCTIONS ARE BLOCK-SCOPED, TOO!

BLOCK-SCOPED FUNCTIONS

```
if (weGonnaParty) {
    function fiesta() {
        console.log('We are partying tonight!');
    }
} else {
    function fiesta() {
        console.log('We are chillin tonight.');
    }
}

fiesta();
```

DEFAULT VALUES

WE'VE MADE LOTS OF ATTEMPTS TO PROPERLY USE DEFAULT VALUES.

```
function blerg(x, y) {
    x = x || 20;
    y = y || 30;

    console.log(x + y);
}

blerg();                    // 50
blerg(7);                   // 37
blerg(null, 16);            // 46
blerg(7, 6);                // 13
```

ES6 DEFAULT VALUES

```
function blerg(x = 20, y = 30) {
    console.log(x + y);
}

blerg();                    // 50
blerg(0, 314);              // 314
blerg(7, 6);                // 13
blerg(5);                   // 35
blerg(undefined, 5); // 25
blerg(5, null);             // 5 (because null -> 0)
```

ES6 DEFAULT VALUES CAN BE EXPRESSIONS OR FUNCTIONS

```
function party(x = 3, y = 8 * x, z = blerg(x)) {
    console.log(x, z);
}
```

DESTRUCTURING

DESTRUCTURING

DESTRUCTURING WAS INVITED FOR SPECIFICALLY ARRAY DESTRUCTURING AND OBJECT DESTRUCTURING

THIS IS WHAT WE DO NOW

```
function party() {
    return [1, 2, 3];
}

function fiesta() {
    return {x : 4, y : 5, z : 6};
}

var temp1 = party(), u = temp1[0], v = temp1[1], w = temp1[2];
console.log(u, v, w); // 1 2 3

var temp2 = fiesta(), x = temp2.x, y = temp2.y, z = temp2.z;
console.log(x, y, z); // 4 5 6
```

THIS IS WHAT ES6 LETS US DO

```
var [a, b, c] = party();
var {x : x, y : y, z : z} = fiesta();

console.log(a, b, c); // 1 2 3
console.log(x, y, z); // 4 5 6
```

THIS IS WHAT ES6 LETS US DO

```
var [a, b, c] = party();
var {x, y, z} = fiesta();

console.log(a, b, c); // 1 2 3
console.log(x, y, z); // 4 5 6
```

THIS IS WHAT ES6 LETS US DO

```
var [a, b, c] = party();
var {x : taco, y : burrito, z : enchilada} = fiesta();

console.log(a, b, c); // 1 2 3
console.log(x, y, z); // ReferenceError
console.log(taco, burrito, enchilada); // 4 5 6
```

THIS IS WHAT ES6 LETS US DO

```
var party1 = 'candy', party2 = 'cake';

var yolo = {x : party1, y : party2};
var {x : fiesta1, y : fiesta2} = yolo;

console.log(fiesta1, fiesta2);        // 'candy' 'cake'
```

BUT WAIT THERE'S MORE

```
var x = 'party', y = 'fiesta';
[y, x] = [x, y];
console.log(x, y);                  // 'fiesta' 'party'
```

WHOA NO TEMP VARIABLE!

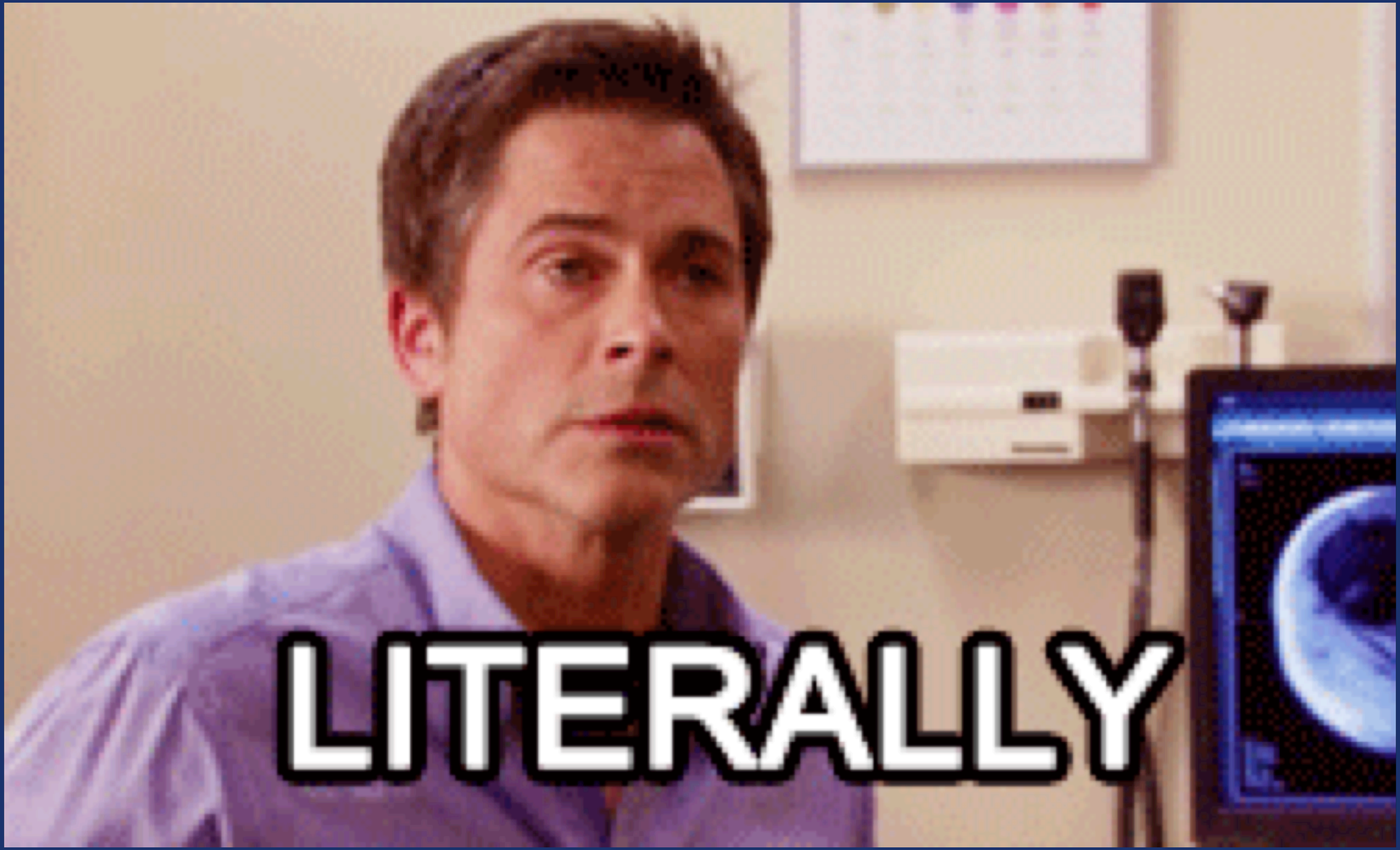
BUT WAIT THERE'S MORE

```
var {party : x, party : y} = {party : 1};

x;  // 1
y;  // 1
```

