Tema 3. Tecnologías del cliente. JavaScript

Programación web

Boni García

Curso 2016/2017

Índice

Índice

Introducción

- Las páginas web pueden incorporar interactividad con el lenguaje **JavaScript**
- Con JavaScript se puede modificar la página y ejecutar código cuando se interactúa con ella a través del modelo de objetos del **DOM** (*Document Object Model*)
- También se pueden hacer peticiones al servidor web en segundo plano y actualizar el contenido de la web con los resultados usando **AJAX** (*Asynchronous JavaScript And XML*)

Introducción

- JavaScript es un lenguaje de programación basado en el estándar **ECMAScript** de **ECMA** (organización diferente al **W3C**)
- Hay ligeras **diferencias** en la implementación de JavaScript de los navegadores, aunque actualmente todos son bastante compatibles entre sí (en el pasado no fue así)
- Aunque algunos elementos de la sintaxis recuerden a Java, no tiene nada que ver con Java
- El nombre **JavaScript** se eligió al publicar el lenguaje en una época en la que Java estaba en auge y fue principalmente por marketing (inicialmente se llamó LiveScript)

http://www.ecma-international.org/

Características

- **Imperativo y estructurado** (como por ejemplo Java)
  - Se declaran **variables**
  - Se ejecutan las sentencias **en orden**
  - Dispone de **sentencias de control** de flujo de ejecución
- **Scripting:** Tradicionalmente JavaScript ha sido interpretado en el navegador
  - Esto no es cierto para todos los navegadores
  - Por ejemplo, el motor de JavaScript de Google Chrome (llamado V8) compila el código JavaScript a código máquina siguiendo un enfoque JIT (*just-in-time*), esto es, compilación en tiempo de ejecución

Características

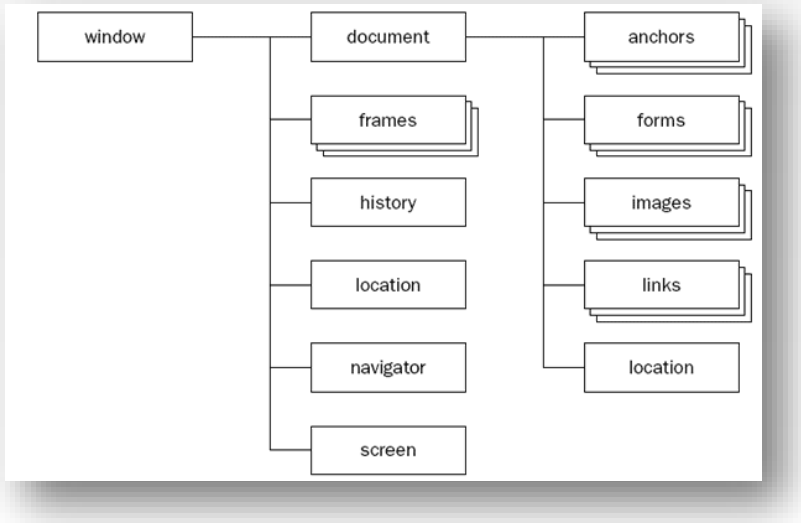
- **Tipado dinámico** (Java es estático)
  - Al declarar una variable no es necesario definir su tipo
  - A lo largo de la ejecución del programa una misma variable puede tener valores de diferentes tipos
- **Orientado a objetos**
  - Todos los **valores son objetos** (no como en Java, que existen tipos primitivos)
  - Existe **recolector de basura** para objetos que no se utilizan (como en Java)
  - La orientación a objetos está **basada en prototipos** (en Java está basada en clases)
  - Se pueden **crear objetos**, añadir atributos y métodos en tiempo de ejecución

Características

- **Funcional**
  - Las funciones en JavaScript son elementos de primera clase, o sea, permite declarar **funciones independientes** (no como en Java)
  - Esas funciones se pueden declarar en cualquier sitio, asignarse a variables, pasarse como parámetro
  - En JavaScript se puede implementar código inspirado en el **paradigma funcional** (aunque no sea puro)
    - Ejemplo de lenguajes de programación funcional: Erlang, Scala, Haskell, entre otros

Características

- Permite la manipulación del **DOM** y del **BOM**
- **DOM (*Document Object Model*)**
  - Librería (API) para manipular el documento HTML cargado en el navegador
  - Permite la gestión de eventos, insertar y eliminar elementos, etc.
- **BOM (*Browser Object Model*)**
  - Acceso a otros elementos del browser: historial, peticiones de red AJAX, etc…
  - El BOM incluye al DOM como uno de sus elementos

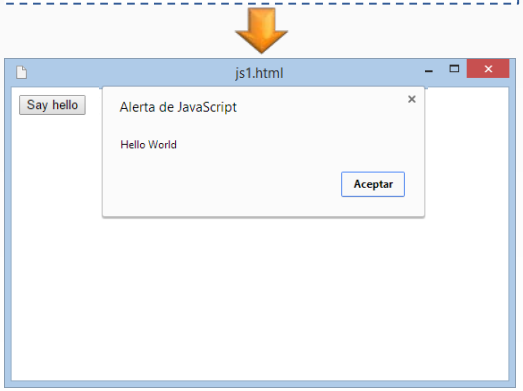


Formas de insertar JavaScript en HTML

1. Incluido en un elemento

- Mediante eventos HTML (*onclick*, …):

```
<!DOCTYPE html>
<html>
<body>
<button
onclick="alert('Hello
World');">Say
hello</button>
</body>
</html>
```



