

Esta clase va a ser

- grabada

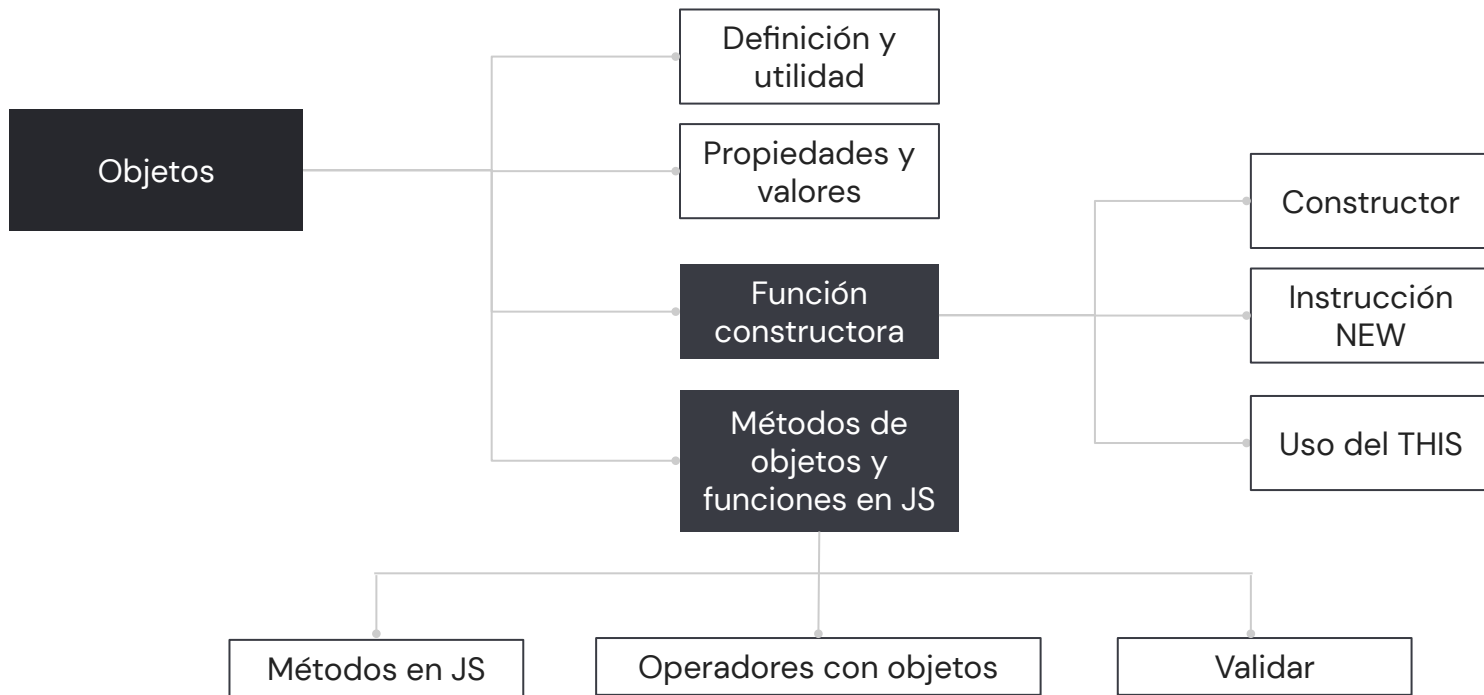
Clase 05. JAVASCRIPT

Objetos

Objetivos de la clase

- Comprender qué es un **objeto** en JavaScript y cómo se usa.
- Conocer qué es una **función constructora** y un objeto creado con ella.
- Analizar cuáles son las propiedades de los objetos y sus **métodos**.
- Diferenciar progresivamente **métodos** y **funciones**.
- Conocer qué es una declaración de **clases** en JavaScript.

