

Javascript #3

WEB 1 TUDAI UNICEN 2024

Organización de los datos - Ejemplo

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

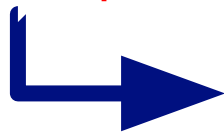
- Título del documento
- Fecha de publicación
- Nombre del autor
- Email del autor

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 - 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



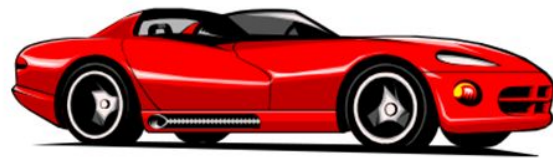
Objetos

REPASO

Los objetos en programación representan “cosas” o “conceptos abstractos” del mundo real, con sus características y comportamientos específicos.

- Un objeto es una combinación de
 - **Atributos (o propiedades):** almacenan datos. Estos datos pueden ser de tipo primitivo y/o otro tipo de objeto.
 - **Métodos:** lleva a cabo una determinada acción o tarea con los atributos.

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



■ Atributos:

- color
- velocidad
- ruedas
- motor

■ Métodos:

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

JSON

JSON: JavaScript Object Notation

JSON es un formato ligero de almacenamiento e intercambio datos, con una estructura específica.

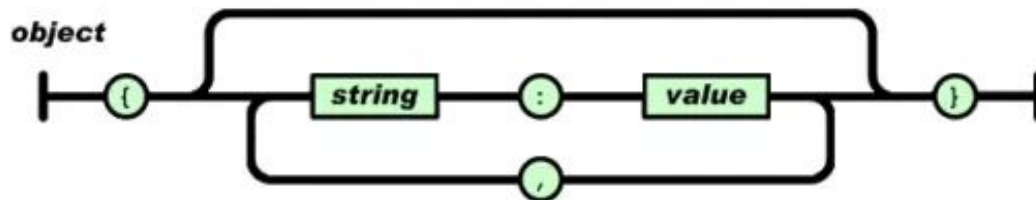
- Es una manera de crear **objetos** en Javascript (no la única).
- Una forma de encapsular **variables (datos)** y **funciones (comportamiento)**

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

Qué es JSON?

{JSON}-

El formato JSON define una manera muy sencilla y natural para representar objetos



//por ejemplo un objeto con dos
atributos (o propiedades)

```
{  
  "nombre": "javier",  
  "materia": "web"  
}
```



No confundir un objeto JSON con una función

Metáfora - Tipos de variables

VARIABLE SIMPLES



ARREGLO



OBJETO



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];
```

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

Declarar un arreglo y acceder a sus datos



Podemos declarar e inicializar un objeto.

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

En JS se recomienda no usar "" (comillas) para las propiedades del objeto

Podemos acceder a sus propiedades directamente

```
console.log(materia.nombre); // "Web"
console.log(materia.carrera); // "TUDAI"
```

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

```
console.log(materia["nombre"]); // "Web"
```

Podemos declarar un objeto vacío y agregar propiedades dinámicamente. (No muy recomendado)

```
let materia = {}; // materia es un objeto vacío
```

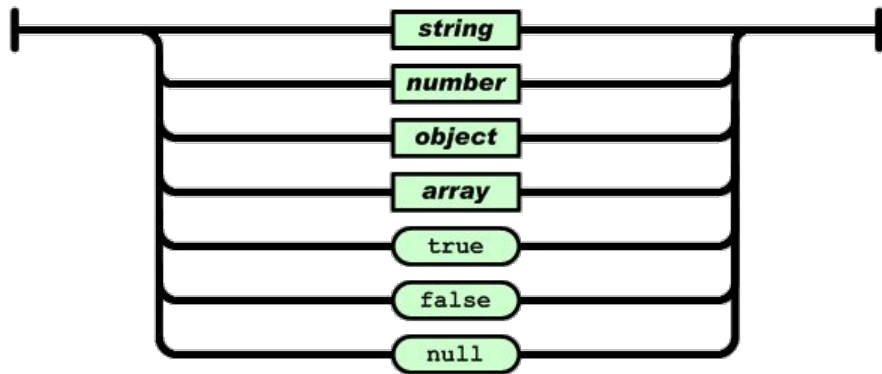
```
materia.nombre = "Web";
```

