

Javascript #2

WEB 1 TUDAI UNICEN 2024



KEEP
CALM
AND
CODE
JAVASCRIPT

**VAMOS A APRENDER CONCEPTOS NUEVOS
SIGUIENDO EJEMPLOS**

Piedra, Papel o Tijera

Ejemplo 1

Problema

Vamos a hacer una app para jugar contra la “computadora” al **piedra, papel o tijera**.



PIEDRA



PAPEL



TIJERA

Creemos un botón, que al apretarlo “simule” la inteligencia de una persona y elija automáticamente una de las tres opciones.

¿Qué vamos a aprender?

- Qué son los arreglos y para que se utilizan
- Declarar un arreglo
- Mostrar un elemento del arreglo

¿Cómo vamos a “simular” que la computadora elija una de las tres opciones disponibles?

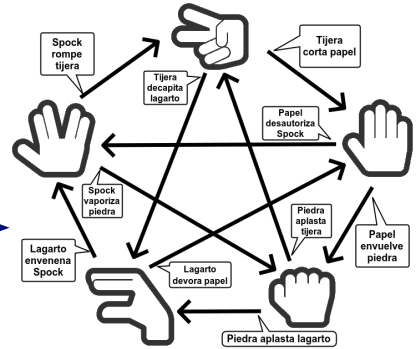
La idea es elegir al azar una de las tres opciones del juego:

Podríamos generar un número al azar y hacer 3 if (o un switch) para saber que opción elegir.

Contra: Si bien tenemos tres opciones puede haber variantes con más opciones. Se hace muy largo el código.

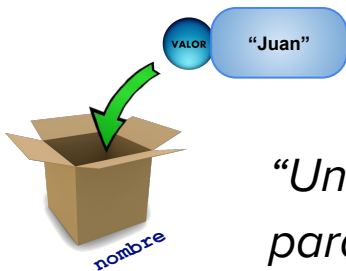
Solución: Hagamos un código que elija al azar de una lista de opciones, sean 3 o la cantidad que sean.

5 opciones



Arreglos

VARIABLE SIMPLES



“Una variable es una caja para guardar un dato.”

ARREGLO



Los arreglos almacenan elementos en una **colección** de datos ordenados.

Son una variable que tiene muchas posiciones.

“En lugar de una caja, ahora tenemos una cajonera.”

Declarar arreglos

Existen maneras de declarar arreglos en JS

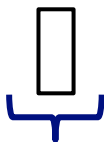
- Mediante la notación de literales

```
let names = [];
```

- Mediante su constructor (no tan utilizada)

```
let names = new Array();
```

**Las dos generan un
arreglo vacío**



A diagram showing a vertical rectangle representing an array, with a blue bracket underneath it. The word 'names' is written below the bracket.

names

**no tiene
posiciones**

Declarar arreglos

También se pueden declarar arreglos con elementos pre-cargados

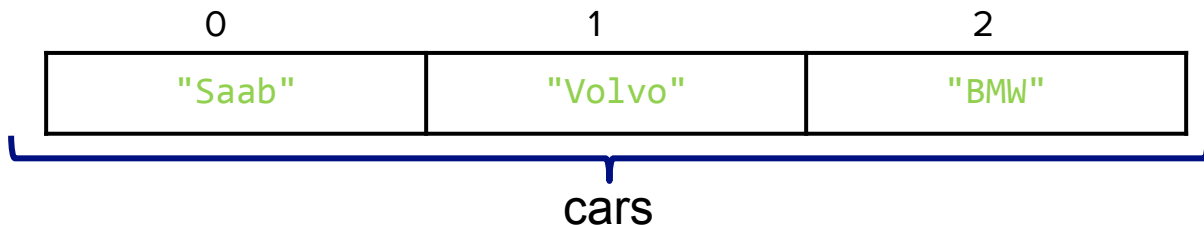
- Mediante la notación de literales

```
let cars = ["Saab", "Volvo", "BMW"];
```

- Mediante su constructor (no tan utilizada)

```
let cars = new Array("Saab", "Volvo", "BMW");
```

tiene 3 posiciones (0, 1, 2)