2. Incluido en la página

- En la sección `head` o `body` con la etiqueta `script`:

```
<!DOCTYPE html>
<html>
<head>
<script>
function hello() {
  alert('Hello World');
}
</script>
</head>
<body>
<button
onclick="hello();">Say
hello</button>
</body>
</html>
```

3. En fichero independiente

- Enlazado con etiqueta `script` en sección `head` o `body`:

```
<!DOCTYPE html>
<html>
<head>
<script
src="script.js"></script>
</head>
<body>
<button
onclick="hello();">Say
hello</button>
</body>
</html>
```

script.js

```
function hello() {
   alert('Hello World');
}
```

Sintaxis básica

- Variables:
  - Es un lenguaje con tipado **dinámico** (las variables se declaran con `var` pero no se indica el tipo):

```
var count = 10;
var found = false;
var name = 'John';
```

  - La variable tiene como ámbito la función, no por bloque. Al finalizar el bloque, la variable sigue presente (en Java es por bloque)
  - Si no se inicializan, las variables tienen el valor `undefined` (en lugar de `null` como en Java)
  - Valores especiales:
    - Las variables pueden además tomar el valor `null` (variable definida pero con estado nulo)
    - Valor `NaN` (“*not a number*”, valor especial que dictamina que una variable no es numérica)

Sintaxis básica

- Tipos de datos:
  - `Number`
    - Números enteros y reales de cualquier precisión
  - `String`
    - Cadenas de caracteres
    - Comillas simples o dobles
  - `Boolean`
    - `true` o `false` (como en Java)

Sintaxis básica

- Escritura de datos en JavaScript:
  - Escribe en el documento HTML:

```
document.write('text');
```

  - Escribe en la consola JavaScript:

```
console.log('text');
console.info('text');
console.warn('text');
console.error('text');
```

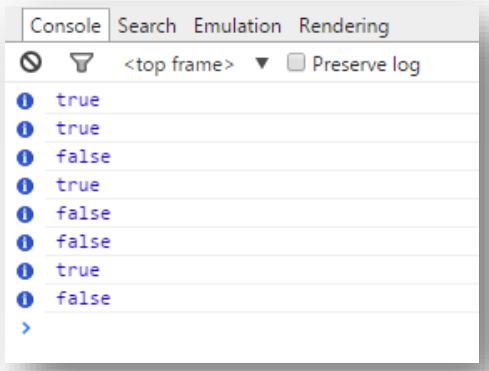
  - Mostrar una alerta JavaScript:

```
alert('text');
```

Sintaxis básica

- Operadores:
  - Similares a Java:
    - Aritméticos: `+ - * / %`
    - Comparación números: `< > <= >=`
    - Lógicos: `&& || !`
    - Comparativo: `?:` (operador Elvis)
    - Modificación: `++ --`
    - Asignación: `= += -= *= /= %=`
  - Diferentes a Java:
    - Igual (valor y tipo): `===`
    - Distinto (valor y tipo): `!==`

```
var x = 5;
console.info(x == "5");
console.info(x == 5);
console.info(x === "5");
console.info(x === 5);
console.info(x != "5");
console.info(x != 5);
console.info(x !== "5");
console.info(x !== 5);
```



Sintaxis básica

- Comentarios:
  - Una línea y multilínea

```
// Comment inline

/*
 * Comment multi-line
 */
```

- Delimitadores:
  - De bloque `{}`
  - De sentencia `;`

Arrays

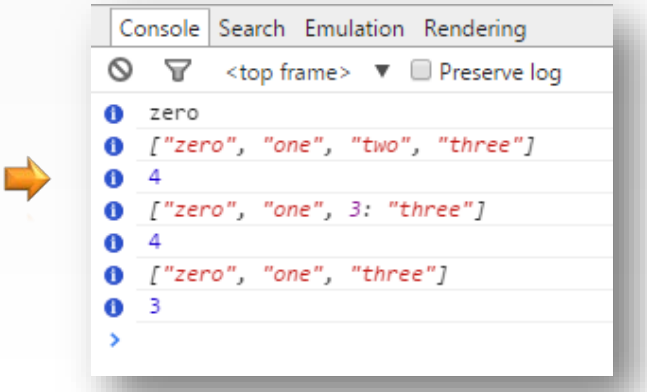
- Características iguales a Java:
  - El acceso para lectura o escritura es con `[]`
  - Tienen la propiedad `length`
  - Empiezan por 0
  - La asignación de variables no copia el array, las variables apuntan al mismo objeto
- Características diferentes a Java:
  - Los literales son con `[]` en vez de `{}`
  - No se pone `new` en el literal
  - Pueden mezclar valores de varios tipos (como un array de `Object` en Java)
  - El acceso a un elemento fuera de los límites es `undefined` (en Java sería un `ArrayIndexOutOfBoundsException`)

```
var empty = [];
var numbers = ['zero','one','two','three'];
```

