
# Tema 3 Objetos Nativos en Javascript

# Practica 04-04 Objetos

| | | | |
|---|---|---|---|
| Nombre | | Curso | |
| Apellidos | | Fecha | |

## Sumario


# 1. Creación de objetos a medida.

En unidades anteriores has visto, como toda la información que proviene del navegador o del
documento está organizada en un modelo de objetos, con propiedades y métodos. Pues
bien, JavaScript también te da la oportunidad de crear tus propios objetos en memoria, objetos con
propiedades y métodos que tú puedes definir a tu antojo.

Estos objetos no serán elementos de la página de interfaz de usuario, pero sí que serán objetos que podrán contener **datos (propiedades) y funciones (métodos)**, cuyos resultados si que se podrán mostrar en el navegador.

El definir tus propios objetos, te permitirá enlazar a cualquier número de propiedades o métodos que tú hayas creado para ese objeto. Es decir, tú controlarás la estructura del objeto, sus datos y su
comportamiento.

hay que dejar claro que, **JavaScript no es un lenguaje orientado a objetos de verdad ennsentido estricto**. Se considera que, JavaScript **es un lenguaje basado en objetos**.

La diferencia entre orientado a objetos y basado en objetos es significante, y tiene que ver sobre todo en cómo los objetos se pueden extender. Quizás en un futuro no muy lejano podamos ver que JavaScript soporte clases, interfaces, herencia, etc.

**Un objeto en JavaScript es realmente una colección de propiedades**. Las propiedades pueden tener
forma de datos, tipos, funciones (métodos) o incluso otros objetos.

De hecho sería más fácil de entender un objeto como un array de valores, cada uno de los cuales está asociado a una propiedad (un tipo de datos, método u objeto). Un

momento: ¿un método puede ser una propiedad de un objeto? Pues en JavaScript parece que sí.

**Una función contenida en un objeto se conoce como un método**. Los métodos no son diferentes de
las funciones que has visto anteriormente, excepto que han sido diseñados para ser utilizados en el
contexto de un objeto, y por lo tanto, tendrán acceso a las propiedades de ese objeto.

Esta conexión entre propiedades y métodos es uno de los ejes centrales de la orientación a objetos.
Los objetos se crean empleando una función especial denominada **constructor**, determinada por el
nombre del objeto. Ejemplo de una función constructor:

```
function Coche(){
// propiedades y métodos
}
```

Aunque esta función no contiene código, es sin embargo la base para crear objetos de tipo Coche.
Puedes pensar en un constructor como un anteproyecto o plantilla, que será utilizada para crear
objetos. Por convención, los nombres de los constructores se ponen generalmente con las iniciales
de cada palabra en mayúscula, y cuando creamos un objeto con ese constructor (**instancia de ese
objeto**), lo haremos empleando minúsculas al principio.

Por ejemplo: **var unCoche = new Coche();**

La palabra reservada **new se emplea para crear objetos en JavaScript**. Al crear la variable **unCoche**,
técnicamente podríamos decir que hemos creado una instancia de la clase Coche, o que hemos
instanciado el objeto Coche, o que hemos creado un objeto Coche, etc. Es decir hay varias formas de
expresarlo, pero todas quieren decir lo mismo.


# 1.1.- Definición de propiedades.

Una vez que ya sabemos como crear un constructor para un objeto, vamos a ver cómo podemos
crear una propiedad específica para ese objeto. **Las propiedades para nuestro objeto se crearán
dentro del constructor** empleando para ello la palabra reservada **this**. Véase el siguiente ejemplo:

```
function Coche(){
// Propiedades
this.marca = "Audi A6";
this.combustible = "diesel";
this.cantidad = 0; // Cantidad de combustible en el depósito.
}
```

La palabra reservada **this**, se utiliza **para hacer referencia al objeto actual**, que en este caso será el
objeto que está siendo creado por el constructor. Por lo tanto, usarás **this**, para crear nuevas
propiedades para el objeto. El único problema con el ejemplo anterior es, que todos los coches que
hagamos del tipo Coche será siempre Audi A6, diésel, y sin litros de combustible en el depósito. Por
ejemplo;

```
var cocheDeMartin = new Coche();
var cocheDeSilvia = new Coche();
```

A partir de ahora, si no modificamos las propiedades del coche de Martin y de Silvia, en el momento
de instanciarlos tendrán ambos un Audi A6 a diesel y sin combustible en el depósito.