Arreglos - Acceso

Acceder a un arreglo

Los elementos dentro de un arreglo se acceden indicando su posición dentro del mismo (**índice**)

```
let cars = ["Saab", "Volvo", "BMW"];
```

MUY NUEVO CON MENOS SOPORTE EN NAVEGADORES VIEJOS

- Mediante operador []

```
let car = cars[0];
```

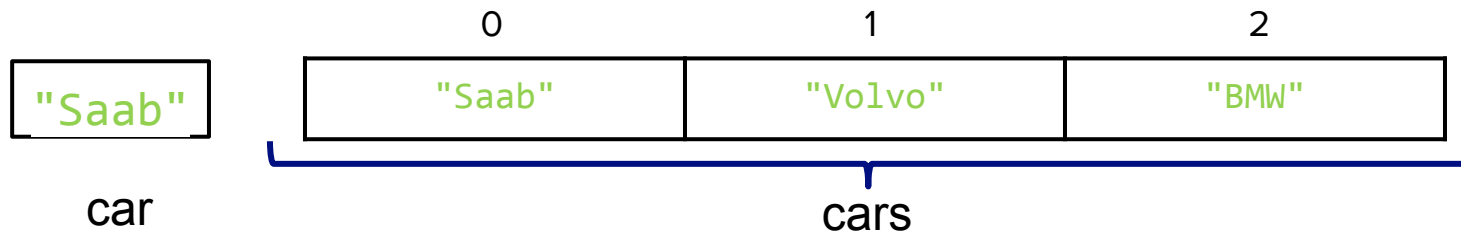
- Mediante metodo .at()

```
let car = cars.at(0);
```

```
console.log(car); // "Saab"
```

```
console.log(cars[1]); // "Volvo"
```

```
console.log(cars[2]); // "BMW"
```



Arreglos - Tamaño

Tamaño de un arreglo

Se puede utilizar la propiedad “length” del arreglo para saber su tamaño

```
let cars = ["Saab", "Volvo", "BMW"];  
let carsSize = cars.length; // 3
```

Resolviendo el problema

En el Piedra, Papel y Tijera, podemos crear un arreglo para almacenar la lista de opciones que tiene la computadora para elegir.

```
let opciones = ["piedra", "papel", "tijera"];
```

-> Luego ejecuto un numero random entre 0 y 2 para elegir al “azar” una de las opciones y con esto simulo que la computadora eligió una.

Random

REPASO

Podemos utilizar también la función matemática para elegir un número al azar entero

```
Math.floor(Math.random() * 3) + 1;
```

En este caso tenemos 3 opciones, o mejor aún `opciones.length` opciones

$0 < \text{Math.random()} * 3 < 2.999$

Como los arreglos empiezan en 0 no hace falta sumar 1



Solución - Piedra, Papel y Tijera

```
"use strict"
```

```
let btn = document.querySelector('#btn-play');  
btn.addEventListener('click', jugar);
```

```
// arreglo con todas las opciones posibles  
let opciones = ['piedra', 'papel', 'tijera'];
```

```
function jugar() {  
  // numero random entre 0 y el tamaño del arreglo  
  let random = Math.floor(Math.random() * opciones.length);  
  let opcion = opciones[random];  
  
  // escribimos el resultado en el html  
  document.querySelector('.result').innerHTML = opcion;  
}
```

Resumen

Aprendimos

- Arreglos
- Acceder a los elementos del arreglo



Aplicación de sorteos

Ejemplo 2

Problema

Tenemos que hacer una app para hacer sorteos donde podamos agregar todas las personas que queramos y nos elija un ganador al azar.

Creemos un formulario y una lista vacía para ir completando con lo que carga el usuario. Además tiene que tener un botón de reset.

¿Qué vamos a aprender?

- Agregar elementos a un arreglo
- Recorrer un arreglo
- Vaciar un arreglo

Escribe los participantes del sorteo:
Debe tener como mínimo dos participantes.

Juan

Ana

Luis

Marcelo

Martin

Andrea

Sofia

Sonia

Sergio

Juana

Analia

Nº de premios:

Introduzca cada nombre en una línea diferente

Resultados del sorteo:

Puesto 1: Marito
Puesto 2: Andrea
Puesto 3: Luis

Modelo vs Vista

¿Por qué necesito un arreglo si puedo ir insertando los nombres directo en el HTML?

