

# Javascript Parte 3

---

# JSON

---

# Organización de los datos - Ejemplo

Supongamos que queremos crear una tabla que contenga documentos con la siguiente información:

- Título del documento
- Fecha de publicación
- Nombre del autor
- Email del autor

Titulo	Fecha	Autor- Nombre	Autor- Email
Documento 1	25/08/1995	Juan	juan@gmail.com
Documento 2	30/02/2003	Ricardo	ricardo@gmail.com

# Organización de los datos - Ejemplo

Supongamos que queremos crear una tabla que contenga documentos con la siguiente información:

- Título del documento
- Fecha de publicación
- Nombre del autor
- Email del autor

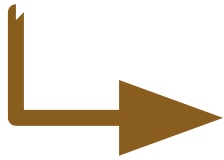
Titulo	Fecha	Autor- Nombre	Autor- Email
Documento 1	25/08/1995	Juan	juan@gmail.com
Documento 2	30/02/2003	Ricardo	ricardo@gmail.com

¿Que necesitamos? → Guardar toda esa información

# Organización de los datos - Como lo hacemos?

¿Cómo guardamos esa información según lo visto hasta el momento?

Usamos una variable para cada tipo de datos que queremos guardar?



En el **ejemplo** tenemos 2 documentos y 4 datos por documento → **necesitamos 8 variables para guardar** los datos

¿Que pasa si tenemos 1000 documentos?  
¿Declaramos 2000 variables?

¿Y si no sabemos cuántos documentos?  
¿Declaramos muchas variables por las dudas?

Variables NO  
Arreglos? Quizá

# Organización de los datos complejos

---

Pero... y si ahora queremos que los **documentos pueden tener más de un autor**, cada uno con su nombre y su email.

- Título del documento
- Fecha de publicación
- Autor 1
  - Nombre del autor
  - Email del autor
- Autor2
  - Nombre del autor
  - Email del autor

# Organización de los datos complejos

Pero... y si ahora queremos que los **documentos pueden tener más de un autor**, cada uno con su nombre y su email.

- Título del documento
- Fecha de publicación
- Autores
  - Autor 1
    - Nombre del autor
    - Email del autor
  - Autor2
    - Nombre del autor
    - Email del autor

¿Nos alcanza con usar arreglos?



Mejor usamos  
**OBJETOS**



# Objetos - ¿Que son?

Una forma de organizar las **variables** y **funciones**

Encapsulan **datos** y **comportamiento**

En JavaScript, se pueden describir a través del formato JSON

Ya lo vimos cuando vimos el DOM, porque el DOM es un “Modelo de **Objetos** del Documento”

## 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);
```



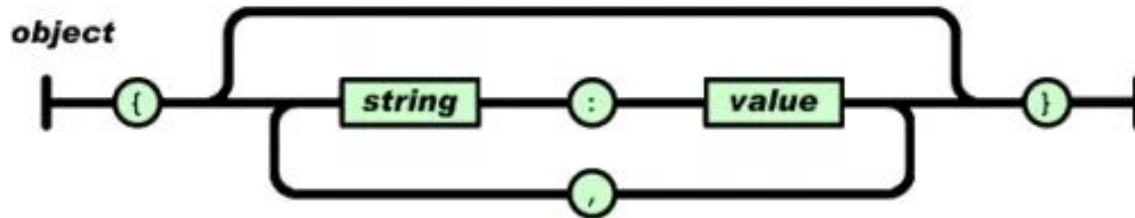
# Qué es JSON?



Es una forma de escribir objetos en Javascript.

Los objetos se caracterizan por ser

- Organizados
- Fáciles de acceder



```
//objeto 'profesor' con dos atributos
{
  "nombre": "javier",
  "materia": "web"
}
```



No confundir un Objeto JSON con una función, tiene llaves, pero no tiene parámetros, ni código, ni la palabra *function* antes

# Tipos de variables

## VARIABLES SIMPLES

3
---

```
let variable = 3;
```

Guardan un único valor de un tipo primitivo (numero, etc)

## ARREGLOS

3
5
77
5

```
let arreglo = [  
  3,  
  5,  
  77,  
  5  
];
```

Guardan muchos valores **del mismo tipo**, ordenados, con una posición para acceder a cada uno.

## OBJETOS

nombre	Web
carrera	TUDAI
cant_profesores	7
cant_alumnos	240

```
let materia = {  
  "nombre": "Web",  
  "carrera": "TUDAI",  
  "cant_profesores": 7,  
  "cant_alumnos": 240  
};
```

Guardan un dato complejo compuesto por diferentes datos de diferentes tipos

# Metáfora - Tipos de variables

## VARIABLE SIMPLES



## ARREGLO



## OBJETO



```
let dado = {}; //dado es un objeto
```

Les podemos agregar miembros dinámicamente.

