ES5 - Nociones importantes

1 - Main Event Loop & Call stack (+setTimeout…)
2 - Callback & Callback hell
3 - This binding

Novedades es6

1 - class
2 - modules / import / export
3 - Fat arrow - () => ...
4 - Variable interpolation with strings (`/${myVariable}/detail`)
5 - Spread/Rest
6 - Destructuring
7 - New features (const, let, new objects(Map, WeakMap, Symbol…)
8 - New methods & Functional JS (Array.findIndex()…), Array.from(…)) - (array map, filter, reduce, find...)

https://github.com/getify/You-Dont-Know-JS

ES5 - Nociones importantes

1 - Main Event Loop & Call stack (+setTimeout…)

RECURSOS

- http://2014.jsconf.eu/speakers/philip-roberts-what-the-heck-is-the-event-loop-anyway.html

INTRO

Javascript es un lenguaje compilado, *single-thread* y se ejecuta en el *main-thread* del navegador. Podemos conseguir "*multi-threading*" con el uso de *Workers*, pero con ciertas limitaciones (por ejemplo, no hay acceso al DOM desde un Worker).

Al ser *single-thread* y ejecutarse en el *main-thread* del navegador, podemos encontrarnos con situaciones de bloqueo por la ejecución de un script síncrono.

Call stack - Event Loop - Task Queue - WebApis (min 13:40 del vídeo anterior)

Ejemplo práctico

```
console.log("Hey");

setTimeout(function() {
    console.log("Ho");
}, 1000);

console.log("Let's go");
```

```
console.log("Hey");

setTimeout(function() {
    console.log("Ho");
}, 0);

console.log("Let's go");
```

Sync vs Async

```
// Synchronous

[1, 2, 3, 4].forEach(function(i) {
    console.log('processing sync');
});
```

```
// Asynchronous

function asyncForEach(array, cb) {
    array.forEach(function() {
        setTimeout(cb, 0);
    });
}

arrayForEach([1, 2, 3, 4], function(){
    console.log('processing async');
});
```

2 - Callbacks & Callback hell

RECURSOS

- http://callbackhell.com/

Callback Hell

https://github.com/getify/You-Dont-Know-JS/blob/master/async%20%26%20performance/ch2.md#nestedchained-callbacks

Control Inversion & Trust Issues

https://github.com/getify/You-Dont-Know-JS/blob/master/async%20%26%20performance/ch2.md#trust-issues

3 - This binding

https://github.com/getify/You-Dont-Know-JS/blob/master/this%20%26%20object%20prototypes/ch2.md#nothing-but-rules

Default binding & strict mode

```
function foo() {
    console.log( this.a );
}

var a = 2;

foo(); // 2
```

```
"use strict";

function foo() {
    console.log( this.a );
}

var a = 2;

foo(); // TypeError: `this` is
`undefined`
```

Implicit binding

```
function foo() {
    console.log( this.a );
}

var obj = {
    a: 2,
    foo: foo
};

obj.foo(); // 2
```

Implicitly lost

```
function foo() {
    console.log( this.a );
}

var obj = {
    a: 2,
    foo: foo
};

var bar = obj.foo; // function
reference/alias!

var a = "oops, global"; // `a` also
property on global object

bar(); // "oops, global"
```

```
function foo() {
    console.log( this.a );
}

function doFoo(fn) {
    // `fn` is just another reference
to `foo`

    fn(); // <-- call-site!
}

var obj = {
    a: 2,
    foo: foo
};

var a = "oops, global"; // `a` also
property on global object

doFoo( obj.foo ); // "oops, global"
```

```
function foo() {
    console.log( this.a );
}

var obj = {
    a: 2,
    foo: foo
};

var a = "oops, global"; // `a` also
property on global object

setTimeout( obj.foo, 100 ); // "oops,
global"
```

Explicit binding

```
function foo() {
   console.log( this.a );
}

var obj = {
   a: 2
};

foo.call( obj ); // 2
```

```
function foo() {
    console.log( this.a );
}

var obj = {
    a: 2
};

var bar = function() {
    foo.call( obj );
};

bar(); // 2
setTimeout( bar, 100 ); // 2

// `bar` hard binds `foo`'s `this` to
`obj`
// so that it cannot be overriden
bar.call( window ); // 2
```

