LET'S CONST TOGETHER, RIGHT NOW WITH ES3

Olov Lassus / @olov

SAID ABOUT A LANGUAGE FROM THE 50'S

" Here is a language so far ahead of its time that it was not only an improvement on its predecessors but also on nearly all its successors. "

_ SAID THIS, ABOUT _?

“ Here is a language so far ahead of its time that it was not only an improvement on its predecessors but also on nearly all its successors. ”

TONY HOARE SAID THIS, ABOUT _?

“ Here is a language so far ahead of its time that it was not only an improvement on its predecessors but also on nearly all its successors. ”

ALGOL

STATIC / LEXICAL SCOPE

Old-school but still going strong (ALGOL 58). Lexical scope basically means that you can match a variable reference to its corresponding declaration, without running the program.

JS SCOPE: 'IT'S COMPLICATED'

- Global object
- `this`
- `with`
- `eval`
- Function scope (hoisting)

GLOBAL OBJECT

```
// x is not defined here

(function() {
    x = "yup"; // oops, we forgot var
})();
console.log(x); // yup, x is now a global
```

Can be dealt with

GLOBAL OBJECT

```
"use strict";

window[Math.random() > 0.5 ? "x" : "y"] = "yup";

console.log(x); // will it crash?
```

Can be dealt with

THIS

`this` is special as in most object oriented languages.

A common JS gotcha is that `this` changes as soon as you make a regular function call (instead of calling a method), becomes `window`/`global` or `null`.

Can be dealt with

WITH

Pretty crazy and destroys static scope

Can be dealt with (use with caution)

EVAL

Can introduce new variable names so destroys static scope

Can be dealt with (use with caution)

FUNCTION SCOPE

SHADOWING

```
var str = "kramer";
function bangify(str) {
    str = str + "!";
    return str;
}
console.log(bangify(str)); // prints "kramer!"
console.log(str); // prints "kramer";
```

SHADOWING

```
var str = "kramer";
function bangify(s) {
    var str = s + "!";
    return str;
}
console.log(bangify(str)); // prints "kramer!"
console.log(str); // prints "kramer";
```

IN MOST OTHER {} LANGUAGES

```
void fn() {
    int y = 0;
    // z is certainly not defined here
    for (int x = 0; x < 10; x++) {
        int y = x * 2;
        int z = y;
    }
    printf("%d\n", y); // prints 0
    // z is not defined here
}
fn();
```

IN JAVASCRIPT

```
function fn() {
    var y = 0;
    // z is active here (undefined)
    for (var x = 0; x < 10; x++) {
        var y = x * 2;
        var z = y;
    }
    console.log(y); // prints 18
    // z is active here too (18)
}
fn();
```

IN JAVASCRIPT (NORMALIZED EQUIVALENT)

```
function fn() {
    var y = undefined, z = undefined;
    y = 0;
    // z is active here (undefined)
    for (var x = 0; x < 10; x++) {
        y = x * 2;
        z = y;
    }
    console.log(y); // prints 18
    // z is active here too (18)
}
fn();
```

FUNCTION SCOPE

We have various ways of dealing with the limitations of function scope in JS, most of them are not that fun.

Manual hoisting is bad for robustness, encapsulation and readability.

ENTER ES6!

LET

`let` is the best thing since sliced arrays.

`let` fixes `var` and gives you block scope!

IN ES6

```
"use strict";
function fn() {
    let y = 0;
    // z is certainly not defined here
    for (let x = 0; x < 10; x++) {
        let y = x * 2;
        let z = y;
    }
    console.log(y); // prints 0
    // z is not defined here
}
fn();
```

RIGHT NOW?

IN NODE.JS

```
~ % cat yup.js
"use strict";
let x = "yup";
{ let x = "nope"; }
console.log(x);

~ % node --harmony yup.js
yup
```

Right now!

IN CHROME (WHILE DEVELOPING)

chrome://flags: check Enable experimental JavaScript

```
> (function(){
    "use strict";
    let x = "yup";
    { let x = "nope"; }
    console.log(x);
})();
yup
```

Right now!

IN ALL BROWSERS?