Arrays

- Gestión como listas (como `java.util.List`)
  - Se pueden establecer elementos en posiciones no existentes y el array crece dinámicamente.
  - El método `push` añade un elemento al final (como el método `add` en Java)
  - La propiedad `length` se puede cambiar para reducir el tamaño del array
  - El operador `delete` borra un elemento (pero deja el hueco)
  - Para borrar y no dejar el hueco se usa el método `splice` indicando el índice desde el que hay que borrar y el número de elementos.

```
var numbers = [ 'zero', 'one', 'two', 'three' ];
console.info(numbers[0]);
console.info(numbers);
console.info(numbers.length);
delete numbers[2];
console.info(numbers);
console.info(numbers.length);
numbers.splice(2, 1);
console.info(numbers);
console.info(numbers.length);
```



Sentencias de control

- Bloques de sentencias
  - Con llaves `{}` (como en Java)
  - Las variables no desaparecen al terminar en bloque (como ocurre en Java)
- Sentencia condicional: `if`
  - Sintaxis como en Java
  - La expresión no tiene que devolver un valor `boolean`
  - Se considera falso: `false`, `null`, `undefined`, `""` (cadena vacía), `0`, `NaN`

Sentencias de control

- Sentencias `switch`, `while` y `do`
  - Sintaxis y semántica como en Java
- Sentencia `for`
  - `for(init; expr; inc)` como en Java, pero la variable se declara fuera del `for`
  - No existe `continue`, pero sí `break`
- Sentencia `return`
  - Rompe el flujo y devuelve como en Java

Funciones

- Se pueden declarar con nombre:

```
function func(param) {
   console.log(param);
}

func(4); // Print 4 in the console
```

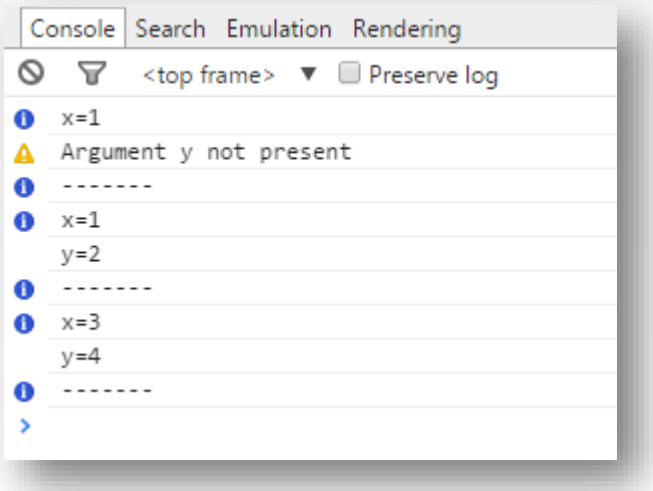
- Se pueden declarar sin nombre (anónimas) y asignarse a una variable o usarse como parámetro:

```
var func2 = function(param) {
   console.log(param);
}

func2(5); // Print 5 in the console
```

Funciones

- El número de parámetros al invocar una JavaScript no tiene por qué coincidir con el número de parámetros que acepta la función
  - Si se pasan menos parámetros, los demás se pasan como `undefined`
  - Si se pasan más, se ignoran

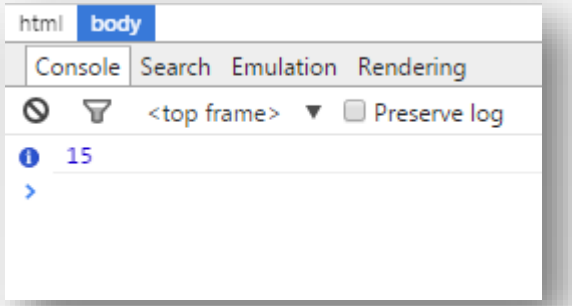
```
function myFunction(x, y) {
   console.info("x=" + x);
   if (!y) {
      console.warn("Argument y not present");
   } else {
      console.log("y=" + y);
   }
   console.info("-------");
}

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



Funciones

- Todas las funciones cuentan con el objeto `arguments`, que contiene el array de los argumentos con la que fue invocada una función:

```
function sumAll() {
   var i, sum = 0;
   for (i = 0; i < arguments.length; i++) {
      sum += arguments[i];
   }
   return sum;
}

console.info(sumAll(1, 2, 3, 4, 5));
```



- Se pueden declarar funciones en el cuerpo de otras funciones:

```
var myFunction1 = function(node) {
   node.onclick = function(e) {
      alert("Alert");
   }
}
```

Excepciones

- Existe un bloque con `try catch finally` (como en Java)
- El operador `throw` eleva la excepción
- A diferencia de Java, se puede lanzar cualquier objeto como excepción