Hard binding

Variación sobre *explicit binding*. El contexto t*his* se especifica internamente de forma explícita en la función, por lo que no podemos modificar el contexto *this* al llamar a esa función.

```
function foo(something) {
    console.log( this.a, something );
    return this.a + something;
}

var obj = {
    a: 2
};

var bar = function() {
    return foo.apply( obj,
arguments );
};

var b = bar( 3 ); // 2 3
console.log( b ); // 5
```

```
function foo(something) {
    console.log( this.a, something );
    return this.a + something;
}

// simple `bind` helper
function bind(fn, obj) {
    return function() {
        return fn.apply( obj,
arguments );
    };
}

var obj = {
    a: 2
};

var bar = bind( foo, obj );

var b = bar( 3 ); // 2 3
console.log( b ); // 5
```

ES5 Function.prototype.bind

```
function foo(something) {
    console.log( this.a, something );
    return this.a + something;
}

var obj = {
    a: 2
};

var bar = foo.bind( obj );

var b = bar( 3 ); // 2 3
console.log( b ); // 5
```

new binding

```
var something = new MyClass(..);
```

¿Qué pasa por detrás?

1 - a brand new object is created (aka, constructed) out of thin air
2 - the newly constructed object is [[Prototype]]-linked
3 - the newly constructed object is set as the this binding for that function call
4 - unless the function returns its own alternate object, the new-invoked function call will automatically return the newly constructed object.

```
function foo(a) {
    this.a = a;
}

var bar = new foo( 2 );
console.log( bar.a ); // 2
```

Ejemplo práctico

```
function MyClass() {
    this.name = “My class”;

    this.init = function() {
        $(‘.myButton’).on(‘click’, this.onClick);
        // VS
        $(‘.myButton’).on(‘click’, this.onClick.bind(this));
    };

    this.onClick = function(event) {
        this.makeRequest(this.onComplete);
        // VS
        this.makeRequest(this.onComplete.bind(this));
    };

    this.makeRequest(onCompleteCallback) {
        // …
        fetch(…).then(function(result){
           onCompleteCallback(result);
        });
    };

    this.onComplete = function(result) {
        // …
        console.log(this.name);
    };
}

var c = new MyClass();
c.init();
```

Novedades es6 - SUGAR!!

1 - class

Básica

Auto binding *this* in class methods

```
class Vehicle {

    constructor(name) {
        this.name = name;
    }

    startEngine() {
        console.log('starting engine…' + this.name);
    }

}

const vehicle = new Vehicle('Super vehicle');
vehicle.startEngine();
```

Herencia

Super calls

```
class Car extends Vehicle{

    constructor(model) {
        super('Car');

        this.model = model; // Not allowed before super() call
    }

    startEngine() {
        super.startEngine();
        console.log('starting engine…' + this.model);
    }

}

const myCar = new Car('VW Golf');
myCar.startEngine();
```

ES6+ class

```
class Vehicle {

    static STATE = {
        IDLE: ‘idle’,
        RUNNING: ‘running’
    };

    engine;
    state;
    // …

    constructor(name) {
        this.name = name;
    }

    startEngine = () => {
        console.log(‘starting engine…’ + this.name);
    }

}

const vehicle = new Vehicle(‘Super vehicle’);
vehicle.startEngine();
```

2 - modules - import / export

Default import/export. We may alias this module while importing

`class Foo {`
`  // …`
`}`
 	
`export default Foo;`
	`import Foo from ‘./Foo’;`
	
	`class Bar extends Foo {`
	`  // …`
	`}`
	
	`export default Bar;`
		`import **Bla** from ‘./Foo’;`
		
		`class Bar extends Bla {`
		`  // …`
		`}`
		
		`export default Bar;`

`function foo() {`
`  // …`
`}`
 	
`export default foo;`
	`import foo from ‘./foo';`
	
	`foo();`
		`import bla from ‘./foo’;`
		
		`bla();`

Explicit import/export

`class Foo {`
`  // …`
`}`
 	
`class Bar {`
`  // …`
`}`
 	
`export {Foo, Bar};`
	`import {Foo} from ‘./Foo’;`
	
	`class Bar extends Foo {`
	`  // …`
	`}`
	
	`export default Bar;`

`function foo() {`
`  // …`
`}`
 	
`export {foo as bar};`	

Exporting & renaming other modules

```
export { foo, bar } from "baz";
export { foo as FOO, bar as BAR } from "baz";
export * from "baz";
```

* A single module may export a default and multiple methods, classes, etc.