MAPA DE CONCEPTOS



Temario

04

Programación con funciones

- ✓ Funciones y propiedades básicas
- ✓ Scope
- ✓ Funciones anónimas y funciones flecha

05

Objetos

- ✓ [Constructores](#)
- ✓ [Métodos y operaciones](#)
- ✓ [Clases](#)

06

Arrays

- ✓ [Array: métodos y propiedades](#)
- ✓ [Arrays de objetos](#)

Glosario

Parámetros: cuando necesitamos enviarle a la función algún valor o dato para que luego la misma lo utilice en sus operaciones, estamos hablando de los parámetros de la función.

Ámbito de una variable (llamado "scope" en inglés): es la zona del programa en la que se define la variable, el contexto al que pertenece la misma dentro de un algoritmo. JavaScript define dos ámbitos para las variables: global y local.

- **Variables locales:** Se crean y se usan siempre en las funciones.
- **Variables globales:** Se definen fuera de las funciones, y se pueden usar en cualquier lugar del código.

Objetos: Conceptos generales



¿Qué es un objeto?

En JS, los objetos son estructuras que podemos definir para agrupar valores bajo un mismo criterio. Podemos decir que **un objeto es una colección de datos relacionados como una entidad**. Se componen de un listado de pares clave-valor, es decir, contienen **propiedades** y **valores** agrupados.

¿Por qué usamos objetos?

La utilidad de los objetos deviene de su composición por varios valores y operaciones comunes (funciones) para todos los elementos de este tipo y sus propiedades".

```
let nombre = "Homero";  
let edad   = 39;  
let calle  = "Av. Siempreviva 742";  
// Las variables anteriores entran relacionados entre sí, entonces mejor usamos un objeto literal  
const personal = { nombre: "Homero", edad: 39, calle: "Av. Siempreviva 742"  
}
```

Anatomía de un objeto

Un objeto literal se define directamente entre llaves `{}`

Los valores que almacenan se listan separados por coma, bajo la forma `propiedad: valor`

```
const personal = {  
  nombre: "Homero",  
  edad: 39,  
  calle: "Av. Siempreviva 742"  
}
```

Obteniendo valores del objeto

Para obtener el valor de una propiedad en un objeto utilizamos la notación punto (.): El nombre de la variable del objeto, seguido de punto y el nombre de la propiedad:

```
const personal = { nombre: "Homer",  
                  edad: 39,  
                  calle: "Av. Siempreviva 742"}  
  
console.log(personal.nombre)  
console.log(personal.edad)  
console.log(personal.calle)
```

Obteniendo valores del objeto

Otra forma de obtener el valor de una propiedad en un objeto utilizamos la notación **corchetes ([])**: El nombre de la variable del objeto, seguido de corchetes y dentro de ellos un string del nombre de la propiedad:

```
const personal = { nombre: "Homero",  
                  edad : 39,  
                  calle : "Av. Siempreviva 742 "}  
  
console.log(personal["nombre"])  
console.log(personal["edad"])  
console.log(personal["calle"])
```

Asignar valores a las propiedades

Es posible usar las dos formas(corchetes y paréntesis) para acceder a las propiedades y asignar nuevos valores a los datos almacenados en las propiedades del objeto.

```
const personal = { nombre: "Homero",  
                  edad: 39,  
                  calle: "Av. Siempreviva 742"}  
  
personal["nombre"] = "Marge"  
personal.edad = 36
```

Objetos: Constructores

Constructores

En JS, **el constructor de un objeto es una función que usamos para crear un nuevo objeto cada vez que sea necesario.** Con esta "función constructora" podemos inicializar las propiedades del objeto al momento de ser instanciado con **new**.

```
function Persona(nombre, edad, calle) {  
  this.nombre = nombre;  
  this.edad    = edad;  
  this.calle   = calle;  
}  
  
const persona1 = new Persona("Homer", 39, "Av. Siempreviva 742");  
const persona2 = new Persona("Marge", 36, "Av. Siempreviva 742");
```

Constructor y New

En el ejemplo anterior, se define la función `Persona`, donde se asignan las diferentes propiedades con los valores recibidos como parámetros.