Lo ideal sería por lo tanto que en el momento de instanciar un objeto de tipo Coche, que le digamos
al menos la marca de coche y el tipo de combustible que utiliza. Para ello tenemos que modificar el
constructor, que quedará de la siguiente forma:

```
function Coche(marca, combustible){
// Propiedades
this.marca = marca;
this.combustible = combustible;
this.cantidad = 0; // Cantidad de combustible inicial por defecto en el depósito.
```

```
}
```

Ahora sí que podríamos crear dos tipos diferentes de coche:

```
var cocheDeMartin = new Coche("Volkswagen Golf","gasolina");
var cocheDeSilvia = new Coche("Mercedes SLK","diesel");
```

Y también podemos acceder a las propiedades de esos objetos, consultar sus valores o modificarlos.
Por ejemplo:

```
document.write("<br/>El coche de Martin es un: "+cocheDeMartin.marca+" a
"+cocheDeMartin.combustible);
document.write("<br/>El coche de Silvia es un: "+cocheDeSilvia.marca+" a "+
cocheDeSilvia.combustible);
// Imprimirá:
// El coche de Martin es un: Volkswagen Golf a gasolina
// El coche de Silvia es un: Mercedes SLK a diesel
// Ahora modificamos la marca y el combustible del coche de Martin:
cocheDeMartin.marca = "BMW X5";
cocheDeMartin.combustible = "diesel";
document.write("<br/>El coche de Martin es un: " + cocheDeMartin.marca + " a " +
cocheDeMartin.combustible);
// Imprimirá: El coche de Martin es un: BMW X5 a diesel
```

## 1.2.- Definición de métodos.

Las propiedades son solamente la mitad de la ecuación de la Orientación a Objetos en JavaScript. La
otra mitad son **los métodos, que serán funciones que se enlazarán a los objetos**, para que dichas
funciones puedan acceder a las propiedades de los mismos.
Nos definimos un ejemplo de método, que se podría utilizar en la clase Coche:

```
function rellenarDeposito (litros){
// Modificamos el valor de la propiedad cantidad de combustible
this.cantidad = litros;
}
```

Fíjate que el método **rellenarDeposito**, que estamos programando a nivel global, hace referencia a la
propiedad this.cantidad para indicar cuantos litros de combustible le vamos a echar al coche. Lo

único que faltaría aquí es realizar la conexión entre el método **rellenarDeposito** y el objeto de tipo
Coche (recuerda que los objetos podrán tener propiedades y métodos y hasta este momento sólo
hemos definido propiedades dentro del constructor). Sin esta conexión la palabra reservada **this** no
tiene sentido en esa función, ya que no sabría cuál es el objeto actual. Veamos cómo realizar la
conexión de ese método, con el objeto dentro del constructor:

```
function Coche(marca, combustible){
// Propiedades
this.marca = marca;
this.combustible = combustible;
this.cantidad = 0; // Cantidad de combustible inicial por defecto en el depósito.
// Métodos
this.rellenarDeposito = rellenarDeposito;
}
```

Aquí se ve de forma ilustrada, que los métodos son en realidad propiedades: se declaran igual que las
propiedades, por lo que son enmascarados como propiedades en JavaScript. Hemos creado una
nueva propiedad llamada **rellenarDeposito** y se le ha asociado el método **rellenarDeposito**. Es muy
importante destacar que el método **rellenarDeposito()** se referencia sin paréntesis dentro del
constructor, **this.rellenarDeposito = rellenarDeposito**.

Ejemplo de uso del método anterior:

```
cocheDeMartin.rellenarDeposito(35);
document.write("<br/>El coche de Martin tiene "+cocheDeMartin.cantidad+ " litros de " +
cocheDeMartin.combustible+ " en el depósito.");
// Imprimirá
// El coche de Martin tiene 35 litros de diesel en el depósito.
```

La forma en la que hemos definido el método **rellenarDeposito** a nivel global, no es la mejor práctica
en la programación orientada a objetos. **Una mejor aproximación sería definir el contenido de la**

**función rellenarDeposito dentro del constructor, ya que de esta forma los métodos al estar**
**programados a nivel local aportan mayor privacidad y seguridad al objeto en general**, por ejemplo:

```
// Definicion de la "Clase" Coche
function Coche(marca, combustible){
// Propiedades
        this.marca = marca;
        this.combustible = combustible;
        this.cantidad = 0;
// Métodos
        this.rellenarDeposito = function (litros){
                this.cantidad=litros;
        };
}

let audi= new Coche("Audi","diesel");
```

## 1.3.- Definición de objetos literales.

Otra forma de definir objetos es hacerlo de forma literal.

