

Javascript #1

Javascript

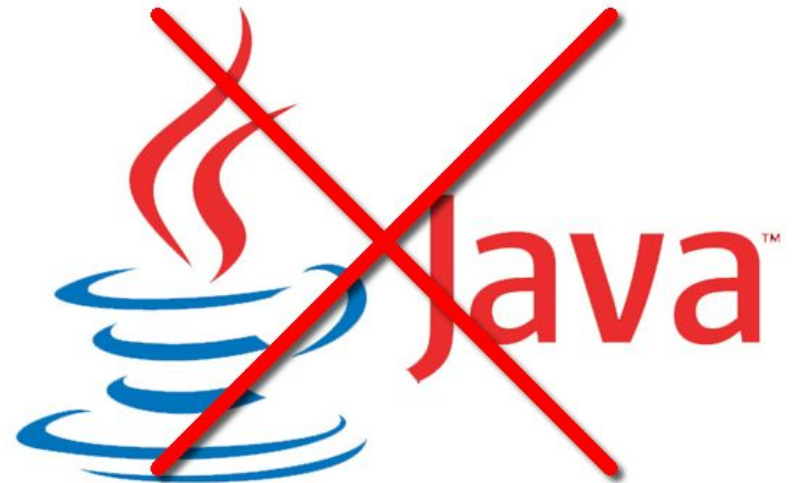
JavaScript es un lenguaje de programación interpretado. Es uno de los lenguajes de programación más utilizados hoy en día.

Nace con la necesidad de generar *dinamismo* en las páginas web.

- Principal uso, **del lado del cliente**. Existe un intérprete javascript en cada máquina (navegadores)
- Por su popularidad se ha extendido a otras aplicaciones y entornos fuera de la web. (por ej. mobile apps)
- Existe también del lado del servidor (node.js)

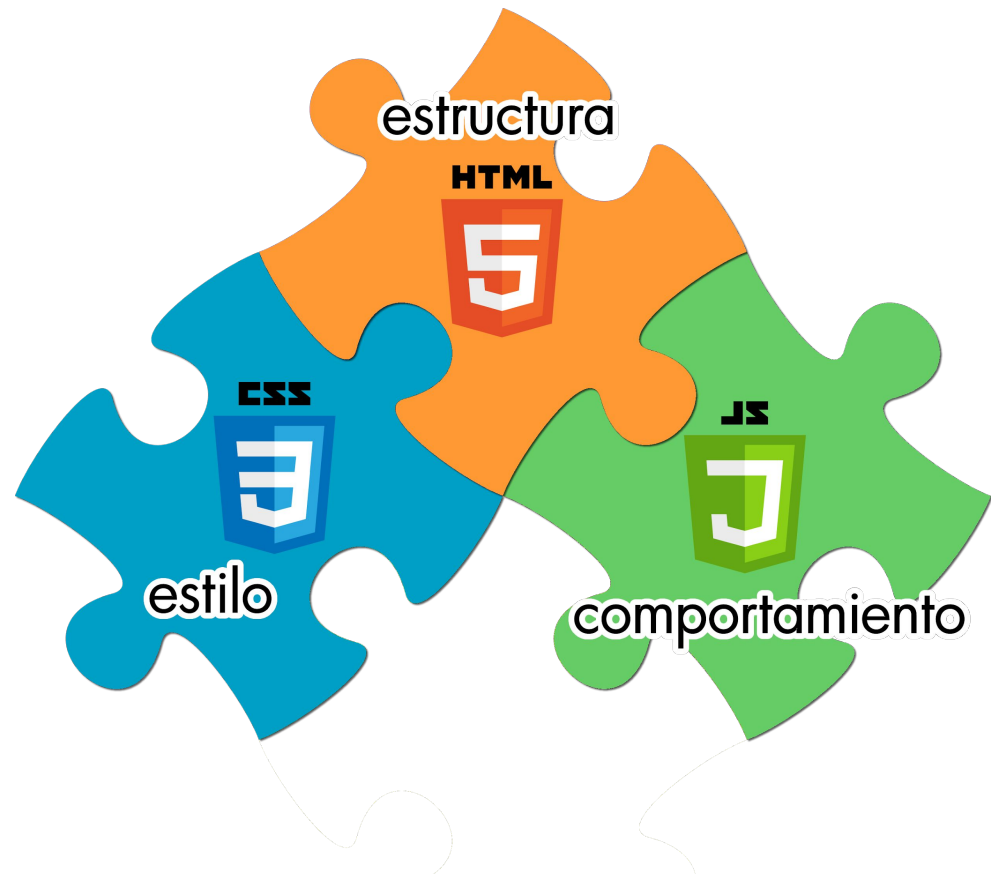
Breve Historia

- Desarrollado por Brendan Eich 1995.
- Se llamó Mocha, después LiveScript.
- Primer navegador con soporte: Netscape 2.
- NO TIENE RELACIÓN CON JAVA



Javascript: Origen

- Se popularizó con DHTML (páginas dinámicas).
- La web dejó de ser un conjunto de markup documents, para tener comportamiento de acuerdo al usuario.
- DHTML = HTML + CSS + JavaScript + DOM

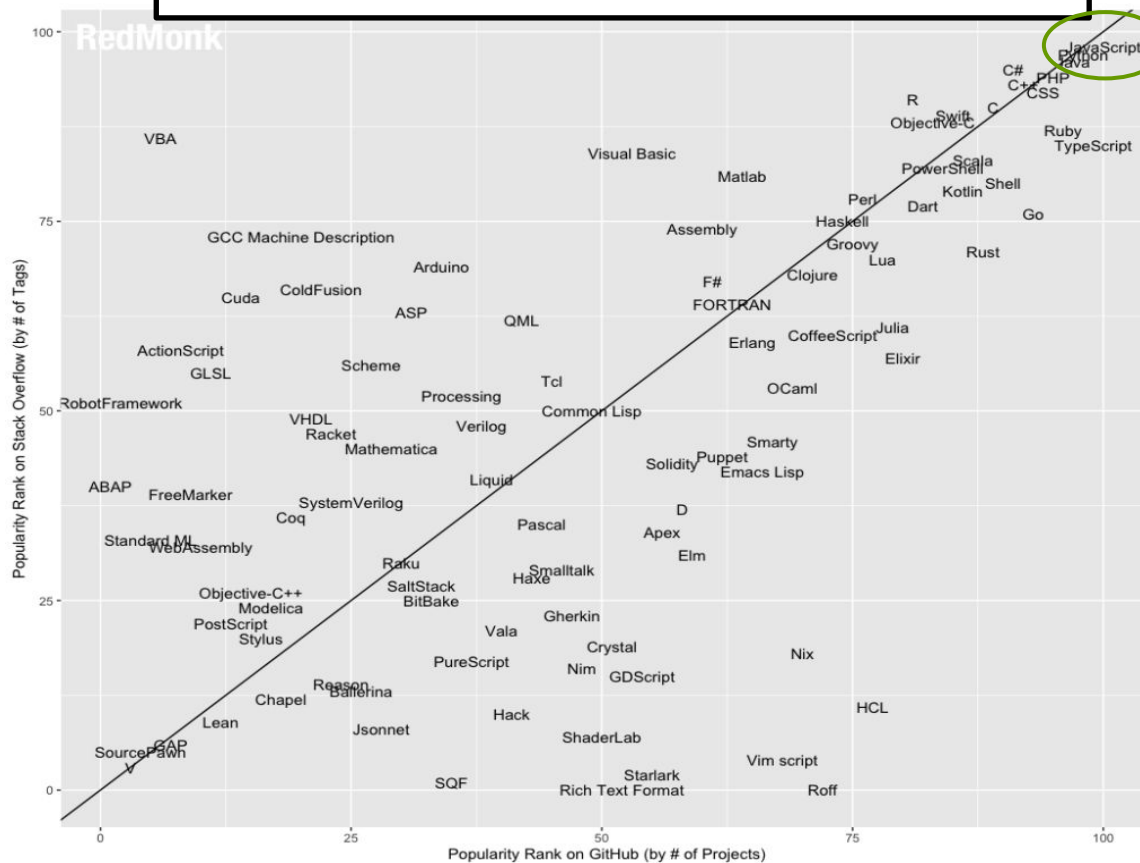


Y está en el mismo lugar desde que esta materia existe!

Redmonk Rank 2022



#tags en StackOverflow



#PR en GitHub

Javascript ¿para qué?

Algunos ejemplos:

- Validar formularios (lo que no permita HTML5)
- Reaccionar a lo que haga el usuario (click, teclear, etc)
- Cambiar algo al pasar el mouse (si no lo permite CSS)
- Partes de páginas que se muestran/ocultan
- Hacer cálculos complejos
- Carga dinámica de contenido (AJAX)
- Hacer Single Page Applications (SPAs)

Atwood's Law



Corolario del *Principle of Least Power*

Any application that can be written in JavaScript, will eventually be written in JavaScript.



ECMAScript 6 (ES6)

ES6 - El estándar

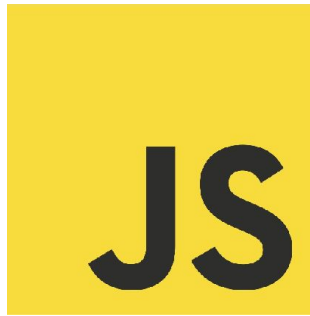
ECMAScript (ES) es el estándar que define a JavaScript.