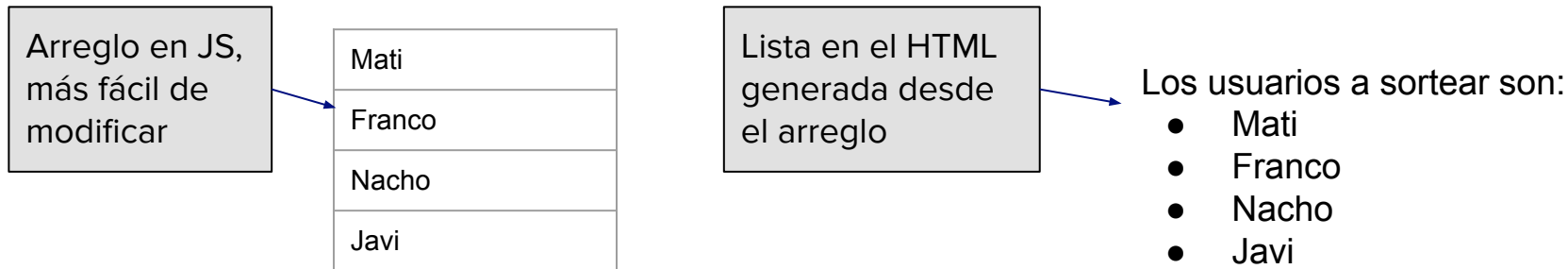
- **Usar el DOM para guardar todo:**
 - todo es string (no puedo guardar otros tipos)
 - difícil de recorrer para buscar
 - imposible en aplicaciones con muchos datos

No es responsabilidad del DOM guardar los datos, sólo mostrarlos!

Modelo vs Vista

Al aumentar la complejidad de mi aplicación, es casi obligatorio guardar el **estado de los datos** en Javascript, y luego representarlo (dibujarlo/escribirlo) en el DOM.

Las variables en Javascript son nuestro “**modelo**” de datos y después se muestra en el DOM, llamado **vista** .



Arreglos: Agregar y borrar elementos

```
let frutas = ["manzana", "pera", "naranja"];
```

Agregar un elemento al final

```
frutas.push("mandarina"); // ["manzana", "pera", "naranja", "mandarina"];
```

Agregar un elemento al inicio

```
frutas.unshift("limón"); // ["limón", "manzana", "pera", "naranja", "mandarina"];
```

Borrar último elemento

```
frutas.pop(); // ["limón", "manzana", "pera", "naranja"];
```

Vaciar el arreglo

```
frutas = []; // lo vuelvo a definir pero vacío
```

Recorriendo Arreglos

Los arreglos se pueden recorrer utilizando alguna estructura de repetición, por ejemplo “for”.

```
let frutas = ["manzana", "pera", "naranja"];
```

// foreach de ES6

```
for (let elem of frutas) {  
    console.log(elem);  
}
```

Itera por el elemento
directamente



// método clásico

```
for (let i=0; i<frutas.length; i++) {  
    let elem = frutas[i];  
    console.log(elem);  
}
```

Resolviendo el problema

Creo un arreglo para **almacenar la lista de personas** que ingresa el usuario desde el formulario.

```
let personas = [];
```

0	"Mati"
1	"Franco"
2	"Nacho"
3	"Javi"
4	"Otro nombre"

Resolviendo el problema

Creo el HTML

- 3 botones (Agregar, Limpiar Todos, Borrar Último)
- 1 input para agregar nombres
- la lista vacía para ir mostrando los nombres

Nombre:

- Mati
- Franco
- Nacho

Creo 4 funciones: `agregar()`, `reset()`, `borrarUltimo()` y `mostrar()`

Llamo a `mostrar()` cada vez que se modifica el arreglo

Solución

```
let nombres = [];
```

variable global para el estado (el modelo)

```
function agregar() {  
  let nombre = document.getElementById("nombre").value;  
  nombres.push(nombre);  
  mostrar();  
}
```

una función por evento

```
function reset() {  
  nombres = [];  
  mostrar();  
}
```

una función que muestre la lista para
evitar duplicar código en cada evento

```
function mostrar() {  
  let lista = document.querySelector("#listado");  
  lista.innerHTML = ""; //borro todo lo que haya  
  for(let n of nombres){  
    lista.innerHTML += `<li>${n}</li>`;  
    //lista.innerHTML = lista.innerHTML + "<li>" + n + "</li>";  
  }  
} //...los event listener
```

generamos HTML desde JS

```
function borrarUltimo() {  
  nombres.pop();  
  mostrar();  
}
```

```
function sortear() {  
  let pos = Math.floor(  
    Math.random() * nombres.length);  
  let g = document.querySelector("#ganador");  
  g.innerHTML = nombres[pos];  
}
```

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

DEMO

Resumen

Aprendimos

- Manipulaciones sencillas sobre arreglos (agregar, vaciar)
- Recorrer arreglos



App Contador de Pasos

Contador de pasos

Queremos realizar una app que lleve la cuenta de la cantidad de pasos que realizo por día:

- En la página web se muestra la cantidad de pasos en el día.
- Hay un botón para incrementar la cantidad de pasos
- Hay un botón para decrementar la cantidad de pasos
- Hay un input de texto para sumar muchas pasos en una sola acción



¿Qué vamos a aprender?

- **tipos de variables**

- Al escribir el texto, voy a tener un texto y lo quiero sumar como número.

- **funciones anónimas**

- **parámetros en las funciones**

Tipos de Datos

Las variables pueden tener tipos:

- String
- Number
- Boolean
- Null
- Undefined
- Object
 - Function
 - Array
 - Date
 - Expresiones Regulares (RegExp)

Tipado de Variables - Tipos

El **tipado estático** nos obliga a definir desde el principio el tipo de una variable. Lenguajes con tipado estático son C++, Java, C# (casi) entre otros.

El **tipado dinámico** nos da la facilidad de no definir los tipos al declarar una variable, algunos ejemplos son PHP, JavaScript, Grooby, Phyton, entre otros.

¿Se les ocurren pros y contras?

Tipos

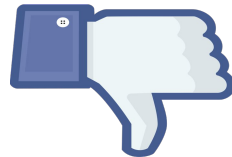
Javascript tiene **tipos dinámicos**.

- Una misma variable puede cambiar de tipo.
- Puede causar confusiones (y errores que no encuentro durante horas).

```
let nombre = "Pepe"; //nombre es un string
```

```
...
```

```
nombre = 2; //nombre es un int (cambia tipo)
```



Mala
práctica

Conversión de tipos

- **Cuidado** con los **tipos**, son dinámicos y no saber de qué tipo es una variable puede cambiar el resultado.

```
5 == "5" //true
```

```
"1" + 2 + 3; //"123"
```

http://www.w3schools.com/jsref/jsref_parseint.asp

```
//Conversion manual de tipos
```

```
parseInt("1", 10) + 2 + 3; //6
```

- **ES6** introduce una nueva forma de trabajar con Strings

```
'<p>Vos sos '+nombre+' '+apellido+'.</p>'
```

```
`<p>Vos sos ${nombre} ${apellido}.</p>`
```

Undefined

- ***undefined*** es un tipo fundamental en Javascript
- Las variables sin inicializar valen ***undefined***
- Variables y miembros sin declarar valen ***undefined*** (salvo que uses “use strict” que causa una falla)
- Las funciones siempre devuelven un valor, si no tienen valor de retorno devuelven ***undefined***



0



null



undefined

Escribiendo código JavaScript más limpio

- ✓ **Funciones con parámetros**
- ✓ **Funciones anónimas**

Parámetros

Podemos hacer el código más **genérico y reusable** utilizando parámetros y devolviendo valores.

```
// función específica para sumar 2 + 4
```

```
function sumarDosMasCuatro {  
    let suma = 2 + 4;  
    console.log(suma);  
}
```

```
// función genérica para sumar dos valores cualquiera
```

```
function sumar(parametro1, parametro2) {  
    return parametro1 + parametro2;  
}
```

```
// la llamamos con los valores que queremos
```

```
let resultado = sumar(2, 8); //devuelve 10  
console.log(resultado);
```

El primer parámetro va a valer 2, el segundo va a valer 8, por lo que la suma dará 10

Los parámetros son la **entrada** del código, y el valor que devuelve es la **salida**.



