

IFCD65

Módulo 2

Javascript ES6

1. Caracterización de los fundamentos de la programación Front End:
JavaScript
2. Aplicación de los elementos más avanzados de JavaScript
3. Interpretación y reescritura de llamadas, datos y código AJAX
4. Reproducción con código, de prototipos realizados con
herramientas de diseño gráfico
5. Manipulación de códigos de terceros
6. Elaboración de documentación técnica y de usuario de lenguaje
Javascript

JavaScript (ES6)

1. Caracterización de los fundamentos de la programación Front End: JavaScript

- Historia y evolución de JavaScript
- Variables
- Tipos de datos: Boolean, Number, String, Date, undefined, BigInt y Symbol
- Objetos
- Valores especiales: null, NaN e Infinity
- Operadores y precedencia de operadores
- Expresiones
- Bloque de control del flujo y control / tratamiento de errores
- Scopes
- Funciones
- Gestión de eventos
- JavaScript y DOM: cómo se relacionan HTML, CSS y JavaScript

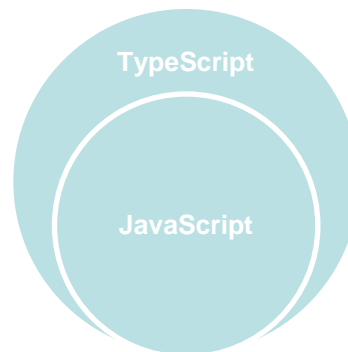
JavaScript

De contenido estático a contenido dinámico e interactivo

- A excepción de los elementos hipervínculos HTML (tag <a>), el contenido es estático, es decir, no es interactivo, pues no cambia frente a acciones del usuario.
- Utilizando reglas de estilo CSS, se enriquece el aspecto, y se crean efectos interactivos limitados, como por ejemplo cambiar el aspecto de un hyperlink al pasar sobre este.
- JavaScript permite crear **contenido interactivo** y **dinámico**:
 - Modificación del aspecto visual del documento web
 - Validación avanzada de formularios
 - Manejo de eventos
 - Interacción con el servidor para descarga de datos (AJAX)
 - Soportado en todos los navegadores

Características

- Originalmente llamado LiveScript
- Es **interpretado** y **orientado a objetos**
- Basado en la especificación del lenguaje, **ECMAScript (ES)**
- Su uso se ha incrementado significativamente por el enfoque **interactivo** y **dinámico** de los sitios web actuales.
- Desde 2009 podemos crear aplicaciones **fullstack** con un solo lenguaje de programación tanto en el **cliente** como en el **servidor**.



JavaScript

Historial de versiones

Versión	Fecha	Características principales
ES1	1997	Primera edición
ES2	1998	Alineación con ISO/IEC 16262
ES3	1999	Expresiones regulares, manejo de cadenas, try / catch, formatos
ES4	Nunca lanzada	
ES5	2009	strict mode, getters / setters, JSON, reflection
ES6	2015	Let y const, arrays tipados, promises
ECMAScript 2016	2016	ES2016, aislado del código, ** y Array.prototype.includes
ECMAScript 2017	2017	Concurrencia, await / async, sobrecarga de operadores, registros
ECMAScript 2018	2018	Propiedades de descanso/propagación, iteración asíncrona

Ejemplo JavaScript 1

```
<html>
<head>
<meta charset="utf-8" />
<title>Ejemplo de código JavaScript en el propio documento</title>
<script type="text/javascript">
    alert("Hola Mundo");
</script>
</head>
<body>
<p>Un párrafo de texto.</p>
</body>
</html>
```

Ejemplo JavaScript 1

Archivo codigo.js guardado en la carpeta js
alert("Un mensaje de prueba");

Documento HTML

```
<html>
<head>
<meta charset="utf-8">
<title>Ejemplo de código JavaScript en el propio documento</title>
<script type="text/javascript" src="js/codigo.js">
</script>
</head>
<body>
<p>Un párrafo de texto.</p>
</body>
</html>
```


<noscript>

<head> ... </head>

<body>

<noscript>

<p>Bienvenido a Mi Sitio</p>

<p>La página que estás viendo requiere para su funcionamiento el uso de JavaScript. Si lo has deshabilitado intencionadamente, por favor vuelve a activarlo.</p> </noscript>

</body>

Normas básicas

- No se tienen en cuenta los espacios en blanco y las nuevas líneas.
- Se distinguen las mayúsculas y minúsculas.
- No se define el tipo de las variables.
- Se pueden incluir comentarios:

Ejemplo de comentario de una sola línea:

```
// a continuación se muestra un mensaje alert("mensaje de prueba");
```

Ejemplo de comentario de varias líneas:

```
/* Los comentarios de varias líneas son muy útiles cuando  
se necesita incluir bastante información en los comentarios */  
alert("mensaje de prueba");
```

Ejercicio JS1

Modificar el primer ejemplo para que:

- Después del primer mensaje, se debe mostrar otro mensaje que diga *"Soy el primer script"*.
- Añadir algunos comentarios que expliquen el funcionamiento del código.
- Añadir en la página HTML un mensaje de aviso para los navegadores que no tengan activado el soporte de JavaScript.

JavaScript

Estructura general

- El tag **<script>** permite añadir un bloque de instrucciones JavaScript
- El bloque JS normalmente en el **<head>** del documento.

```
<script type="text/javascript">  
    //código JavaScript  
</script>
```

type: Define el tipo MIME del script incluido. Debe ser "text/javascript". Actualmente se puede omitir.

- Algunos navegadores **no soportan scripts**, por lo que ignorarán la marca **<script>**, pero no el contenido. La solución es escribir el bloque JS entre los símbolos de comentarios:

```
<script type="text/javascript">  
    <!-- ocultar script  
    //código JavaScript -->  
</script>
```

JavaScript

Código JS externo

- Código JavaScript contenido en archivo de extensión **.js** referenciado desde el documento:

```
<script type="text/javascript" src="myscript.js"></script>
```

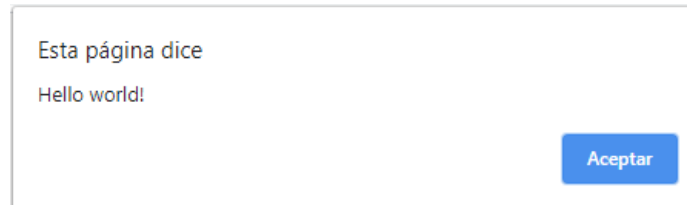
src: URL que referencia al archivo
que contiene el código JavaScript.
Permite rutas relativas y absolutas.

- El contenido de `myscript.js` es directamente el código JavaScript, sin etiquetas.
- Esta opción permite **reutilizar el código** en otras páginas, por lo que es la **recomendable**.

JavaScript

- “Hola Mundo” en JavaScript

```
<script>  
window.alert("Hello world!");  
document.write("<p>Hola Mundo!</p>");  
</script>
```

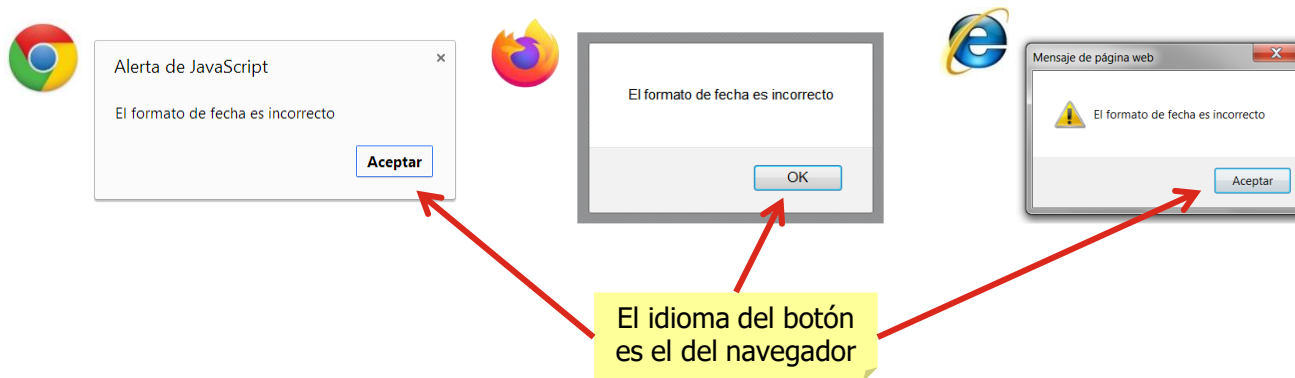


Sintaxis básica JavaScript

Mensajes interactivos

- JavaScript provee comandos para mostrar mensajes interactivos al usuario:
- alert**: mensajes informativos o de advertencia.

```
alert("El formato de fecha es incorrecto");
```



Sintaxis básica JavaScript

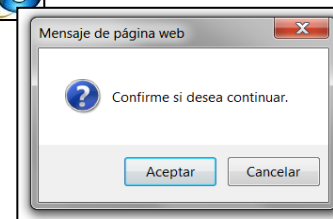
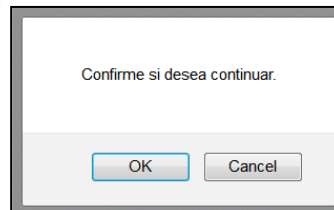
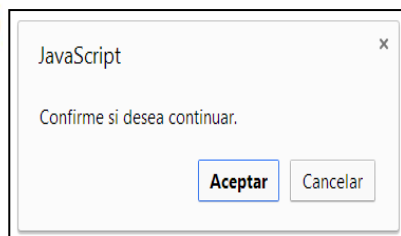
Mensajes interactivos

- **confirm**: mensajes de decisión.

```
if (confirm("Confirme si desea continuar.")) {  
    //lógica si marca Aceptar/OK  
}
```

El retorno indica el
botón presionado:

- Aceptar/OK: true
- Cancelar: false



- **prompt**: permite recoger valores alfanuméricos.

Manejar herramientas online para edición y prueba de código JavaScript, HTML y CSS.

1. Probar las funciones alert, prompt y confirm en el editor de código online codepen.

<https://codepen.io/>

15 min

Sintaxis básica JavaScript

Comentarios

- Los comentarios pueden ser por línea o por bloque:

```
//Comentarios
```

```
//de
```

```
//línea
```

Ejemplo:

```
var a = 56; //assign. básica
```

```
/*
```

```
Comentario de bloque
```

```
*/
```

Ejemplo:

```
/*
```

```
Con el siguiente comando se obtienen  
las propiedades del navegador
```

```
*/
```

```
var nav = window.navigator;
```

Sintaxis básica JavaScript

- Un **identificador** es un nombre para referenciar a un elemento del código (clase, variable, método...)
- Es una secuencia alfanumérica cuyo primer carácter no es un número.
- Se pueden incluir vocales acentuadas, la ñ y letras con símbolos (ç, ...), aunque no es recomendable por problemas de portabilidad entre plataformas, ya que los juegos de caracteres pueden afectar el código.
- Se permiten incluir otros caracteres imprimibles (\$, _) exceptuando aquellos que tienen un significado especial dentro del lenguaje (por ejemplo % y ?). De todas formas, no se recomienda.
- Ejemplos válidos pero no todos cumplen las convenciones:

Arriba, caña, C3P0, àëîò, hola, _barco_

- Ejemplos no válidos:

3com, 123, 4D5v, C#, a!b, r"t, qwe`t

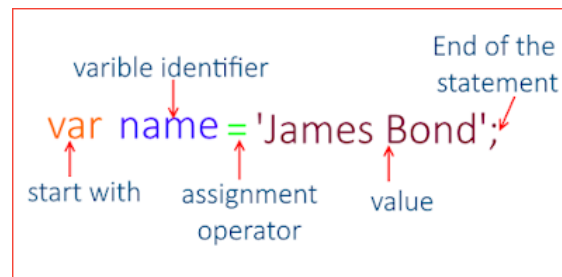
↑	↑	↑	↑	↑	↑	↑	↑
comienzan con número			caracteres no válidos				

Sintaxis básica JavaScript

Variables

- Una **variable** es una referencia a un valor u objeto de un determinado tipo. Para referirse al valor utilizamos su identificador.
- Declaración o definición de una variable con **var**:

```
var <identificador>;  
var x; var letra; var mensaje; var obj;
```



- Asimismo se puede inicializar una variable asignándole un valor u objeto en la declaración, utilizando el símbolo "=":

```
var x = 45;  
var mensaje = "Hola";  
var enabled = true;  
var today = new Date();
```

Sintaxis básica JavaScript

const

- A partir de la versión 2015 de JavaScript (ES6 - ECMAScript 2015) podemos usar **const** para declarar **una constante** (no se puede reasignar).

```
const A_STRING = 'I am a string';  
const A_NUMBER = 100;  
const AN_ARRAY = [ 'var1', 12, { foo: 'bar' } ];  
const AN_OBJECT = { name: 'MyAPP', ver: '1.0' };  
/*
```

Convención: escribir las constantes en mayúsculas para
distinguir las del resto de variables

```
*/
```

Sintaxis básica JavaScript

let

- A partir de la versión 2015 de JavaScript (ES6 - ECMAScript 2015) podemos usar **let** para definir una variable con **alcance restringido**.
- Su ámbito o scope es el bloque en donde está definida:

```
function pets() {  
    var dog = "Fido";  
    if (true) {  
        let cat = "Fluffy";  
    }  
}
```

← Alcance de **var** es en cualquier parte dentro la función

← Alcance de **let** es dentro del bloque {}

const vs let vs var

keyword	const	let	var
Alcance Global	NO	NO	SÍ
Alcance Función	SÍ	SÍ	SÍ
Alcance Bloque	SÍ	SÍ	NO
Re-asignable	NO	SÍ	SÍ

Sintaxis básica JavaScript

Ciclo de vida de una variable

- Una **variable** se crea en el ámbito de un bloque que la contiene:

```
var count;

function inc() {
    count++;

    var message = "La cuenta es " + count;

    alert(message);
}
```

Las variables que están fuera de un bloque son globales dentro de la página.

Inicio bloque

Variable local

Fin bloque

Si una variable se declara sin 'var', se considera global. Esto es **no recomendable**.

- Todas las variables creadas son **destruidas al salir de la página**.
- Si se vuelve a entrar a la página, las variables se crean nuevamente, no estando relacionadas con valores anteriores de las mismas.

Sintaxis básica JavaScript

Tipos de variables

- Una **variable** puede ser de varios tipos, aunque siempre se identifica como var.
- Los tipos utilizados son:
 - **texto (string)**: se asignan con una cadena, con comillas dobles o simples:

```
var mensaje = "El campo 'nombre' es requerido";
```

```
var mensaje = 'Comillas "dobles"';
```

Las comillas simples y dobles son intercambiables, siempre que se mantenga el orden de cierre.