```
dado.valor = 5;
```

```
//es lo mismo que let dado = {valor: 5}
```

En JS los objetos son como arreglos (el índice es un string)

```
console.log(dado.valor); //5
```

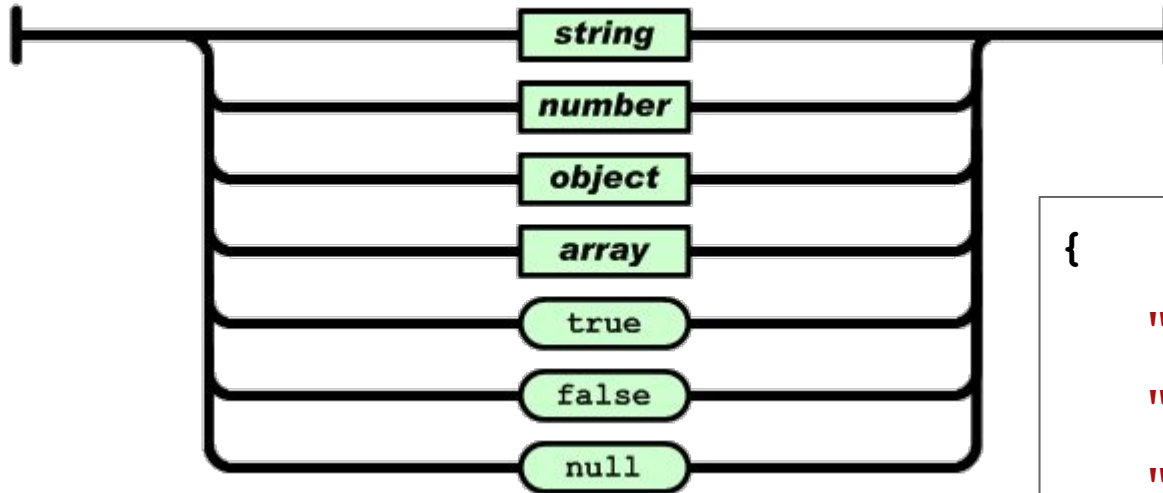
```
//otra forma no tan usada (útil si "valor"  
viene de otra variable)
```

```
console.log(dado["valor"]); //5
```

# Valores en JSON

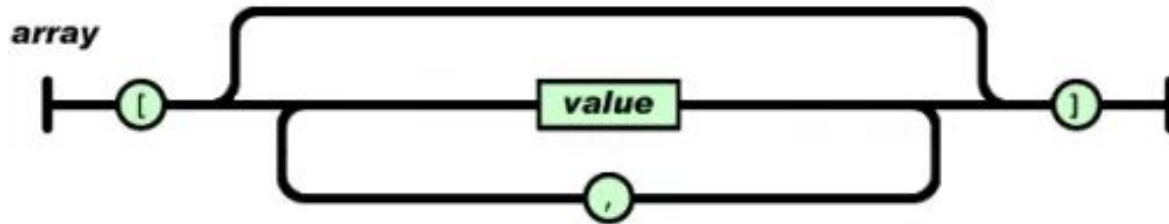
Los valores pueden ser de los siguientes tipos:

*value*



Notar uso de comillas en *value* sólo para cadenas de texto

```
{  
  "cadena": "texto",  
  "numero": 5,  
  "otroObjeto": {...},  
  "arreglo": [5, "a", 1],  
  "verdadero": true,  
  "nada" : null  
}
```



Forma general  
[ { }, { }, ... ]

```
let dados = [ //datos es un arreglo
  { "valor": 5 }, // un dado
  { "valor": 3 } //otro dado
]; // fin del arreglo

console.log(dados[0].valor); // Muestra: 5
console.log(dados[1].valor); // Muestra: 3
```

# Solucion 1- Objetos

---

```
let documento = {  
  "titulo" : "Practico JavaScript",  
  "autor-nombre": "Web",  
  "autor-email": "web@gmail.com"  
}  
  
console.log(documento.titulo) //Practico  
JavaScript  
console.log(documento.autor-email)  
//web@gmail.com
```

## Solucion 2- Objetos anidados

```
let documento = {  
  "titulo" : "Practico JavaScript",  
  "autor" : {  
    "nombre" : "Web",  
    "email" : "web@gmail.com"  
  }  
}  
  
console.log(documento.titulo) //Practico  
JavaScript  
console.log(documento.autor.email)  
//web@gmail.com
```



# Alcanza?

---

[TBC]

## Solución 3 - Arreglos de documentos

```
let documentos = [ {  
  "titulo" : "Practico JavaScript",  
  "autor" : {  
    "nombre" : "Web",  
    "email" : "web@gmail.com"  
  }  
}, ... ]  
  
console.log(documentos[0].titulo) //Practico  
JavaScript  
  
console.log(documentos[0].autor.email)  
//web@gmail.com
```

## Solución 4 - Muchos autores por documento