3 - Fat arrow - () => ...

ES5

```
class Foo {
    constructor() {
        // …
        $('.myButton').on('click', (function() {
            this.foo();
        }).bind(this));
    }

    foo() {
        //…
    }
}
```

ES6 - Auto binding *this*

```
class Foo {
    constructor() {
        // …
        $('.myButton').on('click', () => {
            this.foo();
        });
    }

    foo() {
        //…
    }
}
```

* ¡Cuidado si no necesitamos auto-binding!

4 - Variable interpolation with strings. String literals

Easy!

```
class Foo {
    constructor(name, surname, address) {
        // …
        this.name = name;
        this.surname = surname;
        this.address = address;
    }

    foo() {
        const message = `hey ${this.name} ${this.surname}, you live in $
{this.address}`;
        console.log(message);
    }
}



const p = `<p>This is a multiline text
              broken in lines</p>`;
```

5 - Spread/Rest

```
// Spread values

function foo(x,y,z) {
    console.log( x, y, z );
}

foo( ...[1,2,3] );
```

```
var a = [2,3,4];
var b = [ 1, ...a, 5 ];

console.log( b );    // [1,2,3,4,5]
```

```
// Gather values

function foo(x, y, ...z) {
    console.log( x, y, z );
}

foo( 1, 2, 3, 4, 5 );
```

6 - Default parameter values

```
// ES5

function foo(x,y) {
    x = (x !== undefined) ? x : 11;
    y = (y !== undefined) ? y : 31;

    console.log( x + y );
}

foo( 0, 42 );           // 42
foo( undefined, 6 );    // 17
```

```
// ES6

function foo(x = 11, y = 31) {
    console.log( x + y );
}

foo( 0, 42 );           // 42
foo( undefined, 6 );    // 17
```

7 - Destructuring or *structured assignment. (left to right assignment)*

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

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

var [ a, b, c ] = foo();
var { x: x, y: y, z: z } = bar();

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

```
// If the property name being matched is the same as the variable you want to
declare, you can actually shorten the syntax

var { x, y, z } = bar();

console.log( x, y, z );             // 4 5 6
```

```
// But…

var { x: bam, y: baz, z: bap } = bar();

console.log( bam, baz, bap );       // 4 5 6
console.log( x, y, z );             // ReferenceError
```

```
var aa = 10, bb = 20;

var o = { x: aa, y: bb };
var     { x: AA, y: BB } = o;

console.log( AA, BB );              // 10 20
```

https://github.com/getify/You-Dont-Know-JS/blob/master/es6%20%26%20beyond/ch2.md#object-property-assignment-pattern

Ejemplo práctico

```
const foo = ({ a: 5, b: 3 } = {}) => {
    console.log(a + b);
};

foo(); // 8
foo({ a: 2, b: 10 }); // 12
```

7 - New features (const, let, static, new objects(Map, WeakMap, Symbol…)

```
// const. Inmutable

const name = 'mikel';

name = 'no way'; // TypeError: Assignment to constant variable.
```

```
// let = var. Mutable

let name = 'mikel';

name = 'yep!'; // ok
```

```
// static. Mutable

class VWGolf extends Car {
    static NAME = 'VW Golf';

    // No constructor. It may be not needed

    static run() {
        console.log('running!');
    }
}

console.log(VWGolf.NAME); // 'VW Golf'
VWGolf.run(); // 'running!'
```

```
// Map. Iterable. Strong references -> depends on Garbage Colector.

const references = new Map();
reference.set(key, value);
reference.get(key);

// WeakMap. Not interable. Weak references -> no garbage colector.
```

```
// Symbol

const sym1 = Symbol();
const sym2 = Symbol("foo");
const sym3 = Symbol("foo");

console.log(sym2 === sym3); // false
```

```
// Singleton based on Symbol

const INSTANCE = Symbol( "instance" );

function HappyFace() {
    if (HappyFace[INSTANCE]) return HappyFace[INSTANCE];

    function smile() { .. }

    return HappyFace[INSTANCE] = {
        smile: smile
    };
}

var me = HappyFace(),
    you = HappyFace();

me === you;          // true
```

```
// Symbol Registry

const EVT_LOGIN = Symbol.for( "event.login" );
console.log( EVT_LOGIN );       // Symbol(event.login)
```