- **número (number)**: se asignan con un número de cualquier tipo. No se hace distinción entre enteros o números con decimales.

```
var max = 10;
```

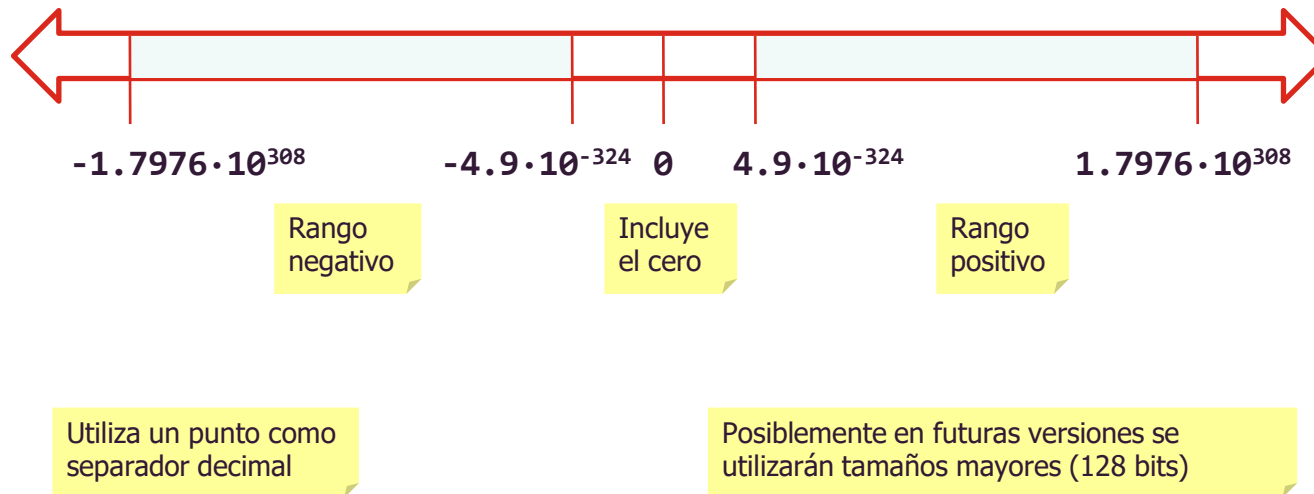
```
var distancia = 12.34;
```

Separador decimal: punto.

Sintaxis básica JavaScript

Número

- JavaScript maneja variables de tipo **numérico (number)**. Internamente se representan como un número de punto flotante de **64 bits**, lo que permite **doble precisión** y el rango siguiente:



Sintaxis básica JavaScript

String

- JavaScript maneja variables de tipo **texto (string)**. Se declaran como textos entre comillas simples o dobles. Internamente se almacenan como una cadena de caracteres, sin un largo máximo establecido. Ejemplo:

```
var mensaje = "Hola";
```

- Las cadenas son objetos que tienen asociadas algunas funciones y propiedades predefinidas útiles para su manejo.
- Para conocer la longitud de un string, es decir, el número de caracteres, se utiliza la propiedad **length**:

```
str = "12345";
```

```
str.length
```



```
5
```

Sintaxis básica JavaScript

Boolean

- **lógico (boolean)**: se asignan con los valores true o false.

```
var enabled = true;  
var isValid = false;
```

Sintaxis básica JavaScript

- JavaScript maneja variables de tipo **lógico (boolean)**. Internamente se representan como un valor equivalente a un bit.
- Ejemplos:

```
var enabled = true;  
  
var isFinished = count > 10;  
  
var hasValue = message != "";
```

Un boolean puede
contener el resultado de
un operador relacional

Asignación

Comparación

Sintaxis básica JavaScript

Conversión de Tipos

- Si se tiene un valor tipo texto, por ejemplo "1", y se quiere manejar como número, JavaScript provee funciones para **convertir** entre tipos de variables. Esto es útil porque al ser todas 'var', no es posible diferenciarlas por su tipo.
- Ejemplos:

```
var strOne = "1";  
var numOne = Number(strOne);  
  
var strEnabled = "true";  
var enabled = Boolean(strEnabled);
```

- También es posible obtener el tipo interno de una variable, utilizando typeof. Ejemplo:

```
if (typeof numOne == "number") {  
    alert("numOne es un number");  
}
```

Sintaxis básica JavaScript

Operadores aritméticos y relacionales

- Un **operador** es un símbolo formado por uno o más caracteres, que permite realizar una determinada operación entre uno o más datos y produce un resultado.

Operadores aritméticos:

- Unarios: negativo (-) y positivo (+)
- Binarios: suma (+), resta (-), producto (*), división (/) y módulo (%)

Operadores relacionales:

- igualdad (==). No confundir con =, que es para asignación.
- desigualdad (!=)
- mayor que (>)
- menor que (<)
- mayor o igual que (>=)
- menor o igual que (<=)



Sintaxis básica JavaScript

Operadores y asignación

- El operador de **asignación** básico es (=) que, a parte de asignar, devuelve el valor asignado. La asignación es asociativa por la derecha.
- Operadores **compuestos**: realizan una operación y asignan el resultado. Existen para la suma (**+=**), resta (**-=**), producto (***=**), división (**/=**), módulo (**%=**).
- Operadores **incrementales**: realizan un incremento (**++**) o decremento (**--**) de una unidad en una variable. Si el operador se coloca antes del nombre de la variable devuelve el valor de la variable antes de incrementarla; si se hace después, se incrementa y devuelve el valor ya incrementado.



Incremento

```
var numero = 5;  
++numero;  
alert(numero); // numero  
= 6
```



```
var numero = 5;  
numero = numero + 1;  
alert(numero); //  
numero = 6
```

Decremento

```
var numero = 5;  
--numero;  
alert(numero); // numero  
= 4
```



```
var numero = 5;  
numero = numero - 1;  
alert(numero); //  
numero = 4
```

Incremento / Decremento

```
var numero1 = 5;  
var numero2 = 2;  
numero3 = numero1++ +  
numero2;  
// numero3 = 7, numero1 = 6
```



```
var numero1 = 5;  
var numero2 = 2;  
numero3 = ++numero1 +  
numero2;  
// numero3 = 8, numero1 =  
6
```

NO ES LO MISMO ++6 QUE 6++

Sintaxis básica JavaScript

Operadores y asignación

- Ejemplos de asignación (todos son **var** tipo **number**):

```
a = 56;           //asignación básica  
  
b = 78 + a;       //asignación básica  
  
c += 23;          //asignación compuesta. Equivale a c = c + 23  
  
i++;              //Incremento. Equivale a i = i + 1  
  
d = i++;           //Asigna e incrementa. Si i vale 3 al inicio, d vale 3.  
  
d = ++i;           //Incrementa y asigna. Si i vale 3 al inicio, d vale 4.
```

Sintaxis básica JavaScript

Concatenación

- Los textos se pueden **concatenar** utilizando el operador "+":
- Como el operador "+" también se utiliza para sumar números, hay ciertos matices a tener en cuenta, los que se ilustran a continuación:

```
var str = "Esto es " + "un string";
```

```
s1 = 1 + 3 + "5" + "7";      457
```

Regla de izquierda a derecha.
Suma 1 + 3, y luego concatena "5" y "7"

```
s2 = 1 + (3 + "5") + 7;      1357
```

Calcula el paréntesis primero.
Concatena 1, el resultado, y 7

```
s3 = "1" + (3 + 5) + 7;      187
```

Como el primer operando es un string,
comienza concatenando

```
s4 = 1 + "3" + 5 + 7;        1357
```

Como la primera operación incluye a un
string, comienza concatenando

```
s5 = "" + 1 + 3 + 5 + 7;      1357
```

Si se quieren concatenar sólo números, se
puede agregar un string vacío

```
s6 = "" + (1 + 3 + 5 + 7);    16
```

Si se quiere convertir una suma a string,
similar a la anterior y también String(...)

```
s6 = String(1 + 3 + 5 + 7);
```



Ejemplo prompt

```
<html>
  <head>
    <meta charset="utf-8" />
    <title>Ejemplo PROMPT</title>
    <script type="text/javascript">
      var numero = prompt("Introduce tu número de DNI
(sin la letra)");
      var letra = prompt("Introduce la letra de tu DNI (en
mayúsculas)");
      alert(numero);
      letra = letra.toUpperCase();
      alert(letra);
    </script>
  </head>
  <body>
    <p>Ejemplo de prompt</p>
  </body>
</html>
```

Ejercicio JS2

Modificar el primer ejemplo para que:

- Utilizando la diapositiva anterior, realizar una suma cualquiera de dos número y mostrar el resultado con una alerta.

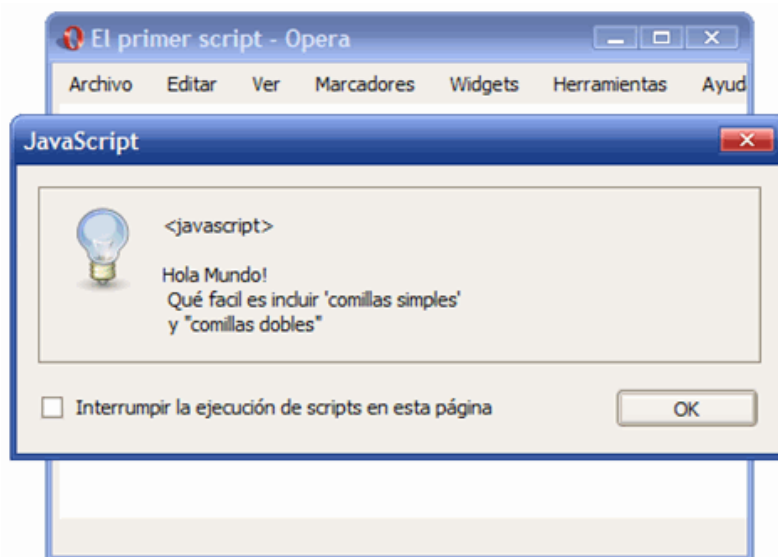
Sintaxis básica JavaScript

Caracteres especiales

- En JavaScript existe una representación específica para algunos caracteres especiales, que permite definirlos y utilizarlos:

Carácter	Significado	Utilización
\n	newline (Nueva línea)	Para saltos de línea en manejo de cadenas
\r	carriage return (Salto de carro)	Idem anterior, en Windows.
\t	tab (tabulación)	Detectar o insertar tabulaciones
\"	comillas (también existe \')	Colocar comillas en un String. Ej: str = "Esto va \"entre comillas\".";
\\	escapa un \	Para escribir un "\" en una cadena. Ej: str = "Para newline se utiliza \\n";
\b	backspace	Poco usado
\f	form feed	Poco usado

Ejercicio JS3



Crear un script para que:

El mensaje que se muestra al usuario se almacene en una variable llamada mensaje y el funcionamiento del script sea el mismo.

El mensaje mostrado sea el de la imagen.

Sintaxis básica JavaScript

Operadores lógicos

- Unarios: not (!)
- Binarios: and (&&) y or (||). Son perezosos (no evalúan el segundo operando cuando se deduce el resultado del primero). No perezosos: & y |, que son poco utilizados.

```
var a = false;  
var b = true;  
var c = true;
```

```
var j = a && (b || c);
```

Evalúa sólo a, y resulta false

```
var k = b || (b && c);
```

Evalúa sólo b, y resulta true

```
var m = !b & (b || c);
```

Evalúa !b y también el paréntesis, del cual sólo evalúa b. Resultado se convierte con Boolean(m)

```
var n = !a | (b && c);
```

Evalúa !a y también el paréntesis, del cual evalúa b y c. Resultado se convierte con Boolean(n)

Sintaxis básica JavaScript

Operador condicional

- Es el único de JavaScript con tres operandos, cuya sintaxis es:

`<condición> ? <expresión1> : <expresión2>`

- Se evalúa `<condición>`
- Si es verdadera se devuelve el resultado de evaluar `<expresión1>`
- Si es falsa, el resultado de evaluar `<expresión2>`
- Ambas expresiones deben ser del tipo al que se asigna el resultado.

- Ejemplo:

`max = (a > b) ? a : b;`

← Si a es mayor que b, devuelve a. En caso contrario, devuelve b

Sintaxis básica JavaScript

Precedencia de operadores

Operadores	Prioridad
!, +, - (unarios)	mayor
*, /, %	
+, -	
<, <=, >=, >	
==, !=	
&&	
= (asignación)	menor

Sintaxis básica JavaScript

Control de flujo

- Sentencia condicional **if**

```
if (<condición>) {  
    <instrucciones>  
} else {  
    <instrucciones>  
}
```

Ejemplo:

```
if (a > b) {  
    alert("A gana");  
} else {  
    alert("B gana");  
}
```

```
if (<condición1>) {  
    <instrucciones>  
} else if (<condición2>) {  
    <instrucciones>  
}  
  
//varias condiciones else if...  
  
} else if (<condiciónN>) {  
    <instrucciones>  
} else {  
    <instrucciones>  
}
```

http://www.w3schools.com/js/js_if_else.asp



Ejercicio JS4

Completar las condiciones de los if del siguiente script para que los mensajes de los alert() se muestren siempre de forma correcta:

```
var numero1 = 5; Solicitar por prompt/input  
var numero2 = 8; Solicitar por prompt /input  
if(numero2>=numero1) { alert("numero1 no es  
mayor que numero2"); }
```

```
if(...) { alert("numero2 es positivo"); }
```

```
if(...) { alert("numero1 es negativo o distinto de  
cero"); }
```

```
if(...) { alert("Incrementar en 1 unidad el valor de  
numero1 no lo hace mayor o igual que numero2"); }
```

Ejercicio JS5

Ejercicio A7

Desarrolle un algoritmo que permita convertir calificaciones numéricas a letras según la siguiente tabla de conversión:

NÚMERO	LETRA
19-20	A
16-18	B
12-15	C
9-11	D
0-8	E

Ejercicio JS6

Ejercicio A8

Desarrolle un algoritmo para determinar el pago de entradas de espectáculo en función del número que compren.

NÚMERO	DESCUENTO
1	-
2	10% $2 \cdot 10 \cdot 0,90$
3	15% $3 \cdot 10 \cdot 0,85$
4	20% $4 \cdot 10 \cdot 0,80$
5 o más	25% $5 \cdot 10 \cdot 0,75$

Ejercicio JS7

Ejercicio A3

Desarrollar un algoritmo que permita leer dos valores distintos y determinar cual de los dos valores es el mayor. Escribir el resultado.

Realizar un algoritmo, además, que sume los dos números.



Sintaxis básica JavaScript

Control de flujo

- Sentencia condicional **switch-case**

```
switch (<expresión>){  
  case <literal1>:  
    <instrucciones>  
    break;  
  case <literal2>:  
    <instrucciones>  
    break;  
  //N valores...  
  case <literalN>:  
    <instrucciones>  
    break;  
  default:  
    <instrucciones>  
}
```

Quando entra a un
"case", continúa hasta el
siguiente "break"

Ejemplo:

```
switch (estado){  
  case 0:  
    alert("Opción 0");  
    break;  
  case 1:  
    alert("Opción 1");  
  case 2:  
    alert("Opción 2");  
    break;  
  default:  
    alert("Ninguna");  
}
```

si estado es 0:
"Opción 0"
si estado es 1:
"Opción 1"
"Opción 2"

Permite cualquier tipo de
variable. Normalmente se
utiliza Number y String

Ya que no hay
break en case 1



Switch

```
switch(variable) {  
  case valor_1:  
    ...  
    break;  
  case valor_2:  
    ...  
    break;  
  ...  
  case valor_n:  
    ...  
    break;  
  default:  
    ...  
    break;  
}
```

```
switch(numero  
) {  
  case 5:  
    ...  
    break;  
  case 8:  
    ...  
    break;  
  case 20:  
    ...  
    break;  
  default:  
    ...  
    break;  
}
```

```
var raw_value = 11.0;  
switch(true) {  
  case (raw_value >  
10.0):  
    height = 48;  
    width = 36;  
    break;  
  case (raw_value >  
5.0):  
    height = 40;  
    width = 30;  
    break;  
  default:  
    height = 16;  
    width = 12;  
}
```

http://www.w3schools.com/js/js_switch.asp

Ejercicio JS5-Switch

Ejercicio A7

Desarrolle un algoritmo que permita convertir calificaciones numéricas a letras según la siguiente tabla de conversión:

NÚMERO	LETRA
19-20	A
16-18	B
12-15	C
9-11	D
0-8	E



Ejercicio JS6-Switch

Ejercicio A8

Desarrolle un algoritmo para determinar el pago de entradas de espectáculo en función del número que compren.