DESTRUCTURING ISN'T JUST FOR TYPING LESS, BUT MORE FOR DECLARATIVE READABILITY.

LITERALS


LITERALLY

LITERALS

BEFORE:

```
var z = {
        x : x,
        y : y
};
```

LITERALS

AFTER:

```
var z = {
          x,
          y
};
```

LITERALS

BEFORE:

```
var z = {
    x: function() {
        // lalala
    },
    y: function() {
        // lalala
    }
}
```

LITERALS

AFTER:

```
var z = {
    x() {
      // lalala
    },
    y() {
      // lalala
    }
}
```

TEMPLATE LITERALS CAME TO THE PARTY

- BETTER DEFINED AS A "STRING LITERAL"
- INTRODUCES THE ` BACKTICK AS THE DELIMITER

TEMPLATE LITERALS

BEFORE:

```
var name = "Cassidy";
var greeting = "Hello, " + name + "!";
console.log(greeting);                    // "Hello, Cassidy!"
```

TEMPLATE LITERALS

AFTER:

```
var name = "Cassidy";
var greeting = `Hello, ${name}!`;
console.log(greeting);              // "Hello, Cassidy!"
```

TEMPLATE LITERALS

```
function capitalize(s) {
    return s.toUpperCase();
}

var you = 'audience';

var text =
`A very ${capitalize( 'big' )} hello
to you, ${capitalize( `my ${you}` )}!`;

console.log(text);
// A very BIG hello
// to you, MY AUDIENCE!
```

ARROW FUNCTIONS

ARROW FUNCTIONS ARE THE GUAC OF THE PARTY

BEFORE:

```
function guac(x,y) {
    return x + y;
}
```

AFTER:

```
var guac = (x,y) => x + y;
```

ARROW FUNCTIONS

- ARROW FUNCTIONS ARE ALWAYS FUNCTION EXPRESSIONS
- THEY ARE ALWAYS ANONYMOUS
- THEY REDEFINE `this`
- THEY ARE BEAUTIFUL

ARROW FUNCTIONS & this

BEFORE:

```
var controller = {
    makeRequest: function(..){
        var self = this;

        btn.addEventListener( "click", function(){
            // ..
            self.makeRequest(..);
        }, false );
    }
};

// Source: ES6 & Beyond, Kyle Simpson
```

ARROW FUNCTIONS & this

AFTER:

```
var controller = {
    makeRequest: function(..){
        btn.addEventListener( "click", () => {
            // ..
            this.makeRequest(..);
        }, false );
    }
};

// Source: ES6 & Beyond, Kyle Simpson
```

ARROW FUNCTIONS

```
var party = [1, 2, 3, 4, 5];
party = party.map( guac => guac * 5 );
console.log(party);                           // [5, 10, 15, 20, 25]
```

THIS IS ALL JUST SYNTAX AND ORGANIZATION STUFF.

ES6 IS JUST THE BEFORE-PARTY.

ADDITIONAL RESOURCES

- YOU DON'T KNOW JS: YOUDONTKNOWJS.COM
- ES6 SHIMS: GITHUB.COM/PAULMILLR/ES6-SHIM
- ES6 KATAS: ES6KATAS.ORG
- ES6 LEARNING: GITHUB.COM/ERICDOUGLAS/ES6-LEARNING

TRANSPILERS

- BABEL: BABELJS.IO
- GOOGLE CAJA: CODE.GOOGLE.COM/P/GOOGLE-CAJA
- ES TRANSPILER: GITHUB.COM/KAISELLGREN/ES-TRANSPILER

THANK YOU!

@CASSIDOO