JAVASCRIPT AVANZADO

ADOLFO SANZ DE DIEGO

OCTUBRE 2015

1 ACERCA DE

1.1 AUTOR

- **Adolfo Sanz De Diego**
  - Blog: asanzdiego.blogspot.com.es
  - Correo: asanzdiego@gmail.com
  - GitHub: github.com/asanzdiego
  - Twitter: twitter.com/asanzdiego
  - Linkedin: in/asanzdiego
  - SlideShare: slideshare.net/asanzdiego

1.2 LICENCIA

- **Este obra está bajo una licencia:**
  - Creative Commons Reconocimiento-CompartirIgual 3.0
- **El código fuente de los programas están bajo una licencia:**
  - GPL 3.0

1.3 EJEMPLOS

- Las slides y los códigos de ejemplo los podéis encontrar en:
  - https://github.com/asanzdiego/curso-javascript-avanzado-2015

2 JAVASCRIPT

2.1 HISTORIA

- Lo crea **Brendan Eich en Netscape en 1995** para hacer páginas web dinámicas
- Aparece por primera vez en Netscape Navigator 2.0
- Cada día más usado (clientes web, videojuegos, windows 8, servidores web, bases de datos, etc.)

2.2 EL LENGUAJE

- Orientado a objetos
- Basado en prototipos
- Funcional
- Débilmente tipado
- Dinámico

3 ORIENTACIÓN A OBJETOS

3.1 ¿QUÉ ES UN OBJETO?

- **Colección de propiedades** (pares nombre-valor).
- Todo son objetos (las funciones también) excepto los primitivos: **strings, números, booleans, null o undefined**
- Para saber si es un objeto o un primitivo hacer **typeof variable**

3.2 PROPIEDADES (I)

- Podemos acceder directamente o como si fuese un contenedor:

```
objeto.nombre === objeto[nombre] // true
```

3.3 PROPIEDADES (II)

- Podemos crearlas y destruirlas en tiempo de ejecución

```
var objeto = {};
objeto.nuevaPropiedad = 1; // añadir
delete objeto.nuevaPropiedad; // eliminar
```

3.4 OBJETO INICIADOR

- Podemos crear un objeto así:

```
var objeto = {
  nombre: "Adolfo",
  twitter: "@asanzdiego"
};
```

3.5 FUNCIÓN CONSTRUCTORA

- O con una función constructora y un new.

```
function Persona(nombre, twitter) {
  this.nombre = nombre;
  this.twitter = twitter;
};
var objeto = new Persona("Adolfo", "@asanzdiego");
```

3.6 PROTOTIPOS (I)

- Las funciones son objetos y tienen una propiedad llamada **prototype**.
- Cuando creamos un objeto con new, la referencia a esa propiedad **prototype** es almacenada en una propiedad interna.
- El prototipo se utiliza para compartir propiedades.

3.7 PROTOTIPOS (II)

- Podemos acceder al objeto prototipo de un objeto:

```
// Falla en Opera o IE <= 8
Object.getPrototypeOf(objeto);

// No es estandar y falla en IE
objeto.__proto__;
```

3.8 EFICIENCIA (I)

- Si queremos que nuestro código se ejecute una sola vez y que prepare en memoria todo lo necesario para generar objetos, la mejor opción es usar una **función constructora solo con el estado de una nueva instancia, y el resto (los métodos) añadirlos al prototipo**.

3.9 EFICIENCIA (II)

- Ejemplo:

```
function ConstructorA(p1) {
  this.p1 = p1;
}

// los métodos los ponenmos en el prototipo
ConstructorA.prototype.metodo1 = function() {
  console.log(this.p1);
};
```

3.10 HERENCIA

- Ejemplo:

```
function ConstructorA(p1) {
  this.p1 = p1;
}

function ConstructorB(p1, p2) {
  // llamamos al super para que no se pierda p1.
  ConstructorA.call(this, p1);
  this.p2 = p2;
}

// Hacemos que B herede de A
// Prototipo de Función Constructora B apunta al
// Prototipo de Función Constructora A
ConstructorB.prototype = Object.create(ConstructorA.prototype);
```

3.11 CADENA DE PROTOTIPOS

- Cuando se invoca una llamada a una propiedad, **JavaScript primero busca en el propio objeto, y si no lo encuentra busca en su prototipo**, y sino en el prototipo del prototipo, así hasta el prototipo de Object que es null.

