

Desarrollo Web en Entorno Cliente

Tema 4





[objetos]

JS está diseñado en un paradigma simple basado en objetos. Un objeto es una colección de propiedades.

Una propiedad es una asociación entre un nombre (o clave) y un valor.

El valor de una propiedad puede ser una función, en cuyo caso la propiedad es conocida como un método.

Los objetos son parte de las piezas principales de construcción en JS.

JS dispone de tipos primitivos (*undefined*, *string*...) y el resto de elementos son objetos (funciones, *arrays*,...)

Un objeto de JS tiene propiedades asociadas a él.

Una propiedad de un Objeto se puede explicar como una variable adjunta al objeto.

Las propiedades de un objeto son básicamente lo mismo que las variables de JS, pero tienen una pertenencia al objeto. Las propiedades de un objeto definen las características del objeto

Un objeto en JS es un contenedor de propiedades, donde una propiedad tiene un nombre y un valor.

El valor de la propiedad puede ser cualquier valor, excepto *undefined*.

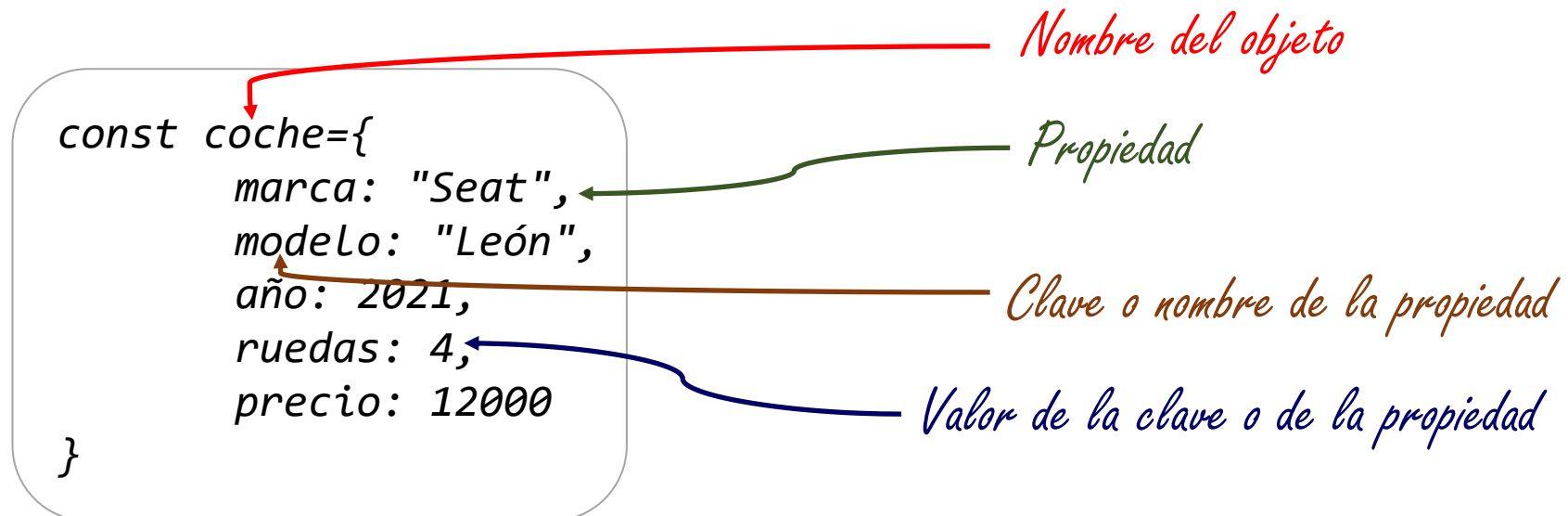
[objetos]

Para crear un objeto (objeto literal) se puede hacer inicializándolo, creando una lista, delimitada entre llaves {}, de de cero o más elementos separados por comas, formados por pares nombre de propiedad, dos puntos :, y su valor.

Una constante definida con llaves indica que es un objeto en JS.

Las propiedades del objeto se definen con el nombre de la propiedad, llamada clave, el signo de dos puntos : y el valor que toma.