```
let documentos = [ {  
  "titulo" : "Practico JavaScript",  
  "autores" : [ {  
    "nombre" : "Web",  
    "email" : "web@gmail.com"  
  } ],  
  ...  
}, ... ]  
  
console.log(documentos[0].titulo) //Practico  
JavaScript  
  
console.log(documentos[0].autores[0].email)  
//web@gmail.com
```

# Acceso al elemento paso a paso

## documentos

titulo	Practico Javascript	
autores		
	nombre	web
	email	web@gmail.com

...

...

# Acceso al elemento paso a paso

documentos [0]

titulo	Practico Javascript											
autores	<table><tr><td>nombre</td><td>web</td></tr><tr><td>email</td><td>web@gmail.com</td></tr><tr><td colspan="2"></td></tr><tr><td colspan="2"></td></tr><tr><td colspan="2"></td></tr></table>		nombre	web	email	web@gmail.com						
	nombre	web										
	email	web@gmail.com										
...												
...												

# Acceso al elemento paso a paso

`documentos[0].autores`

titulo	Practico Javascript											
autores	<table><tr><td>nombre</td><td>web</td></tr><tr><td>email</td><td>web@gmail.com</td></tr><tr><td colspan="2"></td></tr><tr><td colspan="2"></td></tr><tr><td colspan="2"></td></tr></table>		nombre	web	email	web@gmail.com						
	nombre	web										
	email	web@gmail.com										
...												
...												

# Acceso al elemento paso a paso

`documentos[0].autores[0]`

titulo	Practico Javascript											
autores	<table><tr><td>nombre</td><td>web</td></tr><tr><td>email</td><td>web@gmail.com</td></tr><tr><td colspan="2"></td></tr><tr><td colspan="2"></td></tr><tr><td colspan="2"></td></tr></table>		nombre	web	email	web@gmail.com						
	nombre	web										
	email	web@gmail.com										
...												
...												

# Acceso al elemento paso a paso

`documentos[0].autores[0].email`

titulo	Practico Javascript											
autores	<table><tr><td>nombre</td><td>web</td></tr><tr><td>email</td><td>web@gmail.com</td></tr><tr><td colspan="2"></td></tr><tr><td colspan="2"></td></tr><tr><td colspan="2"></td></tr></table>		nombre	web	email	web@gmail.com						
	nombre	web										
	email	web@gmail.com										
...												
...												



# Claves

Las claves pueden o no ir entre comillas

```
{ valor: 4 }
```

O

```
{ "valor": 4 } //Good practice!
```

evita problemas si tenemos un campo que es una palabra reservada

```
{  
    "error": 4,  
    "if": 6 ,  
    "nombre": "pepe"  
}
```



# Obtener todos los documentos que hizo X autor

---

[TBC]

# Ventajas

# {JSON}

- Super - Liviano para transferir
- Datos auto-descriptos
- Legible por el humano
- Fácil de adoptar por los lenguajes orientados a objetos

## {JSON}

```
{
  - Contacts: [
    - {
      FirstName: "Demis",
      LastName: "Bellot",
      Email: "demis.bellot@gmail.com"
    },
    - {
      FirstName: "Steve",
      LastName: "Jobs",
      Email: "steve@apple.com"
    },
    - {
      FirstName: "Steve",
      LastName: "Ballmer",
      Email: "steve@microsoft.com"
    },
    - {
      FirstName: "Eric",
      LastName: "Schmidt",
      Email: "eric@google.com"
    },
    - {
      FirstName: "Larry",
      LastName: "Ellison",
      Email: "larry@oracle.com"
    }
  ]
}
```

## <xml/>

```
<ContactsResponse xmlns:i="http://www.w3.org/2003/05/soap-envelope">
  <Contacts>
    <Contact>
      <Email>demis.bellot@gmail.com</Email>
      <FirstName>Demis</FirstName>
      <LastName>Bellot</LastName>
    </Contact>
    <Contact>
      <Email>steve@apple.com</Email>
      <FirstName>Steve</FirstName>
      <LastName>Jobs</LastName>
    </Contact>
    <Contact>
      <Email>steve@microsoft.com</Email>
      <FirstName>Steve</FirstName>
      <LastName>Ballmer</LastName>
    </Contact>
    <Contact>
      <Email>eric@google.com</Email>
      <FirstName>Eric</FirstName>
      <LastName>Schmidt</LastName>
    </Contact>
    <Contact>
      <Email>larry@oracle.com</Email>
      <FirstName>Larry</FirstName>
      <LastName>Ellison</LastName>
    </Contact>
  </Contacts>
</ContactsResponse>
```



# Ejemplos con JSON

---

# Lista de compras

---

Volvamos a hacer la lista de compras, pero supongamos que tenemos también el precio unitario y la cantidad de items.

## **Manejar 4 arreglos?**

Usemos uno solo de objetos JSON!

# Análisis

---

Qué ventajas/desventajas ven de esta versión VS si hubiéramos usado 4 arreglos?

- [TBC]

# Análisis

Qué ventajas/desventajas ven de esta versión VS si hubiéramos usado 4 arreglos?

- [TBC]

Ventajas JSON:

- Un poco menos de código
- Mas escalable (mas facil agregar campos, etc)
- Mas facil de leer, mas natural para los datos
- Código mas reutilizable

Desventajas

- Confuso si guardo cosas diferentes en el arreglo
- Más largo el acceso a datos `compras[indice].valor` vs `valor[indice]`
- Mayor curva de aprendizaje

TANDIL

# Análisis

Qué ventajas/desventajas ven de esta versión VS si hubiéramos usado 4 arreglos?

- [TBC]

Ventajas JSON:

Mayor Facilidad de manejo

Tenemos un solo objeto que es una arreglo

Mas facil la estructura datos

Mas Robusto, mas legible

Mas adaptable, escalable

TRES  
ARROYOS

Desventajas

- Mayor complejidad en el recorrido del arreglo
- Tenemos que aprender, curva aprendizaje



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



## Extra: console.table(arreglo)

Imprime en la consola un arreglo de objetos en formato de tabla

