Cheatography

ES6 / ECMAScript 2015 / ES2015 Quick Reference Cheat Sheet

by dwapi via cheatography.com/43665/cs/13100/

Hoisting

```
//var declaration example
(funct ion() {
   var foo = 1;
    con sol e.l og(foo + " " + bar);
   var bar = 2;
})();
// Alerts "1 undefi ned " instead of throwing an
error.
// It's aware of bar b/c the declar ation was
hoisted to top of function.
// So no error, but the value is undefined until
after the alert.
//function example
foo();
function foo() {
    ale rt( " Hel lo! ");
}
// Same as above, the function declar ation is
hoisted above the call();
```

Hoisting is JavaScript's default behavior of moving all var and function declarations to the top of the current scope (to the top of the current script or the current function).

Functions

```
// Arrow Function
setTim eout(() => { consol e.l og( 'de layed') },
1000)
// Scoped Functions
{
   let cue = 'Luke, I am your father'
    con sol e.l og(cue)
}
>'Luke, I am your father'
// Scoped Function Equivalent with Immedi ately
Invoked Function Expres sions (IIFE)
(function () {
var cue = 'Luke, I am your father'
consol e.l og(cue) // 'Luke, I am -
}())
consol e.l og(cue) // Reference Error
// Default Params!!
function test(num = 1) { consol e.l og(num) }
```

Promises

```
// Promise itself has three states: Pending,
Fulfilled , Rejected
let example = new Promis e(( res olve, reject) =>
{
        req ues t.g et(url, (error, response,
body) => {
           if (body) {
                res olv e(J SON.pa rse (bo dy)); //
fulfilled
           } else {
               let reason = new Error( 'There wan
an error');
                rej ect (re ason); // reject
           }
       })
}).the n((val) => consol e.l og( " ful fil -
led :", val))
 .ca tch ((err) => consol e.l og( " rej ect -
ed: ", err));



// Run multiple promises in parallel
Promis e.all([
    pro mise1, promise2, promise3
]).then(() => {
   // all tasks are finished
})
```

If you want to use Promises for recurring values or events, there is a better mechanism/pattern for this scenario called streams.

Let vs Var

let variables are limited in scope to the block, statement, or expression on which it is used	var defines variables globally, or locally to an entire function regardless of block scope
Variables declared with let or const do not get hoisted	variables defined with var DO get hoisted

Can NOT be re-decalred.
let a = 5;
let a =6; // SyntaxError: redecl-aration
	CAN be re-decalred.
	var a = 5;
	var a = 6; // no error. a=6

let is more performant, and better for Garbage Collection

Cheatography

ES6 / ECMAScript 2015 / ES2015 Quick Reference Cheat Sheet

by dwapi via cheatography.com/43665/cs/13100/

Let vs Var (cont)

Support: Severside=Everywhere. Browsers IE11+ Android 56+ (altho you can use Babel transpiler)	Universal support

Hoisting is a bit of a quirk in JS. let behaves more like variable declarations in most other langs

Douglas Crockford advises to always use let.

Maps and Sets

What are Maps?

> Basically a Object/hash with some advantages.

Difference between Maps and Objects:

> An Object has a prototype, so there are default keys in the map
> Maps preserve K-V in order they were added -- allows for iteration
> Keys in Objects are treated like Strings Object.keys(myHash) returns a bunch of strings. They can be anything in a Map.

What is a WeakMap?

> A Map where the keys are weak. Meaning if a key is deleted the value will be GC'd

What is a Set?

> Highly performant array that preserves order of insertion, but does not index.

Other New Features

Template Literals	`My dog is ${age} years old`
Default Params	function(greeting='Howdy'){ console.log(greeting) }
Object literals	myHash = {color, size} // same as {color: color, size: size}
Spread Operator	[1, 2, ...more] or list.push(...[3, 4]) or new Date(...[2015,8,1])
Generator Function(*)	function *foo() {} //These can be paused & resumed later????
const variable type	Block scoped, immutable reference (vals in arrays can change)
Symbol variable type	Immuteable (Unclear advantage of these over hash obj)

Classes, Inheritance, Setters, Getters

```
class Rectangle extends Shape {
    con str uct or(id, x, y, w, h) {
        sup er(id, x, y)
        thi s.width = w
        thi s.h eight = h
    }
    // Getter and setter
    set width(w) {
        thi s._ width = w
    }
    get width() {
        return this._ width
    }
}
class Circle extends Shape {
    con str uct or(id, x, y, radius) {
        sup er(id, x, y)
        thi s.r adius = radius
    }
    do_a(x) {
       let a = 12;
        sup er.d o_a(x + a);
    }
    static do_b() { ...
    }
}
Circle.do_b()
```

Spread Operator and Destructuring

```
> const [ cat, dog, ...fish ] = ['schroedinger', 'Laika', 'Nemo', 'Dori']
> fish // -> ['Nemo', 'Dori']
> cat // -> ['schr oed inger']
> let arr = [1, 2, 3]
> [...arr, 4, 5, 6]
> [1, 2, 3, 4, 5, 6]
```