

***RECUERDA PONER A GRABAR LA
CLASE***





¿DUDAS DEL ON-BOARDING?

MIRALO AQUI



Clase 05. JAVASCRIPT

OBJETOS



OBJETIVOS DE LA CLASE

Entender:

- ¿Qué es un objeto en JS y cómo se usa?
- ¿Qué es una función constructora y un objeto creado con ella?
- ¿Cuáles son las propiedades de los objetos y sus métodos?
- ¿Cómo diferenciar método de funciones?
- ¿Qué es una declaración de clase?

GLOSARIO:

Clase 4

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.

MAPA DE CONCEPTOS

MAPA DE CONCEPTOS CLASE 5

¡Para
recordar!



CRONOGRAMA DEL CURSO

Clase 4



Programación avanzada con funciones



EJEMPLO EN VIVO



CREAR UN ALGORITMO
UTILIZANDO FUNCIÓN



SIMULADOR INTERACTIVO

Clase 5



Objetos



EJEMPLOS EN VIVO



CREAR UN OBJETO Y
UTILIZARLO



INCORPORAR OBJETOS

Clase 6



Arrays



EJEMPLOS EN VIVO



CREAR UN ALGORITMO
CON ARRAYS



INCORPORAR ARRAYS



PRIMERA ENTREGA DEL
PROYECTO FINAL



HERRAMIENTAS DE LA CLASE

Les compartimos algunos recursos para acompañar la clase

- Guión de clase N° 5 [aquí](#).
- Quizz de clase N° 5 [aquí](#)
- Booklet de Javascript [aquí](#)
- FAQs de Javascript [aquí](#)

OBJETOS: CONCEPTOS GENERALES



¿QUÉ ES UN OBJETO?

En programación, y también en JS, un objeto es una colección de datos relacionados con funcionalidad, que generalmente consta de variables, denominadas propiedades, y funciones asociadas, llamadas métodos.

Por ejemplo: el objeto *persona*, tendría como una propiedad la altura, y como un método, hablar.

¿POR QUÉ USAMOS OBJETOS?

Cuando tengo que crear un elemento cuya información está compuesta por más de un valor y existen operaciones comunes (funciones) para todos los elementos de este tipo y sus propiedades, debe definirse como un objeto.

```
let nombre = "Homero";  
let edad   = 39;  
let calle  = "Av. Siempreviva 742";  
// Los variables anteriores entran relacionados entre sí, entonces mejor usamos un objeto literal  
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: "Homero",  
                  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 de acceder a las propiedades para asignar nuevos valores a los datos almacenados en la propiedades del objeto.

```
const personal = { nombre: "Homer",  
                    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("Homero", 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!

CODER HOUSE



BREAK

¡5/10 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 únicamente desde el mismo objeto.

FUNCIÓN

```
//Funciones: Generalmente retornar un valor y son de acceso global.  
function fl(){  
    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 nuestro 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 persona1 = new Persona("Homer", 39, "Av. Siempreviva 742");  
const persona2 = new Persona("Marge", 36, "Av. Siempreviva 742");  
persona1.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: "Homer", 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]);
}
```

CLASES

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 tipo 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 reemplaza por el método **constructor**. Los métodos en las clases no referencian a propiedades, se declaran dentro del bloque si 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("Homero", 39, "Av. Siempreviva 742");
personal.hablar();
```

EJEMPLO APLICADO: CLASE PRODUCTO

```
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
en vivo



¡VAMOS A PRACTICAR LO VISTO!

CODER HOUSE

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 para ser invocados por los objetos que lo contienen.
- Las clases son otra forma de crear objetos personalizados en JS.



INCORPORAR OBJETOS

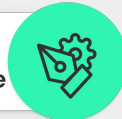
Traslada al proyecto integrador el concepto de objetos, visto en la clase de hoy.

INCORPORAR OBJETOS

Formato: Página HTML y código fuente en JavaScript. Debe identificar el apellido del alumno/a en el nombre de archivo comprimido por “claseApellido”.

Sugerencia: 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.

Desafío
entregable



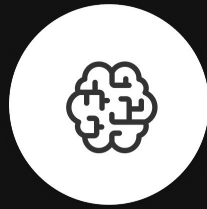
>> Consigna: A partir de los ejemplos mostrados la primera clase, y en función del tipo de simulador que hayas elegido, deberás:

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

>>Aspectos a incluir en el entregable: 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.

>>Ejemplo:

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



¡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*



RECURSOS:

Material
ampliado



- Objetos |

Los apuntes de Majo (Página 25 a 30).

Te lo explico con gatitos.

- Documentación |

Documentación Objetos.

Documentación Clases.

Disponible en [nuestro repositorio](#).

CODER HOUSE

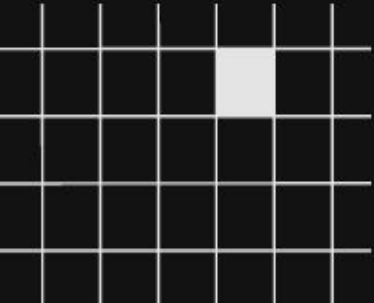
¿PREGUNTAS?





¡MUCHAS GRACIAS!

Resumen de lo visto en clase hoy:

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



OPINA Y VALORA ESTA CLASE

#DEMOCRATIZANDO LA EDUCACIÓN

CODER HOUSE

CODER HOUSE

***¡GRACIAS POR ESTUDIAR CON
NOSOTROS!***
