

Introducción a JavaScript

Víctor Custodio 2015



¿Qué es JavaScript?

- JavaScript es un lenguaje de programación que se utiliza principalmente para crear páginas web dinámicas.
- Una página web dinámica es aquella que incorpora efectos como texto que aparece y desaparece, animaciones, acciones que se activan al pulsar botones y ventanas con mensajes de aviso al usuario.



¿Cómo surgió?

- A principios de los años 90, la mayoría de usuarios que se conectaban a Internet lo hacían con módems a una velocidad máxima de 28.8 kbps. En esa época, empezaban a desarrollarse las primeras aplicaciones web y por tanto, las páginas web comenzaban a incluir formularios complejos.
- Con unas aplicaciones web cada vez más complejas y una velocidad de navegación tan lenta, surgió la necesidad de un lenguaje de programación que se ejecutara en el navegador del usuario. De esta forma, si el usuario no rellenaba correctamente un formulario, no se le hacía esperar mucho tiempo hasta que el servidor volviera a mostrar el formulario indicando los errores existentes



Un poco de historia...

- **Brendan Eich**, un programador de Netscape, pensó que podría solucionar este problema adaptando otras tecnologías existentes al navegador Netscape Navigator 2.0, que iba a lanzarse en 1995. Inicialmente, Eich denominó a su lenguaje *LiveScript*.
- Posteriormente, **Netscape firmó una alianza con Sun Microsystems** para el desarrollo del nuevo lenguaje de programación. Además, justo antes del lanzamiento Netscape decidió cambiar el nombre por el de **JavaScript**. La razón del cambio de nombre fue exclusivamente por marketing, ya que Java era la palabra de moda en el mundo informático y de Internet de la época.
- Netscape Navigator 3.0 incorporó la siguiente versión del lenguaje, la 1.1. Al mismo tiempo, Microsoft lanzó JScript con su navegador Internet Explorer 3. JScript era una copia de JavaScript al que le cambiaron el nombre para evitar problemas legales.



Un poco de historia...

- **Para evitar una guerra de tecnologías**, Netscape decidió que lo mejor sería **estandarizar el lenguaje** JavaScript. De esta forma, en 1997 se envió la especificación JavaScript 1.1 al organismo ECMA (*European Computer Manufacturers Association*).
- ECMA creó el comité TC39 con el objetivo de "*estandarizar de un lenguaje de script multiplataforma e **independiente de cualquier empresa***". El primer estándar que creó el comité TC39 se denominó **ECMA-262**, en el que se definió por primera vez **el lenguaje ECMAScript**.
- Por este motivo, algunos programadores prefieren la denominación *ECMAScript* para referirse al lenguaje JavaScript. De hecho, JavaScript no es más que la implementación que realizó la empresa Netscape del estándar ECMAScript.



¿Por qué estudiar JavaScript?

- JavaScript es uno de los 3 lenguajes que todos los desarrolladores web deben aprender:
 - 1. HTML para definir el contenido de las páginas web
 - 2. CSS para especificar el diseño de páginas web
 - 3. JavaScript para programar el comportamiento de las páginas web
- JavaScript es el lenguaje de programación más popular en el mundo.
- Es un lenguaje para **HTML**, para la Web, para los ordenadores, servidores, portátiles, tabletas, teléfonos inteligentes, televisores y más.



¿Para que sirve?

- Con JavaScript podemos:
 - Cambiar contenido y el estilo de nuestro html de forma dinámica
 - Validar datos de formularios
 - reaccionar a eventos, se puede programar para ejecutarse cuando sucede algo, como cuando ha terminado de cargar la página o cuando un usuario hace clic en un elemento HTML.
 - para detectar el navegador del visitante y dependiendo del navegador cargar otra página diseñada específicamente para ese navegador.
 - para crear cookies: JavaScript puede utilizarse para almacenar y recuperar información en equipo del visitante