Luego, en algún lugar del código posterior a esas líneas, se puede construir un objeto `Persona` declarando una variable y asignando la referencia del objeto instanciado mediante la instrucción `new Persona(...)`

Uso del THIS

La palabra clave `this` ("este") refiere al elemento actual en el que se está escribiendo el código. Cuando se emplea un función constructora para crear un objeto (con la palabra clave `new`), `this` está enlazado al nuevo objeto instanciado.

`This` es muy útil para asegurar que se emplean las propiedades del objeto actual.

```
function Persona(literal) {  
  this.nombre = literal.nombre;  
  this.edad   = literal.edad;  
  this.calle  = literal.calle;  
}  
  
const personal = new Persona({ nombre: "Homero", edad: 39, calle: "Av.Siempreviva 742" });
```



Ejemplo en vivo

¡VAMOS A PRACTICAR LO VISTO!



Break

¡10 minutos y volvemos!

Métodos y operaciones con objetos

Método <> Función

Como vimos anteriormente, las funciones en JS se pueden definir en cualquier parte del código, y pueden ser llamadas desde cualquier otra parte del código posterior.

Los métodos de los objetos también son técnicamente funciones, sólo que se limitan a poder ser ejecutados solo desde el mismo objeto.

Función

```
//Funciones: Generalmente retornar un valor y son de acceso global.  
function f1(){  
    return this;  
}
```

Método

```
//Métodos: Se requiere un objeto y puede no retornar un valor.  
function Persona(nombre, edad, calle) {  
    this.nombre = nombre;  
    this.edad = edad;  
    this.calle = calle;  
}
```

Métodos en objetos JS

JavaScript cuenta con sus propios objetos, incluso ya usamos algunos de ellos sin identificar que son objeto.

Por ejemplo: Cada vez que creamos una cadena de caracteres se crea automáticamente como una instancia del objeto String y, por lo tanto, tiene varios métodos/propiedades comunes disponibles en ella.

```
let cadena = "HOLA CODER";  
//Propiedad de objeto String: Largo de la cadena.  
console.log(cadena.length);  
//Método de objeto String: Pasar a minúscula.  
console.log(cadena.toLowerCase());  
//Método de objeto String: Pasar a mayúscula.  
console.log(cadena.toUpperCase());
```

Métodos personalizados

Podemos crear nuestros propios métodos para objetos personalizados, referenciando funciones por su nombre o definiendo funciones anónimas asociadas a una propiedad de la función constructora.

Llamar a un método es similar a acceder a una propiedad, pero se agrega () al final del nombre del método, posiblemente con argumentos.

```
function Persona(nombre, edad, calle) {  
    this.nombre = nombre;  
    this.edad    = edad;  
    this.calle   = calle;  
    this.hablar = function() { console.log("HOLA SOY " + this.nombre) }  
}  
  
const personal = new Persona("Homero", 39, "Av. Siempreviva 742");  
const persona2 = new Persona("Marge", 36, "Av. Siempreviva 742");  
personal.hablar();  
persona2.hablar();
```


Operador IN y FOR...IN

El operador **in** devuelve true si la propiedad especificada existe en el objeto.

Mientras que el **bucle for...in** permite acceder a todas las propiedades del objeto, obteniendo una propiedad por cada iteración.

```
const personal = { nombre: "Homero", edad: 39, calle: "Av. Siempreviva 742"};
//devuelve true porque la clave "nombre" existe en el objeto personal
console.log( "nombre" in personal);
//devuelve false porque la clave "origen" no existe en el objeto personal
console.log( "origen" in personal);
//recorremos todas las propiedades del objeto con el ciclo for...in
for (const propiedad in personal) {
    console.log(personal[propiedad]);
}
```

Classes

Clases

Las **clases** de JavaScript, introducidas en ES6, proveen una sintaxis mucho más clara y simple para crear objetos personalizados.