Funciones anónimas

Se usan para no crear tantas funciones que se usan en un solo lugar

- Es una función sin nombre que se escribe directamente donde quería pasar parámetros.
- En este caso encapsula a la función que si pasa parámetros.

~~`btn.addEventListener("click", sumar(20, 50));`~~

```
btn.addEventListener("click", function() {  
    let valor1 = 20;  
    let valor2 = 50;  
    let resultado = sumar(valor1, valor2)  
});
```

Declaramos una **función anónima**, para poder llamar a una **función con parámetros**

Resolver el problema

Asignar eventos a los 3 botones

```
<button id="btn-decrementar">-</button>
<button id="btn-incrementar">+</button>
<button id="btn-agregar">AGREGAR</button>
```

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

JS + Función anónima:

```
document.querySelector('#btn-incrementar').addEventListener('click', function(e) {
    incrementar(1)
});
```

```
document.querySelector('#btn-decrementar').addEventListener('click', function(e) {
    incrementar(-1);
});
```

```
document.querySelector('#btn-agregar').addEventListener('click', incrementarManual);
```

Resumen

Aprendimos

- Tipos
- Conversiones de tipos
- Funciones anónimas
- Funciones con parámetros



Variables globales y
DOMContentLoaded

Variables Globales

En el ejemplo anterior, la variable **contador** es una variable global. ¿No habíamos dicho que es mala práctica?

Variables Globales

- Se puede acceder desde cualquier lado de la aplicación. (incluso se puede ver o modificar desde la consola del navegador)
- Si incluyo dos archivos JS podrían usar el mismo nombre de la variable y tener errores.

Evitar variables y funciones globales

Existen varias maneras de evitar variables globales en javascript.

Dos de las más utilizadas son:

- Crear un ámbito local para limitar el alcance y dentro declarar todas las variables y funciones. Se puede hacer utilizando el evento DOMContentLoaded
- Crear módulos Javascript ECMAScript (no los utilizamos en esta materia) <https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Modules>

DOMContentLoaded

Podemos configurar todos los eventos una vez que ya se cargo el DOM. De este modo nos **aseguramos** que estén cargados todos los elementos del DOM antes de manipularlos



```
document.addEventListener("DOMContentLoaded", iniciar);  
  
function iniciar(){  
    let contador = 0; // ESTA VARIABLE ES LOCAL EN ESTE AMBITO  
  
    // código de inicialización de eventos  
  
}
```

Evitar variables y funciones globales

```
"use strict";
document.addEventListener('DOMContentLoaded', iniciar);

function iniciar() {
    let btn = document.getElementById("btn-click");
    btn.addEventListener("click", sumar);

    let contador = 0;

    function sumar() {
        //incrementa el valor de contador
        contador++;
        //es lo mismo que contador = contador + 1
        let valor = document.getElementById("spanContador");
        valor.innerHTML = contador;
    }
}
```

“contador” no puede usarse desde más afuera

**EVERY TIME YOU CREATE JAVASCRIPT GLOBAL
VARIABLES**



**GOD KILLS A
KITTEN**

memegenerator.net

Resumen

Aprendimos a

- Limitar desde donde son accesibles las variables
 - DOMContentLoaded



Reloj

Reloj - Bomba

Simular la cuenta regresiva de una bomba.

Con un botón activarla y dejar 5 segundos para escapar y comenzar la cuenta regresiva.

El valor de la cuenta regresiva se ingresa por un input



Qué vamos a aprender

Qué vamos a aprender?