Las propiedades se separan mediante comas ,



[objetos]

Se puede crear un objeto y enviarlo a la consola del navegador, desde aquí se puede expandir, haciendo clic en la flecha que sale junto a él, con el fin de mostrar todas sus propiedades.

```
const coche={  
  marca: "Seat",  
  modelo: "León",  
  año: 2021,  
  ruedas: 4,  
  precio: 12000  
};  
  
console.log(coche);
```

```
▼ {marca: 'Seat', modelo: 'León', año: 2021, ruedas: 4, precio: 12000}  
  año: 2021  
  marca: "Seat"  
  modelo: "León"  
  precio: 12000  
  ruedas: 4  
  ► [[Prototype]]: Object
```

acceder a las propiedades

Para acceder al valor de las propiedades del objeto se utiliza la notación punto .

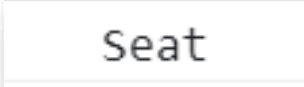
Se corresponde con el nombre del objeto seguido de punto y, a continuación, el nombre de la propiedad.

```
const coche={  
  marca: "Seat",  
  modelo: "León",  
  año: 2021,  
  ruedas: 4,  
  precio: 12000  
};  
  
console.log(coche.marca);
```

También se puede acceder mediante la notación de corchetes [].

Los objetos, a veces son llamados *Arrays Asociativos*, debido a que cada propiedad está asociada con un valor de cadena que se puede utilizar para acceder a ella.

```
console.log(coche["marca"]);
```



Seat

modificar propiedades

Para modificar los valores de las propiedades del objeto se realiza igual que su acceso, pero dando un nuevo valor a la propiedad con el signo igual =

```
const coche={  
    marca: "Seat",  
    modelo: "León",  
    año: 2021,  
    ruedas: 4,  
    precio: 12000  
};  
  
coche.ruedas=5;  
console.log(coche.ruedas);
```

También se puede modificar mediante la notación de corchetes `[]`.

```
coche['ruedas']=5;  
console.log(coche["ruedas"]);
```

crear propiedades

Para crear nuevas propiedades en un objeto, una vez que ya ha sido definido, se realiza igual que cuando se modifica el valor de la propiedad.

```
const coche={  
  marca: "Seat",  
  modelo: "León",  
  año: 2021,  
  ruedas: 4,  
  precio: 12000  
};  
  
coche.disponible=true;  
console.log(coche);
```

También se puede modificar mediante la notación de corchetes `[]`.

```
coche['disponible']=true;  
console.log(coche.disponible);
```

```
▼ {marca: 'Seat', modelo: 'León', año: 2021, ruedas: 4, precio: 12000, ...}  
  año: 2021  
  disponible: true  
  marca: "Seat"  
  modelo: "León"  
  precio: 12000  
  ruedas: 4  
  ► [[Prototype]]: Object
```


(eliminar propiedades)

Para eliminar propiedades en un objeto, una vez que ya ha sido definido, se utiliza la palabra reservada *delete* seguido del nombre de la propiedad.

```
const coche={  
  marca: "Seat",  
  modelo: "León",  
  año: 2021,  
  ruedas: 4,  
  precio: 12000  
};
```

```
delete coche.marca;  
console.log(coche);
```

También se puede modificar mediante la notación de corchetes `[]`.

```
delete coche['marca'];  
console.log(coche.marca);  
  
>undefined
```

```
▼ {modelo: 'León', año: 2021, ruedas: 4, precio: 12000} ⓘ  
  año: 2021  
  modelo: "León"  
  precio: 12000  
  ruedas: 4  
  ► [[Prototype]]: Object
```

[object destructuring]

Destructuring (la desestructuración) es un mecanismo de JS que permite desempaquetar propiedades de objetos en distintas variables.

El mecanismo es crear una nueva variable encerrada entre llaves `{}` que se llama igual que la clave de la propiedad del objeto, y que contendrá el mismo valor.

```
const coche={  
  marca: "Seat",  
  modelo: "León",  
  año: 2021,  
};
```

```
const {modelo}= coche;  
console.log(modelo);
```

```
const {modelo, año}=coche;  
console.log(`El ${modelo} es del año ${año}`);
```

El León es del año 2021

[object destructuring]

En la desestructuración se puede desempaquetar propiedades de un objeto y asignarlas a una variable con un nombre diferente al que tiene la propiedad del objeto.