3.12 CADENA DE PROTOTIPOS DE LA INSTANCIA

- En el ejemplo anterior:

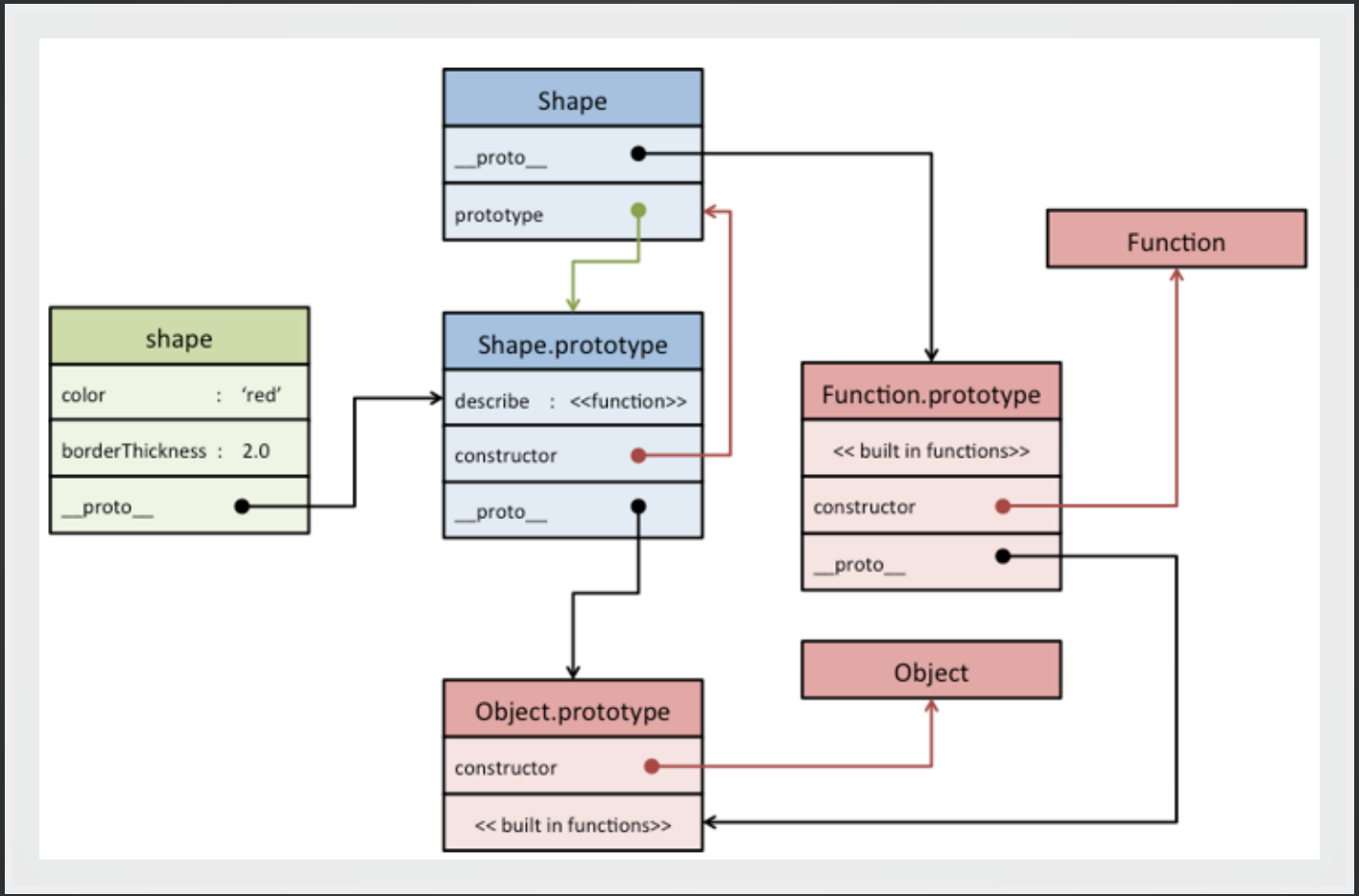
```
instanciaB.__proto__ == ConstructorB.prototype // true
instanciaB.__proto__.__proto__ == ConstructorA.prototype // true
instanciaB.__proto__.__proto__.__proto__ == Object.prototype // true
instanciaB.__proto__.__proto__.__proto__.__proto__ == null // true
```