Un literal es un valor fijo que se especifica en JavaScript. Un objeto literal será un conjunto, de cero o
más parejas del tipo **nombre:valor**.

Por ejemplo:

```
avion = { marca:"Boeing" ,
modelo:"747" , pasajeros:"450" };
```

Es equivalente a:

```
var avion = new Object();
avion.marca = "Boeing";
avion.modelo = "747";
avion.pasajeros = "450";
```

Para referirnos desde JavaScript a una propiedad del objeto avión podríamos hacerlo con:

```
document.write(avion.marca); // o también se podría hacer con:
document.write(avion["modelo"]);
```


Podríamos tener un conjunto de objetos literales simplemente creando un array que contenga en
cada posición una definición de objeto literal:

```
var datos=[
{"id":"2","nombrecentro":"IES A Piringalla" ,"localidad":"Lugo","provincia":"Lugo"},
{"id":"10","nombrecentro":"IES As Fontiñas","localidad":"Santiago","provincia":"A Coruña"},
{"id":"9","nombrecentro":"IES As Lagoas","localidad":"Ourense","provincia":"Ourense"},
{"id":"8","nombrecentro":"IES Cruceiro Baleares","localidad":"Culleredo","provincia":"A Coruña"},
{"id":"6","nombrecentro":"IES Cruceiro Baleares","localidad":"Culleredo","provincia":"A Coruña"},{"id":"4","nombrecentro":"IES de Teis","localidad":"Vigo","provincia":"Pontevedra"},
{"id":"5","nombrecentro":"IES Leliadoura","localidad":"Ribeira","provincia":"A Coruña"},
{"id":"7","nombrecentro":"IES Leliadoura","localidad":"Ribeira","provincia":"A Coruña"},
{"id":"1","nombrecentro":"IES Ramon Aller Ulloa","localidad":"Lalin","provincia":"Pontevedra"},
{"id":"3","nombrecentro":"IES San Clemente","localidad":"Santiago de Compostela","provincia":"A Coruña"}
];
```

De la siguiente forma se podría recorrer el array de datos para mostrar su contenido:

```
for (var i=0; i< datos.length; i++){
document.write("Centro ID: "+datos[i].id+" - ");
document.write("Nombre: "+datos[i].nombrecentro+" - ");
document.write("Localidad: "+datos[i].localidad+" - ");
document.write("Provincia: "+datos[i].provincia+"<br/>");
}
```

Y obtendríamos como resultado:

Centro ID: 2 - Nombre: IES A Piringalla - Localidad: Lugo - Provincia: Lugo
Centro ID: 10 - Nombre: IES As Fontiñas - Localidad: Santiago - Provincia: A Coruña
Centro ID: 9 - Nombre: IES As Lagoas - Localidad: Ourense - Provincia: Ourense
Centro ID: 8 - Nombre: IES Cruceiro Baleares - Localidad: Culleredo - Provincia: A Coruña
Centro ID: 6 - Nombre: IES Cruceiro Baleares - Localidad: Culleredo - Provincia: A Coruña
Centro ID: 4 - Nombre: IES de Teis - Localidad: Vigo - Provincia: Pontevedra


Centro ID: 5 - Nombre: IES Leliadoura - Localidad: Ribeira - Provincia: A Coruña
Centro ID: 7 - Nombre: IES Leliadoura - Localidad: Ribeira - Provincia: A Coruña
Centro ID: 1 - Nombre: IES Ramon Aller Ulloa - Localidad: Lalin - Provincia: Pontevedra
Centro ID: 3 - Nombre: IES San Clemente - Localidad: Santiago de Compostela - Provincia: A
Coruña

# 2. Iterar un objeto

Para **recorrer el contenido de un objeto en JavaScript**, puedes usar for...in, Object.keys(), Object.entries(), o incluso for...of (pero *solo* si lo combinas con esos métodos, porque un objeto **no es iterable** por sí mismo).

Aquí tienes una explicación clara con ejemplos.

### 2.1 Recorrer un objeto con for...in

for...in sirve para recorrer **las claves** (propiedades) de un objeto.

```
const persona = {
  nombre: "Ana",
  edad: 25,
  ciudad: "Lima"
};
```

- for (let clave in persona) {  console.log(clave, persona[clave]);  }

let avion2 = { marca: "Boeing-2", modelo: "747-2", pasajeros: "450-2", 1: "marcha1-2", 2: "marcha2-2" };
for (key in avion2) { document.write(key + "-" + avion2[key] + "<br>");

#### 2.2 Recorrer un objeto con forEach()

Un objeto **no tiene .forEach()**, pero puedes usar Object.keys(), Object.values() o Object.entries().