```
console.table(arreglo)
```

(index)	firstName	lastName
0	"John"	"Smith"
1	"Jane"	"Doe"
2	"Emily"	"Jones"

Un objeto encapsula datos y comportamiento.  
¿Dónde está el comportamiento?

```
let dado =  
{  
  valor : 5,  
  tirar : function(){  
    this.valor = ...random...;  
  }  
}
```

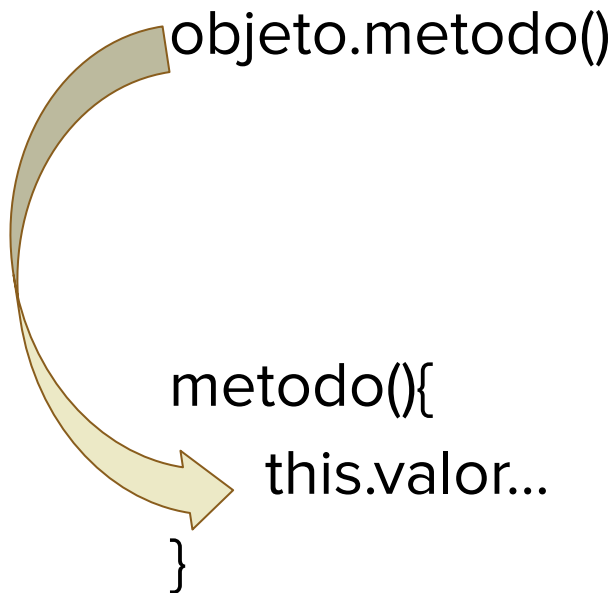
Esto no se usa así, ya que no es fácil tener la misma función en muchos objetos del mismo tipo (de la misma clase - lo que se llama “class”).

Otros lenguajes facilitan este uso. Mesmo Javascript lo facilita, pero no dentro de un JSON.

Esto es la base de la Programación Orientada a Objetos que ven en Programación 2

# This

This hace referencia al objeto que ejecuto el método



Ejecuto un método! Siempre que usemos `variable.funcion()` estamos llamando a un método (una función dentro de un objeto)

La variable especial “this” es la misma variable que “objeto” en la línea que hicimos “objeto.metodo()”

A close-up photograph of a man's face, looking upwards with a wide-eyed, open-mouthed expression of surprise or realization. The lighting is dramatic, with strong highlights on his face and deep shadows in the background.

**I KNOW**

**JSON**

# Funciones

---

# Parámetros

- Tipos primitivos: se pasan por copia-valor
- Tipos objetos: se pasan por referencia

Ejemplo: La función **aumentar** suma 1 a ambos parámetros

```
let primitivo = 5
```

```
let objeto = { valor: 5 }
```

```
aumentar(primitivo,objeto)
```

```
// primitivo: 5 (se copio)
```

```
// objeto.valor: 6 (se usó el mismo)
```

```
function aumentar(primitivo, objeto){  
  primitivo++;  
  objeto.valor++;  
}
```

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

# Parámetros

- Cualquier argumento puede ser omitido o agregado.

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

```
sumar(1, 2, 3); //6
```

```
sumar(1, 2); //NaN : Hace 1+2+undefined
```

```
sumar(1, 2, 3, 4, 5, 6); //6
```



# Lista de parámetros

---

Las funciones pueden llamarse con cualquier cantidad de parámetros. Puedo usar un arreglo para recorrerlos.

- **arguments** : Tiene la lista de **valores** de los parámetros recibidos

```
function sumar(){  
  let suma = 0;  
  for (let arg of arguments){  
    suma += arg  
  }  
  return suma;  
}  
sumar(10,5,2) // 17
```

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

# Una función es un objeto

Declaro una variable y le asigno una función.