3.13 CADENA DE PROTOTIPOS DE LA FUNCIÓN CONSTRUCTORA

- En el ejemplo anterior:

```
expect(ConstructorB.__proto__).toEqual(Function.prototype);
expect(ConstructorB.__proto__.__proto__).toEqual(Object.prototype);
expect(ConstructorB.__proto__.__proto__.__proto__).toEqual(null);
```

3.14 ESQUEMA PROTOTIPOS



Esquema prototipos

3.15 OPERADOR INSTANCEOF

- La expresión **instanciaB instanceof ConstructorA** devolverá true, si el prototipo de la Función ConstructorA, se encuentra en la cadena de prototipos de la instanciaB.
- En el ejemplo anterior:

```
instanciaB instanceof ConstructorB; // true
instanciaB instanceof ConstructorA; // true
instanciaB instanceof Object; // true
```

3.16 EXTENSIÓN

- Con los prototipos podemos extender la funcionalidad del propio lenguaje.
- Ejemplo:

```
String.prototype.hola = function() {
  return "Hola "+this;
}

"Adolfo".hola(); // "Hola Adolfo"
```

3.17 PROPIEDADES Y MÉTODOS ESTÁTICOS (I)

- Lo que se define dentro de la función constructora va a ser propio de la instancia.
- Pero como hemos dicho, en JavaScript, una función es un objeto, al que podemos añadir tanto atributos como funciones.
- **Añadiendo atributos y funciones a la función constructora obtenemos propiedades y métodos estáticos.**

3.18 PROPIEDADES Y MÉTODOS ESTÁTICOS (II)

- Ejemplo:

```
function ConstructorA() {

  ConstructorA.propiedadEstatica = "propiedad estática";
}

ConstructorA.metodoEstatico = function() {
  console.log("método estático");
}
```

3.19 PROPIEDADES Y MÉTODOS PRIVADOS (I)

- La visibilidad de objetos depende del contexto.
- Los contextos en JavaScript son bloques de código entre dos {} y en general, desde uno de ellos, solo tienes acceso a lo que en él se defina y a lo que se defina en otros contextos que contengan al tuyo.

3.20 PROPIEDADES Y MÉTODOS PRIVADOS (II)

- Ejemplo:

```
function ConstructorA(privada, publica) {
  var propiedadPrivada = privada;
  this.propiedadPublica = publica;
  var metodoPrivado = function() {
    console.log("-->propiedadPrivada", propiedadPrivada);
  }
  this.metodoPublico = function() {
    console.log("-->propiedadPublica", this.propiedadPublica);
    metodoPrivado();
  }
}
```

3.21 POLIMORFISMO

- Poder llamar a métodos sintácticamente iguales de objetos de tipos diferentes.
- Esto se consigue mediante herencia.

4 TÉCNICAS AVANZADAS

4.1 FUNCIONES

- Son objetos con sus propiedades.
- Se pueden pasar como parámetros a otras funciones.
- Pueden guardarse en variables.
- Son mensajes cuyo receptor es **this**.

4.2 THIS

- Ejemplo:

```
var nombre = "Laura";

var alba = {
  nombre: "Alba",
  saludo: function() {
    return "Hola "+this.nombre;
  }
}

alba.saludo(); // Hola Alba

var fn = alba.saludo;

fn(); // Hola Laura
```

4.3 CALL Y APPLY

- Dos funciones permiten manipular el this: **call y apply** que en lo único que se diferencian es en la llamada.

```
fn.call(thisArg [, arg1 [, arg2 [...]]])
```

```
fn.apply(thisArg [, arglist])
```

4.4 NÚMERO VARIABLE DE ARGUMENTOS

- Las funciones en JavaScript aunque tengan especificado un número de argumentos de entrada, **pueden recibir más o menos argumentos** y es válido.

4.5 ARGUMENTS

- Es un objeto que **contiene los parámetros** de la función.

```
function echoArgs() {
  console.log(arguments[0]); // Adolfo
  console.log(arguments[1]); // Sanz
}
echoArgs("Adolfo", "Sanz");
```

4.6 DECLARACIÓN DE FUNCIONES

- Estas 2 declaraciones son **equivalentes**:

```
function holaMundo1() {
  console.log("Hola Mundo 1");
}
holaMundo1();

var holaMundo2 = function() {
  console.log("Hola Mundo 2");
}
holaMundo2();
```

4.7 TRANSFIRIENDO FUNCIONES A OTRAS FUNCIONES

- Hemos dicho que las funciones son objetos, así que **se pueden pasar como parámetros**.

```
function saluda() {
  console.log("Hola")
}
function ejecuta(func) {
  func()
}
ejecuta(saluda);
```

4.8 FUNCIONES ANÓNIMAS (I)

- Hemos dicho que las funciones se pueden declarar.
- Pero también **podemos no declararlas y dejarlas como anónimas**.