Para ello, en el nombre de la variable que recoge el valor se pone dos puntos : seguido del nombre de la variable que se desea usar en lugar del nombre de la propiedad del objeto.

```
const coche={  
  marca: "Seat",  
  modelo: "León",  
  año: 2021,  
};  
  
const {modelo: tipo, marca: fabricante}= coche;  
console.log(fabricante);
```

[object destructuring]

Puede suceder que al hacer la desestructuración se intente desempaquetar una propiedad que no existe en el objeto o no tiene un valor definido (*undefined*).

Se puede anticipar a esta situación mediante el nombre de la variable que recoge el valor seguido de un igual = y el valor por defecto que tomará en el caso de que dicha propiedad no tenga valor o no exista en el objeto.

```
const coche={  
    marca: "Seat",  
    modelo: "León",  
    año: 2021,  
};  
  
const {matricula="0000", marca: fabricante="ND"} = coche;  
console.log(matricula);
```

[object destructuring]

Un objeto puede contener otros objetos.

```
const coche={  
  marca: "Seat",  
  modelo: "León",  
  año: 2021,  
  dimensiones: {  
    peso: 1318,  
    altura: 1.4,  
    anchura:1.8  
  }  
};  
  
console.log(coche);  
console.log(coche.dimensiones);
```

```
▼ {marca: 'Seat', modelo: 'León', año: 2021, dimensiones: {...}}  
  año: 2021  
  ► dimensiones: {peso: 1318, altura: 1.4, anchura: 1.8}  
    marca: "Seat"  
    modelo: "León"  
    ► [[Prototype]]: Object  
▼ {peso: 1318, altura: 1.4, anchura: 1.8} ⓘ  
  altura: 1.4  
  anchura: 1.8  
  peso: 1318  
  ► [[Prototype]]: Object
```

[object destructuring]

También se puede hacer desestructuración de las propiedades del objeto contenido dentro de otro objeto.

```
const coche={  
  marca: "Seat",  
  modelo: "León",  
  año: 2021,  
  dimensiones: {  
    peso: 1318,  
    altura: 1.4,  
    anchura:1.8  
  }  
};
```

```
const {dimensiones}=coche;  
console.log(dimensiones);
```

```
▼ {peso: 1318, altura: 1.4, anchura: 1.8}  
  altura: 1.4  
  anchura: 1.8  
  peso: 1318  
  ► [[Prototype]]: Object
```

```
const {peso, altura}=coche.dimensiones;  
console.log(peso);  
console.log(altura);
```

1318

1.4

[object destructuring]

Se puede hacer la desestructuración de las propiedades del objeto contenido dentro de otro objeto mediante el nombre del objeto contenido seguido de dos puntos : y entre llaves {} los nombres de las propiedades a extraer separadas por coma , esto no va a extraer el objeto contenido.

```
const coche={  
  marca: "Seat",  
  modelo: "León",  
  año: 2021,  
  dimensiones: {  
    peso: 1318,  
    altura: 1.4,  
    anchura:1.8  
  }  
};
```

```
const {año, dimensiones:{peso, altura}}=coche;  
console.log(peso);  
console.log(altura);
```

1318

1.4

[congelado de objetos]

Para evitar que los objetos se puedan modificar y, así, actúen como constantes, hay que usar el modo estricto. Para ello se ubica en la parte superior del fichero, donde se define el objeto, la siguiente sentencia *"use strict"*;

El modo estricto, incluido con ES5 (ECMAScript 2009), es un mecanismo para activar una forma restringida de JS, en este modo, por ejemplo, hay que definir las variables antes de utilizarlas.

```
numero=10;  
console.log(numero);
```

10

```
"use strict";  
numero=10;  
console.log(numero);
```

✖ ▶ Uncaught ReferenceError: numero is not defined
at variables.js:104

[congelado de objetos]

El modo estricto habilita una serie de métodos para los objetos (*Object Methods*) de JS.