- ES6 Aprobado en Junio de 2015
 - ES6 fué el cambio más grande en Javascript en años
 - Ya están aprobados hasta la versión 12 => ES12 (2021)
-
- Cambios de sintaxis y estructura para un código más simple y compacto
 - Ya soportado casi 100% por navegadores modernos
 - [Link para chequear compatibilidad](#)
 - [Link a la especificación del lenguaje](#)

Vamos a indicar cambios
del lenguaje así



Ejemplos



KEEP
CALM
AND
CODE
JAVASCRIPT

Ejemplo 1: Mensaje al usuario

Hacer una página que salude al usuario al entrar

Mostrar un mensaje saludando al usuario al entrar a la página

¿Qué vamos a aprender?

- Incluir un archivo Javascript y ejecutarlo
- Mostrar una alerta (cartel) por pantalla

Como incluir un Javascript

- Conviene incluir un archivo Javascript separado
- **Se ejecuta su código en la línea donde se incluye**

```
<!DOCTYPE html>
```

```
<html>
```

```
...
```

```
<body>
```

```
.....
```

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

```
</body>
```

```
</html>
```

- Incluirlo al final del body, luego de que ya se cargo el html con **todos** sus elementos.
- **Se pueden agregar varios archivos .js**



Función Alert

- La función alert nos muestra una alerta en nuestro navegador
- La forma de usarla es:

```
alert("mensaje")
```

- No se suele usar en páginas reales, ya que no se integra visualmente con el resto del sitio

Buscá los cartelitos de Demo:

Recordá que en Codepen están todas las soluciones para experimentar, son lo mismo que hacemos en clase!



<http://codepen.io/webUnicen/pen/eZMvzo>

Pregunta

El código Javascript incluido se ejecuta automáticamente al cargar la página

¿Qué pasa si hay dos alert?

- A. Se muestran los dos mensajes
- B. Se muestra uno y al aceptarlo recién se muestra el segundo

```
/* Mi codigo inicial de Javascript  
muestra un alert para comprobar que el codigo se esta ejecutando.  
*/  
alert("HOLA USUARIO!");
```


Ver la consola

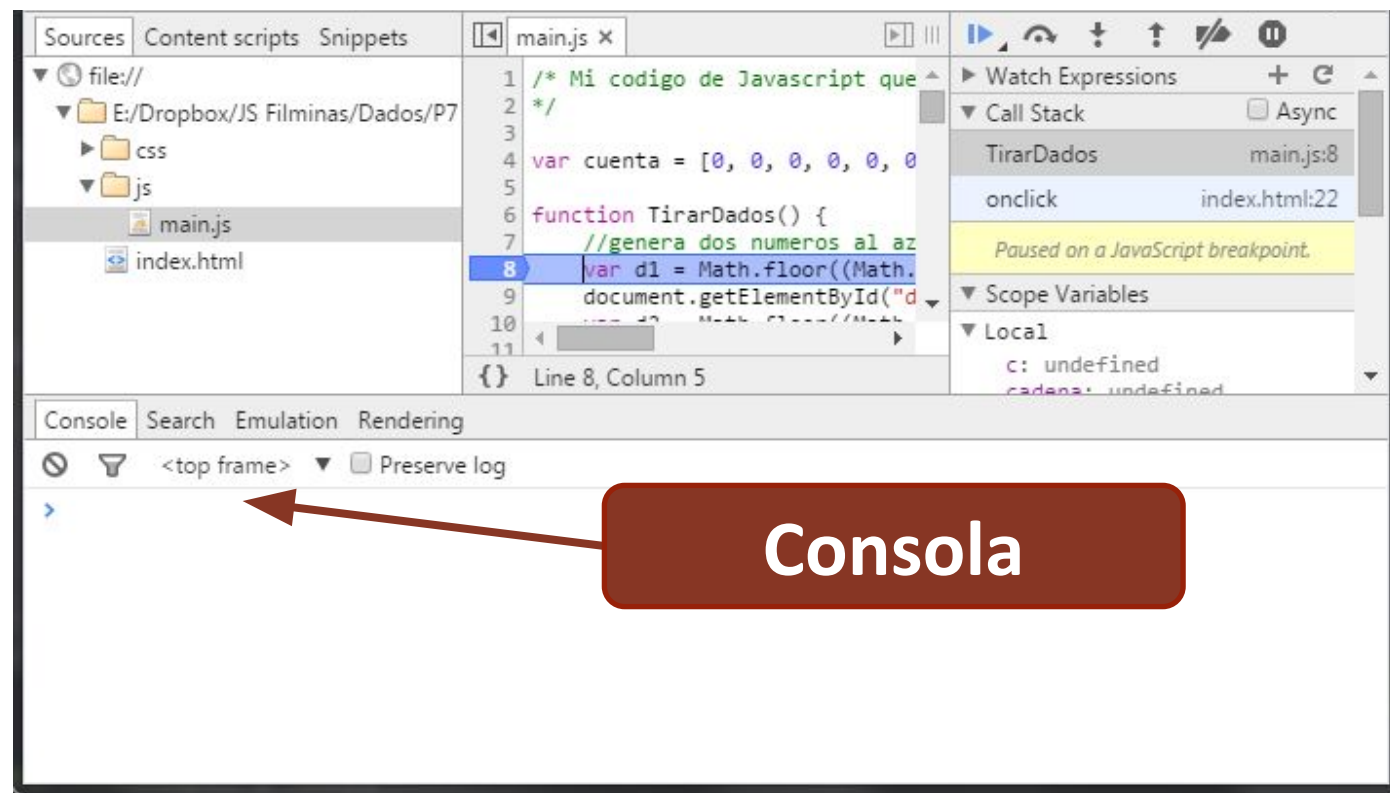
Menú Chrome >

Más herramientas >

Herramientas Desarrollador

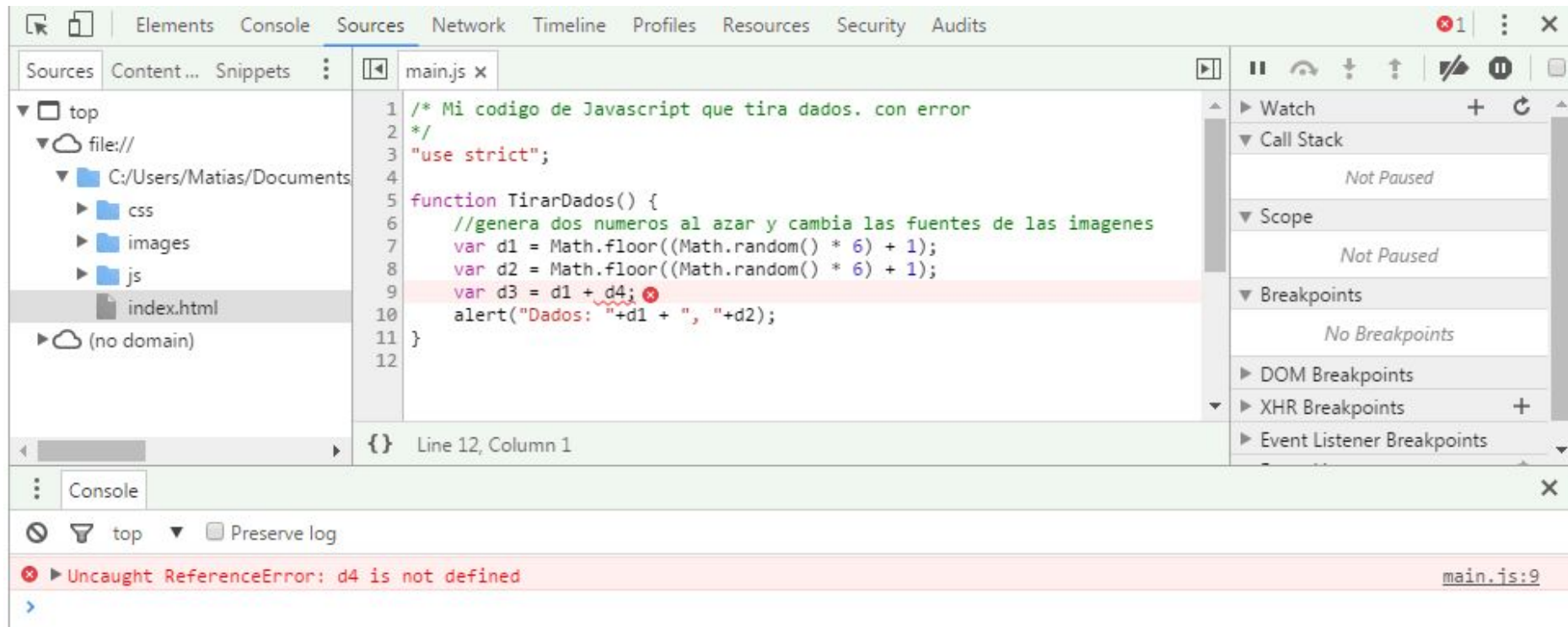
Atajo de Teclado:

- Ctrl + Shift + I
- F12



Syntax Error

- Algunos errores del código se detectan al cargar el JS
- Otros errores luego de ejecutar esa línea
- Si no se tiene abierta a las “Herramientas de desarrollador”, no se ven los errores
- SIEMPRE ABRIRLA AL PROGRAMAR JS!



Resumen

Aprendimos a

- **Incluir un archivo Javascript y ejecutarlo**
- **Mostrar un cartelito por pantalla**



Botón para saludar

Resolver el problema

¿Qué vamos a aprender?

- Ejecutar un código al hacer click en un botón
 - Esto se llama “al pasar un **evento**”
- Para eso necesitamos darle un nombre a una parte del código
 - Esto se llama “declarar una **función**”

Funciones

“Una función es un conjunto de líneas de código que realizan una tarea específica y puede retornar un valor.

Las funciones pueden tomar parámetros que modifiquen su funcionamiento.”