4.9 FUNCIONES ANÓNIMAS (II)

- Una función anónima así declarada **no se podría ejecutar**.

```
function(nombre) {
  console.log("Hola "+nombre);
}
```

4.10 FUNCIONES ANÓNIMAS (III)

- Pero **una función puede devolver una función anónima**.

```
function saludador(nombre) {
  return function() {
    console.log("Hola "+nombre);
  }
}

var saluda = saludador("mundo");
saluda(); // Hola mundo
```

4.11 FUNCIONES AUTOEJECUTABLES

- Podemos autoejecutar funciones anónimas.

```
(function(nombre) {
  console.log("Hola "+nombre);
})("mundo")
```

4.12 CLOUSURES (I)

- Un closure **combina una función y el entorno en que se creó**.

```
function creaSumador(x) {
  return function(y) {
    return x + y;
  };
}

var suma5 = creaSumador(5);
var suma10 = creaSumador(10);

console.log(suma5(2));  // muestra 7
console.log(suma10(2)); // muestra 12
```

4.13 CLOUSURES (II)

- En una closures la función interna almacena una **referencia al último valor** de la variable establecido cuando la función externa termina de ejecutarse.

4.14 EL PATRÓN MODULO

- Se trata de una función que actúa como contenedor para un contexto de ejecución.

```
miModulo = (function() {

  var propiedadPrivada;

  function metodoPrivado() { };

  // API publica
  return {
    metodoPublico1: function () {
    },

    metodoPublico2: function () {
    }
  }
}());
```

4.15 EFICIENCIA (I)

- Si se ejecuta desde el navegador, **se suele pasar como parámetro el objeto window para mejorar el rendimiento**. Así cada vez que lo necesitemos el intérprete lo utilizará directamete en lugar de buscarlo remontando niveles.
- Y también **se suele pasar el parámetro undefined, para evitar los errores que pueden darse si la palabra reservada ha sido reescrita** en alguna parte del código y su valor no corresponda con el esperado.

4.16 EFICIENCIA (II)

```
miModulo = (function(window, undefined) {

  // El código va aquí

})( window );
```

4.17 EL PATRÓN MODULO REVELADO (I)

- El problema del patrón Modulo es pasar un método de privado a público o viceversa.
- Por ese motivo lo que que se suele hacer es definir todo en el cuerpo, y luego **referenciar solo los públicos en el bloque return**.

4.18 EL PATRÓN MODULO REVELADO (II)

```
miModulo = (function() {

  function metodoA() { };

  function metodoB() { };

  function metodoC() { };

  // API publica
  return {
    metodoPublico1: metodoA,
    metodoPublico2: metodoB
  }
}());
```

4.19 ESPACIOS DE NOMBRES (I)

- Para simular espacios de nombres, en JavaScript se anidan objetos.

```
miBiblioteca = miBiblioteca || {};

miBiblioteca.seccion1 = miBiblioteca.seccion1 || {};

miBiblioteca.seccion1 = {
  priopiedad: p1,
  metodo: function() { },
};

miBiblioteca.seccion2 = miBiblioteca.seccion2 || {};

miBiblioteca.seccion2 = {
  priopiedad: p2,
  metodo: function() { },
};
```

4.20 ESPACIOS DE NOMBRES (II)

- Se puede combinar lo anterior con módulos autoejecutables:

```
miBiblioteca = miBiblioteca || {};

(function(namespace) {

  var propiedadPrivada = p1;

  namespace.propiedadPublica = p2;

  var metodoPrivado = function() { };

  namespace.metodoPublico = function() { };

}(miBiblioteca));
```

5 DOCUMENT OBJECT MODEL

5.1 ¿QUÉ ES DOM?

- Acrónimo de **Document Object Model**
- Es un conjunto de utilidades específicamente diseñadas para **manipular documentos XML, y por extensión documentos XHTML y HTML**.
- DOM transforma internamente el archivo XML en una estructura más fácil de manejar formada por una jerarquía de nodos.