Ejemplo: cambiar un elemento de HTML

```
<!DOCTYPE html>

<html>

<body>

<h1>Mi primera pagina con javascript</h1>

<p>JavaScript puede cambiar el contenido de un elemeto HTML</p>

<button type="button" onclick="myFunction()">Clickeame!</button>

<p id="demo">Esto es solo una demostracion</p>

<script>

function myFunction() {

    document.getElementById("demo").innerHTML = "Hello JavaScript!";

}

</script>

</body>

</html>
```



Ejemplo: Cambiar estilos CSS

```
<!DOCTYPE html>
<html>
<body>

<h1>Mi tercera pagina con JS</h1>

<p id="demo">JavaScript puede cambiar el estilo de un elemento HTML</p>

<script>
function myFunction() {
    var x = document.getElementById("demo");
    x.style.fontSize = "25px";
    x.style.color = "red";
}
</script>

<button type="button" onclick="myFunction()">Clickeame!</button>

</body>
</html>
```



Ejemplo: Validar formularios

```
<!DOCTYPE html>
<html>
<body>
<p>Introduce un numero entre 1 y 10</p>
<input id="numeroIntroducido" type="number">
<button type="button" onclick="myFunction()">Submit</button>
<p id="demo"></p>

<script>
function myFunction() {
    var text;

    var x = document.getElementById(" numeroIntroducido ").value;

    if (isNaN(x) || x < 1 || x > 10) {
        text = "Entrada no válida";
    } else {
        text = "OK";
    }
    document.getElementById("demo").innerHTML = text;
}
</script>

</body>
</html>
```





Introducción desde 0

¿Donde se añade JavaScript?

- El código JavaScript en una página web siempre se define dentro de las etiquetas **<script>** **</script>**
- Estas etiquetas pueden estar tanto en el body como en el head.



JavaScript en el Head

```
<!DOCTYPE html>  
<html>  
<head>  
  <script>  
    alert("Hola JavaScript");  
  </script>  
</head>  
<body>  
  <h1>JavaScript en el Head</h1>  
  <p> Un parrafo</p>  
</body>  
</html>
```



JavaScript en el Body

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
</head>
<body>
<h1>JavaScript en el Head</h1>
<p> Un parrafo</p>
<script>
alert("Hola JavaScript");
</script>
</body>
</html>
```



JavaScript externo

- `<!DOCTYPE html>`
`<html>`
`<body>`
`<script src="myScript.js"></script>`
`</body>`
`</html>`



¿Cómo mostramos los resultados?

- Existen cuatro formas de mostrar resultados a través de JS:
- Pintando una caja de alerta: **window.alert()** o **alert()**
- Escribiendo en el html: **document.write()**.
- Escribiendo contenido dentro de una etiqueta html: **innerHTML**.
- Escribiendo en la consola del navegador: **console.log()**.



Sintaxis

- Un programa de computadora es una lista de "instrucciones" para ser "ejecutados" por el ordenador.
- En un lenguaje de programación, estas instrucciones de programa se llaman declaraciones o sentencias.
- JavaScript es un lenguaje de programación.
- Sentencias de JavaScript están separados por punto y coma.
- Ejemplo:

```
var x = 5;  
var y = 6;  
var z = x + y;
```



Más sobre sintaxis

- Sensible a las Mayusculas:

```
lastName = "Doe";  
lastname = "Peterson";
```

- Los comentarios:

```
var x = 5;    // comentario  
/* var x = 6;    Otra forma de comentar, (este código no se ejecutará) el  
comentario puede extenderse varias líneas */
```

- No podemos poner espacios en el nombre de variables: podemos usar 3 formas para nombrar las variables con más de una palabra.
 - Guion :first-name;
 - Guion bajo: first_name;
 - Separación por Mayusculas: firstName;
- No podemos usar palabras clave como nombre de variables:

```
var var = 5;
```



Sintaxis

- Sentencias de JavaScript se componen de:
 - Los valores
 - Operadores
 - Expresiones
 - Palabras clave
 - y Comentarios





Valores

- Los valores en JavaScript pueden ser literales o variables:
- **Literales** : son los números y las cadenas de caracteres:
- Ejemplo:

10.50

1021

“Pepito Grillo”

Numeros, con o sin decimales (con un punto si llevan decimales) y las cadenas de caracteres entre comillas.



Valores

- Los valores en JavaScript pueden ser literales o variables:
- **Variables:** las variables se utilizan para almacenar valores de datos.
- JavaScript utiliza la palabra clave var para definir variables.
- Un signo igual (=) se utiliza para asignar valores a las variables
- Ejemplo:

```
var x;  
x = 6;
```





Operadores



Operadores

- Los operadores realizan algo sobre los literales o variables:
- Para asignar un literal a una variable se utiliza el operador “=”

```
var x = 5;  
var y = 6;
```

Para realizar funciones aritméticas se utilizan los operadores matemáticos (+ - * /):

```
var x = 5 + 6;  
var y = x * 10;
```



Los operadores

Operador	Descripción
+	Suma
-	Resta
*	Multiplicacion
/	Division
%	Modulo
++	Incrementar
--	Decrementar

Ejemplo: Multiplicando

```
var x = 5;  
var y = 2;  
var z = x * y;
```

Diviviendo

```
var x = 20;  
var y = 5;  
var z = x / y;
```

Incrementando

```
var x = 5;  
x++
```

Práctica: Realiza los ejercicios
operadores 1,2,3,4 y5



Operadores de asignación

Operador	Ejemplo	Lo mismo que
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y

Ejemplo:

```
var x = 10;  
x += 5;
```

Ejemplo con palabras(strings):

```
txt1 = "Que día";  
txt1 += "más bonito hace";
```

Práctica: Realiza los ejercicios operadoresAsignacion1,2,3,4 y5



Operadores de comparación

Operador	Descripción
==	Igual a
===	Igual valor e Igual tipo
!=	distinto
!==	No igual valor o no igual tipo
>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que





Palabras clave



Palabras clave

- Palabras clave de JavaScript se utilizan para identificar las acciones a realizar por parte del navegador.
- La palabra clave var indica al navegador que debe crear una nueva variable:

```
var x = 5 + 6;  
var y = x * 10;
```

- En este caso una variable llamada x cuyo contenido será el resultado de 5+6, y posteriormente una variable y cuyo contenido será el valor de la variable x multiplicado por 10.



Palabras Clave

- Existen muchas palabras claves en JS, algunas de las que vamos a estudiar las tienes en al siguiente tabla:

Palabra Clave	Descripción
break	Termina un bucle
do ... while	Ejecuta un bloque de sentencias y repite el bloque mientras se cumpla una condición
for	Ejecuta un bloque de código hasta que la condición deje de ser verdadera
function	Declara una función
if ... else	Identifica un bloque de código que será ejecutado dependiendo de la condición
return	Se sale de la ejecución de una función
var	Declara una variable





Variables y tipos de datos



Las variables

- X Y y Z son variables

```
var x = 5;  
var y = 6;  
var z = x + y;
```

X guarda un 5, y guarda un 6 y z guarda un 11.

- Lo mismo pero con menos algebra y mas vocabulario normal:

```
var precio1= 5;  
var precio2 = 6;  
var total = precio1 + precio2;
```



Las variables

- Las variables deben ser declaradas antes de ser usadas.
- Cuando declaramos una variable, su valor es undefined (indefinido) hasta que le asignamos algún valor.

- Prueba :

- ```
<p id="demo"></p>
<script>
var coche= "Volvo"; /*declaro la variable y le asigno un valor */
document.getElementById("demo").innerHTML = coche;
</script>
```

- y:

- ```
<p id="demo"></p>
<script>
var coche; /* solo declaro la variable*/
document.getElementById("demo").innerHTML = coche;
</script>
```



Las variables

- También podemos declarar las variables separándolas por comas, sin necesidad de repetir la palabra clave var:

```
var person = "John Doe", carName = "Volvo", price = 200;
```

- Podemos usar operadores algebraicos, tanto con números como con palabras:
- `var x = 5 + 2 + 3;`
- `var y = "John" + " " + "Smith";`
 - ¿Cuál es el valor de y?
- Realiza los ejercicios ej1,2,3,4,5 y 6 sobre variables.



TIPOS DE DATOS

- El concepto de tipos de datos en programación es un concepto importante . Para poder operar en las variables , es importante saber algo sobre el tipo .
- Existen 5 tipos de datos en JavaScript:
- ```
var length = 16; // Number
var lastName = "Johnson"; // String
var cars = ["Saab", "Volvo", "BMW"]; // Array
var x {nombre:"John", apellido:"Doe"}; // Object
var verdad= true; //boolean
```



# TIPOS DE DATOS

- JS evalúa las expresiones de izquierda a derecha y de una a una, por lo que si nos encontramos con:
  - `var x = 16 + 4 + "Volvo";`

El valor de x será 20Volvo

- Y si encontramos:
  - `var x = "Volvo" + 16 + 4;`

El valor de x será Volvo164



# TIPOS DE DATOS DINÁMICOS

- JS tiene variables de tipo dinámico, esto significa que la variable será del tipo del valor que se le asigne, de forma dinámica.
- ```
var x;           // Aquí x es undefined  
var x = 5;       // Aquí x es un Number  
var x = "John";  // Aquí x es un String
```
- Esto no ocurre en lenguajes más estrictos como Java



TIPOS DE DATOS :Strings

- Los Strings se definen con comillas, ya sean dobles o simples.
- También podemos introducir comillas dentro de un String, siempre que no lo hayamos definido con el mismo tipo de comillas. Ejemplos:
- `var nombreCoche = "Volvo XC60"; // Comillas dobles`
`var nombreCoche='Volvo XC60'; // Comillas simples`
- `var frase= 'A él le llaman "Jony"'; // Comillas dobles dentro de simples`



TIPOS DE DATOS :Strings, caracteres de escape

- Hay una serie de caracteres especiales que sirven para expresar en una cadena de texto determinados controles como puede ser un salto de línea o un tabulador. Estos son algunos de los caracteres de escape:
 - Salto de línea: `\n`
 - Comilla simple: `\'`
 - Comilla doble: `\"`
 - Tabulador: `\t`



TIPOS DE DATOS : Numbers

- Los Numbers se pueden definir con decimales o sin decimales, para las decimales utilizamos el punto:
- `var x1 = 34.00; // con decimales`
`var x2 = 34;`
- También podemos escribirlos en formato científico para números muy grandes
- `var y = 123e5; // 12300000`
`var z = 123e-5; // 0.00123`



TIPOS DE DATOS :diferencias entre Strings y números.

- Es importante diferenciar entre cadena de caracteres y números ya que nos es lo mismo:

```
x = 5 + 5;
```

```
y = "5" + 5;
```

```
z= "Hola" + 5;
```

- Si lleva comillas siempre será un String, independientemente de su contenido.
- Si a un String le sumamos un numero, el resultado será un String



TIPOS DE DATOS : BOOLEANS

- Los datos booleans pueden tener solamente dos valores:
- `var x = true; //Verdadero`
`var y = false; //Falso`
- Se utilizan principalmente para indicar que se cumple una condición o no, ya profundizaremos más adelante:
- `var mayorDeEdad = true;`



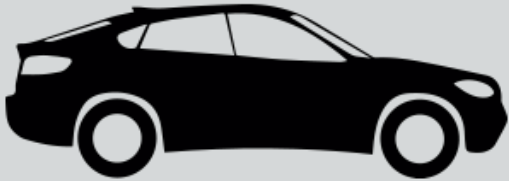
TIPOS DE DATOS : ARRAYS (Matrices)

- Los arrays o matrices son un conjunto de elementos almacenados en una misma variable.
- Matrices de JavaScript se escriben con corchetes. Elementos de matriz están separadas por comas. El código siguiente declara (crea) una matriz llamada coches , que contiene tres elementos (nombres de coche) :
- `var coches= ["Saab", "Volvo", "BMW"];`
- Veremos más sobre los Arrays más adelante...



TIPOS DE DATOS : Objetos

- En la vida real , un coche es un objeto. Un coche tiene propiedades como el peso y el color , y métodos como encender y parar:

Objeto	Propiedades	Métodos
	coche.peso : 1.2 toneladas coche.color : negro coche.numPuertas: 5	coche.arrancar (); coche.frenar(); Coche.cambiarMarcha (2°);

- Hasta ahora hemos visto como definir una variable que tiene un valor simple:
`var coche = "Fiat";`
- Ahora veamos como se crea una variable que lo que tiene es un objeto (conjunto de valores y métodos)



TIPOS DE DATOS: Objetos

- Los Objetos en JavaScript se escriben con llaves. Propiedades del objeto se escriben con pares de valores(nombre : valor) y separados por comas entre ellas.
- `var persona = {nombre:"John", apellido:"Doe", edad:50, colorOjos:"azul"};`
- `var coche = {tipo:"Fiat", modelo:500, color:"blanco"};`



TIPOS DE DATOS: Objetos

- Cuando queramos que una variable sea un objeto vacio, podemos asignarle el valor null (nulo):