WIKIPEDIA
The Free Encyclopedia

Funciones

- En Javascript no hay diferencia entre funciones y procedimientos
- Una función puede o no devolver datos

```
function saludar()  
{  
  alert("Hola");  
}
```

Creación

- Para llamar a esta función tengo que invocar su “firma”.

```
saludar();
```

Ejecución

(llamar a la función)

- Esto va a mostrar un cartel que diga **Hola**.



<http://codepen.io/webUnicen/pen/YZmNwZ>

Funciones: Para qué?

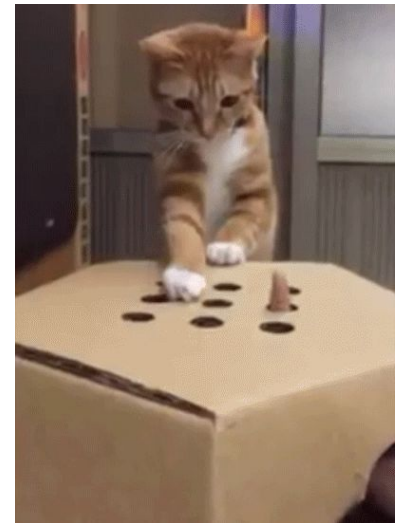
Una función le da un nombre a una porción de código

Sirve principalmente para:

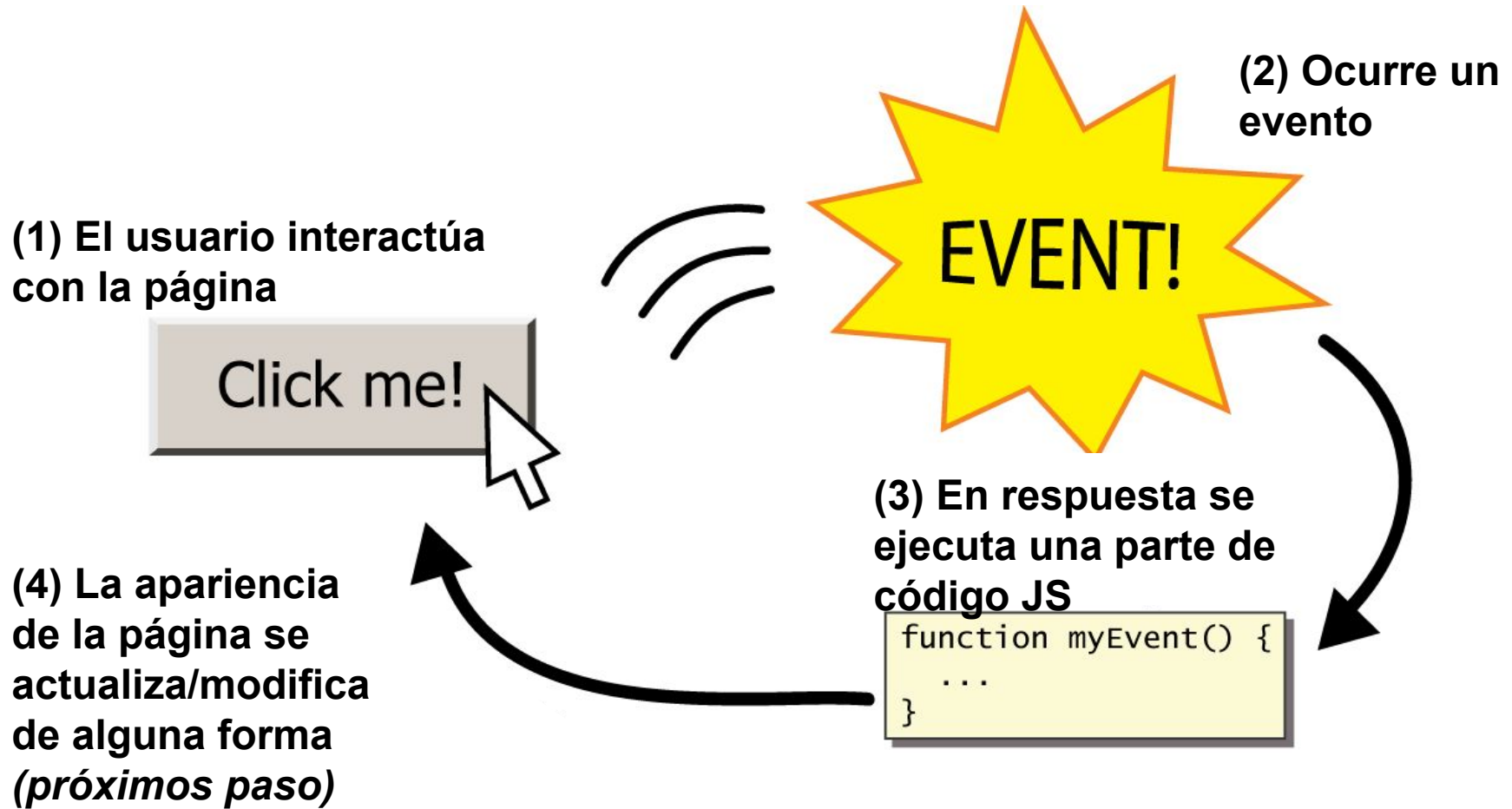
- dividir un problema grande en varios problemas chicos
- facilitar lectura del código
 - Ej: en lugar de 1000 líneas sueltas tengo 50 funciones de 20 líneas
- reutilizar el código (llamandolo dos veces)
 - en un futuro tengo un solo lugar para modificar

Eventos

- Un evento es algo que ocurre en el sistema, originado por el usuario o otra parte del sistema y que se avisa al sistema.
- Ejemplos:
 - El usuario hace click.
 - Se terminó de cargar la página.
 - Pasó un segundo desde que se terminó de procesar.
- Las interfaces gráficas suelen programarse orientada a eventos.



Programación dirigida por eventos



Eventos

- Los eventos son capturados por manejadores (handlers).

- **HTML**

Evento

```
<button onclick="saludar();">Saludar</button>
```

- **Javascript**

**Función
(handler o
callback)**

```
function saludar() {  
  alert("Hola!");  
}
```

**No recomendado
(por ahora se hace así)**

Resultado



Saludar!

Una página incorporada en s.codepen.io dice
Hola!

Aceptar

Podes ver el ejemplo aca:

<https://codepen.io/webUnicen/pen/VXWbWL>

Eventos

Ejemplos de eventos:

- [onclick](#)
- [onkeydown](#)
- [onload](#)
- [onfocus](#)
- [onchange](#) (recomendado para selects)
- [ondrag](#)
- [oncopy](#)
- [onpause](#)(para media)

Hay 50~100 eventos: http://www.w3schools.com/jsref/dom_obj_event.asp

Programación dirigida por eventos

Un programa dirigido por eventos sigue los siguientes pasos:

- Comienza la ejecución del programa
- Se llevan a cabo las inicializaciones y demás código inicial
- El programa queda bloqueado “Escuchando” hasta que se produzca algún evento

Se definen:

- Eventos (Click, Drag, Hover, Load, etc.)
- Funciones que se ejecutan en esos eventos
- Se llama el “controlador de eventos”



Programación dirigida por eventos

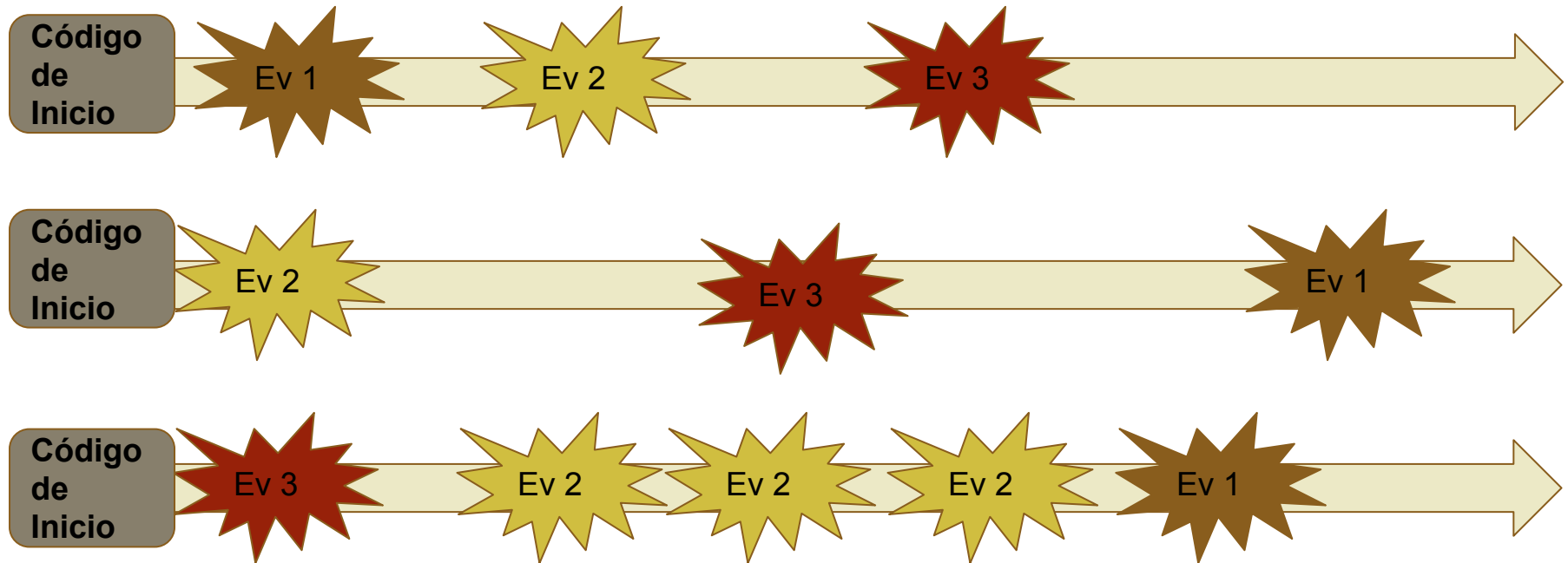
Programación secuencial

- Sabemos el flujo de la ejecución

Programamos dirigida por eventos

- No sabemos la secuencia exacta de ejecución
- Se disparan diferentes códigos con diferentes acciones

Ejemplo de tres ejecuciones diferentes:



Resumen

Aprendimos a

- Usar un **evento**
- Declarar una **función**



Saludo con nombre

Saludar

Consigna:

- Un lugar para escribir en la página web. A medida que escribo mi nombre la página me dice “Bienvenido {NOMBRE}”.
 1. Bienvenido J
 2. Bienvenido Ja
 3. Bienvenido Jav
 4. Bienvenido Javi

¿Qué vamos a aprender?