No obstante, no todos los navegadores admiten el modo estricto, y en los navegadores modernos no hace falta activarlo para habilitar los *Objects Methods*.

Por lo que no hay que confiar plenamente en el modo estricto sin antes hacer pruebas de sus características en los navegadores donde va a utilizarse.

Uno de los métodos de *Objects Methods* es *freeze* (congelar) que “congela” el objeto para que permanezca inalterado.

```
// "use strict";

const coche={
  marca: "Seat",
  modelo: "León",
  año: 2021,
};
Object.freeze(coche);
coche.año=2020;
console.log(coche);
```

```
▼ {marca: 'Seat', modelo: 'León', año: 2021}
  año: 2021
  marca: "Seat"
  modelo: "León"
  ► [[Prototype]]: Object
```

[congelado de objetos]

Un objeto congelado no puede ser modificado, no se le pueden agregar propiedades nuevas ni tampoco se le pueden eliminar las que posee.

Cuando un objeto ha sido congelado ya no se puede descongelar ni alterar de ninguna manera.

Si se intenta modificar un Objeto congelado y se ha habilitado el método estricto dará un error.

```
"use strict";
```

```
const coche={  
    marca: "Seat",  
    modelo: "León",  
    año: 2021,  
};  
Object.freeze(coche);  
coche.año=2020;  
console.log(coche);
```

```
✖ ▶ Uncaught TypeError: Cannot assign to read only property 'año' of object '#  
<Object>'  
    at variables.js:101
```

[congelado de objetos]

Otro de los métodos de *Objects Methods* es *isFrozen* (¿está congelado?) que devuelve *true* o *false* en base a si el objeto sobre el que se pregunta está o no “congelado”.

```
"use strict";  
  
const coche={  
    marca: "Seat",  
    modelo: "León",  
    año: 2021,  
};  
Object.freeze(coche);  
console.log(Object.isFrozen(coche));
```

true

[sellado de objetos]

Existe el método *seal* (sellado) de *Objects Methods* que “sella” el objeto.

A diferencia de *freeze*, *seal*, no deja agregar o eliminar propiedades, pero si deja modificar las existentes.

```
coche.año=2020;  
console.log(coche);
```

```
coche.país= "España";
```

```
"use strict";
```

```
const coche={  
  marca: "Seat",  
  modelo: "León",  
  año: 2021,  
};  
Object.seal(coche);
```

```
▼ {marca: 'Seat', modelo: 'León', año: 2020}  
  año: 2020  
  marca: "Seat"  
  modelo: "León"  
  ► [[Prototype]]: Object
```

```
✖ ► Uncaught TypeError: Cannot add property país, object is not extensible  
   at variables.js:101
```

[sellado de objetos]

De igual manera, existe el método *isSealed* (¿está sellado?) de *Objects Methods* que devuelve *true* o *false* en base a si el objeto sobre el que se pregunta está o no “sellado”.

```
"use strict";

const coche={
  marca: "Seat",
  modelo: "León",
  año: 2021,
};
Object.seal(coche);

console.log(Object.isSealed(coche));
```

true

[copiar objetos]

A partir de dos o más objetos podemos crear otro objeto que posea las características de los objetos iniciales mediante el método *assign* (asignar) de *Objects Methods*.

```
const coche={  
  marca: "Seat",  
  modelo: "León",  
  año: 2021,  
};  
  
const producto={  
  precio: 12000,  
  color: "rojo"  
};  
  
const resultado= Object.assign(coche, producto);  
console.log(resultado);
```

```
▼ {marca: 'Seat', modelo: 'León', año: 2021, precio: 12000, color: 'rojo'}  
  año: 2021  
  color: "rojo"  
  marca: "Seat"  
  modelo: "León"  
  precio: 12000  
  ► [[Prototype]]: Object
```

[copiar objetos]

Otra forma de copiar los objetos es mediante el *Spread Operator* (...)