- **Con claves (Object.keys())**
  - Object.keys(persona).forEach(clave => {  console.log(clave, persona[clave]); });
- **Con valores (Object.values())**
  - Object.values(persona).forEach(valor => {  console.log(valor); });


- **Con claves y valores (Object.entries())**
  - Object.entries(persona).forEach(([clave, valor]) => { console.log(clave, valor); });

**2.3. Recorrer un objeto con for...of**

Un objeto **no es iterable** directamente, pero puedes usarlo junto con Object.entries():

for (let [clave, valor] of Object.entries(persona)) { console.log(clave, valor); }
}

# 3. Prototipos de objetos

Los prototipos son un mecanismo mediante el cual los objetos en JavaScript heredan características entre sí. En este artículo, explicaremos como funcionan los prototipos y también cómo se pueden usar las propiedades de estos para añadir métodos a los contructores existentes.

## 3.1 Modificando prototipos

Para modificar la propiedad prototype de una función constructor (los métodos añadidos a la propiedad prototipo están disponibles en todas las instancias de los objetos creados a partir del constructor).

el siguiente código añade un nuevo método y atributo a la propiedad prototype del constructor:

Person.**prototype**.vivaYo = "VivaYo";

Person.**prototype**.farewell = function () {

alert(this.name.first + " ha dejado el edificio. ¡Adiós por ahora!"+ this.vivaYo );

};

Guarda el código y abre la página en el navegador, e ingresa lo siguiente en la entrada de texto.

person1.farewell();


# 3.2 Prototype X Funciones Flecha

## OJOOOOO : NO SE PUEDEN USAR FUNCIONES FLECHA CON el PROTOTYPE

ESTO ESTARIA MAL

```
String.prototype.tedacuen=()=>("Te da cuen pecador de la pradera¡")
String.tedacuenEstatica=()=>("Te da cuen pecador de la pradera2¡")
console.log("".tedacuen())
console.log(String.tedacuenEstatica())
```

Esto lo podemos usar para cualquier clase por ejemplo la Clase string.

```
String.prototype.tedacuen= function () { return "Te da cuen pecador de la
pradera¡"}
String.tedacuenEstatica= function () { return "Te da cuen pecador de la
pradera¡"}
console.log("".tedacuen())
console.log(String.tedacuenEstatica())
```


# 4. 5 maneras de iterar un objeto en JavaScript

Los objetos son estructuras de datos bastante importantes en JavaScript, internamente el lenguaje administra todo mediante objetos. Recorrerlos o iterarlos es bastante sencillo.

- Recorrer objetos es una tarea bastante básica y sencilla de realizar.
- Recomiendo usar los métodos provistos de la API del lenguaje (entries(), forEach(), keys(), values(), etc) para recorrer objetos. No reinventes la rueda.
- No te limites a usar las meneras de recorrer un objeto descritas en este post, dependiendo a la situación, puede que te sea más útil usar otros métodos. En desarrollo existen varias maneras de llegar a los mismos resultados.

```
let perro = {
  nombre: "Scott",
  color: "Negro",
  macho: true,
  edad: 5
};
```

**1. Usando la propiedad Object.keys()**

Object.keys() obtiene en un arreglo todas las claves del objeto en cuestión.*

```
let claves = Object.keys(perro); // claves =
["nombre", "color", "macho", "edad"]
for(let i=0; i< claves.length; i++){
  let clave = claves[i];
  console.log(perro[clave]);
}
```

**2. Usando la propiedad Object.values()**

Object.values() obtiene los valores del objecto en cuestión.*

```
let valores = Object.values(perro); // valores =
["Scott", "Negro", true, 5];
for(let i=0; i< valores.length; i++){
  console.log(valores[i]);
}
```

**3. Usando un bucle for...in**

```
for (let clave in perro){
  console.log(perro[clave]);
}
```

**4. Usando la propiedad Object.entries() con un forEach()**

```
// valores = [["nombre","Scott"], ["color","Negro"],
["macho",true], ["edad",5]];

Object.entries(perro).forEach(([key, value]) => {
  console.log(value)
});
```

**5. Usando la propiedad Object.entries() y un bucle for...of**

```
for(const [key, value] of Object.entries(perro)){
console.log(value) }
```

```
/*salida:
"Scott"
"Negro"
true
5
*/
```


# 4. Ejercicios

**Ejercicio 1.** Crea un objeto alumno con los siguientes valores metodos y propiedades

1. Propiedades: nombre, apellidos, id, NombreModulos, notasModulos (estos dos son dos arrays paralelos)
2. Metodos:
    1. set y get de las propiedades nombre, apellidos, id
    2. Mostrar modulos(id): muestra el nombre de los modulos
    3. nota(nombremodulo) : muestra la nota del modulo
    4. media(id) te muestra su media
    5. suspensas(id): te muestra las notas suspensas
    6. Aprobadas(id): te muestra las notas aprobadas

       el id es para que se muestre en el InnerElement que tenga ese id.