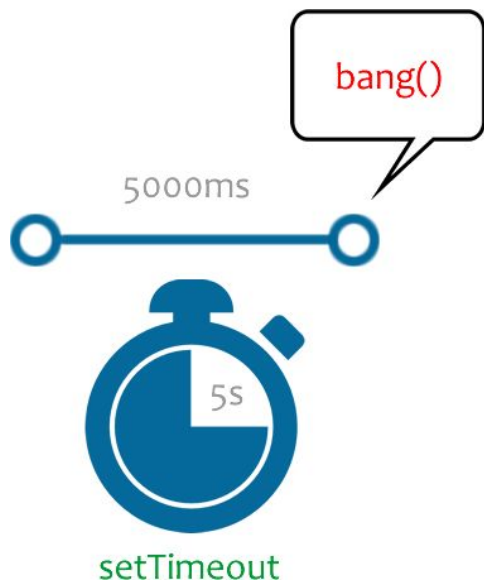
- Ejecutar eventos diferidos en tiempo, o retardados
- Ejecutar eventos que se repiten en intervalos de tiempo hasta que hagamos un reset.

Eventos de tiempo

Se puede programar un evento, para ejecutar una función dentro de M milisegundos.

//dispara (ejecuta bang) en 5 segundos

let timer = setTimeout(bang, 5000);



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

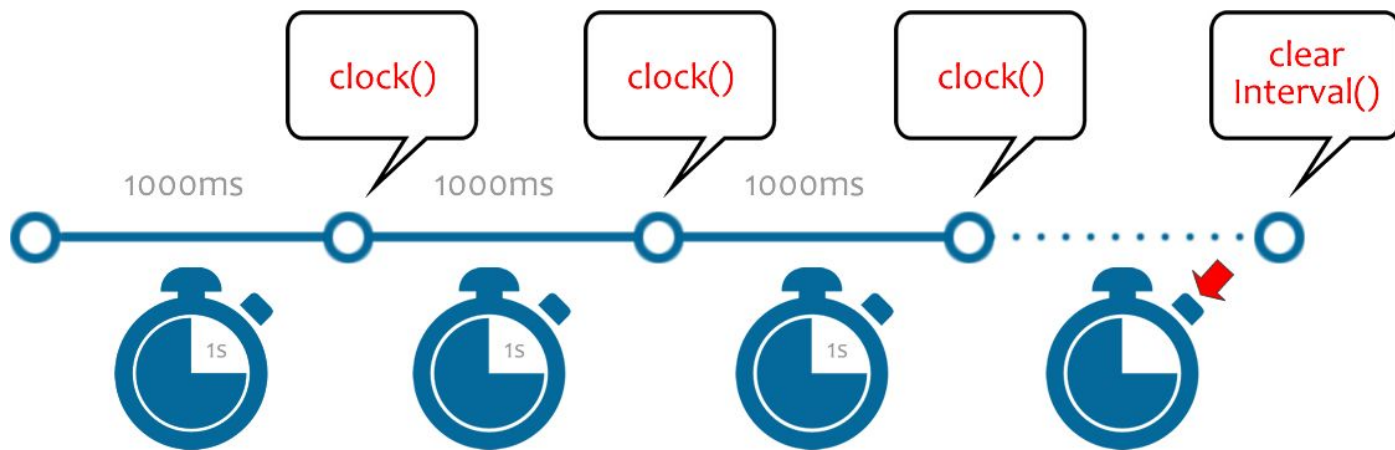
Eventos de tiempo

```
let timer = setInterval(clock, 1000);
```

```
...
```

```
clearInterval(timer);
```

setInterval llama a la función cada 1000 milisegundos, hasta que se limpie el intervalo.



setInterval

Resultado

Cuenta Regresiva

```
function cuentaRegre(){  
  let intervalo = setInterval(function() {  
    if (i === 0) {  
      clearInterval(intervalo); // limpio intervalo  
      alert('BOOOOOOM!!');  
    }  
    else {  
      i--;  
    }  
  }, 1000);  
}
```



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

Implicancias de eventos de tiempo

[TBC]

Resumen

Aprendimos

- Usar temporizadores e intervals



Javascript - Buenas prácticas

Separar los "event handlers"

// Mal hecho

```
function handleClick(event) {  
    let popup = document.getElementById("popup");  
    popup.style.left = event.clientX + "px";  
    popup.style.top = event.clientY + "px";  
    popup.className = "reveal";  
}
```

// Mejor, pero sigue estando mal

```
function handleClick(event) {  
    showPopup(event);  
}  
  
function showPopup(event) {  
    let popup = document.getElementById("popup");  
    popup.style.left = event.clientX + "px";  
    popup.style.top = event.clientY + "px";  
    popup.className = "reveal";  
}
```