8 - New methods & Functional JS (Array.findIndex()…), Array.from(…)) - (array map, filter, reduce, find...)

```
// Array.of(...) static method

var a = Array( 3 );
a.length;                       // 3
a[0];                           // undefined

var b = Array.of( 3 );
b.length;                       // 1
b[0];                           // 3

var c = Array.of( 1, 2, 3 );
c.length;                       // 3
c;                              // [1,2,3]
```

```
// Array.from(...) static method
// ES5

// array-like object
var arrLike = {
    length: 3,
    0: "foo",
    1: "bar"
};

var arr =
Array.prototype.slice.call( arrLike );
```

```
// Array.from(...) static method
// ES6

var arr = Array.from( arrLike );
```

```
// copyWithin(...) prototype method

[1,2,3,4,5].copyWithin( 3, 0 );          // [1,2,3,1,2]

[1,2,3,4,5].copyWithin( 3, 0, 1 );       // [1,2,3,1,5]

[1,2,3,4,5].copyWithin( 0, -2 );         // [4,5,3,4,5]

[1,2,3,4,5].copyWithin( 0, -2, -1 );     // [4,2,3,4,5]
```

```
// fill(...) prototype method

const hey = 'ho';

const a = Array( 3 ).fill( hey );
a; // ['ho', 'ho', 'ho']



var a = [ null, null, null, null ].fill( 42, 1, 3 );

a;   // [null,42,42,null]
```

```
// find(...) & some(...) prototype
methods
// ES5

var a = [1,2,3,4,5];

(a.indexOf( 3 ) != -1);              //
true
(a.indexOf( 7 ) != -1);              //
false

(a.indexOf( "2" ) != -1);            //
false
```

```
// find(...) & some(...) prototype
methods
// ES6

var a = [1,2,3,4,5];

a.find( function matcher(v){
    return v == "2";
} );                                 /
/ 2

a.find( function matcher(v){
    return v ==
7;                    // undefined
});



var a = [1,2,3,4,5];

a.some( function matcher(v){
    return v == "2";
} );                                 /
/ true

a.some( function matcher(v){
    return v == 7;
} );                                 /
/ false
```

```
// findIndex(...) prototype method

var points = [
    { x: 10, y: 20 },
    { x: 20, y: 30 },
    { x: 30, y: 40 },
    { x: 40, y: 50 },
    { x: 50, y: 60 }
];

points.findIndex( function matcher(point) {
    return (
        point.x % 3 == 0 &&
        point.y % 4 == 0
    );
} );
```

```
// entries() values() & keys() prototype methods

var a = [1,2,3];

[...a.values()];                    // [1,2,3]
[...a.keys()];                      // [0,1,2]
[...a.entries()];                   // [ [0,1], [1,2], [2,3] ]

[...a[Symbol.iterator]()];          // [1,2,3]
```

Functional JS

```
// map() prototype method

const numbers = [1, 4, 9, 16];

const sqrtNumbers = numbers.map((number) => {
    return Math.sqrt(number);
});

console.log(sqrtNumbers); // [1, 2, 3, 4]. numbers hasn't changed.
```

```
// filter() prototype method

const numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

const evenNumbers = numbers.filter((number) => {
    return number % 2;
});

console.log(evenNumbers); // [2, 4, 6, 8, 10];
```

```
// reduce() prototype method

const numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

const summ = numbers.reduce((prevValue, currentValue, index, array) => {
    return prevValue + currentValue;
}, 0); // initial value

console.log(summ); // 55
```