```
materia.carrera = "TUDAI";
```

Valores en JSON

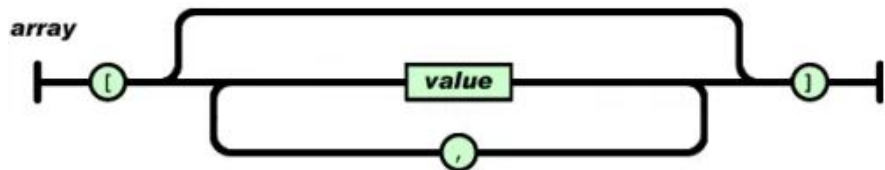
Los valores de las propiedades en un JSON pueden ser de los siguientes tipos:

value



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

```
{  
  cadena: "texto",  
  numero: 5,  
  otroObjeto: {...},  
  arreglo: [5,6,1],  
  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	<table><tr><td>nombre</td><td>web</td></tr><tr><td>email</td><td>web@gmail.com</td></tr></table>		nombre	web	email	web@gmail.com
	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></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></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></table>		nombre	web	email	web@gmail.com				
	nombre	web								
	email	web@gmail.com								
...										
...										

Obtener todos los documentos que hizo X autor

[TBC]

Ejemplos con JSON

Lista de compras

Vamos a hacer una lista de compras. Además de que item vamos a comprar el usuario quiere tener tres botones para indicar la cantidad de unidades (1, 2 o 3).
(si, el usuario es raro)

Manejar multiples arreglos?

Usemos uno solo de objetos!

Paso a Paso

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



Análisis

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

- [TBC]

Análisis (Respuesta)

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

- [TBC]

Ventajas JSON:

- Un poco menos de código
- Mas adaptable (más facil agregar campos, etc)
- Mas legible, más 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

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,  
    color : "rojo",  
    caras : ["a","b"]  
    tirar : function() {  
      this.valor = 4;  
    }  
  }
```

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.

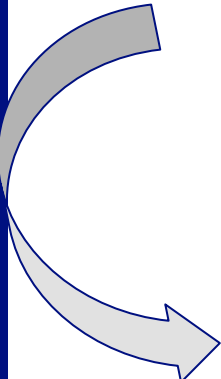
Esto es la base de la Programación Orientada a Objetos que ven en Programación 2

This

This hace referencia al objeto que ejecuto el método

objeto.metodo()

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



```
metodo() {  
    this.valor...  
}
```

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 and slightly to the right. He has a surprised or awe-struck expression, with his mouth slightly open and eyes wide. The lighting is dramatic, with strong highlights on his face and deep shadows in the background. The image is framed by a white border.

I KNOW

JSON

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");
```

sin el punto



Selector de CSS



Retorna uno o más

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

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

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.firstChild` , encuentra el primer hijo
- `element.lastChild` , 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);  
}
```

Manejo de Atributos en ES6



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);
```

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

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()
```

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



Eliminar elementos del DOM

Eliminar elementos:

- método `remove()`



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>



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
```