```
var person = null;
```

- O utilizar el valor undefined:

```
var person = undefined;
```





FUNCIONES



FUNCIONES

- Una función de JavaScript es un bloque de código diseñado para realizar una tarea en particular. Una función de JavaScript se ejecuta cuando "algo" lo invoca (alguien o algo lo llama) .
- `function` miFunction(p1, p2) {
 `return` p1 * p2; // la función devuelve el producto de p1 y p2.
}



FUNCIONES: Sintaxis

- `function nombre(parametro, parametro2, parametro3) {
 código a ejecutar
}`
- Una función JavaScript está definida con la palabra clave de la función, seguido por un nombre, seguido de paréntesis ().
- Los nombres de funciones pueden contener letras , dígitos, guiones y signos de dólar (mismas reglas que las variables) .
- Los paréntesis pueden incluir nombres de parámetros separados por comas: (parámetro1, parámetro2 , ...)
- El código que se ejecutará , por la función , se coloca dentro de llaves : { }



FUNCIONES: Invocando una función

- El código dentro de la función se ejecutará cuando se invoca a ésta, es decir "algo" llamadas a la función:
 - Cuando se produce un evento (cuando un usuario hace clic en un botón)
 - Cuando se invoca (llamada) desde otro código JavaScript
 - Automáticamente (auto invocado)



FUNCIONES: Retorno de la función

- Cuando JavaScript alcanza una sentencia return, la función detendrá la ejecución .
- Si la función se invoca desde código JavaScript se " volverá a la línea siguiente de código para ejecutar el código de después de la invocación .
- Las funciones menudo calculan un valor de retorno. El valor de retorno es " devuelto" de nuevo a la " persona que llama " :

```
var x = myFunction(4, 3);           // Se llama a myFunction, ésta  
devuelve un valor, que será asignado a x
```

```
function myFunction(a, b) {  
    return a * b;                   // La función myFuction devuelve el  
    producto de los argumentos recibidos(parametros).  
}
```



¿Para que funciones?

- Puedes volver a utilizar código: Definir el código una vez, y lo utilizarlo muchas veces.
- Puedes utilizar el mismo código muchas veces con diferentes argumentos, para producir resultados diferentes.

- Ejemplo: Una función que convierte de grados fahrenheit a celsius:

```
function toCelsius(fahrenheit) {  
    return (5/9) * (fahrenheit-32);  
}  
  
var x = toCelsius(32);  
document.getElementById("demo").innerHTML = x;
```

- Llamar al metodo toCelsius pasandole el argumento deseado es mucho más sencillo que realizar siempre $(5/9) * (fahrenheit-32)$;



Funciones: Ahorrando variables

- En lugar de asignar el retorno de una función a una variable, podemos utilizar la llamada a la función en si misma como un valor:

```
var text = "La temperatura es " + toCelsius(32) + "  
Centigrados";
```

- Sería lo mismo que:

```
var x = toCelsius(32);  
var text = "The temperature is " + x + " Centigrade";
```





Objetos

- En JavaScript los objetos los reyes. Si entiendes bien los objetos, entiendes JS. Ya vimos como definir objetos:

```
var persona = {nombre:"John", apellidos:"Doe", edad:50,  
colorDeOjos:"azul"};
```

- Se suele definir en varias lineas por claridad:

```
var persona = {  
  nombre:"John",  
  apellidos:"Doe",  
  edad:50,  
  colorDeOjos:"blue"  
};
```



Objetos

- También se pueden definir objetos utilizando la palabra clave de JavaScript ***new***.
- `var persona = new Object();`
`persona.nombre = "John";`
`persona.apellidos = "Doe";`
`persona.edad = 50;`
`persona.colorOjos = "azules";`



Objetos: Propiedades

- Además de variables, las propiedades de los objetos también pueden ser funciones u otros objetos, veamos posibles valores de un objeto persona:

Propiedad	Valor de la Propiedad
nombre	John
apellido	Doe
edad	50
colorOjos	blue
nombreCompleto	function() {return this.nombre + " " + this.apellido;}
Pareja	persona2 (otro objeto de la clase persona)



Objetos: Accediendo a sus propiedades

- Podemos acceder a las propiedades de los objetos de dos formas distintas:

nombreObjeto.nombrePropiedad

- o

nombreObjeto[nombrePropiedad]

Ejemplo:

`persona["apellidos"]`; o `persona.apellido`;



Objetos: Accediendo a sus propiedades

- Para acceder a las funciones definidas en el objeto, deberemos de usar siempre los paréntesis al final para indicar que lo que estamos buscando es la ejecución de una función.
 - `var nombreDeLaPersona = persona.nombreCompleto();`



Objetos: El Constructor

- Existe una función especial que crea instancias de objetos específicos, es la función constructor.
- Por ejemplo un constructor para nuestro objeto persona podría ser el siguiente.

```
▪ function persona(nombre, apellido, edad, ojo) {  
    this.nombre = nombre;  
    this.apellido = apellido;  
    this.edad = edad;  
    this.colorOjos = ojos;  
}
```

- Y podríamos usarlo para crear otros objetos con las mismas propiedades haciendo uso de la palabra clave new

```
var miPadre = new persona("John", "Doe", 50, "blue");  
var miMadre = new persona("Sally", "Rally", 48, "green");
```



Objetos: La palabra clave *this*

- La palabra clave ***this*** es utilizada para referenciar al objeto al que pertenece el código JS.
 - El Valor de ***this*** usado en una función es el objeto propietario de dicha función
 - Ej: `function nombreCompleto() {return this.nombre + " " + this.apellido;}`
 - El ***this*** usado en un constructor no tiene un valor, solo sustituye al futuro objeto creado por el constructor.



Objetos: El Constructor por defecto

- JavaScript tiene constructores ya creados para trabajar con sus objetos, y casi todo en JavaScript son o pueden ser objetos:

```
var x1 = new Object();    // nuevo Objecto object
var x2 = new String();    //un objeto String
var x3 = new Number();    //un objeto Numero
var x4 = new Boolean()    // un objeto Booleano
var x5 = new Array();     // un objeto Array
var x6 = new RegExp();    // un Objeto de expression regular

var x7 = new Function();  // un objeto función
var x8 = new Date();      // un objeto de fecha.
```



Objetos: El Constructor por defecto

- A pesar de tener objetos para trabajar con los tipos más básicos de JS, es mucho mas eficiente trabajar con tipos primitivos
- ```
var x1 = {}; // new object
var x2 = ""; // new primitive string
var x3 = 0; // new primitive number
var x4 = false; // new primitive boolean
var x5 = []; // new array object
var x6 = /()/ // new regexp object
var x7 = function(){}; // new function object
```



# Objetos vs Tipos Primitivos

- ¿Por qué es mejor trabajar con tipos primitivos que con objetos?
  - Los objetos realmente son referencias a los datos (una dirección)
  - Los tipos primitivos son los datos en si mismos
- Si trabajamos con objetos primitivos nos estamos ahorrando realizar un redireccionamiento cada vez que usemos la variable. Por eso es más eficiente.
- Hagamos un ejemplo para entender las diferencias:





# Objetos vs Tipos Primitivos

- ¿Por qué es mejor trabajar con tipos primitivos que con objetos?
  - Los objetos realmente son referencias a los datos (una dirección)
  - Los tipos primitivos son los datos en si mismos
- Si trabajamos con objetos primitivos nos estamos ahorrando realizar un redireccionamiento cada vez que usemos la variable. Por eso es más eficiente.
- Hagamos un ejemplo para entender las diferencias:
- Ejecuta [ejemploObjetosVsPrimitivos.html](#)





**Alcance (scope)**

# Alcance

- En JavaScript el alcance es el conjunto de variables, objetos y funciones al que tienes acceso.
- No en todas las partes del código tienes acceso a todas las variables, objetos y funciones. Existen distintos Scopes.



# Alcance Local

- Las variables, funciones y objetos definidos dentro de una función, son accesibles solamente desde dentro de dicha función. Cuando la ejecución de la función se finaliza se eliminarán.