```
try {
   var error = "...";
   if (error) {
      throw "An error";
   }
   return true;
} catch (e) {
   console.error(e);
   return false;
} finally {
   //do cleanup, etc here
}
```

Document Object Model (DOM)

- El *Document Object Model* (DOM) es la API para manipular documentos HTML desde JavaScript
- Cuando se carga una página HTML en un navegador, se convierte en un objeto tipo `document`
- Todos los elementos HTML están representado por el objeto `element`
- Acceso a un elemento del documento usando el atributo `id` (el atributo `id` debería ser único en la página. Si no lo fuese, `getElementById` devuelve el primer elemento con dicho `id`):

```
var obj1 = document.getElementById("objId");
```

- Acceso a un elemento del documento por clase CSS (si hay varios elementos, devuelve el primero que encuentra):

```
var obj2 = document.querySelector(".mycssclass");
```

Document Object Model (DOM)

- Acceso a un conjunto de elementos del documento:
  - Por el atributo `name`
  - Por el atributo `class`
  - Por el tipo de etiqueta

```
var objArray = document.getElementsByName("objName");
```

```
var liArray = document.getElementsByTagName("li");
```

```
var classArray = document.getElementsByClassName("myclass");
```

- Modificación de elementos a través del DOM:
  - Modificar el contenido HTML de un elemento:

```
var element = document.getElementById("txt");
element.innerHTML = "<p>Nuevo texto</p>"
```

  - Cambiar el estilo CSS de un elemento:

```
var element = document.getElementById("img1");
element.style.borderWidth = "3px";
```

http://stackoverflow.com/documentation/javascript/185/getting-started-with-javascript

Document Object Model (DOM)

- Los eventos HTML pueden ser manejados en JavaScript:

Tipo	Evento	Ocurrencia
Ratón	`onclick`	Al hacer click con el ratón
	`ondblclick`	Al hacer doble click con el ratón
	`onmouseover`	Al pasar por encima el cursor
Teclado	`onkeydown`	Al presionar una tecla
Página	`onload`	Al empezar a cargar una página
	`onbeforeunload`	Justo antes de abandonar una página
Formulario	`onchange`	Cuando un campo de formulario cambia de valor
	`onsubmit`	Justo antes de enviar un formulario al servidor

- Más eventos: http://www.w3schools.com/jsref/dom_obj_event.asp

Document Object Model (DOM)

- Ejemplo de captura de eventos:
  - Validación de un formulario

```
<form name="myForm" action="demo_form.jsp" onsubmit="return validateForm();" method="post">
  Name: <input type="text" name="fname">
  <input type="submit" value="Submit">
</form>
```

```
function validateForm() {
   var x = document.forms["myForm"]["fname"].value;
   if (!x) {
      alert("Name must be filled out");
      return false;
   }
   return true;
}
```

Browser Object Model (BOM)

- El objeto `window` representa una ventana/pestaña del navegador
- Algunas propiedades importantes:
  - `document` : Vista anteriormente
  - `history` : Historial de navegación
  - `location`: URL de la página
- Algunos métodos importantes:
  - `alert()` : Genera un cuadro de diálogo de tipo alerta
  - `confirm()` : Genera un cuadro de diálogo de tipo confirmación (“aceptar-cancelar)
  - `open()`: Navega hacia una URL

Cookies

- Las cookies son piezas de información enviadas por una aplicación web y almacenada en la caché del cliente (navegador)
- Están formadas por clave=valor
- Desde mayo de 2011 una normativa europea obliga a todos los sitios webs alojados en Europa a informar del uso de cookies a sus usuarios
  - Más información en: http://www.cookielaw.org/the-cookie-law/
- Las cookies se pueden gestionar desde JavaScript mediante la propiedad `cookie` del objeto `document`

Modo estricto

- Usar el modo estricto (desde ECMAScript 5) significa optar explícitamente por una variante restringida de JavaScript
  - Atrapa errores comunes de programación, lanzando excepciones
  - Deshabilita algunas características que no son recomendables
- El modo estricto se invoca mediante el siguiente comando:

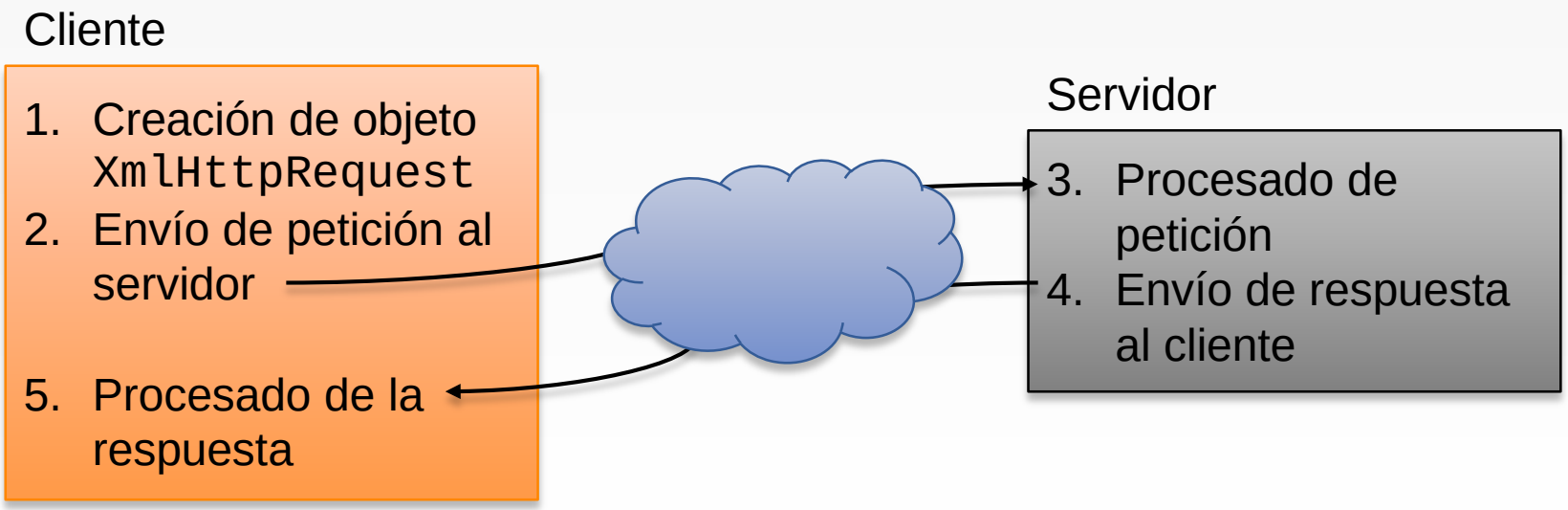
```
"use strict";
```

- Algunos ejemplos de modo estricto:
  - Las variables tienen que ser declaradas para ser utilizadas
  - Las palabras reservadas no pueden utilizarse como identificadores
  - …

https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Modo_estricto

AJAX

- Como ya sabemos, **AJAX** (*Asynchronous JavaScript And XML*), es una técnica que nos permite hacer peticiones en segundo plano (sin recargar la página) a un servidor web
- En los navegadores modernos, se implementa a través del objeto `XMLHttpRequest`



AJAX

- Constructor del objeto `XMLHttpRequest`:

```
var xhr = new XMLHttpRequest();
```

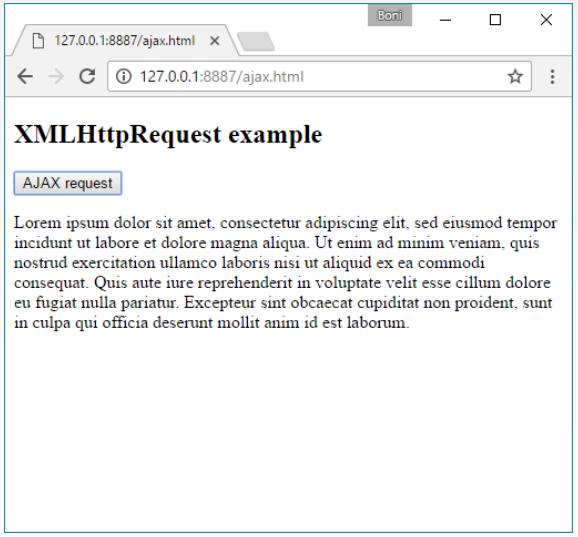
- Métodos importantes del objeto `XMLHttpRequest`:
  - `xhr.open(method, url, async);` (especifica la petición)
    - `method` puede ser `"GET"` o `"POST"`
    - `url` es la ruta en el servidor destino
    - `async` determina el tipo de petición: síncrona (`false`) o asíncrona (`true`)
  - `xhr.send();` (envía la petición)
  - `xhr.abort();` (cancela la petición actual)

AJAX

- Propiedades importantes del objeto `XMLHttpRequest`:
  - `xhr.onreadystatechange` (define la función que atiende a la respuesta (*callbak*)
  - `xhr.readystate` (especifica el estado de la petición a nivel AJAX)
    - `0` petición no inicializada
    - `1` conexión establecida con el servidor
    - `2` petición recibida
    - `3` petición en proceso
    - `4` petición finalizada y respuesta lista
  - `xhr.status` (especifica el estado de la petición a nivel HTTP)
    - `200` ok
    - `403` prohibido
    - `404` no encontrado

AJAX

```
<!DOCTYPE html>
<html>

<body>
    <h2>XMLHttpRequest example</h2>
    <button type="button" id="mybutton">AJAX request</button>
    <p id="content"></p>
</body>

<script>
    var button = document.getElementById("mybutton");
    button.addEventListener("click", function () {
        var xhr = new XMLHttpRequest();
        xhr.onreadystatechange = function () {
            if (this.readyState == 4 && this.status == 200) {
                document.getElementById("content").innerHTML +=
                   this.responseText + "<br>";
            }
        };
        xhr.open("GET", "info.txt", true);
        xhr.send();
    });
</script>