NÚMERO	DESCUENTO
1	-
2	10% $2 \cdot 10 \cdot 0,90$
3	15% $3 \cdot 10 \cdot 0,85$
4	20% $4 \cdot 10 \cdot 0,80$
5 o más	25% $5 \cdot 10 \cdot 0,75$



Sintaxis básica JavaScript

Control de flujo

- Sentencia iterativas **while** y **do-while**

```
while (<condición>) {  
    <instrucciones>  
}
```

Ejemplo:

```
var a = 3;  
  
while (a > 0) {  
    alert(a);  
    a--;  
}
```

```
do {  
    <instrucciones>  
while (<condición>);
```

Ejemplo:

```
var a = 0;  
  
do {  
    alert(a);  
    a--;  
} while (a > 0);
```

Pasa al menos
una vez

Sintaxis básica JavaScript

Control de flujo

- Sentencia iterativa **for**

```
for (<declaración>; <condición>; <instrucción>) {  
    <instrucciones>  
}
```

```
for (var i = 0; i < 3; i++) {  
    alert(i);  
}
```

```
for (var <índice> in <iterable>) {  
    <instrucciones>  
}
```

```
var v = [4, 7, 9, 1];  
  
for (var a in v){  
    alert(v[a]);  
}
```

Lista de números (array)



Estructura for..in

```
for(indice in array)
{
    ...
}
```

```
var dias = ["Lunes", "Martes",  
"Miércoles", "Jueves", "Viernes",  
"Sábado", "Domingo"];
```

```
for(i in dias)
{
    alert(dias[i]);
}
```

http://www.w3schools.com/js/js_loop_for.asp

Sintaxis básica JavaScript

Break y continue

- Existen dos elementos de control para los ciclos:
 - break**: detiene el ciclo, continuando con las instrucciones posteriores.
 - continue**: detiene la iteración, continuando con la siguiente.

<pre>for (var i = 0; i < 5; i++) { if (i == 3) { break; } alert(i); }</pre>	0 1 2
<pre>for (var i = 0; i < 5; i++) { if (i == 3) { continue; } alert(i); }</pre>	0 1 2 4

Sintaxis básica JavaScript

Arrays

- Conjunto de valores accesibles mediante un índice que indica su posición dentro del array.

```
var nums = [1, 2, 3];
```

- Otra forma de declarar un array:

```
var nums = new Array();  
nums[0] = 1;  
nums[1] = 2;  
nums[2] = 3;
```

```
var miArray = [ "valorA", "valorB", . . . ];
```

Diagram illustrating the structure of an array declaration:

- `miArray` is the **identificador** (identifier).
- `[` is the opening bracket.
- `"valorA"` is at **index 0**.
- `"valorB"` is at **index 1**.
- `. . .` represents the **lista valores** (list of values).
- `]` is the closing bracket.

- Otros ejemplos de array:

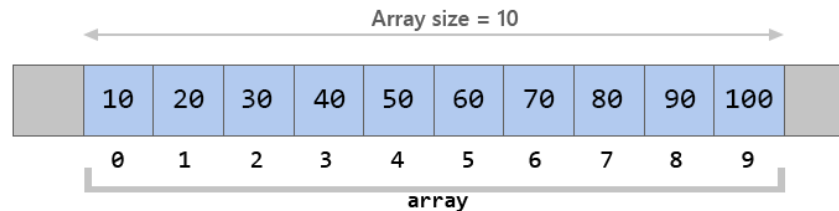
```
var finDeSemana = ["sábado", "domingo"];
```

```
var random = ["sábado", 5, 7, "domingo",];
```



Sintaxis básica JavaScript

Atributo length de Arrays



```
var nums = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100];  
for (var i=0; i < nums.length; i++) {  
    //se accede con nums[i]  
}
```



Ejercicio JS8

El cálculo de la letra del Documento Nacional de Identidad (DNI) es un proceso matemático sencillo que se basa en obtener el resto de la división entera del número de DNI y el número 23. El array de letras es: A partir del resto de la división, se obtiene la letra seleccionándola dentro de un array de letras.

```
var letras = ['T', 'R', 'W', 'A', 'G', 'M', 'Y', 'F', 'P', 'D', 'X', 'B', 'N',  
'J', 'Z', 'S', 'Q', 'V', 'H', 'L', 'C', 'K', 'E', 'T'];
```

Por tanto si el resto de la división es 0, la letra del DNI es la T y si el resto es 3 la letra es la A.

$38000000 \% 23 = 21 \rightarrow \text{letras}[21] \text{ --- K}$

Ejercicio JS8

Con estos datos, elaborar un pequeño script que:

Almacene en una variable el número de DNI indicado por el usuario y en otra variable la letra del DNI que se ha indicado. (Pista: si se quiere pedir directamente al usuario que indique su número y su letra, se puede utilizar la función `prompt()`)

En primer lugar (y en una sola instrucción) se debe comprobar si el número es menor que 0 o mayor que 99999999. Si ese es el caso, se muestra un mensaje al usuario indicando que el número proporcionado no es válido y el programa no muestra más mensajes.

Si el número es válido, se calcula la letra que le corresponde según el método explicado anteriormente. (`var resto = operador1 % operador2;`)

Una vez calculada la letra, se debe comparar con la letra indicada por el usuario. Si no coinciden, se muestra un mensaje al usuario diciéndole que la letra que ha indicado no es correcta. En otro caso, se muestra un mensaje indicando que el número y la letra de DNI son correctos.



Ejercicio JS9

El factorial de un número entero n es una operación matemática que consiste en multiplicar todos los factores $n \times (n-1) \times (n-2) \times \dots \times 1$. Así, el factorial de 5 (escrito como $5!$) es igual a: $5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$

```
For (i=1; i<=numfactoriorial; i++) {  
    factorial = factorial*i  
}
```

Utilizando la estructura for, crear un script que calcule el factorial de un número entero.

Ejercicio JS10

Ejercicio A4

Desarrolle un algoritmo que lea cuatro números distintos y determine cual de los cuatro es mayor.

Pista: Utilizar una tabla

Sintaxis básica JavaScript

Búsqueda de elementos en Arrays

- El método de los arrays `indexOf()` devuelve la posición del elemento pasado como parámetro, si no lo encuentra devuelve -1:

```
días = ["lunes", "martes", "miércoles", "jueves"]  
var posicion = días.indexOf("jueves");
```

- Otra forma de buscar un elemento recorriendo el array a través del método `forEach()`:

```
<script>  
var días = ["lunes", "martes", "miércoles", "jueves"];  
días.forEach(function(dato, indice){  
    document.write("Hoy es " + dato + "indice: " + índice + "<br/>");  
});  
</script>
```

Sintaxis básica JavaScript

Métodos útiles de Arrays

- `push()` añade un elemento al final del array
- `pop()` elimina el último elemento del array
- `shift()` elimina el primer elemento del array
- `unshift()` inserta un elemento al principio del array
- `concat()` fusiona dos arrays en uno
- `split()` convierte un string en array pasando como parámetro el carácter separador a utilizar
- `join()` convierte un array en un string
- `sort()` ordena los elementos del array en orden ascendente
- `reverse()` invierte el orden de un array
- `splice()` reemplaza elementos del array desde una posición concreta, si se omiten elementos a reemplazar se puede usar para eliminar elementos en cualquier posición del array

http://www.w3schools.com/js/js_array_methods.asp

Funciones

```
var resultado;  
var numero1 = 3;  
var numero2 = 5;  
// Se suman los números y se muestra  
el resultado  
resultado = numero1 + numero2;  
alert("El resultado es " + resultado);  
  
numero1 = 10;  
numero2 = 7;  
// Se suman los números y se muestra  
el resultado  
resultado = numero1 + numero2;  
alert("El resultado es " + resultado);  
  
numero1 = 5;  
numero2 = 8;  
// Se suman los números y se muestra  
el resultado  
resultado = numero1 + numero2;  
alert("El resultado es " + resultado);
```

...

```
// Definición de la función  
function  
suma_y_muestra(num1,num2) {  
    var resultado = num1 + num2;  
    alert("El resultado es " +  
resultado);  
}
```

Funciones

```
// Definición de la función
function suma_y_muestra(num1,num2)
{
    var resultado = num1 + num2;
    alert("El resultado es " + resultado);
}

// Declaración de las variables
var numero1 = 3;
var numero2 = 5;
// Llamada a la función
suma_y_muestra(numero1, numero2);

numero1 = 10;
numero2 = 7;
suma_y_muestra(numero1, numero2);

numero1 = 5;
numero2 = 8;
suma_y_muestra(5, 8);
...
```

Funciones

```
// Definición de la función
function calculaPrecioTotal(precio) {
  var impuestos = 1.21;
  var gastosEnvio = 10;
  var precioTotal = ( precio * impuestos ) +
gastosEnvio;
  // percioTotal=38,2414
}

// Llamada a la función
calculaPrecioTotal(23.34);
```

```
function calculaPrecioTotal(precio) {
  var impuestos = 1.21;
  var gastosEnvio = 10;
  var precioTotal = ( precio * impuestos ) +
gastosEnvio;
  return precioTotal;
}

// El valor devuelto por la función, se
guarda en una variable
var precioTotal = calculaPrecioTotal(23.34);
// Seguir trabajando con la variable
"precioTotal"
```

Funciones

```
function calculaPrecioTotal(precio, porcentajImpuestos)
{
    var gastosEnvio = 10;
    var precioConImpuestos = (1 +
porcentajImpuestos/100) * precio;
    var precioTotal = precioConImpuestos +
gastosEnvio;
    return precioTotal;
}

var precioTotal = calculaPrecioTotal(23.34, 21);
var otroPrecioTotal = calculaPrecioTotal(15.20, 4);
alert(precioTotal.round(2));
alert(otroPrecioTotal.round(2));
```

Rendondear a dos decimales el precio total devuelto por la función

Ejercicios JS11

Escribir el código de una función a la que se pasa como parámetro un número entero y devuelve como resultado una cadena de texto que indica si el número es par o impar. Mostrar por pantalla el resultado devuelto por la función.

Funciones - Variables

```
function creaMensaje()  
{  
    var mensaje =  
    "Prueba";  
}  
creaMensaje();  
alert(mensaje);
```

mensaje es una variable
local de la función
mensaje="";

```
function creaMensaje()  
{  
    mensaje =  
    "Prueba";  
}  
creaMensaje();  
alert(mensaje);
```

mensaje es una variable
global
mensaje = "Prueba"

Funciones - Variables

```
var mensaje = "gana la de  
fuera";  
function muestraMensaje()  
{  
    var mensaje = "gana la  
de dentro";  
    alert(mensaje);  
}  
alert(mensaje);  
muestraMensaje();  
alert(mensaje);
```

gana la de fuera
gana la de dentro
gana la de fuera

```
var mensaje = "gana la de  
fuera";  
function muestraMensaje()  
{  
    mensaje = "gana la de  
dentro";  
    alert(mensaje);  
}  
alert(mensaje);  
muestraMensaje();  
alert(mensaje);
```

gana la de fuera
gana la de dentro
gana la de dentro

http://www.w3schools.com/js/js_function_definition.asp

Sintaxis básica JavaScript

Funciones

- Declaración de una función en JavaScript:

```
function mensaje() {  
  alert("Se ha presionado el botón  
'Enviar'");  
}
```

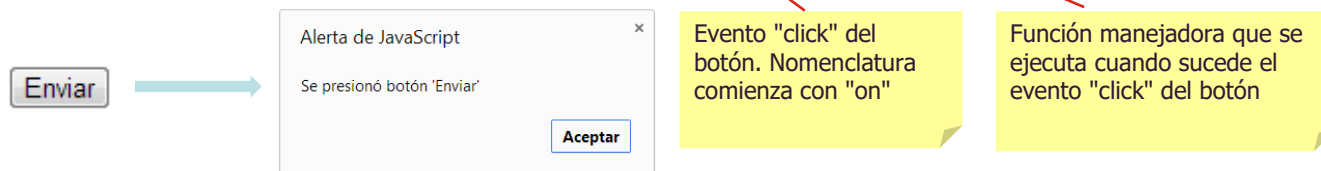
nombre parámetros

```
function add ( a, b ) {  
  return a + b;  
}
```

retorno cuerpo/scope

- Llamada a la función (en el ejemplo asociamos la función a un evento de un botón):

```
<input type="button" value="Enviar" onclick="mensaje()" />
```





Sintaxis básica JavaScript

Función asignada a una variable

```
var suma = function(a, b) {  
  return a * b;  
}
```

← **Declaración**
de la función

```
alert(suma(7, 9));
```

← **Llamada**
a la
función

Sintaxis básica JavaScript

Funciones Flecha

- Definición de una función (ECMA5):

```
function suma(x, y) {  
  return x + y;  
}
```

nombre parámetros flecha / scope

var add = (x, y) => x + y;

retorno directo

- ECMA6 permite declarar funciones de forma **abreviada** llamada **función flecha**:

```
var suma = (x, y) => { x + y };
```

- Sintaxis:

Las funciones flecha son siempre anónimas
Sintaxis más limpia y simplificada

(param1, param2, ..., param n) => { expression; }

https://www.w3schools.com/js/js_arrow_function.asp

Sintaxis básica JavaScript

Tipo String: funciones

- Para buscar la primera posición de un string dentro de otro, se utiliza **indexOf**, y la última, con **lastIndexOf**:

```
str = "ser o no ser, esa es la cuestión";  
      0123456789      18      26  
  
str.indexOf("ser");      => 0      str.lastIndexOf("es");      => 26  
  
str.indexOf("ser", 3);      => 9      str.lastIndexOf("es", 25);      => 18  
  
str.indexOf("hamlet");      => -1      -1 cuando no lo encuentra
```

Sintaxis básica JavaScript

Tipo String: funciones

- Continuando:

- indexOf se utiliza también para saber si un string contiene una cadena:

```
str = "Cinema Paradiso";
```

```
str.indexOf("a P") != -1  $\Rightarrow$  true
```

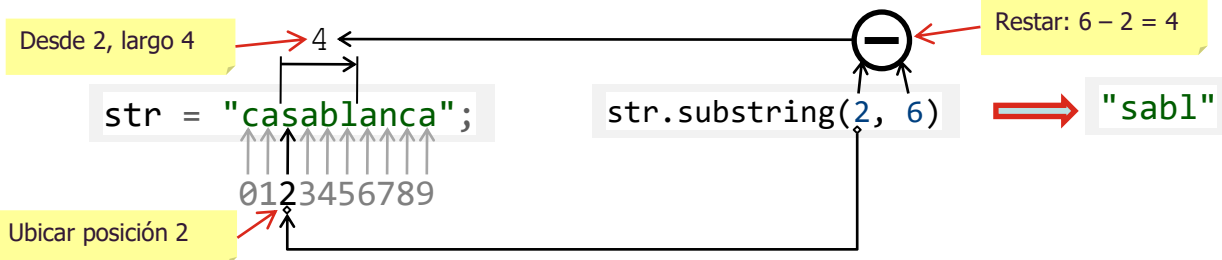
- Para extraer un caracter en una posición, se utiliza charAt:

```
str = "Cinema Paradiso";
```

```
str.charAt(2)  $\Rightarrow$  'n'
```

0 a n-1

- Para extraer un trozo de una cadena, se utiliza substring:



Sintaxis básica JavaScript

Tipo String: funciones

- Continuando:
 - Para pasar el contenido todo a minúsculas, se utiliza `toLowerCase()`, y a mayúsculas es con `toUpperCase()`:

```
str = "Artificial Intelligence";
```

```
str.toLowerCase() ➡ "artificial intelligence"
```

```
str.toUpperCase() ➡ "ARTIFICIAL INTELLIGENCE"
```

- Para comparar dos textos, se utiliza `"=="`:

```
var str1 = "euro";
```

```
var str2 = "\u0065\u0075\u0072\u006f";
```

"euro" en unicode

```
str1 == str2 ➡ true
```



Sintaxis básica JavaScript

Tipo String: funciones

- Para comparar dos textos, ignorando mayúsculas y minúsculas (case-insensitive), se deben pasar a mayúsculas o minúsculas y utilizar "==":

```
str1 = "lower or UPPER";  
str2 = "Lower or Upper";
```

```
str1.toLowerCase() == str2.toLowerCase() ➡ true
```

- Si se compara un texto y un número con el mismo valor, sucede lo siguiente:

```
str1 = "12";  
str2 = 12;
```

```
str1 == str2 ➡ true
```

Compara valores

```
str1 === str2 ➡ false
```

Compara valores y tipos

```
str1 === String(str2) ➡ true
```

```
Number(str1) === str2 ➡ true
```

Sintaxis básica JavaScript

Tipo String: funciones

- Para reemplazar en un String las ocurrencias de una cadena de caracteres por otra, se utiliza el método `replace`:

```
name = "les aventures de tintin et milou";
```

```
name = name.replace("tin","tan");
```

Por default, sólo reemplaza la primera ocurrencia

- El método `replace` admite expresiones regulares (se ven más adelante), lo que flexibiliza su uso:

```
name = "Déjà vu";
```

```
name.replace(/[a-z]/, "_");
```

Encuentra las minúsculas de la "a" a la "z"

⇒ "Dé_à vu"

Reemplaza la primera por un "_"

```
name.replace(/[a-z]/g, "_");
```

⇒ "Dé_à _"

Reemplaza todas las minúsculas por un "_"

```
name.replace(/[a-z]/gi, "_");
```

⇒ "_é_à vu"

Reemplaza todas, ignorando mayúsculas y minúsculas

http://www.w3schools.com/js/js_string_methods.asp

JavaScript

Template Strings (ECMA6)

```
<script>
var numUno = 10;
var numDos = 20;
document.write(`La suma de ${numUno} + ${numDos} =`, numUno + numDos);
</script>
```

Las variables numUno y numDos son sustituidas por sus valores cuando escribimos el nombre entre `${ }`

Ejercicios JS12

Escribir un Javascript que solicita una cadena de caracteres y la devuelve letra a letra.