- Usar otro evento (que no es onclick)
- Editar la página web desde Javascript
- Calcular el largo de una cadena

Variables y Constantes

Variables:

- Una variable es un nombre que le damos a un valor que puede cambiar (o no) con el tiempo (durante la ejecución del programa)
- El nombre no es el contenido, es como llamamos a ese valor, pero sin saber el valor exacto mientras escribimos

Constantes:

- Son un nombre que le damos a un valor
- Nunca cambia con el tiempo
- Se usan para aumentar la legibilidad del programa

Antes de ES6

Declarar una variable

```
var nombre = "Pepe";
```



Declarar una constante

No hay

ES6

Declarar una variable

```
let nombre = "Pepe";
```



Declarar una constante

```
const cantDados = 2;
```

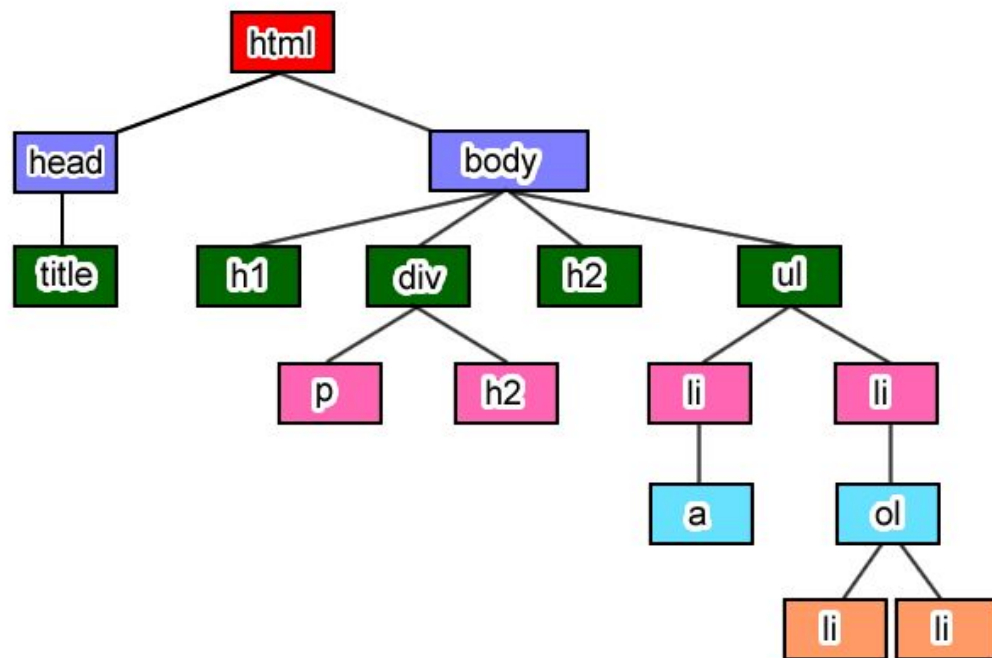
Usar "let" es mejor, porque reduce las posibilidades de introducir errores indeseados. La diferencia la vamos a ver más adelante porque es un detalle muy fino.



Arbol HTML - DOM

Una manera de comprender las dependencias y relaciones entre elementos es mediante un diagrama de árbol.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Ejemplo Arbol</title>
    <link rel="stylesheet" href="estilo.css">
  </head>
  <body>
    <h1>Titulo 1</h1>
    <div>Div1
      <p>Parrafo dentro de div</p>
      <h2>Titulo 2 en div</h2>
    </div>
    <h2>Titulo 2</h2>
    <ul>
      <li>Elemento 1 <a href="...">Link1</a></li>
      <li>Elemento 2
        <ol>
          <li>Primero</li>
          <li>Segundo</li>
        </ol>
      </li>
    </ul>
  </body>
</html>
```



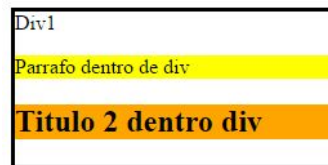
Introducción a DOM

El **D**ocument **O**bject **M**odel es una API (**A**pplication **P**rogramming Interface) para documentos HTML y XML.

- Representación estructurada del documento
- Permite modificar el contenido
- Es lo que conecta las páginas web con Javascript.

El DOM es un árbol de objetos...

Titulo 1



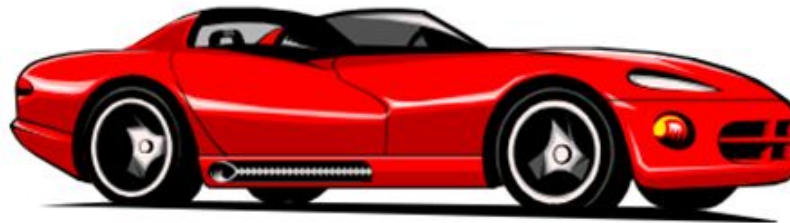
Titulo 2

- Elemento 1 [Link1](#)
- Elemento 2
 1. Primero
 2. Segundo



Objetos

- En la vida real todos los objetos tienen una serie de características y un comportamiento.
- En programación, un objeto es una combinación de
 - **Campos o atributos:** almacenan datos. Estos datos pueden ser de tipo primitivo y/o otro tipo de objeto
 - **Rutinas o métodos:** lleva a cabo una determinada acción o tarea con los atributos.



■ Atributos:

- color
- velocidad
- ruedas
- motor

■ Métodos:

- arranca()
- frena()
- dobla()

Acceso a atributos y llamado de métodos con .

```
let auto = ...  
auto.arranca();  
auto.color = rojo;  
alert(auto.ruedas);
```

Objetos en el DOM y JS

Existen muchos objetos ya predefinidos

Los más usados son:

- **document:** El DOM de los elementos del body y header de este archivo HTML.
- **Window:** La ventana/pestaña del navegador. Es quien tiene el método “alert” que usamos antes.
- **History:** El historial, nos permite ir adelante, atrás, etc
- **Location:** La URL de la barra de navegación.

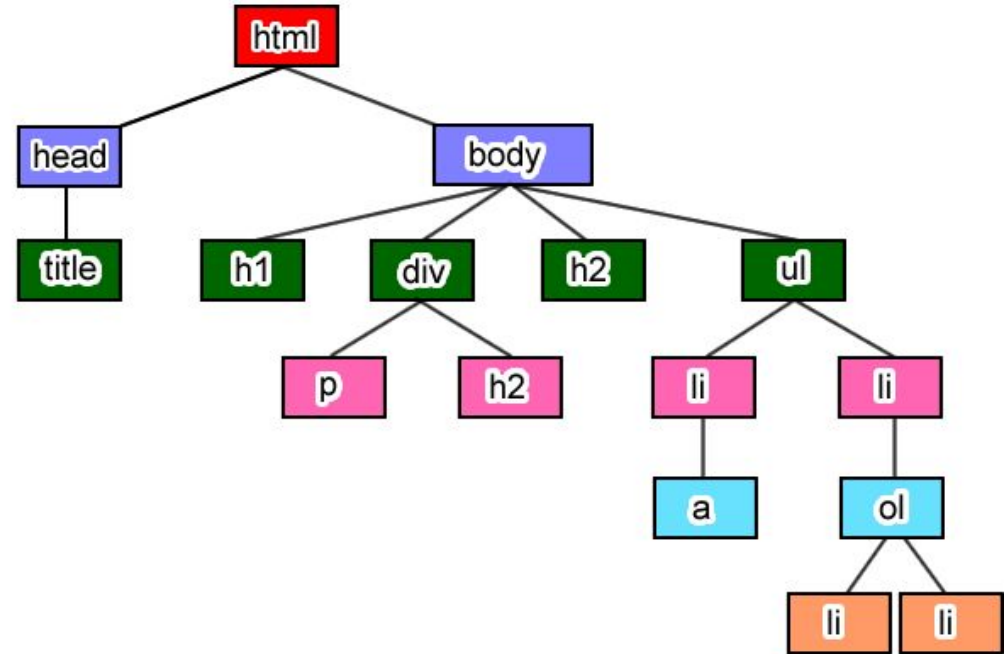
Como editar el DOM

1. Al documento le pedimos el nodo del elemento que queremos editar
2. A ese objeto (el nodo del arbol en cuestion) le modificamos los atributos que necesitamos con un nuevo valor



DOM - Edición