</html>
```

Orientación a objetos

Clases vs prototipos

- La mayoría de lenguajes (Java, C#, C++, Python…) implementan **POO basada en clases**
- JavaScript implementa la **POO basada en prototipos**

POO basada en clases

- Los objetos son instancias de una clase
- La clase se utiliza para definir las propiedades y métodos que tendrán los objetos de esa clase
- Cada objeto se diferencia entre sí por el valor de los atributos
- Todos los objetos de una clase tienen los mismos métodos

POO basada en prototipos

- No existen las clases pero sí los **prototipos**
- Se le pueden **añadir** y **borrar** atributos y métodos en cualquier momento
- La herencia se realiza mediante **cadenas de prototipos**

Orientación a objetos

Creación de objetos

- En JavaScript tres formas principales de crear un objeto:
  1. Usando un objeto literal
  2. Creando un objeto `Object` con la palabra reservada `new`
  3. Definiendo un constructor y usando la palabra reservada `new`

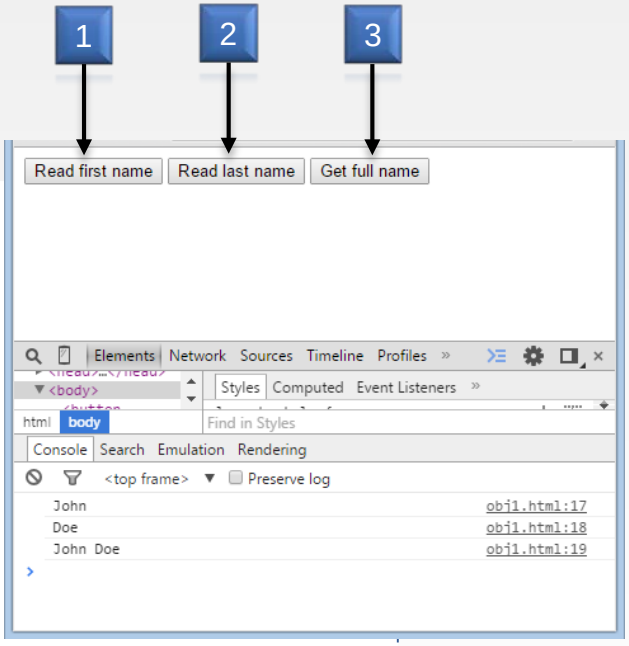
- Desde la versión ECMAScript 5 hay una cuarta forma de crear objetos, usando `Object.create()` equivalente a usar `new`

Orientación a objetos

Creación de objetos

1. Usando un objeto literal

```
<!DOCTYPE html>
<html>
<head>
<script>
var person = {
   firstName : "John",
   lastName : "Doe",
   fullName : function() {
      return this.firstName + " " + this.lastName;
   }
};
</script>
</head>
<body>
<button onclick="console.log(person.firstName);">Read first name</button>
<button onclick="console.log(person['lastName']);">Read last name</button>
<button onclick="console.log(person.fullName());">Get full name</button>
</body>
</html>
```

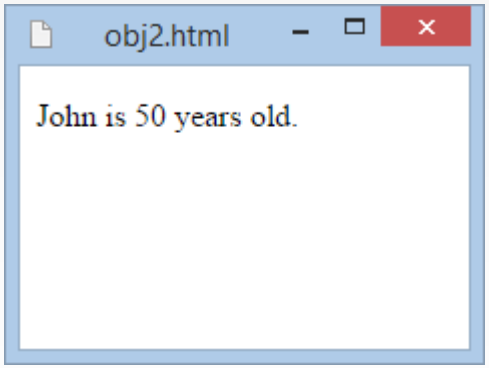


Orientación a objetos

Creación de objetos

2. Creando un objeto `Object` con la palabra reservada `new`

```
<!DOCTYPE html>
<html>
<head>
<script>
window.onload = function() {
    var person = new Object();
    person.firstName = "John";
    person.lastName = "Doe";
    person.age = 50;
    person.eyeColor = "blue";

    document.getElementById("demo").innerHTML = person.firstName
        + " is " + person.age + " years old."
}

</script>
</head>
<body>
<p id="demo"></p>
</body>
</html>
```

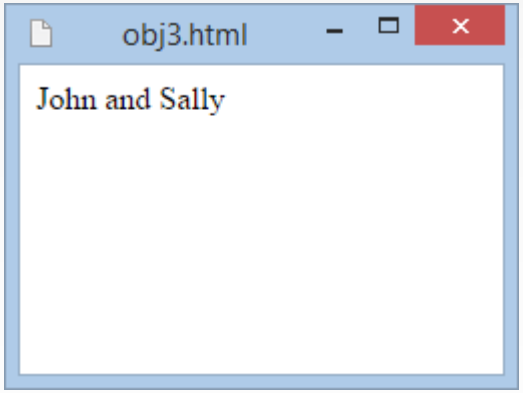


Orientación a objetos

Creación de objetos

3. Definiendo un constructor y usando la palabra reservada new