Ejercicios JS13

Definir una función que muestre información sobre una cadena de texto que se le pasa como argumento. A partir de la cadena que se le pasa, la función determina si esa cadena está formada sólo por mayúsculas, sólo por minúsculas o por una mezcla de ambas.

Eventos

- Señales generadas cuando ocurren acciones específicas.
- Son la base para la **interacción con el usuario**.
- Los eventos se pueden asociar a una función que se ejecute cuando el evento se produzca: **funciones manejadoras**.

Función manejadora:

```
<input type="text" onChange="validarCampo(this)">
```

hace referencia
al evento



Bloque de instrucciones:

```
<input type="text" onChange="
    if (parseInt(this.value) <= 5)
    { alert('Escriba un número mayor que 5. '); }">
```

Eventos

Tipos de eventos

Evento	Descripción
blur	Cuando el usuario hace click fuera de un campo en un formulario
click	Cuando el usuario hace click en un botón, un link o un elemento de un formulario
change	Cuando el usuario cambia el valor de un campo
focus	Cuando se activa el foco en un campo de entrada
load	Cuando se carga una página en el navegador
mouseover	Cuando el puntero del ratón pasa por encima de un hipervínculo
select	Cuando el usuario selecciona un campo de un elemento de formulario
submit	Cuando el usuario envía un formulario
unload	Cuando un usuario abandona una página (para cerrar la ventana o cambiar de página)

Eventos

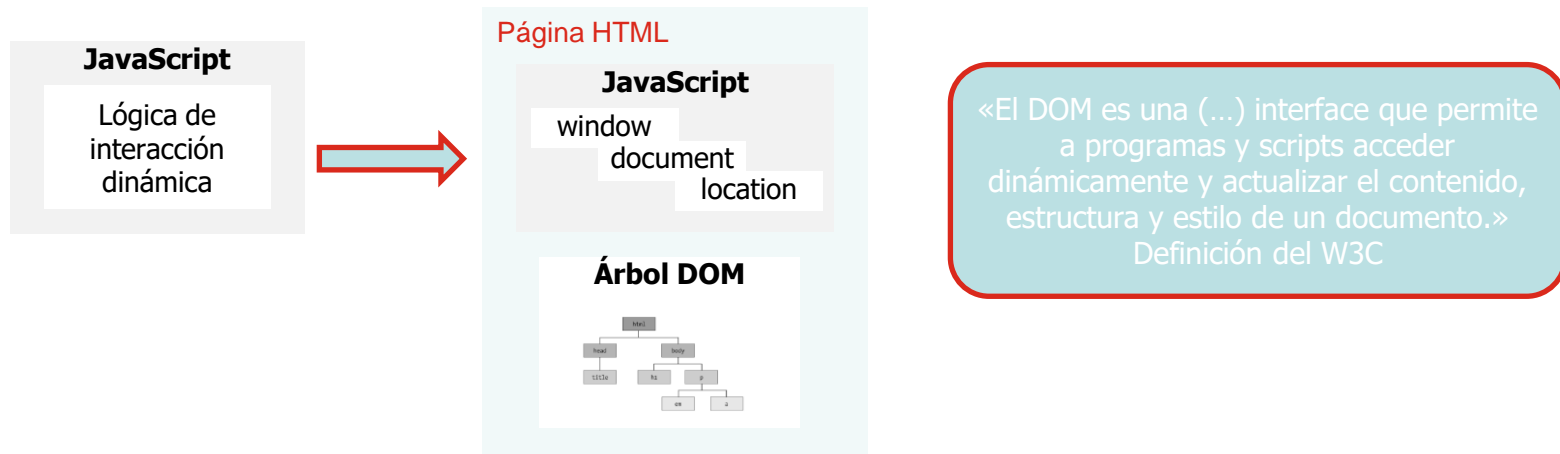
Ejemplo eventos onLoad y onUnload

```
<html>
<head><title>Ejemplo Eventos </title></head>
<body      onLoad="alert('Bienvenido! ');"
           onUnload="alert('Adios! ');">
    <h1>Página con eventos</h1>
</body>
</html>
```

DOM: Document Object Model

Modelo de Objetos del Documento

- El DOM (Modelo de Objetos del Documento) es un estándar del W3C (World Wide Web Consortium) que nos **permite el acceso y la modificación** de los diferentes elementos de un documento HTML.



DOM: Document Object Model

Funcionamiento

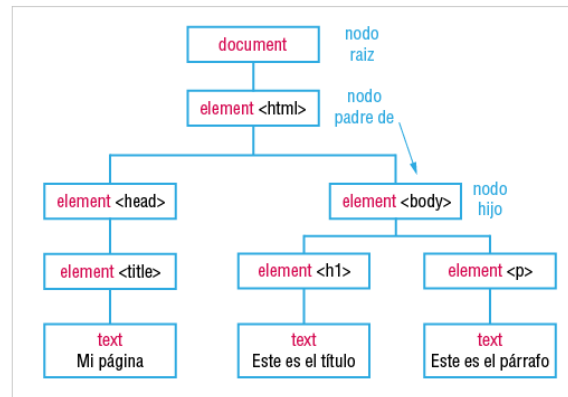
- Cuando creamos un documento HTML, este se compone de diferentes etiquetas que el navegador debe interpretar para poder mostrar los elementos visuales que queremos que el usuario vea en pantalla.
- El navegador realiza la tarea de interpretar las etiquetas y genera una estructura (el DOM) en la que organiza jerárquicamente todos los elementos que conforman el documento (<html>, <body>, <a>...).
- La estructura DOM o **diagrama en árbol** está compuesta por nodos que pueden ser padres, hijos o hermanos de otros nodos. Existen diferentes tipos de nodo según su contenido.

DOM: Document Object Model

Documento HTML

```
<html>
  <head>
    <title> Mi página </title>
  </head>
  <body>
    <h1> Este es el título </h1>
    <p> Este es el párrafo </p>
  </body>
</html>
```

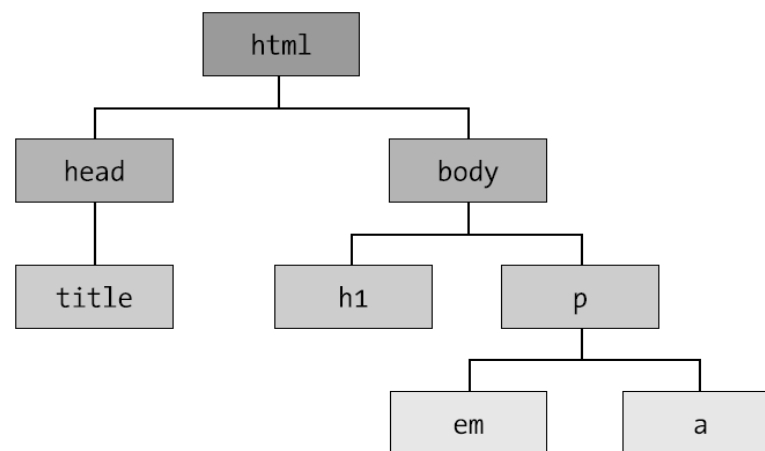
Presentación DOM



DOM: Document Object Model

Árbol DOM del documento

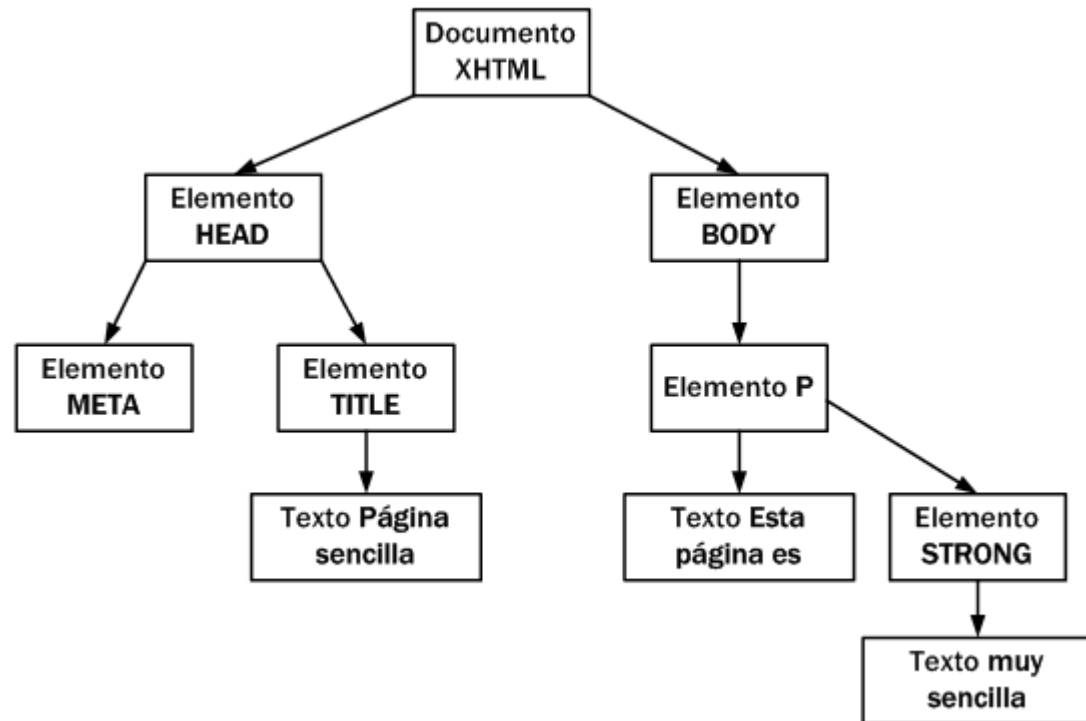
```
<html>
  <head>
    <title>DOM</title>
  </head>
  <body>
    <h1>DOM</h1>
    <p>
      Ejemplo <em>árbol DOM</em>
      <a href="#">Volver</a>
    </p>
  </body>
</html>
```



DOM

```
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1" />
    <title>Página sencilla</title>
  </head>
  <body>
    <p>Esta página es <strong>muy sencilla</strong></p>
  </body>
</html>
```

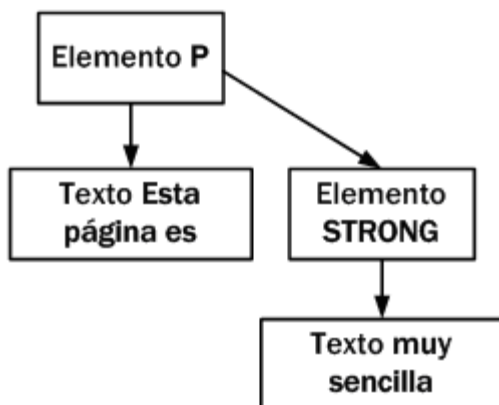
DOM



DOM

<p>Esta página es muy sencilla</p>

Genera los siguientes nodos:



Nodo de tipo "Elemento" correspondiente a la etiqueta <p>. Nodo de tipo "Texto" con el contenido textual de la etiqueta <p>. Como el contenido de <p> incluye en su interior otra etiqueta HTML, la etiqueta interior se transforma en un nodo de tipo "Elemento" que representa la etiqueta y que deriva del nodo anterior.

El contenido de la etiqueta genera a su vez otro nodo de tipo "Texto" que deriva del nodo generado por .

DOM

La especificación completa de DOM define 12 tipos de nodos, aunque las páginas HTML habituales se pueden manipular manejando solamente cuatro o cinco tipos de nodos:

- Document, nodo raíz del que derivan todos los demás nodos del árbol.
- Element, representa cada una de las etiquetas HTML. Se trata del único nodo que puede contener atributos y el único del que pueden derivar otros nodos.
- Attr, se define un nodo de este tipo para representar cada uno de los atributos de las etiquetas HTML, es decir, uno por cada par atributo=valor.
- Text, nodo que contiene el texto encerrado por una etiqueta HTML.
- Comment, representa los comentarios incluidos en la página HTML.

DOM: Document Object Model

Objetos predefinidos

- Un documento HTML tiene objetos JavaScript predefinidos, que pueden ser manipulados de forma programática con JavaScript y la **interfaz de programación de aplicaciones (DOM API)**.

window: referencia a la ventana propiamente tal. Todos los elementos pertenecen implícitamente al objeto window. Si se abre una nueva ventana o pestaña, corresponde a otro window.

Por ejemplo, el método `alert(...)` pertenece al objeto window

document: es la raíz del documento HTML, y el "padre" de todos los elementos. Provee atributos y métodos para acceder a los elementos de la página, lo que permite modificarlos dinámicamente, y de este modo realizar los efectos interactivos.

Por ejemplo, `document.write()` imprime texto

DOM: Document Object Model

Objetos predefinidos

- **navigator**: contiene información sobre el navegador. Permite particularizar la programación, lo que es útil cuando existen diferencias entre los navegadores. También permite restringir una aplicación a un tipo o versión de navegador.
- **location**: contiene información sobre la URL de la página. Si se modifica, se abre una nueva URL, lo que permite utilizarlo para navegar programáticamente.
- **screen**: contiene información de la pantalla, incluyendo alto y ancho. Permite ajustar una página en función de la resolución de pantalla.
- **history**: contiene el historial de URLs visitadas, lo que permite volver atrás programáticamente.