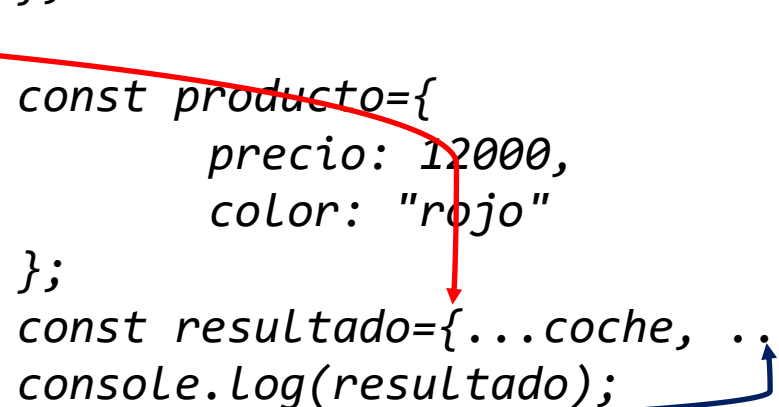
El *Spread Operator* (o *Spread Syntax*) permite a un elemento iterable (un elemento que se puede recorrer por sus elementos o propiedades) ser expandido.

Creamos un nuevo objeto, por eso usamos las llaves

Usamos el Spread Operator para expandir los componentes de cada objeto (propiedades) y rellenar con ellas el objeto resultado

```
▼ {marca: 'Seat', modelo: 'León', año: 2021, precio: 12000, color: 'rojo'}  
  año: 2021  
  color: "rojo"  
  marca: "Seat"  
  modelo: "León"  
  precio: 12000  
  ► [[Prototype]]: Object
```

```
const coche={  
  marca: "Seat",  
  modelo: "León",  
  año: 2021,  
};  
  
const producto={  
  precio: 12000,  
  color: "rojo"  
};  
  
const resultado={...coche, ...producto};  
console.log(resultado);
```



[this]

Los objetos, además de propiedades, pueden tener funciones, llamadas métodos.

Se tiene que usar la palabra clave *this* (*este*) dentro del método (la función) para acceder a las propiedades del objeto debido al ámbito local de las mismas.

```
const coche={  
  marca: "Seat",  
  modelo: "León",  
  año: 2021,  
  mostrarInfo: function() {  
    console.log(`Marca: ${this.marca} - Modelo: ${this.modelo}`)  
  }  
};  
coche.mostrarInfo();
```

Se volverá con *this* más adelante cuando se estudien las funciones con más profundidad.

[object constructor]

Hasta ahora, se han creado objetos simplemente inicializándolos mediante llaves {}.

Existe otra forma de crear objetos a través de funciones, esta forma de crear objetos recibe el nombre *Object Constructor*.

Esta forma resulta muy útil cuando se tienen que crear muchos objetos parecidos.

Se utiliza una función a modo de constructor del objeto y la palabra reservada *new*, que crea el objeto.

```
const coche={  
  — marca: "Seat",  
  — modelo: "León",  
};
```

```
function Coche2(marca, modelo){  
  this.marca=marca;  
  this.modelo=modelo;  
};
```

```
const leon=new Coche2('Seat', 'León');  
const ibiza=new Coche2('Seat', 'Ibiza');  
console.log(leon);  
console.log(ibiza);
```

```
▼ Coche2 {marca: 'Seat', modelo: 'León'}  
  marca: "Seat"  
  modelo: "León"  
  ► [[Prototype]]: Object  
▼ Coche2 {marca: 'Seat', modelo: 'Ibiza'}  
  marca: "Seat"  
  modelo: "Ibiza"  
  ► [[Prototype]]: Object
```

[object constructor]

También se puede utilizar un objeto ya creado como plantilla para crear un nuevo objeto igual al primero.

En este caso se utiliza el método `Object.create()` que crea un objeto nuevo, utilizando un objeto existente que se le pasa como argumento.