```
let f = function() { doSomething(); }
```

```
f(); //ejecuta f
```

```
let f2 = f; //f2 es la misma función que f
```

```
f2(); //ejecuta f2
```

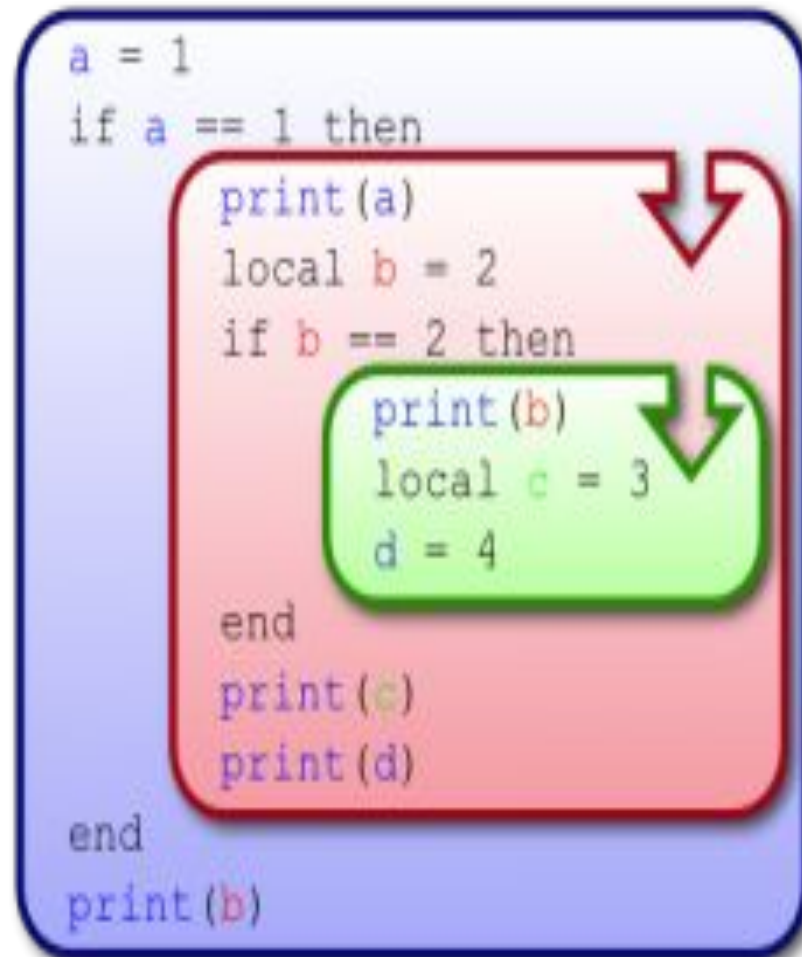
```
f2.call(); //le digo a f2 que se ejecute
```

```
//es lo mismo que f2()
```



# Ámbitos

El ámbito de una variable es el conjunto de líneas donde está variable y es accesible.



# Ámbitos - VAR

Para las variables “var” (pre-ES6), la única forma de crear un ámbito es con funciones.

Para crear variables locales (ej: no ensuciar el espacio global), se suele usar una función anónima (sin nombre) y llamarla.

```
(function () {  
    //Creo una función anónima  
    //nuevo ambito  
    ...  
})(); //la ejecuto  
); //fin sentencia
```

# Ámbitos - VAR

Error típico:

```
for(var i=2; i<10; i++)  
{  
  ...  
}  
console.log(i)
```

Equivale a

```
var i;  
for(i=2; i<10; i++)  
{  
  ...  
}  
console.log(i)
```

//el for no crea un scope

//para “var” lo único que crea scopes son las funciones

# LET vs VAR



**let** alcance de bloque

**var** alcance de función

```
console.log( foo ); // ReferenceError
console.log( bar ); // undefined
if (true) {
  let foo = 2;
  var bar = 2;
  console.log( bar ); // 2
}
console.log( foo ); // ReferenceError
console.log( bar ); // 2
```

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

<https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Statements/let>

# Ámbitos - VAR vs LET

Ejemplo práctico de diferencia:

```
console.log("Con var");  
for(var i = 0; i < 5; i++) {  
    setTimeout(function () {  
        console.log(i);  
    }, 0)  
}
```

```
console.log("Con let");  
for(let i = 0; i < 5; i++) {  
    setTimeout(function () {  
        console.log(i);  
    }, 0)  
}
```

El setTimeout usa la variable, pero después

Con VAR es siempre la misma variable, así que usa el último valor (5).  
Imprime 5 veces 5

Con LET cada ciclo usa una variable diferente  
Imprime del 0 al 4



# Ámbitos - Closures

Forma de crear variables “ocultas”

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

```
function crearFuncionContadora() {  
    //nuevo ámbito  
    let x = 0;  
    return function() { x++; return x; }  
};  
  