Casos de usos del lenguaje JavaScript

- Modificar el contenido **HTML** y los estilos **CSS**, a través de la utilización del **API DOM**.
- Controlar la **visibilidad** de una sección de la página, lo que permite hacer efectos como pestañas, menús o acordeón.
- Cambio dinámico de **estilo css**, lo que permite por ejemplo mostrar selecciones, o efectos de habilitación y deshabilitación.
- Crear **controles personalizados** que no existen nativamente en HTML, como barras de progreso.
- **Actualizar** secciones del documento pantalla, como por ejemplo una de noticias, sin tener que recargar la página.

DOM - Funciones Javascript

```
//Obtener elementos P  
var parrafos =  
document.getElementsByTagName("p");
```

```
//Obtener el primer párrafo  
var primerParrafo = parrafos[0];
```

```
//Obtener y tratar todos los párrafos  
for(var i=0; i<parrafos.length; i++) {  
    var parrafo = parrafos[i];  
}
```

```
//Obtener nodos por Id  
document.getElementById("demo")
```

```
//Contenido HTML  
document.getElementById("demo").innerHTML
```

DOM - Funciones Javascript

```
<html>
<body>
<p>An unordered list:</p>
<ul>
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ul>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction() {
  var x = document.getElementsByTagName("LI");
  document.getElementById("demo").innerHTML =
x[1].innerHTML;
}
</script>
</body>
</html>
```



Ejercicios JS15-a

A partir del ejemplo anterior definir una lista de bebidas ordenadas y numeradas. Solicitar al usuario que introduzca el número de bebida que quiere seleccionar y mostrar ese dato en pantalla.

DOM Crear Nodos

```
// Crear nodo de tipo Element
var parrafo = document.createElement("p");

// Crear nodo de tipo Text
var contenido = document.createTextNode("Hola
Mundo!");

// Añadir el nodo Text como hijo del nodo Element
parrafo.appendChild(contenido);

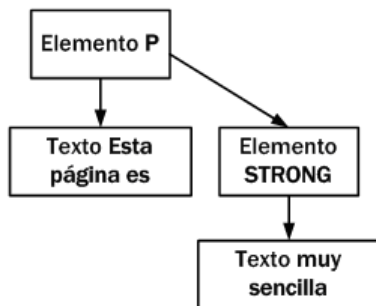
// Añadir el nodo Element como hijo de la pagina
document.body.appendChild(parrafo);
```



Ejercicios JS15-b

A partir del ejemplo anterior definir una lista de bebidas ordenadas y numeradas. Solicitar al usuario que introduzca el nombre de una nueva bebida y mostrar ese dato en pantalla añadiéndolo a la lista de bebidas.

DOM Eliminar Nodos



```
var negrita =  
document.getElementById("provisional");  
negrita.parentNode.removeChild(negrita);
```

```
<p>Esta página es <strong  
id="provisional">muy  
sencilla</strong></p>
```

Ejercicios JS15-c

A partir del ejemplo anterior definir una lista de bebidas ordenadas y numeradas. Solicitar al usuario que introduzca el nombre de una nueva bebida y mostrar ese dato en pantalla añadiéndolo a la lista de bebidas.

Además solicitar al usuario el número de una bebida a eliminar y eliminarla.

Interfaz de programación de aplicaciones (DOM API)

Cambio de contenido

- Para cambiar el contenido HTML de una parte de una pantalla en forma dinámica, se utiliza la propiedad **innerHTML**. Aunque no es parte del estándar, todos los navegadores conocidos manejan la propiedad. Por ejemplo, se puede modificar el contenido de un elemento.

Dynamic

Valor aleatorio: 4



Cuando se presiona el botón,
cambia dinámicamente el valor

Cambiar



Interfaz de programación de aplicaciones (DOM API)

Cambio de contenido

- Código HTML y JavaScript:

```
<h1>Dynamic</h1>
<p id="par">
  Valor aleatorio: 0
</p>
```

```
<input type="button" value="Cambiar"
onclick="change()">
```

Al presionar el botón, se invoca la función change()

```
<script>
  function change() {
    var p = document.getElementById("par");
    var num = 1 + Math.floor(9*Math.random());
    p.innerHTML = "Valor aleatorio: " + num;
  }
</script>
```

Obtiene el objeto asociado al párrafo, con el id="par"

Genera un número aleatorio entre 1 y 9

Cambia el contenido del objeto p, que corresponde al párrafo

Interfaz de programación de aplicaciones (DOM API)

Cambio de regla de estilo

- Podemos cambiar dinámicamente la clase de estilo de una sección de pantalla. Por ejemplo, un párrafo tiene un estilo normal y uno destacado, y se quiere que al pasar el puntero sobre él, cambie.

Párrafo que cambia
al pasar el mouse.



**Párrafo que cambia
al pasar el mouse**



```
.sec {  
  font-family: Courier;  
  border: 1px solid blue;  
  padding: 3px;  
  cursor: default;  
  width: 200px;  
}  
  
.pnormal {  
  font-weight: normal;  
}  
  
.phigh {  
  font-weight: bold;  
}
```



Interfaz de programación de aplicaciones (DOM API)

Cambio de regla de estilo

- Código HTML y JavaScript:

```
<div class="sec"  
  onmouseover="change(true)"  
  onmouseout="change(false)">  
  <p id="par" class="pnormal">  
    Párrafo que cambia al pasar el mouse.  
  </p>  
</div>
```

Cuando se pasa el mouse sobre el área del div, se invoca la función change(true)

Cuando se saca el mouse del área del div, se invoca la función change(false)

```
<script>  
  function change(opt) {  
    var p = document.getElementById("par");  
    p.className = (opt) ? "phigh" : "pnormal";  
  }  
</script>
```

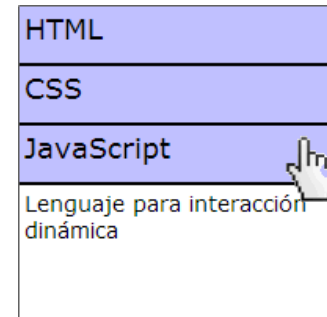
Obtiene el objeto asociado al párrafo, con el id="par"

Cambia dinámicamente la clase de estilo del párrafo, según el parámetro

Interfaz de programación de aplicaciones (DOM API)

Control de visibilidad

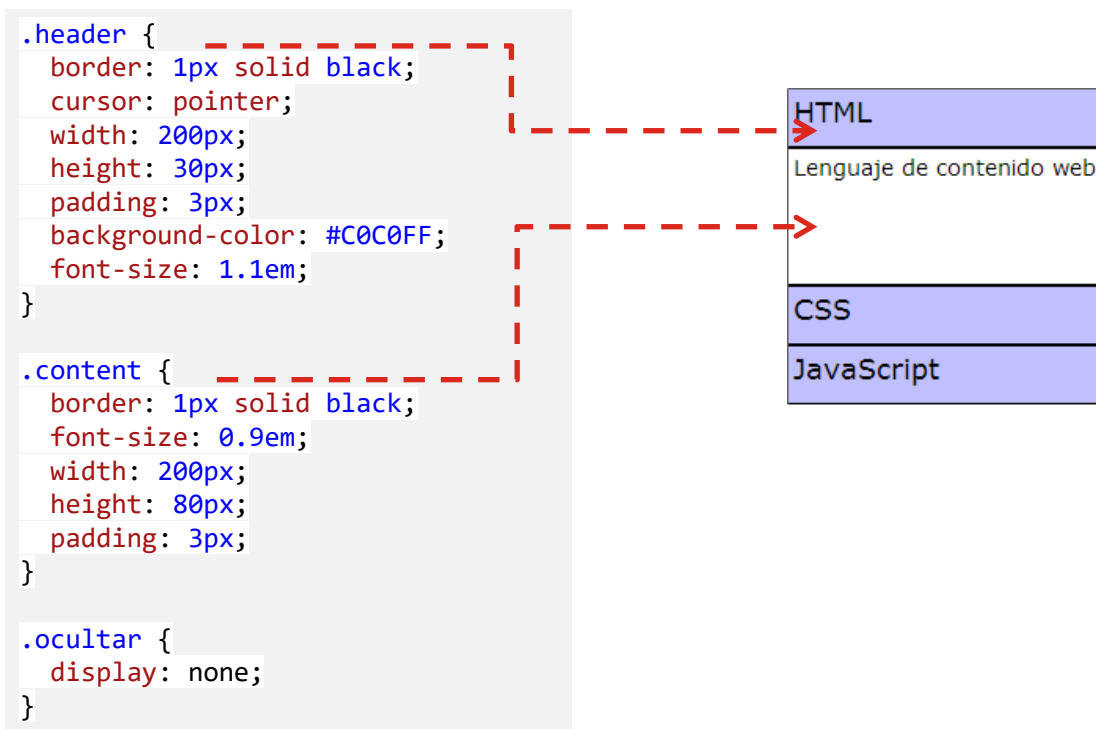
- Para controlar la **visibilidad** de una parte de una página, se controla el atributo de estilo llamado **display** asociado a la rama del árbol DOM correspondiente.



Interfaz de programación de aplicaciones (DOM API)

Control de visibilidad

- Estilos:





Interfaz de programación de aplicaciones (DOM API)

Control de visibilidad

- Elementos HTML:

Al hacer click sobre el div, se ejecuta el método JavaScript "visib"

```
<div class="header" onclick="visib('ht')">  
  HTML  
</div>  
<div class="content" id="ht">  
  Lenguaje de contenido web  
</div>  
<div class="header" onclick="visib('cs')">  
  CSS  
</div>  
<div class="content" id="cs" style="display: none">  
  Estilos gráficos  
</div>  
<div class="header" onclick="visib('js')">  
  JavaScript  
</div>  
<div class="content" id="js" style="display: none">  
  Lenguaje para interacción dinámica  
</div>
```

HTML
Lenguaje de contenido web
CSS
JavaScript

Inicialmente oculta el contenido del div y el espacio que utiliza

Interfaz de programación de aplicaciones (DOM API)

Control de visibilidad

- Función JavaScript:

```
function visib(divId) {  
  var ar = ["ht", "cs", "js"];  
  for (var i=0; i < ar.length; i++) {  
    var divElem = document.getElementById(ar[i]);  
    if (divElem == ar[i]) {  
      divElem.classList.remove('ocultar')  
    } else {  
      divElem.classList.add('ocultar')  
    }  
  }  
};
```

array con ids de
los contenidos

Accede al objeto del
árbol DOM con el
contenido, dado el id.

Cambia el atributo de estilos
display, en función del parámetro:
• remove: mostrar contenido
• add: ocultar contenido

divId = "ht"



HTML
Lenguaje de contenido web
CSS
JavaScript

divId = "cs"

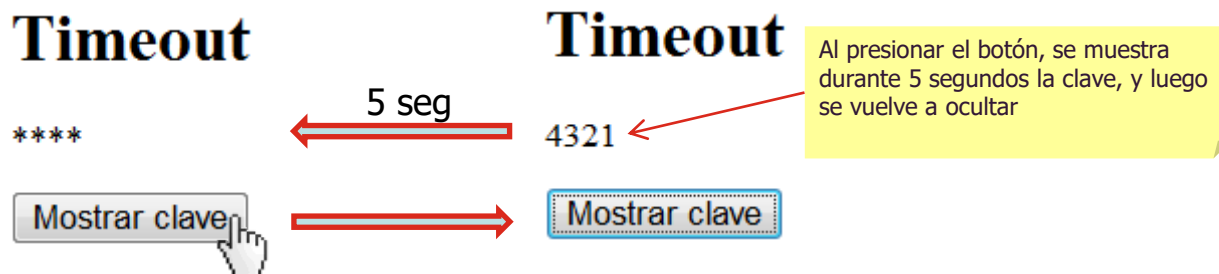


HTML
CSS
Estilos gráficos
JavaScript

Interfaz de programación de aplicaciones (DOM API)

Timeout

- A través del método setTimeout del objeto implícito window, se pueden ejecutar funciones un tiempo después. Esto permite controlar efectos en el tiempo. Por ejemplo, una página que muestra una clave durante 5 segundos, y luego la oculta.
- Resultado:



Interfaz de programación de aplicaciones (DOM API)

Cambio de contenido

- Código HTML y JavaScript:

```
<h1>Timeout</h1>
<p id="pwd">****</p>
<input type="button" value="Mostrar clave" onclick="showPwd()">
```

Al presionar el botón se invoca la función manejadora showPwd()

```
<script>
function showPwd() {
    var pwd = "4321";
    document.getElementById("pwd").innerHTML = pwd;
    setTimeout(hidePwd, 5000);
}

function hidePwd() {
    document.getElementById("pwd").innerHTML = "****";
}
</script>
```

Cambia el contenido del objeto con id="pwd", que corresponde al párrafo que muestra la clave

5000 milisegundos después, invoca la función hidePwd. Se coloca la referencia, sin paréntesis

Vuelve a colocar los **** en el contenido del párrafo con la clave

DOM Acceso atributos

```
<a id="enlace" href="http://www...com">Enlace</a>
```

```
var enlace = document.getElementById("enlace");  
alert(enlace.href); // muestra http://www...com
```

```

```

```
var imagen = document.getElementById("imagen");  
alert(imagen.style.margin);
```

DOM Acceso atributos

```
<p id="parrafo" style="font-weight: bold;">...</p>
```

```
var parrafo = document.getElementById("parrafo");  
alert(parrafo.style.fontWeight); // muestra "bold"
```

- font-weight se transforma en fontWeight
- line-height se transforma en lineHeight
- border-top-style se transforma en borderTopStyle
- list-style-image se transforma en listStyleImage



DOM Acceso atributos

```
<p id="parrafo" class="normal">...</p>
```

```
var parrafo = document.getElementById("parrafo");  
alert(parrafo.class); // muestra "undefined"  
alert(parrafo.className); // muestra "normal"
```

DOM Modificar atributos

```
document.body.style.fontSize="24px"
```

```
document.getElementById("parrafo").className="Clase_Nueva"  
;
```



DOM Modificar atributos

Window.getComputedStyle

Representa el valor final de la propiedad CSS del elemento

```
var style = window.getComputedStyle(element, pseudoElt);
```

```
var x = document.getElementsByTagName("h1");  
alert(window.getComputedStyle(x[0],null).getPropertyValue('font-size'));
```

<http://md360.es/sas/accesoestilos.html>

Ejercicio JS16

Crear una página web con texto y crear dos botones “+” y “-” . Cuando se aprieta en “+” se aumenta la fuente que se muestra y cuando se hace en “-” al revés. La página debe contener un texto en el body y un h1.

Ejercicio JS17

A partir de la página web proporcionada y utilizando las funciones DOM, mostrar por pantalla la siguiente información:

Número de enlaces de la página

Dirección a la que enlaza el penúltimo enlace

Numero de enlaces que enlazan a <http://prueba>

Número de enlaces del tercer párrafo

Página web base: <http://md360.es/sas/js17.html>

Casos de uso JavaScript

Geo-Localización

- Con `navigator.geolocation`, controlamos el soporte, a la vez que programamos las peticiones y respuestas.
- Una llamada a este objeto desde JavaScript, nos devolverá un valor falso si no existe.
- Para programar el funcionamiento para un navegador podríamos usar:

```
<head>
  <meta charset="utf-8" />
  <title>Geo-localización nativa con HTML 5</title>
</head>
<body onload="comprobarNavegador()">
  <h1>Ejemplo de Geo-localización nativa con HTML 5</h1>
  <p id="nivelSoporte">La Geo-localización nativa NO está soportada en este
navegador.</p>
  <h2>Ubicación actual:</h2>
  <h5>Latitud: <span id="latitud">n/c</span></h5>
  <h5>Longitud: <span id="longitud">n/c</span></h5>
  <h5>Precisión: <span id="precision">n/c</span></h5>
```

Casos de uso JavaScript

Geo-Localización

- Primero, comprobamos el soporte en el navegador llamando a la función de geolocalización:

```
<script>
function comprobarNavegador() {
    if(navigator.geolocation) {
        document.getElementById("nivelSoporte").innerHTML =
            "La Geo-Localización HTML5 está soportada en este navegador.";
        navigator.geolocation.getCurrentPosition(updateLocation);
    }
}
```

- A continuación, consultamos los datos geográficos.