```
<!DOCTYPE html>
<html>
  <head>
    <title>Ejemplo Arbol</title>
    <link rel="stylesheet" href="estilo.css">
  </head>
  <body>
    <h1>Titulo 1</h1>
    <div>Div1
      <p>Parrafo dentro de div</p>
      <h2>Titulo 2 en div</h2>
    </div>
    <h2>Titulo 2</h2>
    <ul>
      <li>Elemento 1 <a href="...">Link1</a></li>
      <li>Elemento 2
        <ol>
          <li id="prim">Primero</li>
          <li id="seg">Segundo</li>
        </ol>
      </li>
    </ul>
  </body>
</html>
```

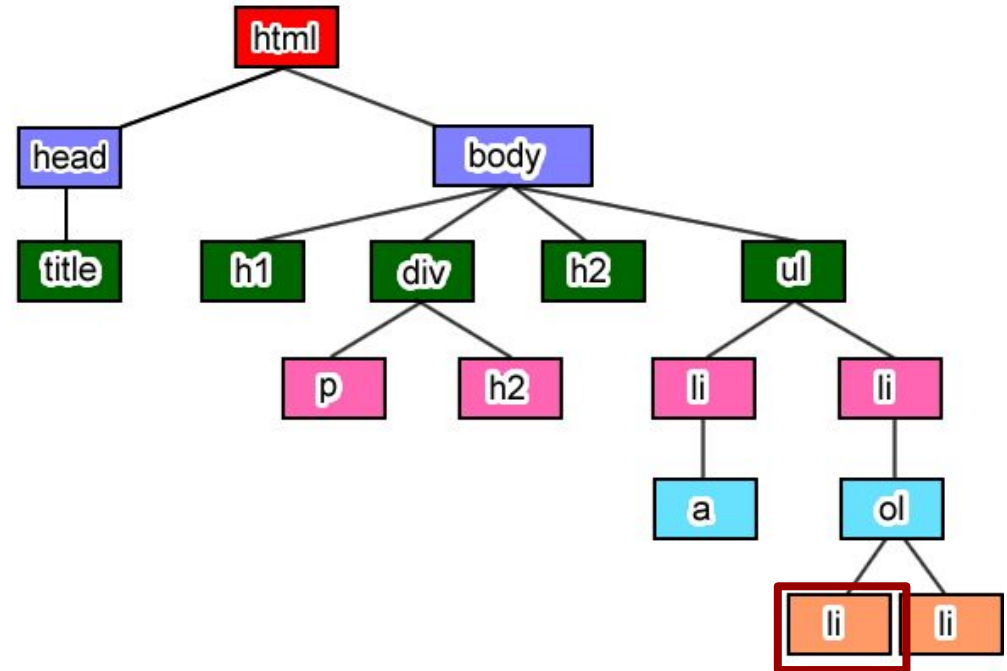


```
document.querySelector('#prim').innerHTML = "1- Primero";
```

Al documento

DOM - Edición

```
<!DOCTYPE html>
<html>
  <head>
    <title>Ejemplo Arbol</title>
    <link rel="stylesheet" href="estilo.css">
  </head>
  <body>
    <h1>Titulo 1</h1>
    <div>Div1
      <p>Parrafo dentro de div</p>
      <h2>Titulo 2 en div</h2>
    </div>
    <h2>Titulo 2</h2>
    <ul>
      <li>Elemento 1 <a href="...">Link1</a></li>
      <li>Elemento 2
        <ol>
          <li id="prim">Primero</li>
          <li id="seg">Segundo</li>
        </ol>
      </li>
    </ul>
  </body>
</html>
```

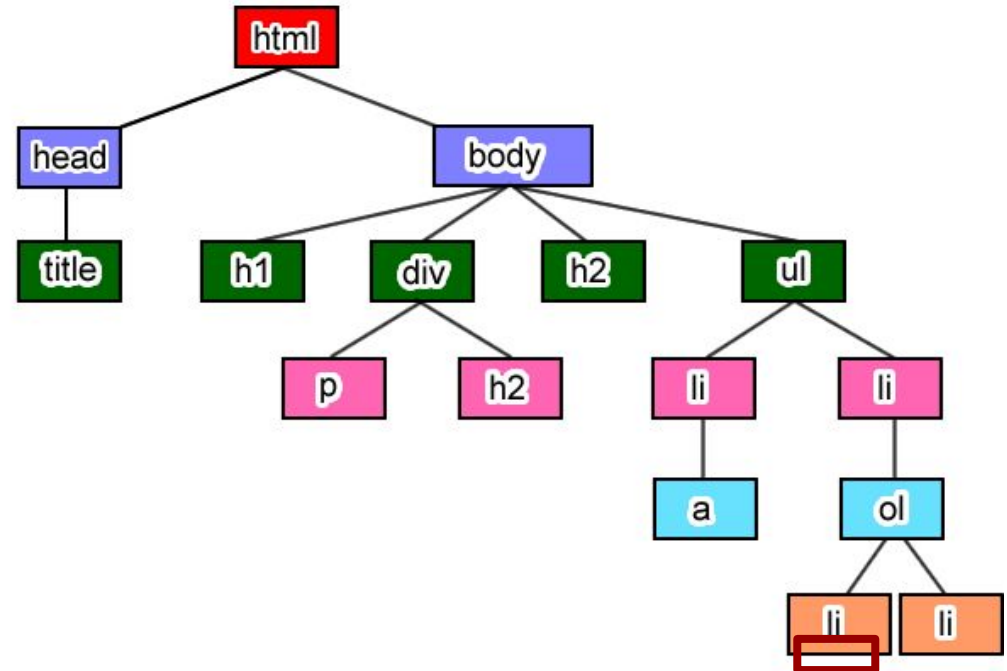


```
document.querySelector('#prim').innerHTML = "1- Primero";
```

Al documento le pido el elemento con ID prim

DOM - Edición

```
<!DOCTYPE html>
<html>
  <head>
    <title>Ejemplo Arbol</title>
    <link rel="stylesheet" href="estilo.css">
  </head>
  <body>
    <h1>Titulo 1</h1>
    <div>Div1
      <p>Parrafo dentro de div</p>
      <h2>Titulo 2 en div</h2>
    </div>
    <h2>Titulo 2</h2>
    <ul>
      <li>Elemento 1 <a href="...">Link1</a></li>
      <li>Elemento 2
        <ol>
          <li id="prim">Primero</li>
          <li id="seg">Segundo</li>
        </ol>
      </li>
    </ul>
  </body>
</html>
```

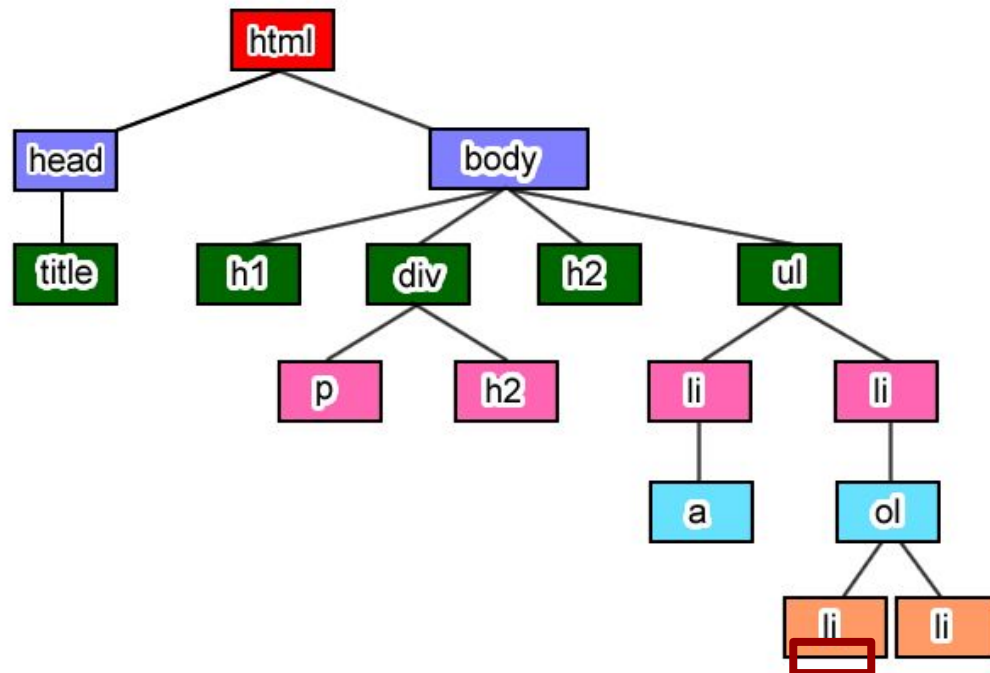


```
document.querySelector('#prim').innerHTML = "1- Primero";
```

Al documento le pido el elemento con ID prim, y a su contenido

DOM - Edición

```
<!DOCTYPE html>
<html>
  <head>
    <title>Ejemplo Arbol</title>
    <link rel="stylesheet" href="estilo.css">
  </head>
  <body>
    <h1>Titulo 1</h1>
    <div>Div1
      <p>Parrafo dentro de div</p>
      <h2>Titulo 2 en div</h2>
    </div>
    <h2>Titulo 2</h2>
    <ul>
      <li>Elemento 1 <a href="...">Link1</a></li>
      <li>Elemento 2
        <ol>
          <li id="prim">1- Primero</li>
          <li id="seg">Segundo</li>
        </ol>
      </li>
    </ul>
  </body>
</html>
```



```
document.querySelector('#prim').innerHTML = "1- Primero";
```

Al documento le pido el elemento con ID prim, y a su contenido se lo piso con un nuevo valor

Obtener nodos del DOM

- Se pueden obtener elementos del DOM consultando por un ID, nombre, clase o un selector.
- Por ahora solo vamos a acceder a elementos mediante IDs

```
let elem = document.getElementById("identificador");
```

O la versión de ES6 que usa un selector CSS:

```
let elem = document.querySelector("#identificador");
```



Leer/Editar el DOM

Las propiedades del DOM (HTML) se pueden leer/editar desde Javascript.

Ejemplo de editar el **contenido**:

```
let unDiv = document.querySelector("unDiv");  
unDiv.innerHTML = "Contenido Nuevo";
```

Ejemplo de editar un **atributo**:

```
let lampImg = document.querySelector("lamp");  
let lampImgAnterior = lampImg.src;  
lampImg.src = "foto.png";
```

Los input tienen el atributo value, que vale lo que tiene escrito por el usuario en la interfaz.