//no la puedo acceder desde afuera  
let inc = crearFuncionContadora();  
inc(); //x es local a “ámbito”
```

En JS, declarar una variable es “crear una nueva cada vez que se pasa por esa sentencia”.



# Arrow Functions



Es una forma abreviada para escribir funciones:

`function(param) { }`

se escribe como

`(param) => { }`

Ejemplo:

```
unArreglo.forEach(elem =>
  console.log(elem)
)
```



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

Mostrar/Ocultar detalles

## Ejemplo

---

Crear un botón Ver Más, que muestre / oculte el contenido de un div.

El botón debe poder reutilizarse y funcionar de manera independiente del resto de los botones de la página.

# Qué vamos a aprender

---

Qué vamos a aprender?

- Obtener múltiples elementos del DOM
- this (código más genérico)
- Recorrer el DOM

# Obtener múltiples 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");
```

Retorna uno o más

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

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

sin el punto



Selector de CSS



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

# Obtener múltiples nodos del DOM

---

Obteniendo elementos del DOM con la misma clase

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

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

`manyElements` es un **arreglo** con los elementos que poseen la clase

`manyElements.length` largo del arreglo y cantidad de nodos con esa clase

`manyElements[0]` es el primer elemento con clase `.myclass`



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

# Recorrer el árbol DOM

---

Los elementos del DOM se pueden recorrer como un árbol y ser localizados:

- `element.children`, encuentra los elementos hijos
- `element.parentElement` , encuentra el elemento padre
- `element.nextElementSibling` , encuentra el siguiente hermano
- `element.previousElementSibling` , encuentra el hermano anterior
- `element.firstElementChild` , encuentra el primer hijo
- `element.lastElementChild` , el último hijo

# this

En el contexto de Eventos *this* representa el elemento involucrado en el evento

```
let el = document.getElementById('miDiv');  
el.addEventListener('click', function(e){  
    this.classList.toggle("clase");  
    //toggle de clase del div miDiv click  
});
```



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



# Resolver el problema

---

Debemos localizar todos los elementos que correspondan a una clase, y luego asignarle a cada uno el evento.

// Búsqueda de todos los botones con una clase

```
let btns = document.querySelectorAll('.btn');
```

// asignación de evento a todos los elementos

```
for(let i = 0; i < btns.length; i++) {  
    btns[i].addEventListener('click', miFuncion);  
}
```

# Resolver el problema

Luego, mediante una función anónima individualizamos el botón que dispara el evento y buscamos su hermano en el DOM.

```
for(let i = 0; i < btns.length; i++) {  
  btns[i].addEventListener('click', function(e){  
    //busca el hermano inmediato  
    let el = this.nextElementSibling;  
    //toggle de clase del hermano  
    el.classList.toggle("ver");  
  });  
}
```



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

# Eliminar elementos del DOM

---

Eliminar elementos:

- método `remove()`

En cada elemento se puede hacer el `.remove()` para eliminarlo del DOM

```
document.querySelector("#id").remove()
```





# jQuery



Old way?

- Librería más extendida de Javascript.
- Lema "Write less, do more" (Escribí menos, hace más)
- Open source software.
- Nace en 2006, acompañando el boom de Ajax y las aplicaciones web.
- Busca facilitar tareas muy frecuentes, y lograr mayor compatibilidad entre navegadores.



# Librería

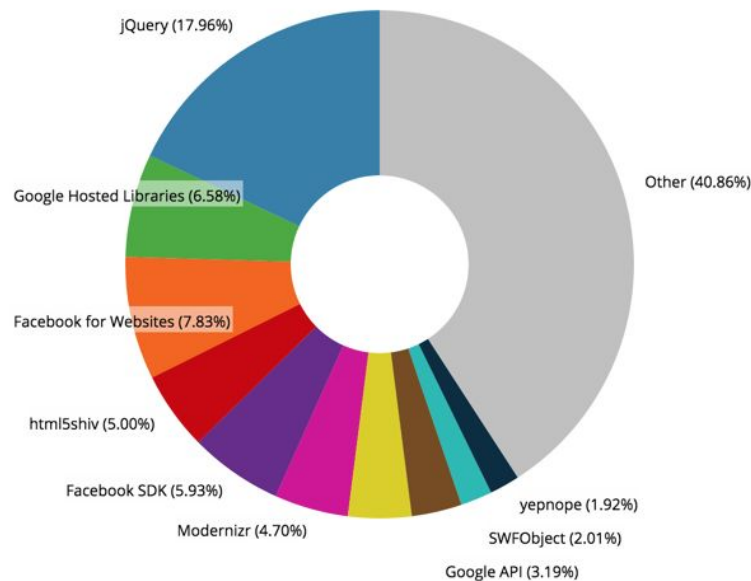
---

- Una librería es un código (uno o más archivos) que contienen funciones en un lenguaje.
- Se incluye en el código del programa y se puede llamar a esas funciones.
- Permite reutilizar código entre diferentes proyectos.
- Hay librerías estándar para cosas que se hacen muchas veces en todos los sitios.