**Ejercicio 2.** Realiza el anterior para que los metodos sean funciones anonimas. Usando la notacion () =>

**Ejercicio 3.** Crea un ejemplo que Muestra cada elemento.

**Ejercicio 4.** Crea un objeto clase que sea un array de alumnos con los metodos (con notacion anonima)

1. matricular(alumno)
2. eliminar( alumnos)
3. numero(alumnos)
4. Mostrar(id)

**Ejercicio 5.** Crea un objeto instituto con nombre , direccion e id de manera literal.

**Ejercicio 6.** Añade al anterior una metodo cambiarnombre(direccion) y mostrar(id) de manera literal usando funciones anonimas.

**Ejercicio 7.** Añade al anterior una metodo cambiarnombre(direccion) y mostrar(id) de manera literal usando funciones Flechas

**Ejercicio 8.** Ejecuta este codigo y observa la diferencia entre el this usando f. Anonimas y flecha

```
var p = new Person1();
    window.age = 10; // <- Age en ventana
    function Person1() {
      this.age = 41; // <- Age en Persona1
                        setTimeout(function () {
                          // <-- La función  ejecutando en el ámbito de
window
```


```
        document.getElementById('msg').innerHTML
+= "<br> funcion anonima this.age1:" + this.age; //
genera "10" porque la función se ejecuta en el ámbito
de window
    }, 100);    }
function Person2() {
    this.age = 42; // <-- ¿<- Age en Persona2
    setTimeout(() => {
        // <-- La función flecha   ejecutando en
Person2
        document.getElementById('msg').innerHTML
+= "<br> funcion flecha this.age2:" + this.age; // genera
";" porque la función se ejecuta en el ámbito de
window
    }, 100);    }
  var p = new Person2();
```

**Ejercicio 9.** Añade al varios metodos de manera dinamica usando prototipe.

**Ejercicio 10.** Especial santa claus

Santa Claus 🎅 está revisando el inventario de su taller para preparar la entrega de regalos. Los elfos han registrado los juguetes en un array de objetos, pero la información está un poco desordenada. Necesitas ayudar a Santa a organizar el inventario.

Recibirás un array de objetos, donde cada objeto representa un juguete y tiene las propiedades:

name: el nombre del juguete (string).
quantity: la cantidad disponible de ese juguete (entero).
category: la categoría a la que pertenece el juguete (string).
Escribe una función que procese este array y devuelva un objeto que organice los juguetes de la siguiente manera:

Las claves del objeto serán las categorías de juguetes.
Los valores serán objetos que tienen como claves los nombres de los juguetes y como valores las cantidades totales de cada juguete en esa categoría.
Si hay juguetes con el mismo nombre en la misma categoría, debes sumar sus cantidades.
Si el array está vacío, la función debe devolver un objeto vacío {}.

```
const inventory = [
  { name: 'doll', quantity: 5, category: 'toys' },
  { name: 'car', quantity: 3, category: 'toys' },
  { name: 'ball', quantity: 2, category: 'sports' },
  { name: 'car', quantity: 2, category: 'toys' },
  { name: 'racket', quantity: 4, category: 'sports' }
  { name: 'racket', quantity: 7, category: 'sports' }
]

organizeInventory(inventory)
// Resultado esperado:
// {
//   toys: {
//     doll: 5,
//     car: 5
//   },
//   sports: {
//     ball: 2,
//     racket: 11
//   }

const inventory2 = [
```


```
  { name: 'book', quantity: 10, category: 'education' },
  { name: 'book', quantity: 5, category: 'education' },
  { name: 'paint', quantity: 3, category: 'art' }
]

organizeInventory(inventory2)
// Resultado esperado:
// {
//   education: {
//     book: 15
//   },
//   art: {
//     paint: 3
//   }
// }
/**
 * @param {{ name: string, quantity: number, category: string }[]} inventory
 * @returns {object} The organized inventory
 */
function organizeInventory(inventory) {
  // Code here
  return {}
}
```