```
function myFunction() {
 // El código aquí no podrá usar la variable coche ya que
 // el código se ejecuta secuencialmente y aun no ha sido
 // definida
 var coche= "Volvo";
 //El código aquí puede usar la variable coche.
}

// El código aquí no podrá usar la variable coche
function myOtraFunction() {
 //el código aquí no puede usar la variable coche
}
```



# Alcance Global

- Las variables, funciones y objetos definidos fuera de una función tienen un alcance global y pueden ser accedidos desde cualquier lugar de la página web.

```
var coche= " Volvo";
```

```
// el código aquí puede usar la variable coche
```

```
function myFunction() {
```

```
 //el código aquí puede usar la variable coche
```

```
 coche = "renault";
```

```
}
```

```
function myOtraFunction() {
```

```
 //el código aquí puede usar la variable coche
```

```
 coche = "seat";
```

```
}
```

- Los elementos globales no se eliminan hasta que se cierra la página web





# Eventos HTML



# Eventos HTML

- Los eventos HTML son las cosas que ocurren en los elementos HTML
- Cuando JavaScript se usa en paginas HTML, JS puede reaccionar a estos eventos.
- Algunos ejemplos de eventos son:
  - Una página web HTML ha terminado de cargarse.
  - Un campo de entrada de un formulario (<input>) ha cambiado su valor.
  - Un botón ha sido pulsado.



# Eventos HTML

- Html permite a través de atributos definir que código JavaScript se encargará de definir un evento concreto.
- Para ello utiliza la siguiente sintaxis sobre la etiqueta fuente del evento:
  - `<etiqueta-html algun-evento =“código JavaScript”>` (o comillas simples)
  - `<etiqueta-html algun-evento =‘código JavaScript’>`
- Ejemplo:





# Eventos HTML

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<button onclick="getElementById('demo').innerHTML=Date()">¿Qué hora
es?</button>
```

```
<p id="demo"></p>
```

```
</body>
```

```
</html>
```



# Eventos HTML

- También podemos actuar sobre la propia etiqueta fuente del evento con el atributo `this`.

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<button onclick="this.innerHTML=Date()">¿Qué hora es?</button>
```

```
<p id="demo"></p>
```

```
</body>
```

```
</html>
```



# Eventos HTML

- Cómo el código JavaScript que trata el evento suele ser más largo que una línea, lo más común es que este código se defina en una función y desde el atributo de la etiqueta html.

```
<!DOCTYPE html>
<html>
<body>
<p>Click the button to display the date.</p>
<button onclick="displayDate()">The time is?</button>
<script>
function displayDate() {
 document.getElementById("demo").innerHTML = Date();
}
</script>
<p id="demo"></p>
</body>
</html>
```



# Eventos HTML

- Algunos de los eventos más comunes en html son

onchange	Un element HTML cambia
onclick	Un element html ha sido clickado
onmouseover	El usuario mueve el raton por encima del elemento
onmouseout	El usuario mueve el raton fuera del elemento
onkeydown	El usuario pulsa una tecla del teclado
onload	El navegador ha finalizado de cargar la página

- Realiza los ejercicios: eventos 1,2,3 y 4.

