Objetos literales

DigitalHouse >
Coding School

Podemos decir que los objetos literales son la **representación** en **código** de un **elemento** de la **vida real**.

Índice

1. Estructura, propiedades y métodos
2. Funciones constructoras

1 Estructura, propiedades y métodos

Estructura básica

Un **objeto** es una estructura de datos que puede contener **propiedades** y **métodos**.

Para crearlo usamos llave de apertura y de cierre { }.

```
{}
let auto = {
    patente : 'AC 134 DD';
};
```

Propiedad

Definimos el nombre de la **propiedad** del objeto.

Dos puntos

Separa el nombre de la propiedad, de su valor.

Valor

Puede ser cualquier **tipo de dato** que conocemos.

Propiedades de un objeto

Un objeto puede tener la cantidad de propiedades que queramos. Si hay más de una, las separamos con comas `,`.

Con la notación **objeto.propiedad** accedemos al valor de cada una de ellas.

```
let tenista = {
    nombre: 'Roger',
    apellido: 'Federer'
};

console.log(tenista.nombre) // Roger
console.log(tenista.apellido) // Federer
```

Métodos de un objeto

Una propiedad puede almacenar cualquier tipo de dato.

Si una propiedad almacena una **función**, diremos que es un **método** del objeto.

```
{}
let tenista = {
    nombre: 'Roger',
    edad: 38,
    activo: true,
    saludar: function() {
        return '¡Hola! Me llamo Roger';
    }
};
```

Ejecución de un método de un objeto

Para ejecutar un método de un objeto usamos la notación **objeto.metodo()**. Los paréntesis del final son los que hacen que el método se ejecute.

```
{}
let tenista = {
    nombre: 'Roger',
    apellido: 'Federer',
    saludar: function() {
        return '¡Hola! Me llamo Roger';
    }
};

console.log(tenista.saludar()); // ¡Hola! Me llamo Roger
```

Trabajando dentro del objeto

La palabra reservada **this** hace referencia al objeto en sí donde estamos parados. Es decir, el objeto en sí donde escribimos la palabra.

Con la notación **this.propiedad** accedemos al valor de cada propiedad interna de ese objeto.

```
{}
let tenista = {
    nombre: 'Roger',
    apellido: 'Federer',
    saludar: function() { return '¡Hola! Me llamo ' + this.nombre; }
};

console.log(tenista.saludar()); // ¡Hola! Me llamo Roger
```

2 Funciones constructoras

Funciones constructoras

JavaScript nos da una opción más para crear un objeto, a través del uso de una **función constructora**.

La función constructora nos permite armar un molde y luego crear todos los objetos que necesitemos.

La función recibe un parámetro por cada propiedad que queramos asignarle al objeto.

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

Estructura de una función constructora

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

Nombre

Definimos un **nombre** para la función, que será el nombre de nuestro **constructor**.

Por convención, solemos nombrar a las funciones constructoras con la primera letra mayúscula. Esto es para diferenciarlas de las funciones normales.

Estructura de una función constructora

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

Parámetros

Definimos la cantidad de parámetros que consideremos necesarios para crear nuestro objeto.

Todos los parámetros serán obligatorios para poder crear el objeto, a menos que definamos lo contrario.

Estructura de una función constructora

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

Propiedades

Con la notación **this.propiedad** definimos la propiedad del objeto que estamos creando en ese momento.

Por lo general los valores de las propiedades serán los que vengan por parámetros.

Instanciar un objeto

La función constructora **Auto()** espera dos parámetros: marca y modelo. Para crear un objeto Auto debemos usar la palabra reservada **new** y llamar a la función pasándole los parámetros que espera.

```
{}  let miAuto = new Auto('Ford', 'Falcon');
```

Cuando ejecutamos el método **new** para crear un objeto, lo que nos devuelve es una instancia. Es decir, en la variable miAuto tendremos almacenada una instancia del objeto Auto. Usando la misma función, podemos instanciar cuantos autos queramos.

```
{}  let miOtroAuto = new Auto('Chevrolet', 'Corvette');
```


DigitalHouse>
Coding School