```
<!DOCTYPE html>
<html>
<head>
<script>
function Person(first, last, age, eyecolor) {
   this.firstName = first;
   this.lastName = last;
   this.age = age;
   this.eyeColor = eyecolor;
}
var man = new Person("John", "Doe", 50, "blue");
var women = new Person("Sally", "Rally", 48, "green");

document.write(man.firstName + " and " + women.firstName);
</script>
</head>
<body>
</body>
</html>
```

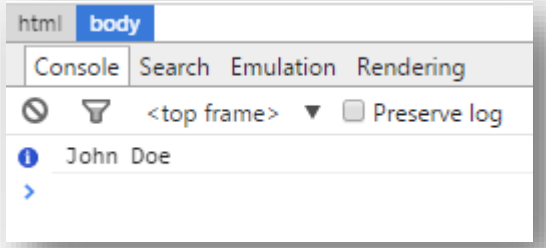


Orientación a objetos

Propiedades de objetos

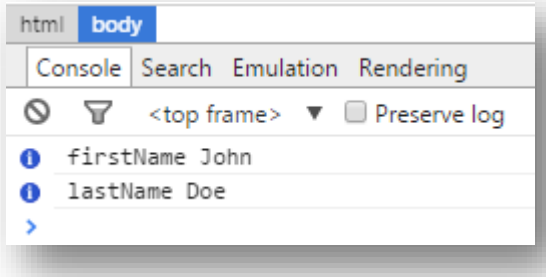
- Acceso:

```
var person = {
   firstName : "John",
   lastName : "Doe"
};
console.info(person.firstName + " " + person["lastName"]);
```



- Acceso mediante bucle `for … in`:

```
var person = {
   firstName : "John",
   lastName : "Doe"
};

var i;
for (i in person) {
   console.info(i + " " + person[i]);
}
```

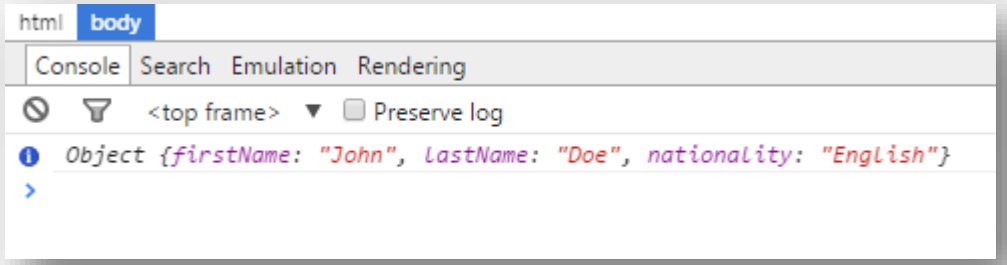


Orientación a objetos

Propiedades de objetos

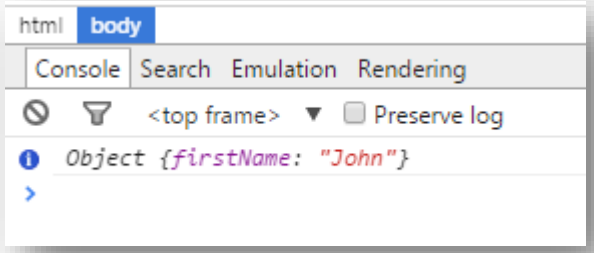
- Añadir nuevas propiedades:

```
var person = {
   firstName : "John",
   lastName : "Doe"
};
person.nationality = "English";
console.info(person);
```



- Eliminar propiedades existentes:

```
var person = {
   firstName : "John",
   lastName : "Doe"
};
delete person.lastName;
console.info(person);
```

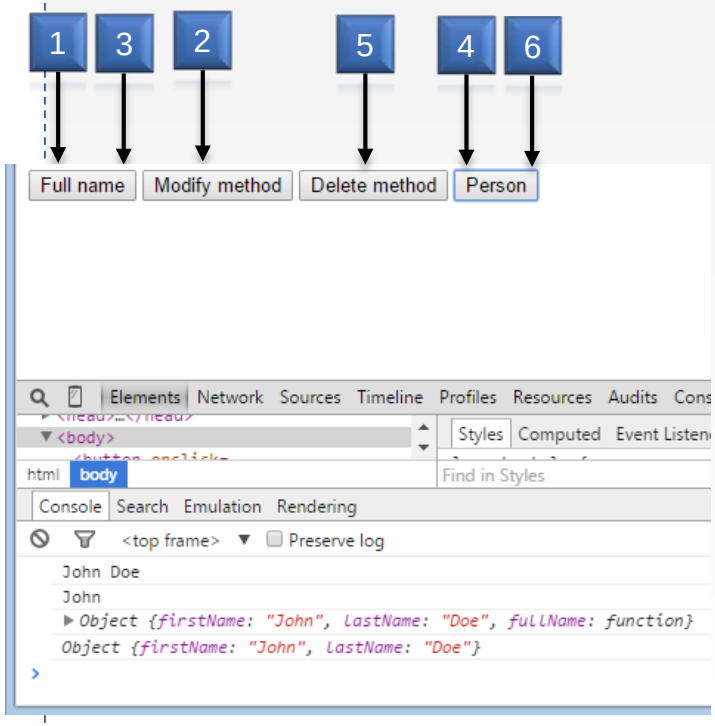


Orientación a objetos

Métodos de objeto

- Se pueden modificar los métodos de un objeto dinámicamente