5.2 TIPOS DE NODOS

- Los más importantes son:
  - **Document**: representa el nodo raíz.
  - **Element**: representa el contenido definido por un par de etiquetas de apertura y cierre y puede tener tanto nodos hijos como atributos.
  - **Attr**: representa el atrributo de un elemento.
  - **Text**: almacena el contenido del texto que se encuentra entre una etiqueta de apertura y una de cierre.

5.3 RECORRER EL DOM

- JavaScript proporciona **funciones** para recorrer los nodos:

```
getElementById(id)
getElementsByName(name)
getElementsByTagName(tagname)
getElementsByClassName(className)
getAttribute(attributeName)
querySelector(selector)
querySelectorAll(selector)
```

5.4 MANIPULAR EL DOM

- JavaScript proporciona **funciones** para la manipulación de nodos:

```
createElement(tagName)
createTextNode(text)
createAttribute(attributeName)
appendChild(node)
insertBefore(newElement, targetElement)
removeAttribute(attributename)
removeChild(childreference)
replaceChild(newChild, oldChild)
```

5.5 PROPIEDADES NODOS (I)

- Los nodos tienen algunas **propiedades** muy útiles:

```
attributes[]
className
id
innerHTML
nodeName
nodeValue
style
tabIndex
tagName
title
```

5.6 PROPIEDADES NODOS (II)

- Los nodos tienen algunas **propiedades** muy útiles:

```
childNodes[]
firstChild
lastChild
previousSibling
nextSibling
ownerDocument
parentNode
```

6 LIBRERÍAS Y FRAMEWORKS

6.1 JQUERY

- jQuery: libreria que reduce código ("write less, do more").

```
// Vanilla JavaScript
var elem = document.getElementById("miElemento");

//jQuery
var elem = $("#miElemento");
```

6.2 JQUERY UI & MOBILE

- jQuery UI: diseño interfaces gráficas.
- jQuery Mobile: versión adaptada para móviles (eventos y tamaño).

6.3 FRAMEWORKS CSS

- Bootstrap y Foundation.
- Fácil maquetación, sistema rejilla, clases CSS, temas, etc.

6.4 MVC EN EL FRONT

- BackboneJS: ligero y flexible.
- EmberJS: "Convention over Configuration", muy popular entre desarrolladores Ruby on Rails.
- AngularJS extiende etiquetas HML (g-app, ng-controller, ng-model, ng-view), detrás está Google, tiene gran popularidad, abrupta curva de aprendizaje.

6.5 NODEJS

- NodeJS permite ejecutar JS fuera del navegador.
- Viene con su propio gestor de paquetes: npm

6.6 AUTOMATIZACIÓN DE TAREAS

- GruntJS: más popularidad y más plugins.
- GulpJS: más rápido tanto al escribir ("Code over Configure") como al ejecutar (streams).

6.7 GESTIÓN DE DEPENDENCIAS

- Bower: para el lado cliente. Puede trabajar con repositorios Git.
- Browserify: permite escribir módulos como en NodeJS y compilarlos para que se puedan usar en el navegador.
- RequeriJS: las dependencias se cargan de forma asíncrona y solo cuando se necesitam.
- WebPack: es un empaquetador de módulos

6.8 APLICACIONES DE ESCRITORIO MULTIPLATAFORMA

- AppJS, y su fork DeskShell: los más antiguos, un poco abandonados.
- NW.js: opción más popular y madura hoy en día.
- Electron: creada para el editor Atom de GitHub: está creciendo en popularidad.

6.9 APLICACIONES MÓVILES HÍBRIDAS

- cordova: una de los primeros. Hoy en día, otros frameworks se basan en él.
- ionic: utiliza AngularJS, tiene una CLI, muy popular.
- React Native: recién liberado por facebook.

6.10 WEBCOMPONENTS

- WebComponents es una especificación de la W3C para permitir crear componentes y reutilizarlos.
- polymer: proyecto de Google para poder empezar a usar los WebComponents en todos los navegadores.