```
let userIn = document.querySelector("#input");  
console.log(userIn.value);
```

Concatenar Strings

El + concatena las cadenas de texto.

"HOLA" + "ALUMNOS"

es lo mismo que

"HOLAALUMNOS" (sin espacio)

Tiene más sentido hacerlo con variables:

let nombre = ...

"HOLA " + nombre //genera el saludo con mucho espacio

Resultado

En negrita marcado lo nuevo

EVENTO INPUT

```
<input type="text" id="txtNombre" oninput="ActualizarSaludo()" />
```

```
<p id="txtSaludo">ACA VA EL SALUDO</p>
```

DA NOMBRE A LOS NODOS

```
function ActualizarSaludo() {
```

```
  //lee el nombre
```

PIDE NODO

```
  let nodoInput = document.getElementById("txtNombre");
```

```
  let nombre = nodoInput.value;
```

LEE VALOR

```
  //lo muestra en consola (opcional, para debug)
```

```
  console.log(nombre);
```

```
  //lo muestra en el DOM
```

PIDE NODO

```
  let nodoSaludo = document.getElementById("txtSaludo");
```

```
  nodoSaludo.innerHTML = "Hola " + nombre;
```

ESCRIBE VALOR

```
}
```

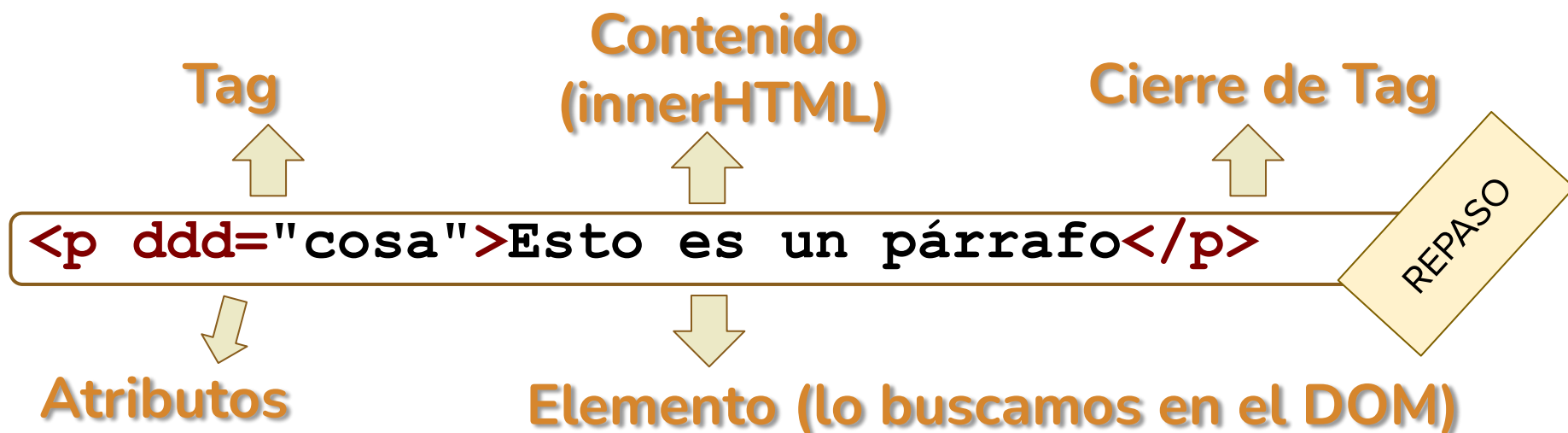
DEMO

Resumen DOM

Las propiedades del DOM (HTML) se pueden leer/editar desde Javascript.

Estas líneas resumen todo lo que van a necesitar en esta etapa.

```
let unDiv = document.getElementById("unDiv");  
unDiv.innerHTML = "Contenido Nuevo";
```



Naming

- Variables:
 - se nombran con sustantivos
- Funciones:
 - Comienzan con verbos
 - En el caso de funciones booleanas, deben comenzar con "is", ej: isValid()
- **Usar nombres descriptivos**
 - nunca son demasiado largos!
- Evitar nombres sin significado como "aux" y "temp"



Variables

- Las variables pueden declararse de forma implícita.
- La primera vez que uso una variable se declara “automáticamente”.

```
numero = 2; //declaró variable número mágicamente
```

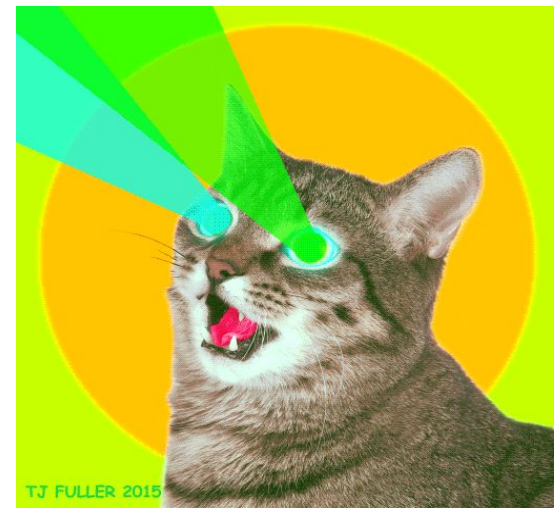
```
let nombre = "Pepe";
```

```
...
```

```
nombbre = "Juan"; //error, tipeo mal la variable
```

```
//crea una variable global nueva
```

```
alert(nombre) //imprime Pepe
```



Use Strict

- Es una buena práctica escribir al comenzar un archivo Javascript

`"use strict";`



- Convierte en obligatoria la declaración de variables
- Restringe otros posibles errores de sintaxis



“El use strict hace que el navegador se ponga la gorra!”

Resumen

Aprendimos a

- Que es el DOM
- Editar la página web desde Javascript
 - Manipular el DOM
- Usar otro evento (que no es onclick)



Debug



Debug

Debug es el término conocida para encontrar los errores de un programa y solucionarlos.

DEBUG

DE(lete) **BUG**

“Eliminar Bichos”

También se dice “debug” al proceso de hacer un seguimiento del código en la máquina y comprender mejor cómo funciona.

Veamos cómo se ejecutan las cosas con la consola



¿Qué vamos a aprender?

- Cómo imprimir en la consola
- Cómo usarla para ver cómo se están ejecutando las cosas

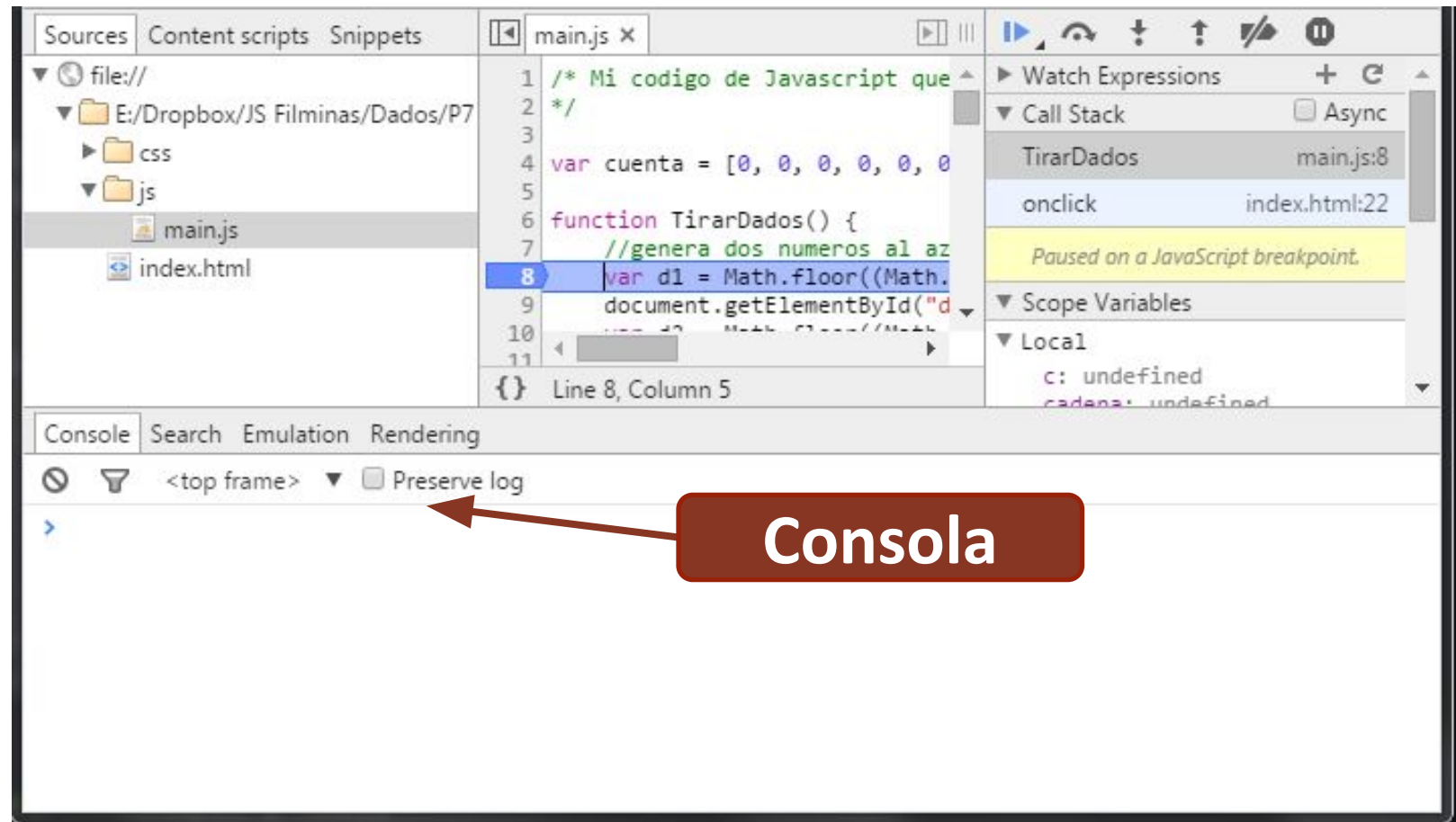
Ver la consola



Menú Chrome > Más herramientas > Herramientas Desarrollador

Atajo de Teclado:

- Ctrl + Shift + I
- F12





- `console.log("cadena")` imprime algo en la consola del debugger.

Resultado