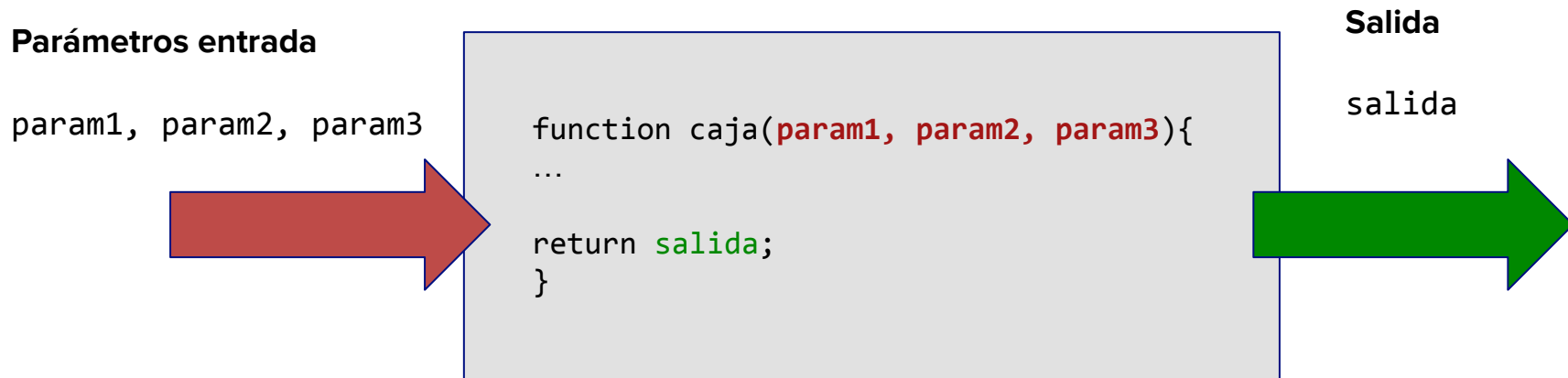
Lista de parámetros

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
```

Funciones



Parámetros entrada: ninguno, 2 o más

Resultado de salida: ninguno, 1 o más

<https://codepen.io/Matias-Presso/pen/poBMPxa>

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()
```



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/NWVGMMVy>

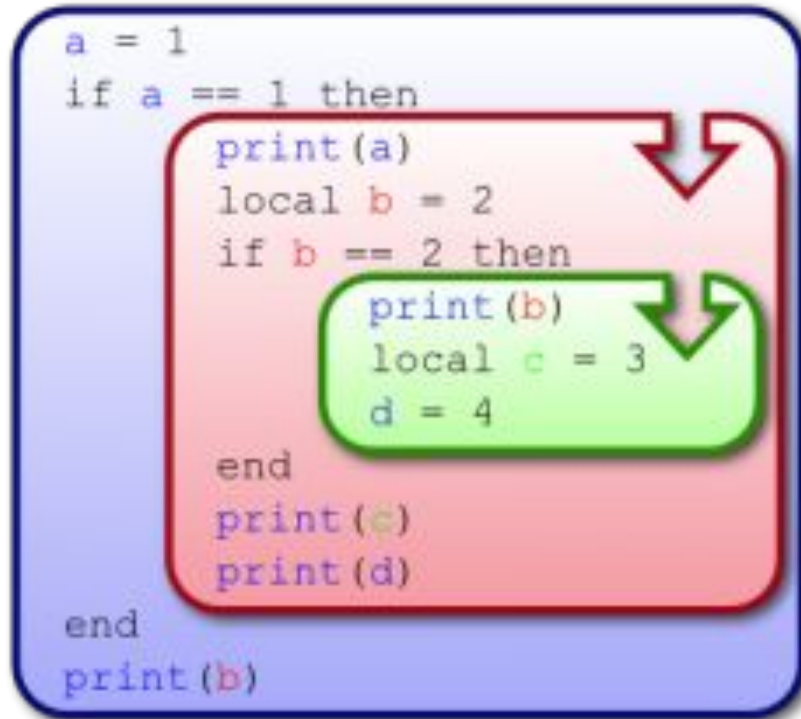
<https://codepen.io/webUnicen/pen/abrvKbo>

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



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



<https://codepen.io/webUnicen/pen/JvpLxK?editors=1111>

Ámbitos - Closures

Forma de crear variables “ocultas”

```
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”.

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

Bibliografía

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

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

AHORA LES TOCA PRACTICAR :D