6.11 OTROS

- React: librería hecho por Facebook para crear interfaces que se renderizan muy rápido, ya sea en cliente o servidor.
- Flux: framework hecho por Facebook que utiliza React.
- Meteor: es una plataforma que permite desarrollar aplicaciones real-time con JS Isomófico (se ejecuta en front y back)

7 EVENTOS

7.1 PRINCIPALES EVENTOS (I)

Evento	Descripción
onblur	Un elemento pierde el foco
onchange	Un elemento ha sido modificado
onclick	Pulsar y soltar el ratón
ondblclick	Pulsar dos veces seguidas con el ratón

7.2 PRINCIPALES EVENTOS (II)

Evento	Descripción
onfocus	Un elemento obtiene el foco
onkeydown	Pulsar una tecla y no soltarla
onkeypress	Pulsar una tecla
onkeyup	Soltar una tecla pulsada
onload	Página cargada completamente

7.3 PRINCIPALES EVENTOS (III)

Evento	Descripción
onmousedown	Pulsar un botón del ratón y no soltarlo
onmousemove	Mover el ratón
onmouseout	El ratón "sale" del elemento
onmouseover	El ratón "entra" en el elemento
onmouseup	Soltar el botón del ratón

7.4 PRINCIPALES EVENTOS (IV)

Evento	Descripción
onreset	Inicializar el formulario
onresize	Modificar el tamaño de la ventana
onselect	Seleccionar un texto
onsubmit	Enviar el formulario
onunload	Se abandona la página

7.5 SUSCRIPCIÓN

- Para añadir o eliminar un **Listener** de un evento a un elemento:

```
var windowOnLoad = function(e) {
  console.log('window:load', e);
};

window.addEventListener('load', windowOnLoad);

window.removeEventListener('load', windowOnLoad);
```

7.6 EVENTOS PERSONALIZADOS (I)

- Podemos crear **eventos personalizados**:

```
var event = new Event('build');

elem.addEventListener('build', function (e) { ... }, false);
```

7.7 EVENTOS PERSONALIZADOS (II)

- Podemos crear **eventos personalizados con datos**:

```
var event = new CustomEvent('build', { 'detail': detail });

elem.addEventListener('build', function (e)  {
  log('The time is: ' + e.detail);
}, false);
```

7.8 DISPARAR UN EVENTO

- Podemos **disparar** eventos:

```
function simulateClick() {
  var event = new MouseEvent('click');
  var element = document.getElementById('id');
  element.dispatchEvent(event);
}
```

7.9 PROPAGACIÓN

7.10 EL PATRÓN PUBSUB (I)

```
var EventBus = {
  topics: {},

  subscribe: function(topic, listener) {
    if (!this.topics[topic]) this.topics[topic] = [];
    this.topics[topic].push(listener);
  },

  publish: function(topic, data) {
    if (!this.topics[topic] || this.topics[topic].length < 1) return;
    this.topics[topic].forEach(function(listener) {
      listener(data || {});
    });
  }
};
```

7.11 EL PATRÓN PUBSUB (II)

```
EventBus.subscribe('foo', alert);
EventBus.publish('foo', 'Hello World!');
```

7.12 EL PATRÓN PUBSUB (III)

```
var Mailer = function() {
  EventBus.subscribe('order/new', this.sendPurchaseEmail);
};

Mailer.prototype = {
  sendPurchaseEmail: function(userEmail) {
    console.log("Sent email to " + userEmail);
  }
};

var Order = function(params) {
  this.params = params;
};

Order.prototype = {
  saveOrder: function() {
    EventBus.publish('order/new', this.params.userEmail);
  }
```

7.13 EL PATRÓN PUBSUB (IV)

```
var mailer = new Mailer();
var order = new Order({userEmail: 'john@gmail.com'});
order.saveOrder();
"Sent email to john@gmail.com"
```

7.14 WEBSOCKETS

8 AJAX

8.1 JSON, JSONP, CORS

8.2 USO DE APIS

9 INYECCIÓN DE DEPENDENCIAS

9.1 AMD (REQUIREJS)

9.2 COMMONJS (BROWSERIFY)

10 ECMASCRIPT6

10.1 PRINCIPALES NOVEDADES

10.2 COMO USARLO HOY

11 ENLACES

11.1 GENERAL (ES)

- http://developer.mozilla.org/es/docs/Web/JavaScript/G
- http://cevichejs.com/
- http://www.arkaitzgarro.com/javascript/
- http://www.etnassoft.com/category/javascript/

11.2 GENERAL (EN)

- http://www.javascriptkit.com/
- http://javascript.info/
- http://www.howtocreate.co.uk/tutorials/javascript/

11.3 ORIENTACIÓN OBJETOS (ES) (I)

- http://www.programania.net/diseno-de-software/entendiendo-los-prototipos-en-javascript/
- http://www.programania.net/diseno-de-software/creacion-de-objetos-eficiente-en-javascript/
- http://blog.amatiasq.com/2012/01/javascript-conceptos-basicos-herencia-por-prototipos/