// La solucion correcta

```
function handleClick(event) {  
    showPopup(event.clientX, event.clientY);  
}  
  
function showPopup(x, y) {  
    let popup = document.getElementById("popup");  
    popup.style.left = x + "px";  
    popup.style.top = y + "px";  
    popup.className = "reveal";  
}
```





**Las consultas al DOM son MUY lentas.
Editarlo también.**



Usar Cache de Selectores

- Salvar los resultados de selectores optimizan mucho el código.

```
function sinCache () {  
    let i, val;  
    for(i=0; i<50; i++){  
        val =  
document.getElementById('title').innerHTML  
TML  
    }  
}
```

```
function conCache () {  
    let i, val;  
    let h1 =  
document.getElementById('title');  
    for(i=0; i<50; i++){  
        val = h1.innerHTML;  
    }  
}
```

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

DEMO

JS y sus detalles

Obtener nodos del DOM

- Se pueden obtener elementos del DOM consultando por un ID, nombre, clase o un selector.
- Podemos obtener como resultado de uno o múltiples elementos del DOM

Retorna un nodo

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

```
let singleElem = document.querySelector(".myclass");
```

sin el punto



Selector de CSS



Retorna uno o más

```
let manyElements = document.getElementsByClassName("myclass");
```

```
let manyElems = document.querySelectorAll(".myclass");
```

Comparaciones cortas en JS

?

- Operador ternario (lf más corto):

```
let tmp = (bool) ? 1 : 2
```

es igual que

```
let tmp = 0;  
if(bool)  
    tmp=1;  
else  
    tmp=2;
```

- Solo sirve cuando es un valor que se asigna en la misma variable en las dos ramas.

Falsey evaluation

En Javascript, hay diferentes cosas que al convertirla a `bool`, se transforma a `false` automáticamente.

```
null == undefined == 0 == false == ""
```

No es tan así, pero es una buena simplificación.

```
let a = null; let b; //undefined
```

```
let c = 0; let d = false;
```

```
let e = "";
```

```
if (a), if (b), if (c), if (d), if (e) //false
```



Puedo pasarme horas revisando un bug en una comparación, que era por un `undefined` en la variable.

Falsey Evaluation

- Por costumbres de otros lenguajes, es normal escribir condicionales tipo C
- Los condicionales JS son más cortos y eficientes

```
1  /* C-style conditional */
2  if (val != null && val.length > 0){
3      ...
4  }
5
6  /* JavaScript style conditional */
7  if (val) {
8      ...
9  }
10
```

Largo de una cadena

Existen muchas funciones que ya trae Javascript

Para calcular el largo de una cadena puedo usar:

```
let largo = str.length("cadena");
```

El valor calculado se **devuelve** y debe guardarse en una variable

Ejercicios

Ejercicios

Ejercicio #1

- Utilizando lo visto en esta clase, crear una función Javascript que oculte y muestre un div que contiene información.
- Analizar cómo modificar el ejercicio para que sea un código reutilizable (poder poner muchos botones que oculten o muestren un div respectivo)

Ejercicio #2

- Tengo una lista de tareas, y quiero dinámicamente (sin refrescar la página) agregar tareas.

AHORA LES TOCA PRACTICAR :D



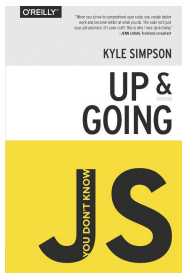
Más Información

Libros

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

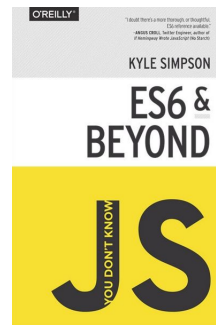
O'Reilly “You don’t know JS, up going”

<https://github.com/getify/You-Dont-Know-JS/blob/master/up%20&%20going/README.md#you-dont-know-js-up--going>



O'Reilly “You don’t know JS, ES6 and beyond”

<https://github.com/getify/You-Dont-Know-JS/tree/master/es6%20%26%20beyond>



Eventos

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