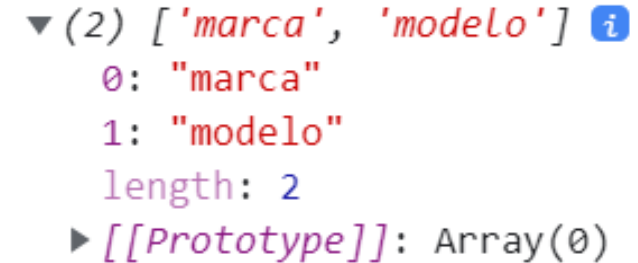
```
const coche={  
  marca: "Seat",  
  modelo: "León",  
};  
  
const coche2= Object.create(coche);  
  
coche2.marca='Seat';  
coche2.modelo='Ibiza';  
  
console.log(coche2);
```

```
▼ {marca: 'Seat', modelo: 'Ibiza'}  
  marca: "Seat"  
  modelo: "Ibiza"  
  ► [[Prototype]]: Object
```

[keys]

Existen otros métodos dentro de *Object*; uno de ellos es *keys()* que devuelve un array con el nombre de las propiedades de un objeto existente que se le pasa como argumento.

```
const coche={  
  marca: "Seat",  
  modelo: "León",  
};  
  
console.log(Object.keys(coche));
```

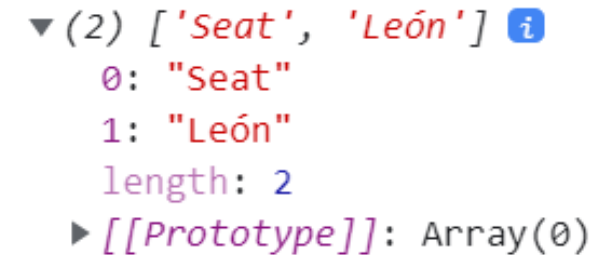


```
▼ (2) ['marca', 'modelo'] ⓘ  
  0: "marca"  
  1: "modelo"  
  length: 2  
  ► [[Prototype]]: Array(0)
```

[values]

Otros de los métodos de *Object* es *values()* que devuelve un array con los valores de las propiedades de un objeto existente que se le pasa como argumento.

```
const coche={  
  marca: "Seat",  
  modelo: "León",  
};  
  
console.log(Object.values(coche));
```



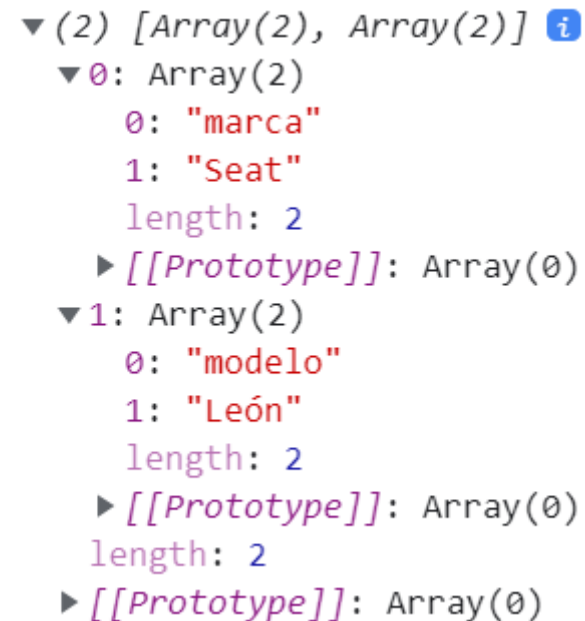
```
▼ (2) ['Seat', 'León'] ⓘ  
  0: "Seat"  
  1: "León"  
  length: 2  
  ► [[Prototype]]: Array(0)
```

[entries]

`entries()` es el último de los métodos de *Object* que se va a estudiar,

`entries()` devuelve un array que contiene arrays formados por pares *clave-valor* que se corresponden con las propiedades y sus valores de un objeto existente que se le pasa como argumento.

```
const coche={  
  marca: "Seat",  
  modelo: "León",  
};  
  
console.log(Object.entries(coche));
```



```
▼ (2) [Array(2), Array(2)] ⓘ  
  ▼ 0: Array(2)  
    0: "marca"  
    1: "Seat"  
    length: 2  
    ► [[Prototype]]: Array(0)  
  ▼ 1: Array(2)  
    0: "modelo"  
    1: "León"  
    length: 2  
    ► [[Prototype]]: Array(0)  
length: 2  
► [[Prototype]]: Array(0)
```

Bibliografía y recursos online

- <https://platzi.com/blog/metodologia-scrum-fases/>
- https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Grammar_and_types
- https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Working_with_Objects