```
<!DOCTYPE html>
<html>
<head>
<script>
var person = {
   firstName : "John", lastName : "Doe",
   fullName : function() {
      return this.firstName + " " + this.lastName;
   }
};
function modifyMethod() {
   person.fullName = function() {
      return this.firstName;
   }
}
function deleteMethod() {
   delete person.fullName;
}
</script>
</head>
<body>
<button onclick="console.log(person.fullName());">Full name</button>
<button onclick="modifyMethod();">Modify method</button>
<button onclick="deleteMethod();">Delete method</button>
<button onclick="console.log(person);">Person</button>
</body>
</html>
```



Orientación a objetos

Prototipos en JavaScript

- La POO en JavaScript está basada en prototipos
- Todos los objetos en JavaScript **heredan** propiedades y métodos de su prototipo
- La forma estándar de crear un prototipo es mediante su **constructor**
- Hasta ahora hemos visto como añadir/eliminar dinámicamente propiedades y métodos de un objeto, pero también se puede modificar dinámicamente el prototipo

```
function person(first, last, age, eyecolor) {
   this.firstName = first;
   this.lastName = last;
   this.age = age;
   this.eyeColor = eyecolor;
}
person.prototype.name = function() {
   return this.firstName + " " + this.lastName;
};
```

Orientación a objetos

Palabra clave `this`

- La palabra clave `this` tiene un comportamiento diferente al de otros lenguajes
- El valor de `this` está determinado por cómo se llama a la función:
  1. En el contexto de ejecución global (fuera de cualquier función), `this` se refiere al objeto global
  2. Dentro de una función/objeto, `this` se refiere al llamante
  3. Dentro de un constructor, el valor de `this` se refiere al objeto creado

https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Operadores/this

Orientación a objetos

Palabra clave `this`

1. En el contexto de ejecución global (fuera de cualquier función), `this` se refiere al objeto global
2. Dentro de una función/objeto, `this` se refiere al llamante
3. Dentro de un constructor, el valor de `this` se refiere al objeto creado

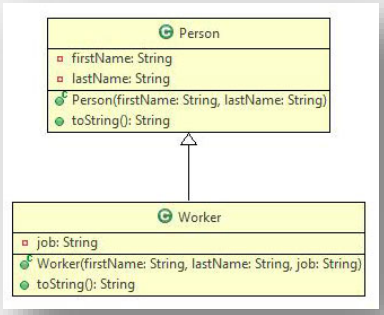
```
console.log("1 " + this); // Window

function f1() {
   return this;
}
console.log("2.a " + f1()); // Window

function f2() {
   "use strict";
   return this;
}
console.log("2.b " + f2()); // undefined
console.log("2.c " + window.f2()); // Window

var foo = {
   baz : function() {
      console.log("2.d " + this);
   }
}
foo.baz(); // Object

function myObject() {
   this.a = 37;
}
var o = new myObject();
console.log("3 " + o.a); // 37
```

```
1 [object Window]
2.a [object Window]
2.b undefined
2.c [object Window]
2.d [object Object]
3 37
```

Orientación a objetos

Herencia

- La herencia en JavaScript se hace usando **cadenas de prototipos**
- Vamos a analizarlo mediante un ejemplo. Sean las clases Java:




```java
public class Person {

   private String firstName;
   private String lastName;

   public Person(String firstName, String
lastName) {
      this.firstName = firstName;
      this.lastName = lastName;
   }

   @Override
   public String toString() {
      return firstName + " " + lastName;
   }
}
```

```java
public class Worker extends Person {

   private String job;

   public Worker(String firstName, String
lastName, String job) {
      super(firstName, lastName);
      this.job = job;
   }

   @Override
   public String toString() {
      return super.toString() + " - Job: "
+ this.job;
   }
}
```

Orientación a objetos

Herencia

- El equivalente del ejemplo anterior pero en JavaScript sería:

```
function Person(firstName, lastName) {
   this.firstName = firstName;
   this.lastName = lastName;
}

Person.prototype.toString = function() {
    return this.firstName + " "
       + this.lastName;
};
```

```
function Worker(firstName, lastName, job) {
   Person.call(this, firstName, lastName);
   this.job = job;
}

Worker.prototype = new Person();

Worker.prototype.toString = function() {
   return Person.prototype.toString.call(this)
      + " - Job: " + this.job;
};
```

La función `call` permite llamar a un método de un objeto sobrescribiendo el valor de `this`

Orientación a objetos

TypeScript

- **TypeScript** es un superconjunto de JavaScript
- Es *open source* (licencia Apache), desarrollado por Microsoft
- Añade tipado estático y objetos basados en clases

```
class Greeter {
    constructor(public greeting: string) { }
    greet() {
        return "<h1>" + this.greeting + "</h1>";
    }
};

var greeter = new Greeter("Hello, world!");

document.body.innerHTML = greeter.greet();
```

```
var Greeter = (function () {
    function Greeter(greeting) {
        this.greeting = greeting;
    }
    Greeter.prototype.greet = function () {
        return "<h1>" + this.greeting + "</h1>";
    };
    return Greeter;
}());

var greeter = new Greeter("Hello, world!");
document.body.innerHTML = greeter.greet();
```

TypeScript

http://www.typescriptlang.org/