Casos de uso JavaScript

Geo-Localización

```
function updateLocation(position) {  
    var lat = position.coords.latitude;  
    var lon = position.coords.longitude;  
    var pre = position.coords.accuracy;  
    if (!lat || !lon) {  
        document.getElementById("nivelSoporte").innerHTML =  
        "Geo-Localizacion HTML5 soportada, pero no en este momento.";  
        return;  
    }  
    document.getElementById("latitud").innerHTML = lat;  
    document.getElementById("longitud").innerHTML = lon;  
    document.getElementById("precision").innerHTML= pre + " ms.";  
}  
</script>  
</body>  
</html>
```



JavaScript (ES6)

2. Aplicación de los elementos más avanzados de JavaScript

- Herencia basada en prototipos
- Herencia en ES6
- Programación funcional: map, filter y reduce
- Módulos y elementos introducidos en ES6

Objetos

POO (Programación Orientada a Objetos):
Paradigma de programación en la que todo se representa como una entidad u objeto de la vida real o imaginario.

En su definición se especifican:

- **Propiedades:** características que distinguen a los objetos del mismo tipo o **Clase**.
- **Métodos:** funcionalidades del objeto. Tareas que se pueden realizar con las propiedades de un objeto.



- Superclase: Teléfonos
- Clase: Teléfono celular
- Subclase: SmartPhone

Propiedades o atributos

- Tipo de pantalla
- Espacio de memoria
- Cantidad de tonos
- Tipo de antena
- Cantidad de idiomas
- Otros

Métodos o comportamientos

- Iniciar alarma
- Asignar tonos
- Registrar llamadas
- Iniciar juego
- Utilizar calculadora
- Enviar mensajes
- Otros

"La orientación a objetos será la más importante de las tecnologías que surjan en los años noventa"
Bill Gates

Clase o Tipo

- Una **clase** especifica qué propiedades y métodos caracterizan a sus objetos, pero no asigna valores a sus miembros.

- Automóvil
 - Marca
 - Modelo
- Persona
 - Nombre
 - Edad
- Estudiante
 - Nivel
 - Curso
 - Especialidad

Ejemplos de clases
y sus propiedades



Persona

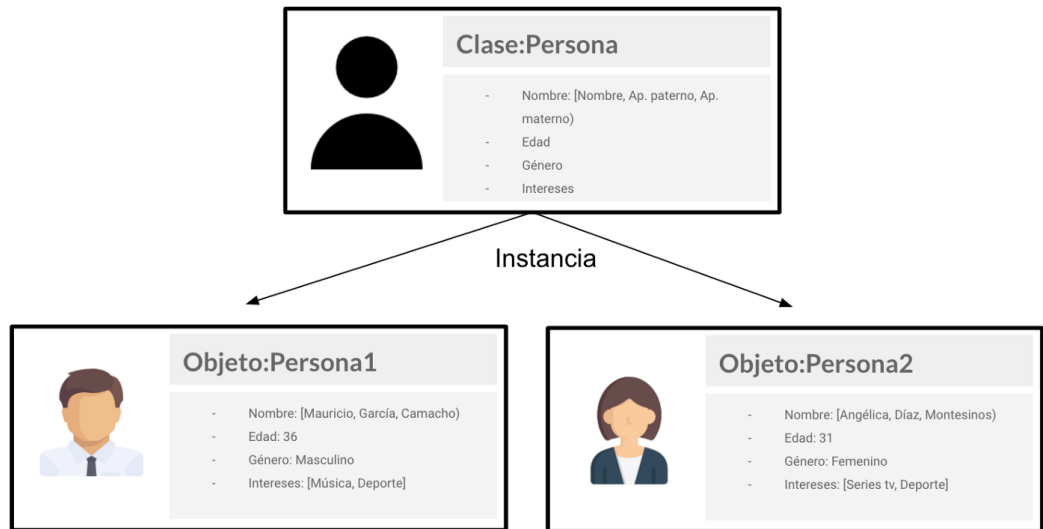
- Nombre (Nombre, apellido paterno, apellido materno)
- Edad
- Género
- Intereses

Objeto o Instancia

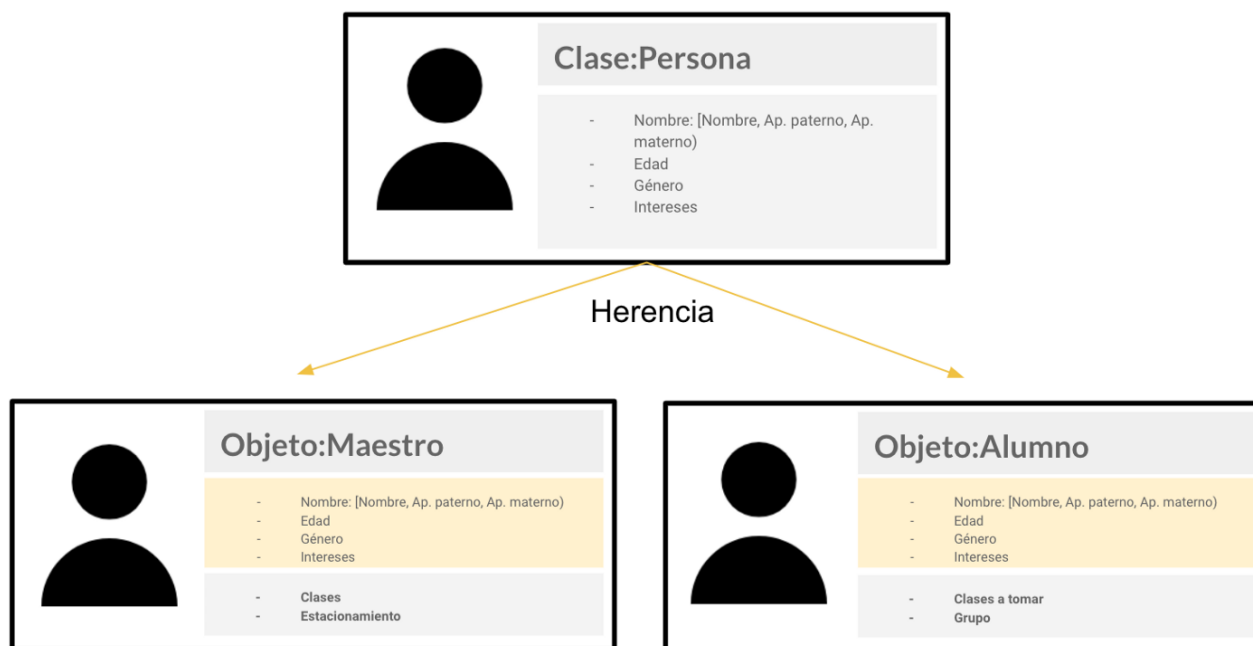
- Un **objeto** es una instancia de clase definida, con valores en su propiedades o atributos.

- persona
 - Nombre: Luis
 - Edad: 36
- Alumno
 - Nombre: Juan
 - Edad: 19
 - Carrera: Diseño Gráfico

Ejemplos de objetos y sus estados
(valores de sus propiedades)



Herencia y Jerarquía de clases





Sintaxis básica JavaScript

Objetos

- Definición de un objeto:

```
var person = {  
  firstName: "John",  
  lastName : "Denver",  
  id       : 12345,  
  fullName : function() {  
    return this.firstName + " " + this.lastName;  
  }  
};
```

Propiedades del objeto

Método del objeto

{JSON}

```
{  
  "id" :1,  
  "name" : "Me",  
  "email" : me@gmail.com  
}
```

Sintaxis básica JavaScript

Objetos

- Formas de acceder a **propiedades** y **métodos** de un objeto:

```
// person.id  
nombreObjeto.nombrePropiedad;  
  
// person["id"]  
nombreObjeto["nombrePropiedad"]  
  
// person.fullName()  
nombreObjeto.nombreMetodo();
```



JavaScript ES6

Clases

- Nueva sintaxis para escribir clases en JavaScript
- Declaración explícita de clases con **class**

```
1 // constructor
2 function Product(name, year, price) {
3   this.name = name;
4   this.year = year;
5   this.price = price;
6 }
7
8 // método
9 Product.prototype.generateBarcode = function() {
10  console.log('Método generateBarcode');
11 }
12
13 // instancia clase
14 var iphone = new Product('iPhone', 2018, 999);
15
16 // llamada método
17 iphone.generateBarcode(); // Método generateBarcode
18 console.log(iphone.name); // iPhone
```

Declaración de
clases en JS
tradicional

```
1 // clase
2 class Product {
3   // constructor
4   constructor(name, year, price) {
5     this.name = name;
6     this.year = year;
7     this.price = price;
8   }
9
10  // método
11  generateBarcode() {
12    console.log('Método generateBarcode');
13  }
14 }
15
16 // instancia clase
17 let iphone = new Product('iPhone', 2018, 999);
18
19 // llamada método
20 iphone.generateBarcode(); // Método generateBarcode
21 console.log(iphone.name); // iPhone
```

Declaración de
clases en ES6

JavaScript ES6

Herencia

- Utilizar **extends** después de la declaración de la clase (class) e indicar el nombre de la clase de la cual queremos que herede todos los métodos y propiedades:

```
1 // constructor
2 function Product(name, year, price) {
3   this.name = name;
4   this.year = year;
5   this.price = price;
6 }
7
8 // método
9 Product.prototype.generateBarcode = function() {
10  console.log('Método generateBarcode');
11 }
12
13 // constructor
14 function Smartphone(name, year, price, os) {
15   Product.call(this, name, year, price);
16   this.os = os;
17 }
18
19 // herencia
20 Smartphone.prototype = Object.create(Product.prototype);
21 Smartphone.prototype.constructor = Smartphone;
22
23
24 // instancia clase
25 var iphone = new Smartphone('iPhone', 2018, 999, 'iOS');
26
27 // llamada método
28 console.log(iphone.name); // iPhone
29 console.log(iphone.os); // iOS
30 iphone.generateBarcode(); // Método generateBarcode
```

Herencia con JS
tradicional



Herencia en ES6

```
1 // clase
2 class Product {
3   // constructor
4   constructor(name, year, price) {
5     this.name = name;
6     this.year = year;
7     this.price = price;
8   }
9
10  // getter
11  get price() {
12    return this._price;
13  }
14
15  // setter
16  set price(price) {
17    // https://blog.abelotech.com/posts/number-currency-formatting-javascript/
18    this._price = price.toFixed(2).replace('.', ',').replace(/(\d)(?=\d{3})+(!\d))/g, '$1.') + ' €';
19  }
20
21  // método
22  generateBarcode() {
23    console.log('Método generateBarcode');
24  }
25 }
26
27 // herencia
28 class Smartphone extends Product {
29   // constructor
30   constructor(name, year, price, os) {
31     super(name, year, price);
32     this.os = os;
33   }
34 }
35
```

Programación Funcional

Código funcional

- Es un sub-paradigma del paradigma declarativo, esto afecta la forma en que escribes código funcional. Generalmente conduce a menos código, porque JavaScript ya tiene muchas de las funciones integradas que normalmente necesitas
- Reglas del código funcional: Diseñar a partir de funciones puras y aisladas + Evitar mutabilidad y efectos secundarios

```
// Ejemplo no funcional
const edad = [12,32,32,53]
for (var i=0; i < edad.length; i++) {
    edadFinal += edad[i];
}

// Ejemplo funcional
const edad = [12,32,32,53]
const edadTotal = edad.reduce( function(primerEdad, segundaEdad){
    return primerEdad + segundaEdad;
})
```



Programación Funcional

map, reduce y filter

- Las funciones de arreglos incluidas en JavaScript `.map`, `.reduce` y `.filter` aceptan una función. Son ejemplos de funciones de orden superior, ya que iteran sobre un arreglo y llaman a la función que recibieron para cada elemento del arreglo.

```
// Ejemplos de cada uno
const arreglo = [1, 2, 3];

const arregloMap = arreglo.map(function(elemento){
  return elemento + 1;
});
// arregloMap es [2, 3, 4]

const reducido = arreglo.reduce(function(primer, segundo){
  return primer + segundo;
});
// reducido es 6

const filtrarArreglo = arreglo.filter(function(elemento){
  return elemento !== 1;
});
// filtrarArreglo is [2, 3]
```



JavaScript (ES6)

3. Interpretación y reescritura de llamadas, datos y código AJAX

- Callback
- Promises
- Programación asíncrona, HTTP, API REST y Fetch API

Callback

Funciones Callback

- Una **callback** es una función anónima que se le pasa como argumento a otra función y se ejecuta en un determinado momento (**asíncrona**).
- Utilidad: cuando no sabemos lo que tarda en responder una función con el valor de retorno.
- Casos de uso: programación asíncrona y reactiva, funciones de escritura en ficheros y en peticiones HTTP en servicios web de tipo REST.

```
function saludar(nombre, callback){  
  let saludo = 'hola ' + nombre  
  callback(saludo)  
}  
  
saludar('Joss', function(resultado){  
  console.log(resultado)  
})
```

Función principal

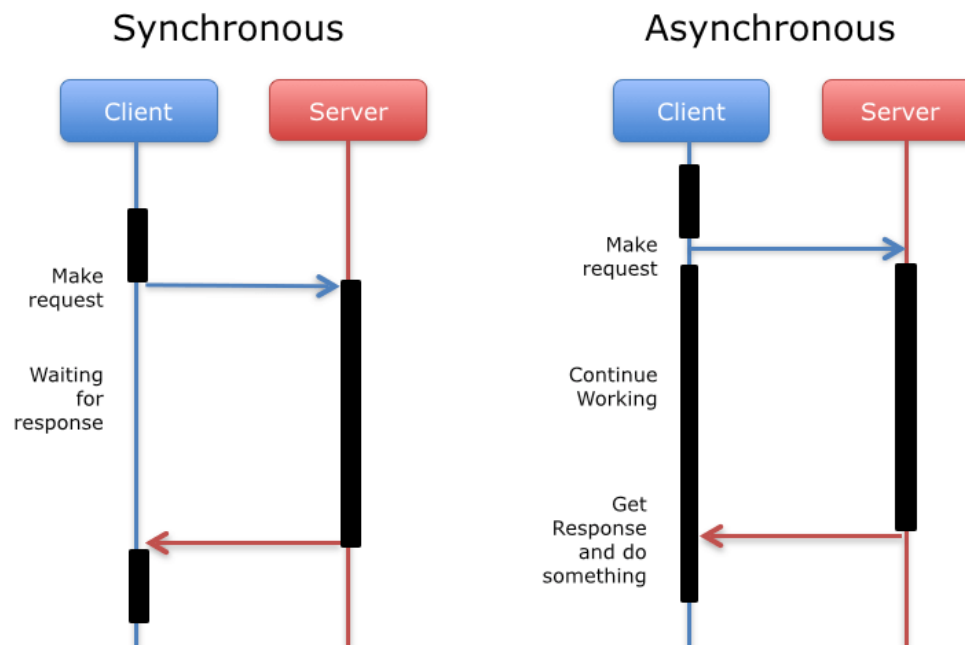
función a recibir como parámetro(callback)

Ejecución de la función (callback) con el valor generado

Ejecución de la función principal

Definición de la callback

Programación Asíncrona



Sintaxis básica JavaScript

JavaScript ES5: Closures

- En JavaScript toda variable declarada fuera de una función, pertenece a este objeto.
- Esto puede entrar en conflicto con otros elementos declarados en librerías de soporte cargadas por la página.
- Pero una variable declarada dentro de una función es local a la función...
- La solución se presenta en forma de una construcción llamada **closure**.

```
var nombre_numero_c = function () {  
    var numeros = ['cero', 'uno', 'dos', 'tres', 'cuatro'];  
    return function (n) {  
        return numeros[n];  
    };  
}();  
alert(nombre_numero_c(1));
```

Casos de uso JavaScript

Programación Reactiva

- Programación **Asíncrona**
- Considera el procesamiento de datos de manera asíncrona
- Procesamiento de datos = **Flujo de datos asíncrono** (Stream)
- Establece los mecanismos para manipular streams (**Observable** y **Observer**)
- Implementación **ReactiveX**: API for asynchronous programming with observable streams
- Soporte para distintos lenguajes de programación
- **RxJX** Reactive API para lenguaje JavaScript

Casos de uso JavaScript

API de AJAX con soporte de datos JSON

```
{ "id" :1,  
  "name" : "Me",  
  "email" : "me@gmail.com" }
```

{JSON}

- AJAX significa **Asynchronous JavaScript And XML**.
- La API de AJAX permite la ejecución de llamadas asíncronas al servidor, que devuelven datos para actualizar fragmentos de una página.
- Se basan en el objeto XMLHttpRequest.
- Existen dos versiones o niveles de llamada.
- El nivel 1, es prácticamente compatible con todos los navegadores, incluyendo los más antiguos.
- El nivel 2, ya disponible en todos los navegadores modernos, permite controlar el nivel de progreso de la llamada, con lo que se simplifica el código.
- Mediante este tipo de llamadas, podemos conseguir actualizaciones parciales de la página, logrando una experiencia de usuario mucho más agradable.