- Simplifica el manejo del DOM.
  - Da métodos para seleccionar uno o más elementos.
  - Ejecutando una acción sobre todos ellos.
- Manejo de eventos.
- Animaciones.
- Ajax.
- Anda en todos los browsers.
  - Nuevas versiones dejan de soportar a versiones viejas de IE

## 2018

jQuery Market Position, 9 May 2018, W3Techs.com



<http://w3techs.com/technologies/details/js-jquery/all/all>  
<http://trends.builtwith.com/javascript>



# JQuery - Vale la pena usarlo?

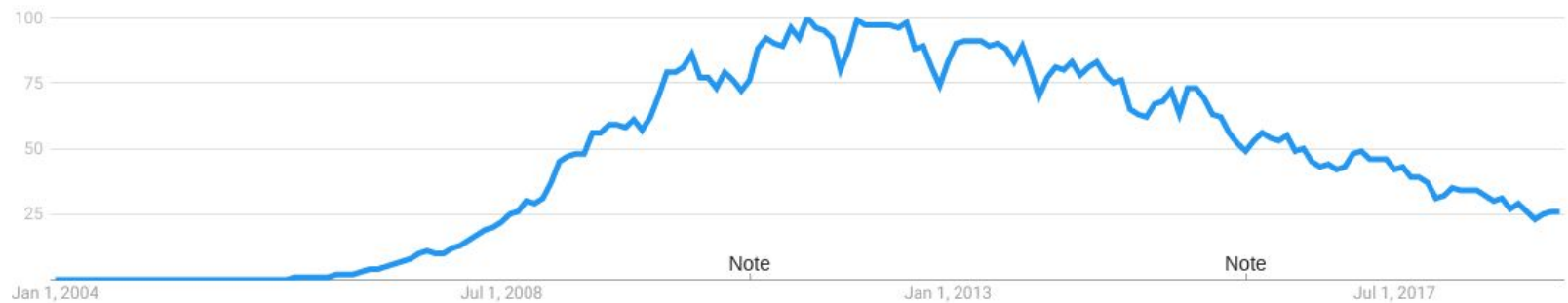
---

- Javascript evolucionó mucho, agregando cosas para manejo del DOM.
- Hay muchas nuevas bibliotecas para hacer diferentes cosas.
- Aunque el mundo está cambiando:
  - Muchísimo sitios lo siguen usando.
  - Es una de las mejores bibliotecas de Javascript.

# JQuery - Por qué lo vemos?

- 

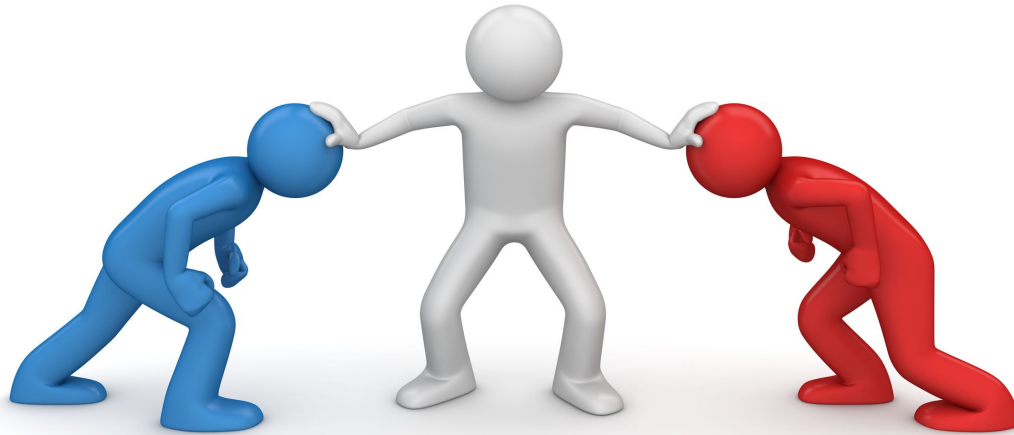
Interest over time ?



<https://trends.google.com/trends/explore?date=all&geo=US&q=%2Fm%2F0268gyp>

# Vamos a dar algunas comparativas

---



Debemos incluir el archivo antes de nuestros scripts.

- Usar el JS nuestro
  - Bajarse los archivos de [www.jquery.com](http://www.jquery.com).
- Usar un CDN



- \$ es la función jQuery. Todo comienza con \$.
  - Modo de trabajo: buscar y editar.
  - Con \$(selector) buscamos algo.
- Podemos obtener múltiples elementos (una colección).
- Las acciones se aplican a todos los elementos.