DEFS.JS

Static scope analysis and transpilation of ES6 block scoped `const` and `let` variables, to ES3

`https://github.com/olov/defs`

MIT licensed

IN THE BROWSER USING DEFS.JS

```
~ % npm install -g defs

~ % defs yup.js
"use strict";
var x = "yup";
{ var x$0 = "nope"; }
console.log(x);
```

Plain ES3 that runs everywhere

IN THE BROWSER USING DEFS.JS

Going back to the previous example

```
~ % cat example.js
"use strict";
function fn() {
    let y = 0;
    // z is certainly not defined here
    for (let x = 0; x < 10; x++) {
        let y = x * 2;
        let z = y;
    }
    console.log(y); // prints 0
    // z is not defined here
}
fn();
```

IN THE BROWSER USING DEFS.JS

Transpiles to

```
~ % defs example.js
"use strict";
function fn() {
    var y = 0;
    // z is certainly not defined here
    for (var x = 0; x < 10; x++) {
        var y$0 = x * 2;
        var z = y$0;
    }
    console.log(y); // prints 0
    // z is not defined here
}
fn();
```

AHA, I CAN FOOL YOU!

```
"use strict";
function fn() {
    let y = 0;
    console.log(z);
    for (let x = 0; x < 10; x++) {
        let y = x * 2;
        let z = y;
    }
    console.log(y); // prints 0
    // z is not defined here
}
fn();
```

NOT REALLY

```
~ % defs example.js

line 4: reference to unknown global variable z
```

WHATEVER, THIS TIME I'LL FOOL YOU!

```
"use strict";
function fn() {
    let y = 0;
    // z is certainly not defined here
    for (let x = 0; x < 10; x++) {
        console.log(z);
        let y = x * 2;
        let z = y;
    }
    console.log(y); // prints 0
    // z is not defined here
}
fn();
```

NOT REALLY

```
~ % defs example.js

line 6: z is referenced before its declaration
```

CONST

If you liked `let`, you're gonna love `const`

LIKE LET, JUST BETTER

`const` is the same thing as `let`, with one major difference.

A variable initialized as `const` can never be modified.

Note: Not the same thing as deep immutability

THAT IS AWESOME

WAIT, WHY IS THAT?

Well, it turns out that in most programs the vast majority of your variables are actually `const`s, just that you have had no way of expressing it or automatically verifying it.

Note: Compare with `final` (Java), `val` (Scala), `const` C/C++`

LET'S GO BACK TO THE EXAMPLE

```
"use strict";
function fn() {
    let y = 0;
    for (let x = 0; x < 10; x++) {
        let y = x * 2;
        let z = y;
    }
    console.log(y); // prints 0
}
fn();
```

USING CONST INSTEAD

```
"use strict";
function fn() {
    const y = 0;
    for (let x = 0; x < 10; x++) {
        const y = x * 2;
        const z = y;
    }
    console.log(y); // prints 0
}
fn();
```

WOHOO! BREAKING YOU!

```
"use strict";
function fn() {
    const y = 0;
    for (let x = 0; x < 10; x++) {
        const y = x * 2;
        y++; // this should't pass
        const z = y;
    }
    console.log(y); // prints 0
}
```

NOT REALLY

```
~ % defs example.js

line 6: can't assign to const variable y
```

CAVEAT 1: REFERENCED BEFORE DECLARATION

```
const fns = [function() { return x; }, function() { return 1; }];
function zeroOrOne() { return Math.floor(Math.random() * 2); }

fns[zeroOrOne()]();
let x = 1;
fns[zeroOrOne()]();
```

Impossible to catch with static analysis so requires run-time checking. Not a problem in practice.

CAVEAT 2: LOOP CLOSURES

```
for (let x = 0; x < 10; x++) {
    let y = x;
    arr.push(function() { return y; });
}
// line 3: can't transform closure. y is defined outside closure,
//         inside loop
```

`let y` is a new binding per iteration while a transformed `var y` won't be.

CAVEAT 2: LOOP CLOSURES

```
for (let x = 0; x < 10; x++) {
    (function(y) {
        arr.push(function() { return y; });
    })(x);
}
```

The solution is to create the binding manually as we've always done.

CONFIGURING DEFS

Example `defs-config.json`:

```
{
    "environments": ["node", "browser"],

    "globals": {
        "my": false,
        "hat": true
    },
    "disallowVars": false,
    "disallowDuplicated": true,
    "disallowUnknownReferences": true
}
```

WRAPPING UP

- `let` and `const` are my favorite ES6 features
- Works in Node today already (`--harmony`)
- `defs.js` transpiles it to beautiful ES3 code
- Thus it works in the browsers today already

GOOD TO GO?

- I don't use `var` any longer in my non-public node.js code
- I still use `var` in most of my open sourced npm modules
- All my shared node/client code is `constlet` style
- I'm migrating to `constlet` style for my client side code
- `defs.js` itself is `constlet` style (and self-transpiles)
- `browserify` and `defs.js` is an awesome combo and a plugin is coming
- I'd love to see more `constlet` style modules on npm

REMEMBER?

_ is not the new _, _ is

REMEMBER!

`let` is not the new _, _ is

REMEMBER!

`let` is not the new `var`, _ is

REMEMBER!

`let` is not the new `var`, `const` is

THANK YOU!

@olov

https://github.com/olov/defs

```
npm install -g defs
```