Casos de uso JavaScript

API de AJAX con soporte de datos JSON

- Hay que tener en cuenta que, por motivos de seguridad, este API genera un error en la respuesta cuando se intenta llamar a servicios de este tipo en un dominio distinto.
- En otros entornos, como Silverlight, los ficheros de servidor CrossDomain.xml, permiten establecer los permisos para este tipo de llamadas.
- Un objeto XMLHttpRequest puede devolver datos en formatos diversos, como XML, JSON, Texto plano, etc.
- La petición adopta un formato como el siguiente:

```
GET /recurso HTTP/1.1  
Host: host:puerto  
User-Agent: Mozilla/5.0
```

Casos de uso JavaScript

API de AJAX con soporte de datos JSON

```
function leerVentas() {  
    var url = "http://localhost:1565/DemosJS5/ventas.json";  
    var request = new XMLHttpRequest();  
    request.responseType = "application/json"; //Predeterminado  
    request.open("GET", url); //Solo configura la llamada  
    request.onload = function() {  
        if (request.status == 200) { //200 significa correcto.  
            actualizarIU(request.responseText); }  
    };  
    request.send(null);  
}
```

Casos de uso JavaScript

API de AJAX con soporte de datos JSON

```
function actualizarIU(respuestaJSON) {  
    var DivVentas = document.getElementById("DIV_ventas");  
    var ventas = JSON.parse(respuestaJSON);  
    for (var i = 0; i < ventas.length; i++) {  
        var venta = ventas[i];  
        var div = document.createElement("div");  
        div.setAttribute("class", "FormatoVenta");  
        div.innerHTML = "Total ventas de " + venta.name + ": " + venta.TVentas;  
        DivVentas.appendChild(div);  
    }  
}
```

Casos de uso JavaScript

API de AJAX con soporte de datos JSON

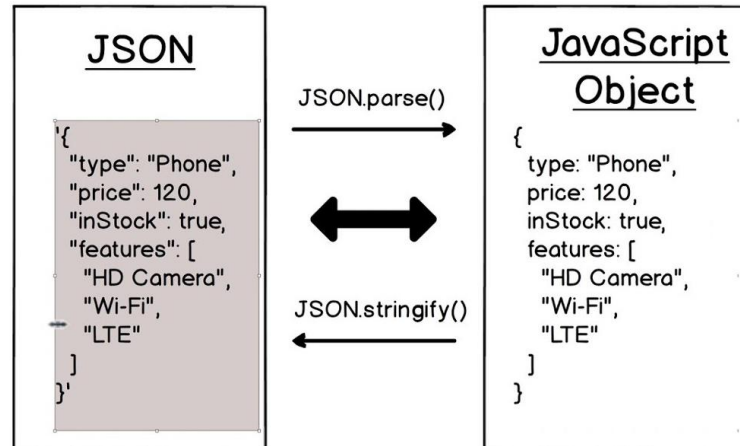
- En el caso de que tengamos que dar soporte a algunos navegadores muy antiguos deberíamos comprobar el evento `onreadystatechange`

```
function leerVentas_XHRv1() {  
    var url = "http://localhost:1565/DemosJS5/ventas.json";  
    var request = new XMLHttpRequest();  
    request.open("GET", url);  
    request.onreadystatechange = function() {  
        if (request.readyState == 4 && request.status == 200) {  
            actualizarIU(request.responseText);  
        }  
    };  
    request.send(null);  
}
```


Casos de uso JavaScript

API de AJAX con soporte de datos JSON

- Para la manipulación de datos JSON, las implementaciones de JavaScript de los navegadores actuales, incluyen un objeto del mismo nombre, que incorpora métodos útiles para facilitar su manejo.
 - stringify()** convierte un objeto JavaScript a cadena.
 - parse()** realiza la operación contraria.



Casos de uso JavaScript

API Storage: Almacenamiento local y de sesión

- El objetivo de estas API es la persistencia de información entre distintas peticiones Web.
- Almacenar datos entre peticiones, de una forma distinta y más adecuada a la que es posible hacer ahora mediante cookies.
- Estos servicios son accedidos mediante objetos vinculados con el objeto window:
window.sessionStorage y **window.localStorage**
- El funcionamiento es a través de una "base de datos" manejada internamente por el navegador y accedida mediante estas API.
- El primer paso es la comprobación del soporte por parte del navegador utilizado.



Casos de uso JavaScript

Almacenamiento local y de sesión

```
function comprobarSoporte() {  
    //sessionStorage  
    if (window.sessionStorage) {  
        alert('Este navegador soporta sessionStorage');  
    } else {  
        alert('Este navegador NO soporta sessionStorage');  
    }  
    //localStorage  
    if (window.localStorage) {  
        alert('Este navegador soporta localStorage');  
    } else {  
        alert('Este navegador NO soporta localStorage');  
    }  
}
```

Casos de uso JavaScript

Almacenamiento local y de sesión

- **sessionStorage** y **localStorage** disponen de varios métodos para asignación (escritura) y recuperación (lectura) de información.
- En la actualidad solo encontramos soporte para cadenas.
- La instrucción de escritura utilizada es **setItem(clave, valor)**.
- Recibe parejas clave/valor y las almacena para recuperación posterior en la base de datos asignada (al navegador).
- La lectura se realiza mediante el método **getItem(clave)** que permite recuperar cualquier valor guardado a partir de su clave y la devuelve en formato cadena.

Casos de uso JavaScript

Almacenamiento local y de sesión

```
<body>
  <form action="valida.html" id="Formulario" >
    <p>Almacenamiento de valores de sesión</p>
    <input type="text" id="claveini" />
    <input type="button" value="Guardar"
      onclick="escribirClave('Dato');" /><br />
      <input type="text" id="claveLeida" />
    <input type="button" value="Leer" onclick="leerClave('Dato');"
    />
  </form>
</body>
```

Casos de uso JavaScript

Almacenamiento local y de sesión

```
function escribirClave(datos) {  
    var valor = document.getElementById("claveini").value;  
    window.sessionStorage.setItem(datos, valor);  
}  
  
function leerClave(datos) {  
    var valor = window.sessionStorage.getItem(datos);  
    document.getElementById("claveLeida").value = valor;  
}
```

Casos de uso JavaScript

Almacenamiento local y de sesión

- También existen limitaciones a la cantidad de información que puede guardarse en el apartado valor vinculado con una clave dada.
- Existen unos valores de cuota que deberán de ser configurables por el usuario:

sessionStorage	localStorage
La persistencia se limita a la página o solapa de navegación dentro del sitio	La persistencia se mantiene incluso después de cerrar el navegador
Los valores almacenados o recuperados solo son visibles desde la página o solapa que los creó.	Los valores se mantienen entre distintas ventanas o solapas ejecutando el mismo origen URL.

localStorage vs sessionStorage

localStorage	sessionStorage
Stores data with no expiration date	Stores data only for a session (until the tab/browser is closed)
Gets cleared through JS or Browser cache / Locally Stored Data	Storage limit is larger than a cookie (at least 5 MB). Max 4KB
Changes available for all current and future visits to the site	Persists over page reloads and restores. Opening a new tab/window will initiate a new session.
	Data is never transferred to the server*
<ul style="list-style-type: none">▪ Both extend Storage▪ Both can only be read on client-side▪ Web storage is per origin (per domain and protocol)▪ Both allow to storage JS primitive types (integers, strings, ...) but nor arrays neither objects → we must convert them to JSON	

localStorage methods

Method	Description
<code>setItem()</code>	Add key and value to local storage
<code>getItem()</code>	Retrieve a value by the key
<code>removeItem()</code>	Remove an ítem by key
<code>clear()</code>	Clear all storage
<code>localStorage.length</code>	Nº de elementos almacenados en el espacio local.



HTML Web Storage Objects

Check browser **support for localStorage and sessionStorage:**

```
if (typeof(Storage) !== "undefined") {  
    // Code for localStorage/sessionStorage.  
} else {  
    // Sorry! No Web Storage support...  
}
```

IndexedDB API

IndexedDB persistently allows to store objects indexed with a "key"

- It's an asynchronous API
- IndexedDB is object-oriented
- IndexedDB databases store key-value pairs
- It's build on a transactional database model
- It does not use SQL

Casos de uso JavaScript

API para acceso a archivos locales (File API)

- Podemos programar una sencilla interfaz que habilite la selección de uno o más archivos por parte del usuario y muestre posteriormente la información de detalles de esos archivos en la página.
- En la parte HTML 5 tendríamos:

```
<!DOCTYPE html >
<html>
<head>
  <title>API de ficheros</title>
</head>
<body>

  <h3>Seleccionar fichero(s)</h3>
  <p><input id="files-upload" type="file" multiple /></p>
  <h3>Ficheros subidos</h3>
  <ul id="file-list"><li>(no hay ficheros todavía)</li></ul>
```

Casos de uso JavaScript

API para acceso a archivos locales (File API)

```
<script>
(function () {
// Recuperamos las referencias a los dos elementos implicados
var filesUpload = document.getElementById("files-upload"),
    fileList = document.getElementById("file-list");

// Función para recorrer el conjunto de archivos
// y actualizar la interfaz de usuario con la información
// sobre ellos.
function traverseFiles(files) {
    var li,file,fileInfo;
    fileList.innerHTML = "";
```

Casos de uso JavaScript

API para acceso a archivos locales (File API)

```
for (var i = 0, il = files.length; i < il; i++) {  
    li = document.createElement("li");  
    file = files[i];  
    fileInfo = "<div><strong>Fichero:</strong> " + file.name + "</div>";  
    fileInfo += "<div><strong>Tamaño:</strong> " + file.size + " bytes</div>";  
    fileInfo += "<div><strong>Tipo:</strong> " + file.type + "</div>";  
    li.innerHTML = fileInfo;  
    fileList.appendChild(li);  
};  
};  
filesUpload.onChange = function () {  
    traverseFiles(this.files);  
};  
})();
```



JavaScript (ES6)

5. Manipulación de código de terceros

6. Elaboración de documentación técnica y de usuario de lenguaje Javascript

- Documentación destinada al usuario final de una aplicación
- Documentación técnica destinada a programadores

Documentación de aplicaciones web

¿Para qué documentar? / Beneficios

- Evitar problemas por falta de planificación y diseño previo del componente / API
- Evitar inconsistencia entre los objetos y métodos
- Evitar agujeros de seguridad
- Uso de herramientas que tienen en cuenta la usabilidad y necesidades de los consumidores/aplicaciones que van a utilizar los servicios
- Realizar mocks testeables
- Posibilitar el versionado
- Crear de forma conjunta al desarrollo de la documentación

Herramientas de documentación técnica

- La documentación técnica va en paralelo con el proceso de desarrollo y no es necesario esperar a que termine todo el trabajo.



- **Javadoc:** generador de documentación creado por Sun Microsystems para el lenguaje Java para generar documentación de API en formato HTML desde el código fuente de Java.
- **Markdown:** lenguaje de marcado suave con sintaxis de formato de texto plano. Permite convertir a HTML y muchos otros formatos utilizando una herramienta con el mismo nombre. Markdown se usa a menudo para formatear archivos "readme", para escribir mensajes en foros de discusión online y para crear texto usando un editor de texto simple.
- **Swagger:** ayuda en el desarrollo de todo el ciclo de vida de la API, desde el diseño y la documentación, hasta la prueba y la implementación.
- **RAML:** lenguaje de definición de APIs que permite escribir su especificación siguiendo un estándar.

Herramientas para la creación y diseño de APIs

RAML (RESTful API Modeling Language)

- Lenguaje de definición de APIs que permite escribir su especificación siguiendo un estándar.
- Lenguaje de modelado para definir APIs REST de sintaxis sencilla y fácilmente comprensibles para seres humanos y software.
- Especificación no propietaria e independiente basada en YAML y JSON.
- Permite definir versión, recursos, métodos, parámetros de URL, seguridad respuestas, tipos de medios y otros componentes HTTP básicos.
- Genera la documentación de la API, casos de prueba, implementa un mock para acelerar el desarrollo y genera el esqueleto de nuestra aplicación.
- Permite definir las respuestas y ejemplos escritos en la especificación como documentación.



For every API, start by defining which version of RAML you are using, and then document basic characteristics of your API - the title, baseURI, and version.

Create and pull in namespaced, reusable libraries containing data types, traits, resource types, schemas, examples, & more.

Annotations let you add vendor specific functionality without compromising your spec

Traits and resourceTypes let you take advantage of code reuse and design patterns

Easily define resources and methods, then add as much detail as you want. Apply traits and other patterns, or add parameters and other details specific to each call.

Describe expected responses for multiple media types and specify data types or call in pre-defined schemas and examples. Schemas and examples can be defined via a data type, in-line, or externalized with !include.