Son una equivalencia al empleo de **función constructora y permite definir distintos tipos de métodos.**

```
class Persona{  
  constructor(nombre, edad, calle) {  
    this.nombre = nombre;  
    this.edad   = edad;  
    this.calle  = calle;  
  }  
}  
  
const personal = new Persona("Homero", 39, "Av. Siempreviva 742");
```

Clases y Métodos

En la declaración de clase, **la función constructora es reemplazada por el método constructor.** Los métodos en las clases no referencian a propiedades, se declaran dentro del bloque sin la palabra **function**.

```
class Persona{
  constructor(nombre, edad, calle) {
    this.nombre = nombre;
    this.edad   = edad;
    this.calle  = calle;
  }
  hablar(){
    console.log("HOLA SOY "+ this.nombre);
  }
}

const personal = new Persona("Homer", 39, "Av. Siempreviva 742");
personal.hablar();
```

```
class Producto {  
    constructor(nombre, precio) {  
        this.nombre = nombre.toUpperCase();  
        this.precio = parseFloat(precio);  
        this.vendido = false;  
    }  
    sumaIva() {  
        this.precio = this.precio * 1.21;  
    }  
    vender() {  
        this.vendido = true;  
    }  
}  
  
const producto1 = new Producto("arroz", "125");  
const producto2 = new Producto("fideo", "50");  
  
producto1.sumaIva();  
producto2.sumaIva();  
producto1.vender();
```

Ejemplo aplicado: Clase producto





Ejemplo en vivo

¡VAMOS A PRACTICAR LO VISTO!

Objetos

Resumen

- ✓ Los objetos tienen propiedades y métodos.
- ✓ El método constructor de un objeto sirve para crear el mismo y asignarle sus propiedades. Permite crear varios objetos usando el mismo constructor.
- ✓ Las funciones de JS son generalmente de acceso global y los métodos son únicamente utilizados para ser invocados por los objetos que lo contienen.
- ✓ Las clases son otra forma de crear objetos personalizados en JS.



Actividad en clase: Incorporar objetos

Duración: 25/30 minutos



Incorporar objetos

Consigna

A partir de los ejemplos mostrados la primera clase, y en función del tipo de simulador que hayas elegido para tu proyecto, deberás:

- ✓ Crear al menos un objeto para controlar el funcionamiento de tu simulador.
- ✓ Incorporar sus propiedades y su constructor.
- ✓ Invocar a ese objeto en algún momento donde el usuario realice alguna acción.
- ✓ Utilizar sus métodos.

Formato

- ✓ Página HTML y código fuente en JavaScript.



Incorporar objetos

Aspectos a incluir

- ✓ Archivo HTML y Archivo JS, referenciado en el HTML por etiqueta `<script src="js/miarchivo.js"></script>`, que incluya la definición de un algoritmo en JavaScript que emplee objetos para elementos con propiedades y métodos comunes.

Sugerencias

- ✓ Reconocer elementos en el simulador cuya información está compuesta por más de un valor y existen operaciones comunes (funciones) para todos los elementos de este tipo y sus propiedades.

Ejemplo

- ✓ Algunos objetos a identificar que forman parte del simulador pueden ser: Producto, Persona, Libro, Auto, Comida, Bebida, Tarea, etc.

¿Preguntas?



Para pensar

¿Te gustaría comprobar tus conocimientos de la clase?

Te compartimos a través del chat de zoom el enlace a un breve quiz de tarea.

Para el profesor:

Acceder a la carpeta "Quizzes" de la camada.

Ingresar al formulario de la clase.

Pulsar el botón "Invitar".

Copiar el enlace.

Compartir el enlace a los alumnos a través del chat.



MATERIAL AMPLIADO

Recursos

Objetos

- ✓ [Los apuntes de Majo \(Página 25 a 30\).](#)
- ✓ [Te lo explico con gatitos](#)

Documentación

- ✓ [Documentación Objetos](#)
- ✓ [Documentación Clases](#)

Disponible en nuestro repositorio.

Resumen de la clase hoy

- ✓ Objetos.
- ✓ Constructor y new.
- ✓ Métodos y propiedades.

Muchas gracias.

Opina y valora
esta clase

#DemocratizandoLaEducación