Esto es lo que ES6 implementó  
como querySelectorAll

```
// Buscar por Id y ocultarlo  
$("#something").hide();
```



- Los métodos se aplican a toda la colección.
  - Si no se encuentra nada no hace nada.
  - No da un error.
- Se puede consultar igual que un arreglo.
  - `$(selector).length`
  - `$(selector)[0]`

```
<div class="clase1 clase2"></div>  
<div class="clase1"></div>
```

```
$(".clase1").hide();  
//esconde ambos divs
```

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

- Diferentes métodos para editar el DOM.
- Muchos de ellos, llamados sin parámetros, devuelve el valor de la propiedad.
- Edita el texto interno.  
`$("li").text("texto")`
- Edita el HTML.  
`$("ul").html("<li>texto</li>")`

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

- Edita el texto interno.

```
document.getElementById("name").innerText("texto")
```

- Edita el HTML.

```
let itemsLista = document.getElementsByTagName("li")
for(let item of itemsLista){
  item.innerHTML("<li>texto</li>")
}
```

Una gran diferencia es que ES6 obliga a escribir el FOR

- + Legible, entiendo lo que pasa
- Más largo

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



- Cambia atributos (propiedades)
  - `$("img").attr("src", url)`
- Elimina atributos
  - `$("img").removeAttr("src")`
- Value
  - `val = $("input").val()`

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

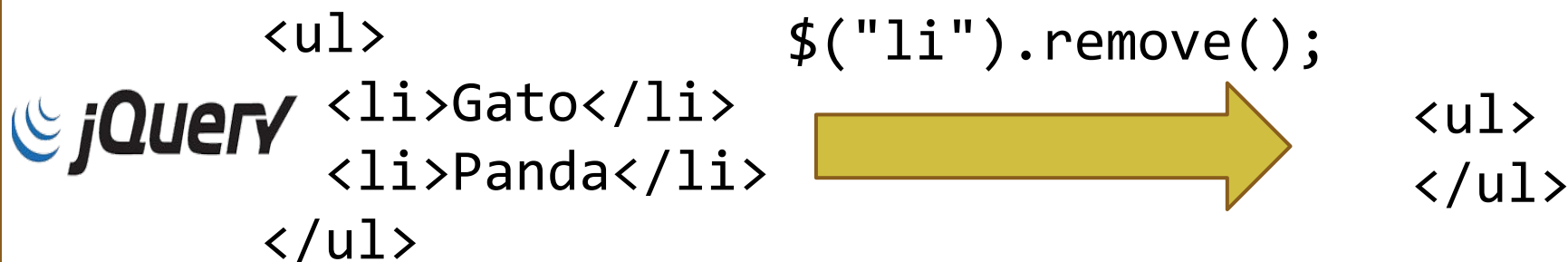
En ES6 esto es el `setAttribute` y otros

```
let element = ...  
element.setAttribute(name, value);  
let attribute =  
element.getAttribute(attributeName);  
element.removeAttribute(attrName);  
let result = element.hasAttribute(name);
```

# Eliminar elementos del DOM

Eliminar elementos:

- método remove()



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



En ES6:

- no se define remove() para colecciones, obliga a hacer el FOR  
`document.querySelectorAll("li").forEach(x=> x.remove())`



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



Diferentes métodos para editar los estilos.

- `addClass("clase")`
- `removeClass("clase")`
- `toggleClass("clase")` //agrega-quita ("niega")



Esto es lo que ES6 implementó  
como classList

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



- jQuery provee métodos propios para recorrer el DOM
- `$("div.section").parent()`
- `$("div.section").next()`
- `$("div.section").prev()`
- `$("div.section").nextAll("div")`
- `$("h1:first").parents()`



Esto es lo que ES6 implementó  
como `nextElementSibling`, etc

```
$("#button").on("click",  
    function(ev) {  
        $(this).addClass("orange");  
        return false; // Or ev.preventDefault();  
    }  
); //se aplica a todos los button
```



*ev* tiene información del click (x, y, etc)  
*this* se refiere al elemento que lanzó el evento.



Esto es lo que ES6 implementó  
como `addEventListener`

Asegura que está cargado el DOM, y aísla el código dentro de una función

```
$(document).ready(function(){  
    // código de inicialización de eventos  
});
```

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



Esto es lo que ES6 implementó como evento DOMContentLoaded



# jQuery Plugins

---

Existe un [gran catálogo de plugins](#).

Algunos Ejemplos:

- [Loading Spin](#)
- [jQuery UI](#)
  - Framework para animaciones e interacciones, efectos y componentes.
- [jQuery Mobile](#)
  - Componentes pensados para aplicaciones móviles con eventos táctiles.



# Ejercicio

---

Hacer una lista de documentos, con un formulario para agregar documentos con su autor y mail del autor.

Usar JQuery y ECMA Script 6 (dos implementaciones)



# Bibliografía

---

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

<http://ejohn.org/apps/workshop/intro/?#5>

**AHORA LES TOCA PRACTICAR :D**