Write human-readable, markdown-formatted descriptions throughout your RAML spec, or include entire markdown documentation sections at the root.

```

1  #%RAML 1.0
2  title: World Music API
3  baseUri: http://example.api.com/{version}
4  version: v1
5
6  uses:
7    Songs: libraries/songs.raml
8
9  annotationTypes:
10    monitoringInterval:
11      parameters:
12        value: integer
13
14  traits:
15    secured: !include secured/accessToken.raml
16
17  /songs:
18    is: [ secured ]
19    get:
20      (monitoringInterval): 30
21      queryParameters:
22        genre:
23          description: filter the songs by genre
24    post:
25      /{songId}:
26        get:
27          responses:
28            200:
29              body:
30                application/json:
31                  type: Songs.Song
32                application/xml:
33                  schema: !include schemas/songs.xml
34                  example: !include examples/songs.xml

```

Songs Library

```

1  #%RAML 1.0 Library
2  types:
3    Song:
4      properties:
5        title: string
6        length: number
7    Album:
8      properties:
9        title: string
10       songs: Song[]
11    Musician:
12      properties:
13        name: string
14       discography: (Song | Album)[]

```

songs.xml

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
3    elementFormDefault="qualified" attributeFormDefault="unqualified">
4    <xs:element name="song">
5      <xs:complexType>
6        <xs:sequence>
7          <xs:element name="title" type="xs:string">
8            </xs:element>
9          <xs:element name="artist" type="xs:string">
10            </xs:element>
11          </xs:sequence>
12        </xs:complexType>
13      </xs:element>
14    </xs:schema>

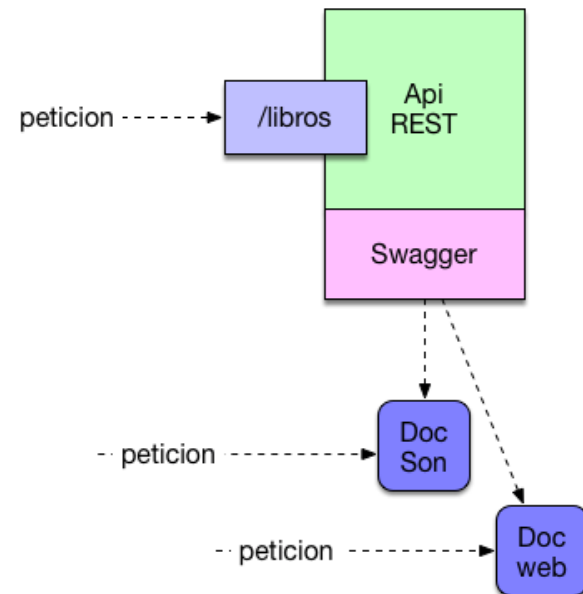
```



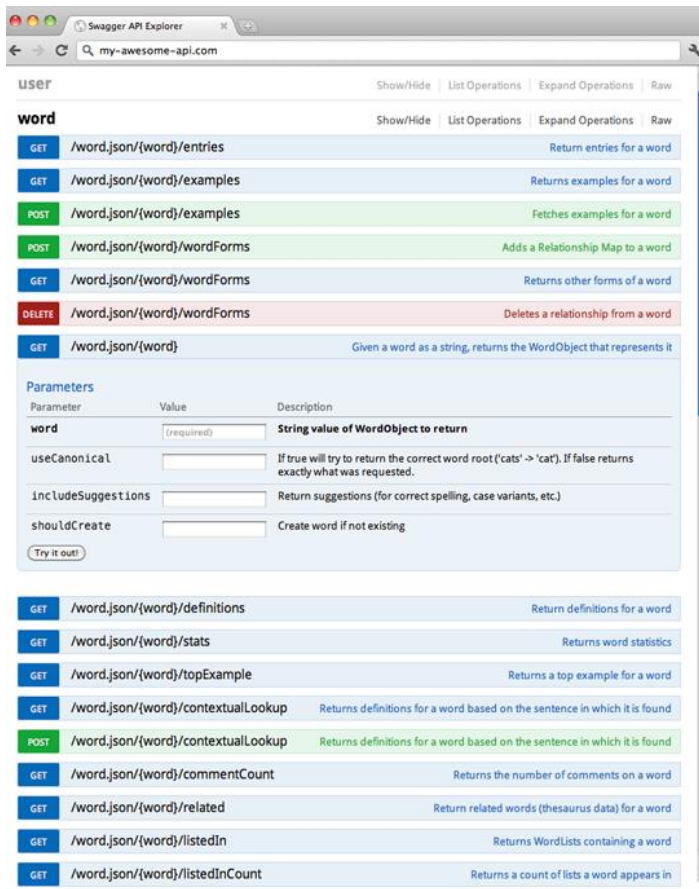
Herramientas para la creación y diseño de APIs

Swagger

- Permite describir, producir, consumir y visualizar APIs RESTful.
- Framework open source para generar documentación de APIs RESTful.
- Interfaz visual a modo de sandbox para testear llamadas al API, y consulta de su documentación en el navegador.
- Código del servidor sincronizado automáticamente con la documentación generada.
- Generación de documentación para Java, Javascript, Ruby, PHP, Scala, ActionScript y su sandbox correspondiente.



Swagger



The screenshot shows the Swagger API Explorer interface for the API 'my-awesome-api.com'. The interface is organized into sections for different API endpoints, each with a list of operations (GET, POST, DELETE) and their descriptions. The 'word' section is expanded, showing a list of operations and a detailed view of the 'GET /word.json/{word}' endpoint.

word

- GET /word.json/{word}/entries: Return entries for a word
- GET /word.json/{word}/examples: Returns examples for a word
- POST /word.json/{word}/examples: Fetches examples for a word
- POST /word.json/{word}/wordForms: Adds a Relationship Map to a word
- GET /word.json/{word}/wordForms: Returns other forms of a word
- DELETE /word.json/{word}/wordForms: Deletes a relationship from a word
- GET /word.json/{word}: Given a word as a string, returns the WordObject that represents it

Parameters

Parameter	Value	Description
word	(required)	String value of WordObject to return
useCanonical		If true will try to return the correct word root ('cats' -> 'cat'). If false returns exactly what was requested.
includeSuggestions		Return suggestions (for correct spelling, case variants, etc.)
shouldCreate		Create word if not existing

Try it out!

- GET /word.json/{word}/definitions: Return definitions for a word
- GET /word.json/{word}/stats: Returns word statistics
- GET /word.json/{word}/topExample: Returns a top example for a word
- GET /word.json/{word}/contextualLookup: Returns definitions for a word based on the sentence in which it is found
- POST /word.json/{word}/contextualLookup: Returns definitions for a word based on the sentence in which it is found
- GET /word.json/{word}/commentCount: Returns the number of comments on a word
- GET /word.json/{word}/related: Return related words (thesaurus data) for a word
- GET /word.json/{word}/listedIn: Returns WordLists containing a word
- GET /word.json/{word}/listedInCount: Returns a count of lists a word appears in

Swagger2

Documentación formato Web

- <http://localhost:8080/swagger-ui.html>



controlador-persona : Controlador Persona



swagger.json

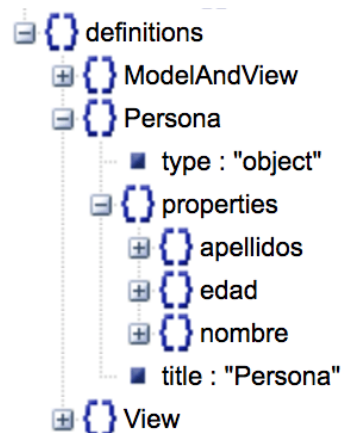
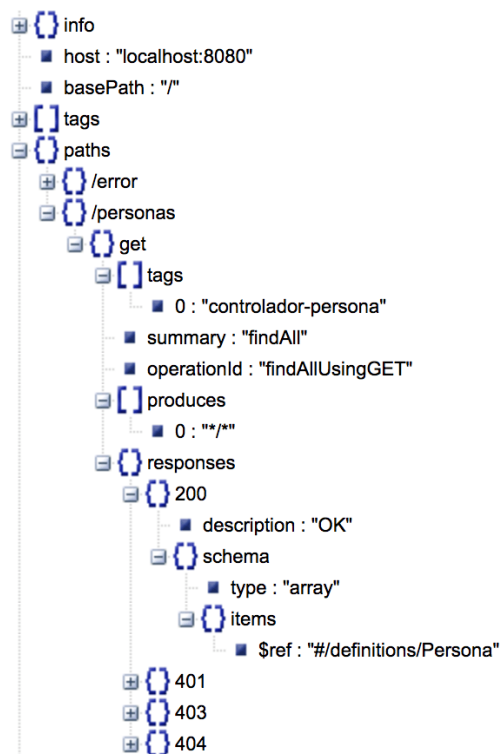
Documentación formato JSON

- /v2/api-docs

```
{
  "swagger": "2.0",
  "info": {
    "description": "Api Documentation",
    "version": "1.0",
    "title": "Api Documentation",
    "termsOfService": "urn:tos",
    "contact": {},
    "license": {
      "name": "Apache 2.0",
      "url": "http://www.apache.org/licenses/LICENSE-2.0"
    },
    "host": "localhost:8080",
    "basePath": "/",
    "tags": [
      {
        "name": "basic-error-controller",
        "description": "Basic Error Controller"
      },
      {
        "name": "controlador-persona",
        "description": "Controlador Persona"
      }
    ],
    "paths": {
      "/error": {
        "get": {
          "tags": [
            "basic-error-controller"
          ],
          "summary": "error",
          "operationId": "errorUsingGET",
          "produces": [
            "*"
          ],
          "responses": {
            "200": {
              "description": "OK",
              "schema": {
                "type": "object",
                "additionalProperties": {
                  "type": "object"
                }
              },
              "401": {
                "description": "Unauthorized",
                "403": {
                  "description": "Forbidden",
                  "404": {
                    "description": "Not Found"
                  }
                }
              },
              "head": {
                "tags": [
                  "basic-error-controller"
                ],
                "summary": "error",
                "operationId": "errorUsingHEAD",
                "consumes": [
                  "application/json"
                ],
                "produces": [
                  "*"
                ],
                "responses": {
                  "200": {
                    "description": "OK",
                    "schema": {
                      "type": "object",
                      "additionalProperties": {
                        "type": "object"
                      }
                    },
                    "204": {
                      "description": "No Content",
                      "401": {
                        "description": "Unauthorized",
                        "403": {
                          "description": "Forbidden",
                          "post": {
                            "tags": [
                              "basic-error-controller"
                            ],
                            "summary": "error",
                            "operationId": "errorUsingPOST",
                            "consumes": [
                              "application/json"
                            ],
                            "produces": [
                              "*"
                            ],
                            "responses": {
                              "200": {
                                "description": "OK",
                                "schema": {
                                  "type": "object",
                                  "additionalProperties": {
                                    "type": "object"
                                  }
                                },
                                "201": {
                                  "description": "Created",
                                  "401": {
                                    "description": "Unauthorized",
                                    "403": {
                                      "description": "Forbidden",
                                      "404": {
                                        "description": "Not Found"
                                      }
                                    }
                                },
                                "put": {
                                  "tags": [
                                    "basic-error-controller"
                                  ],
                                  "summary": "error",
                                  "operationId": "errorUsingPUT",
                                  "consumes": [
                                    "application/json"
                                  ],
                                  "produces": [
                                    "*"
                                  ],
                                  "responses": {
                                    "200": {
                                      "description": "OK",
                                      "schema": {
                                        "type": "object",
                                        "additionalProperties": {
                                          "type": "object"
                                        }
                                      },
                                      "201": {
                                        "description": "Created",
                                        "401": {
                                          "description": "Unauthorized",
                                          "403": {
                                            "description": "Forbidden",
                                            "404": {
                                              "description": "Not Found"
                                            }
                                          }
                                        },
                                        "delete": {
                                          "tags": [
                                            "basic-error-controller"
                                          ],
                                          "summary": "error",
                                          "operationId": "errorUsingDELETE",
                                          "produces": [
                                            "*"
                                          ],
                                          "responses": {
                                            "200": {
                                              "description": "OK",
                                              "schema": {
                                                "type": "object",
                                                "additionalProperties": {
                                                  "type": "object"
                                                }
                                              },
                                              "204": {
                                                "description": "No Content",
                                                "401": {
                                                  "description": "Unauthorized",
                                                  "403": {
                                                    "description": "Forbidden",
                                                    "patch": {
                                                      "tags": [
                                                        "basic-error-controller"
                                                      ],
                                                      "summary": "error",
                                                      "operationId": "errorUsingPATCH",
                                                      "consumes": [
                                                        "application/json"
                                                      ],
                                                      "produces": [
                                                        "*"
                                                      ],
                                                      "responses": {
                                                        "200": {
                                                          "description": "OK",
                                                          "schema": {
                                                            "type": "object",
                                                            "additionalProperties": {
                                                              "type": "object"
                                                            }
                                                          },
                                                          "204": {
                                                            "description": "No Content",
                                                            "401": {
                                                              "description": "Unauthorized",
                                                              "403": {
                                                                "description": "Forbidden",
                                                                "definitions": {
                                                                  "ModelAndView": {
                                                                    "type": "object",
                                                                    "properties": {
                                                                      "empty": {
                                                                        "type": "boolean",
                                                                        "model": {
                                                                          "type": "object",
                                                                          "modelMap": {
                                                                            "type": "object",
                                                                            "additionalProperties": {
                                                                              "type": "object"
                                                                            },
                                                                            "reference": {
                                                                              "type": "boolean",
                                                                              "status": {
                                                                                "type": "string",
                                                                                "enum": [
                                                                                  "100",
                                                                                  "101",
                                                                                  "102",
                                                                                  "103",
                                                                                  "200",
                                                                                  "201",
                                                                                  "202",
                                                                                  "203",
                                                                                  "204",
                                                                                  "205",
                                                                                  "206",
                                                                                  "207",
                                                                                  "208",
                                                                                  "226",
                                                                                  "300",
                                                                                  "301",
                                                                                  "302",
                                                                                  "303",
                                                                                  "304",
                                                                                  "305",
                                                                                  "307",
                                                                                  "308",
                                                                                  "400",
                                                                                  "401",
                                                                                  "402",
                                                                                  "403",
                                                                                  "404",
                                                                                  "405",
                                                                                  "406",
                                                                                  "407",
                                                                                  "408",
                                                                                  "409",
                                                                                  "410",
                                                                                  "411",
                                                                                  "412",
                                                                                  "413",
                                                                                  "414",
                                                                                  "415",
                                                                                  "416",
                                                                                  "417",
                                                                                  "418",
                                                                                  "419",
                                                                                  "420",
                                                                                  "421",
                                                                                  "422",
                                                                                  "423",
                                                                                  "424",
                                                                                  "426",
                                                                                  "428",
                                                                                  "429",
                                                                                  "431",
                                                                                  "451",
                                                                                  "500",
                                                                                  "501",
                                                                                  "502",
                                                                                  "503",
                                                                                  "504",
                                                                                  "505",
                                                                                  "506",
                                                                                  "507",
                                                                                  "508",
                                                                                  "509",
                                                                                  "510",
                                                                                  "511"
                                                                                ]
                                                                              },
                                                                              "view": {
                                                                                "$ref": "#/definitions/View",
                                                                                "viewName": {
                                                                                  "type": "string"
                                    },
                                    "title": "ModelAndView",
                                    "Persona": {
                                      "type": "object",
                                      "properties": {
                                        "apellidos": {
                                          "type": "string",
                                          "edad": {
                                            "type": "integer",
                                            "format": "int32",
                                            "nombre": {
                                              "type": "string"
                                            },
                                            "title": "Persona",
                                            "View": {
                                              "type": "object",
                                              "properties": {
                                                "contentType": {
                                                  "type": "string",
                                                  "title": "View"
                                                }
                                              }
                                            }
                                          }
                                        }
                                      }
                                    }
                                  }
                                }
                              }
                            }
                          }
                        }
                      }
                    }
                  }
                }
              }
            }
          }
        }
      }
    }
  }
}
```

Swagger2

Documentación con visor JSON



- ECMA International

<https://www.ecma-international.org/>

- World Wide Web Consortium (W3C)

<https://www.w3.org/>

- Principios de Accesibilidad (WAI)

<https://www.w3.org/WAI/fundamentals/accessibility-principles/es>

- Web Content Accessibility Guidelines (WCAG)

<https://www.w3.org/WAI/standards-guidelines/wcag/>

- RAML Specification: <https://raml.org/developers/raml-100-tutorial>
- Swagger: <https://swagger.io>
- Otras webs de referencia:
 - <http://www.w3schools.com/>
 - <http://stackoverflow.com/>
 - <http://www.codecademy.com/>



Generalitat de Catalunya
**Consorci per a la Formació Contínua
de Catalunya**

pime
Coneixement



IFCD65

Módulo 2

Javascript ES6