```
console.log("Paso 1: declarando funciones");
function ActualizarSaludo() {
    //lee el nombre
    let nodoInput = document.getElementById("txtNombre");
    let nombre = nodoInput.value;
    //lo muestra en consola (opcional, para debug)
    console.log(nombre);
    //lo muestra en el DOM
    let nodoSaludo = document.getElementById("txtSaludo");
    console.log("Paso 3: Saludo:" + nodoSaludo.innerHTML);
    nodoSaludo.innerHTML = "Hola " + nombre;
    console.log("Paso 4: Saludo:" + nodoSaludo.innerHTML);
}
console.log("Paso 2: continua ejecución");
```



DEMO

Programa correcto

Si un programa no compila, no se va a ejecutar
Los navegadores no muestran estos errores al usuario



ABRI LA CONSOLA PARA VERLOS!



**SIEMPRE PROBA EL CÓDIGO JAVASCRIPT
CON LA CONSOLA ABIERTA**



Prende y Apaga

Prende y Apaga: consigna

Consigna:

- Botones para “prender” y “apagar” la página que
 - Cambiar el color de fondo (blanco/negro)
 - Cambiar la imagen de una lámpara

¿Qué vamos a aprender?

- Cambiar el estilo desde Javascript
- Quitar el Javascript del archivo HTML

Eventos

Asignar handlers en HTML es mala práctica porque mezcla HTML y JavaScript.

Esto es código Javascript adentro del .HTML!!!

```
<button type="button" onclick="verificarFormulario();" >
```

Enviar

```
</button>
```



No
recomendado

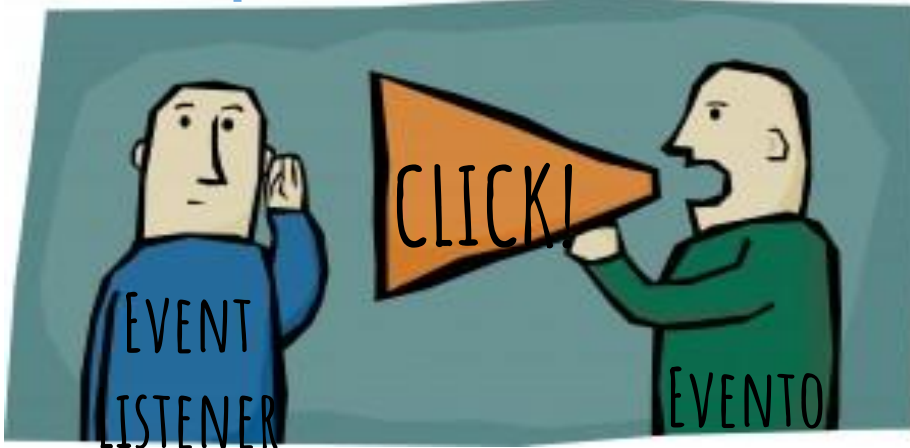
En **ES6**, primero buscamos el elemento y luego le asociamos el handler

HTML

```
<button type="button" id="btn-enviar">Enviar</button>
```

JS

```
let btn = document.getElementById("btn-enviar");  
btn.addEventListener("click", verificarFormulario);
```



La función sin ()
No la está llamando en esta sentencia
Le dice que función llamar (después)

Está pendiente de escuchar un evento para ejecutarlo solo en el momento en que suceda

```
<button type="button" onclick="verificarFormulario();">
```

Enviar

```
</button>
```



Es lo mismo, pero en lugar de escribir Javascript en el HTML, **desde JS** pedimos el nodo del DOM y le agregamos el handler del evento. El código Javascript queda en el JS!

```
<button type="button" id="btn-enviar">Enviar</button>
```

```
let btn = document.getElementById("btn-enviar");
```

```
btn.addEventListener("click", verificarFormulario);
```



https://developer.mozilla.org/en-US/docs/Web/API/EventTarget/addEventListener#Why_use_addEventListener

Why use `addEventListener()`?

`addEventListener()` is the way to register an event listener as specified in W3C DOM. The benefits are as follows:

- It allows adding more than a single handler for an event. This is particularly useful for [AJAX](#) libraries, JavaScript modules, or any other kind of code that needs to work well with other libraries/extensions.
- It gives you finer-grained control of the phase when the listener is activated (capturing vs. bubbling).
- It works on any DOM element, not just HTML elements.

The alternative, [older way to register event listeners](#), is described below.

Editar estilo desde Javascript



Para editar los estilos desde Javascript podemos modificar las clases de los elementos del DOM

```
// div es una referencia a un elemento <div>
div.classList.add("clase");
div.classList.remove("clase");
div.classList.toggle("clase");
```

```
alert(div.classList.contains("clase"));
```

```
// agregar o quitar múltiples clases
div.classList.add("clase-1", "clase-2", "clase-3");
```

```
// estilos vía JS (Mala práctica)
document.getElementById("id").style.font-size = "20px";
```



Buena Práctica!

Cambiar estilos con clases



<https://codepen.io/webUnicen/pen/qmVoMV>

Resumen: conexión entre HTML, CSS y JS

- La única conexión debieran ser las clases
- Las clases son el contrato entre los tres lenguajes
- En lo único que se tienen que poner de acuerdo es en qué significa cada clase
 - HTML le va a poner las clases a lo que corresponda
 - CSS va a hacer que se vea como dice el acuerdo
 - JS va a hacer que se comporte como dice el acuerdo
- El nombre de la clase debe ser representativo de este contrato

Resultado

```
"use strict";  
function prender() {  
    document.querySelector("body").classList.add("prendido");  
    document.querySelector("body").classList.remove("apagado");  
    document.querySelector("#lampara").src = "img/lampara_on.png";  
}  
function apagar() {  
    document.querySelector("body").classList.remove("prendido");  
    document.querySelector("body").classList.add("apagado");  
    document.querySelector("#lampara").src = "img/lampara_off.png";  
}  
document.querySelector("#btnPrender")  
    .addEventListener("click", prender);  
document.querySelector("#btnApagar")  
    .addEventListener("click", apagar);
```

CAMBIA CLASES Y SRC DE LA IMAGEN

DEMO

Extra

Funciones con parámetros

```
function saludar(nombre)
{
  alert("Hola" + nombre);
}
```

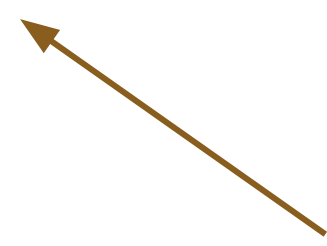
parámetro
formal



- Al llamarlo debo decirle el valor que tiene “nombre”

```
saludar("Javier1");
saludar("Javier2");
```

parámetro real,
o “lo que le
paso como
parametro”



- En los eventos no podemos tener parámetros
- El único parámetro es uno que el navegador le pasa a la función con datos del evento.

JS