11.4 ORIENTACIÓN OBJETOS (ES) (II)

- http://albertovilches.com/profundizando-en-javascript-parte-1-funciones-para-todo
- http://albertovilches.com/profundizando-en-javascript-parte-2-objetos-prototipos-herencia-y-namespaces
- http://www.arkaitzgarro.com/javascript/capitulo-9.html
- http://www.etnassoft.com/2011/04/15/concepto-de-herencia-prototipica-en-javascript/

11.5 ORIENTACIÓN OBJETOS (EN)

- http://www.codeproject.com/Articles/687093/Understa
  JavaScript-Object-Creation-Patterns
- http://javascript.info/tutorial/object-oriented-programm
- http://www.howtocreate.co.uk/tutorials/javascript/obje

11.6 TÉCNICAS AVANZADAS (ES) (I)

- http://www.etnassoft.com/2011/03/14/funciones-autoejecutables-en-javascript/
- http://www.etnassoft.com/2012/01/12/el-valor-de-this-javascript-como-manejarlo-correctamente/
- https://developer.mozilla.org/es/docs/Web/JavaScript/
- http://www.variablenotfound.com/2012/10/closures-en-javascript-entiendelos-de.html

11.7 TÉCNICAS AVANZADAS (ES) (II)

- http://www.webanalyst.es/espacios-de-nombres-en-javascript/
- http://www.etnassoft.com/2011/04/11/el-patron-de-modulo-en-javascript-en-profundidad/
- http://www.etnassoft.com/2011/04/18/ampliando-patron-modulo-javascript-submodulos/
- http://notasjs.blogspot.com.es/2012/04/el-patron-modulo-en-javascript.html

11.8 DOM (ES)

- http://cevichejs.com/3-dom-cssom#dom
- http://www.arkaitzgarro.com/javascript/capitulo-13.html

11.9 DOM (EN)

- http://www.javascriptkit.com/domref/
- http://javascript.info/tutorial/dom

11.10 FRAMEWORKS (ES)

- https://carlosazaustre.es/blog/frameworks-de-javascri
- https://docs.google.com/drawings/d/1bhe9-kxhhGvWU0LsB7LlJfMurP3DGCIuUOmqEOklzaQ/edit
- http://www.lostiemposcambian.com/blog/javascript/ba
  vs-angular-vs-ember/
- http://blog.koalite.com/2015/06/grunt-o-gulp-que-uso/

11.11 FRAMEWORKS (EN)

- http://www.slideshare.net/deepusnath/javascript-fram
- http://stackshare.io/stackups/backbone-vs-emberjs-vs
- http://www.hongkiat.com/blog/gulp-vs-grunt/
- https://mattdesl.svbtle.com/browserify-vs-webpack
- http://hackhat.com/p/110/module-loader-webpack-vs-
- http://devzum.com/2014/02/10-best-node-js-mvc-fram
- http://www.tivix.com/blog/nwjs-and-electronjs-web-tec
- http://stackshare.io/stackups/phonegap-vs-ionic-vs-re
- https://developer.salesforce.com/page/Native,_HTML5

11.12 EVENTOS (ES)

- http://cevichejs.com/3-dom-cssom#eventos
- http://www.arkaitzgarro.com/javascript/capitulo-15.html
- http://codexexempla.org/curso/curso_4_3_e.php

11.13 EVENTOS (EN)

- https://developer.mozilla.org/en-US/docs/Web/API/EventTarget
- https://developer.mozilla.org/en-US/docs/Web/API/Event
- http://dev.housetrip.com/2014/09/15/decoupling-javascript-apps-using-pub-sub-pattern/
- https://stackoverflow.com/questions/5963669/whats-the-difference-between-event-stoppropagation-and-event-preventdefault

11.14 ES6 (ES)

- http://rlbisbe.net/2014/08/26/articulo-invitado-ecmascript-6-y-la-nueva-era-de-javascript-por-ckgrafico/
- http://carlosazaustre.es/blog/ecmascript-6-el-nuevo-estandar-de-javascript/
- http://asanzdiego.blogspot.com.es/2015/06/principios-solid-con-ecmascript-6-el-nuevo-estandar-de-javascript.html

11.15 ES6 (EN)

- http://es6-features.org/
- http://kangax.github.io/compat-table/es5/