```
let inputEmail, inputConsulta...
```

```
...
```

```
btn.addEventListener("click", sumarCinco)
```

```
function sumarCinco(evento) {
```

```
    sumar(5);
```

```
}
```

```
function sumar(cantidad)
```

```
{
```

```
...
```



Prendamos y apagamos con el teclado

- Detectar “key-press”
- Según la tecla prender o apagar

Resumen

Aprendimos a

- Cambiar el estilo desde Javascript
 - `classList`
- Agregar eventos desde Javascript
 - Y así poder no tener nada de Javascript en el archivo HTML



Resumen!

RESUMEN JAVASCRIPT

- Agregamos script con ruta al archivo JS al HTML
- Obtenemos el/los elementos del DOM que va a tener eventos
- Le decimos el/los elemento que función debe ejecutar en que evento (addEventListener)
- Programamos las funciones
 - Obtenemos los datos que necesitamos del dom
 - Procesamos
 - Actualizamos el DOM



El código Javascript no debería aparecer en el archivo HTML
El código CSS no debería aparecer en el archivo JS

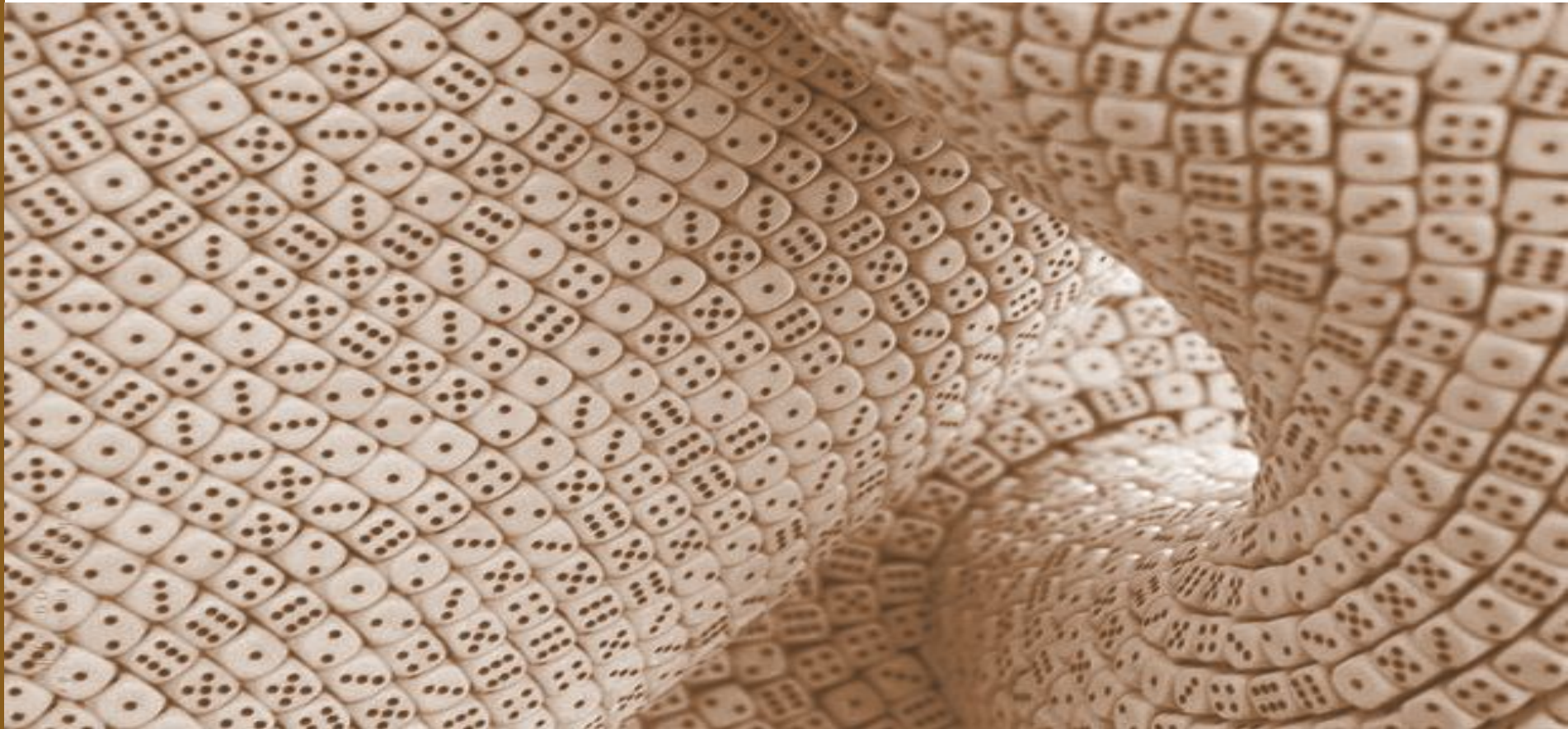


Dados

Ejemplo de
la práctica

Ejemplo

Hacer una aplicación web que al apretar un botón simule el lanzamiento de dos dados, sumarlos y que muestre en el HTML si salió 8.





**KEEP
CALM
AND
GET
ORGANIZED**

Qué vamos a aprender

Qué vamos a aprender?

- Generar un número aleatorio utilizando funciones matemáticas de JS
- Uso de estructuras de control
 - if
- Cómo comparar y formas de comparar

¿Qué tenemos que hacer?

Lo primero que tenemos que hacer es analizar el problema y separarlo en partes pequeñas que sean más fáciles de resolver

Analicemos el problema

- Crear el html
- Un boton que sea lanzar dado
- Dos imagenes para mostrar los dados
- funcion Lanzar
- Verificar con el IF el resultado

Random

- Para generar números al azar podemos usar `Math.random()`;

Así obtenemos un número al azar entre 0 y 1 (0 inclusive, 1 no).
Las caras de un dado son del 1 al 6, ¿cómo hacemos?



Ejemplo: <http://codepen.io/webUnicen/pen/RaMpoV>

Más info: <https://developer.mozilla.org/en-US/docs/Web/JavaScri...>

Random

- Podemos utilizar también la función matemática

```
Math.floor();
```

Que redondea al menor entero

```
Math.floor((Math.random() * 6) + 1);
```

```
0 < Math.random() * 6 < 5.999
```



<http://codepen.io/webUnicen/pen/RaMpoV>

<https://developer.mozilla.org/en-US/docs/Web/JavaScript...>

Random

Arrancamos del random

$0 \leq \text{Math.random}() \leq 0.99999999$

Multiplico por 6

$0 \leq \text{Math.random}() * 6 \leq 5.99999999$

Sumo 1

$1 \leq \text{Math.random}() * 6 + 1 \leq 6.99999999$

Aplico floor

$\text{Math.floor}(\text{Math.random}() * 6 + 1) \in \{1, 2, 3, 4, 5, 6\}$



<http://codepen.io/webUnicen/pen/RaMpoV>

<https://developer.mozilla.org/en-US/docs/Web/JavaScript...>

Mostrar imágenes de los dados



Debo mostrar la imagen del dado que corresponde al número que salió.

Nombre de mis imágenes



dado1.png



dado6.png



¿ Dónde están esas imágenes ?

<http://tudai1-1.alumnos.exa.unicen.edu.ar/web-1/material/archivos/>

<http://tudai1-1.alumnos.exa.unicen.edu.ar/web-1/material/archivos/dado1.png>

Mostrar imágenes de los dados



Debo generar un **string** con el nombre de la imagen que se corresponda con el número del dado que salió.

Concatenamos sabiendo que d1 y d2 son las variables a las que se les asignan los valores de los dados y modificamos en el documento la fuente de la imagen

```
document.getElementById("dado1").src = "images/dado"+d1+".png";
```



¿ Ruta relativa o externa ?

Si estoy en Codepen o no están las imágenes en mi carpeta images el enlace es externo, debo concatenar usando:

<http://tudai1-1.alumnos.exa.unicen.edu.ar/web-1/material/archivos/>



<https://codepen.io/webUnicen/pen/bvXWxo>

Resolviendo el problema



Subdividimos el problema, primero verifiquemos si la suma de los dados da 8.

Necesitamos utilizar un condicional y comparar

Sintaxis JavaScript - if , else

```
if(condicion) {  
    // codigo SI se cumple  
    // la condicion  
}  
else {  
    // codigo si la condicion  
    // NO se cumple  
}
```



Operadores de comparación

- = Asignación

```
let materia = "Web"
```

- == Igualdad (!=)

- Convierte tipos

```
if( 5 == "5") {  
    //true  
}
```

- === Identidad (!==)

```
if( 5 === "5") {  
    //false  
}
```



Siempre usar ===

- Recordemos que (==) hace conversión automática de tipos
- A veces el resultado de esta conversión no es obvia.
- Evitar esto usando (===)
- Lo mismo para != (usar !==)



Uso de ; en Javascript

- El ; es opcional, pero es una buena práctica usarlo.
- Obligatorio cuando minificas el script.

Siempre usar ;



Resultado

```
let suma = d1 + d2;
```

```
if (suma === 8){ //verificacion si es 8 e incremento cuenta  
    document.getElementById("resultado").innerHTML  
        = "salio el 8";  
}
```



<https://codepen.io/webUnicen/pen/yrLBdr>

Resumen

Aprendimos a

- Generar un número aleatorio en JS
- Uso de estructuras condicionales
- Cómo comparar valores de variables



Extra: 1000 veces!

Resolver el problema



Tirar los dados muchas veces, 1000.

Bucle



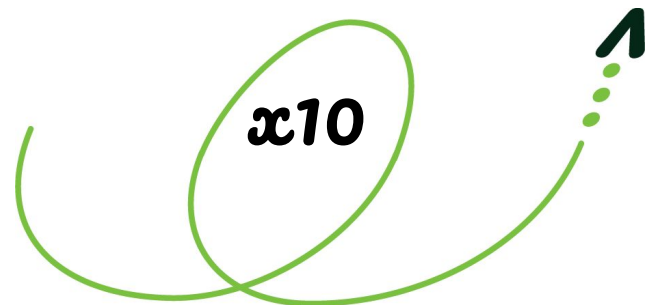
Debemos ejecutar muchas veces la función que arroja los dados.

```
for (let i = 0; i < 10; i++) {  
    //código ejecucion durante el bucle  
}
```

Inicialización: el bucle se ejecuta desde un valor inicial ($i = 0$)

Condición: Se ejecuta siempre que la condición sea verdadera ($i < 10$)

Actualización: En cada pasada del bucle i se modifica, $i++$ es sumarle 1 al anterior



Resultado



<https://codepen.io/webUnicen/pen/jROOOd>

Resumen

Aprendimos a

- Ejecutar N veces las mismas sentencias (iterar)
 - “for”



Más Información

Libros

- Javascript and JQuery : Interactive Front-End Web Development, Jon Duckett Willey 2014
- Learning Web Design: A Beginner's Guide to HTML, CSS, JavaScript, and Web Graphics, Jennifer Niederst Robbins O'Reilly Media 2012
- Standard: <http://standardjs.com/rules.html>
- Tutorial W3 Schools: <http://www.w3schools.com/js/>
- [Javascript from birth to closure](#)

Eventos

- <http://www.elcodigo.net/tutoriales/javascript/javascript5.html>
- <http://dev.opera.com/articles/view/handling-events-with-javascript-es>

AHORA LES TOCA PRACTICAR :D

