CLASES EN JAVASCRIPT (Class)

ECMAScript 2015, también conocido como ES6, introdujo las clases de JavaScript.

Las clases de JavaScript son plantillas para objetos JavaScript.

1. Declarar una clase

Utilice la palabra clave para crear una clase.`class`

Agregue siempre un método denominado :`constructor()`

```
class ClassName {
  constructor() { ... }
}
```

Ejemplo

```
class Car {
  constructor(name, year) {
    this.name = name;
    this.year = year;
  }
}
```

El ejemplo anterior crea una clase denominada "Car".

La clase tiene dos propiedades iniciales: "nombre" y "año".

Una clase JavaScript **no** es un objeto.

Es una **plantilla** para objetos JavaScript.

2. Uso de una clase

Cuando tiene una clase, puede usarla para crear objetos:

Ejemplo

```
let myCar1 = new Car("Ford", 2014);
let myCar2 = new Car("Audi", 2019);
```

Pruébelo usted mismo »

En el ejemplo anterior se utiliza la **clase Car** para crear dos **objetos Car.**

Se llama automáticamente al método constructor cuando se crea un nuevo objeto.

3. El método Constructor

El método constructor es un método especial:

- Tiene que tener el nombre exacto "constructor"
- Se ejecuta automáticamente cuando se crea un nuevo objeto
- Se utiliza para inicializar las propiedades del objeto

Si no define un método constructor, JavaScript agregará un método constructor vacío.

4. Métodos de clase

Los métodos de clase se crean con la misma sintaxis que los métodos de objeto.

Utilice la palabra clave para crear una clase.`class`

Añade siempre un método.`constructor()`

A continuación, agregue cualquier número de métodos.

Sintaxis

```
class ClassName {
  constructor() { ... }
  method_1() { ... }
  method_2() { ... }
  method_3() { ... }
}
```

Cree un método Class denominado "age", que devuelva la antigüedad del coche:

```
class Car {
  constructor(name, year) {
    this.name = name;
    this.year = year;
  }
  age() {
    let date = new Date();
    return date.getFullYear() - this.year;
  }
}
```

```
let myCar = new Car("Ford", 2014);
document.getElementById("demo").innerHTML =
"My car is " + myCar.age() + " years old.";
```

Puede enviar parámetros a los métodos Class:

```
class Car {
  constructor(name, year) {
    this.name = name;
    this.year = year;
  }
  age(x) {
    return x - this.year;
  }
}

let date = new Date();
let year = date.getFullYear();

let myCar = new Car("Ford", 2014);
document.getElementById("demo").innerHTML=
"My car is " + myCar.age(year) + " years old.";
```

Métodos estáticos

La *palabra clave* static define un método estático para una clase. Los métodos estáticos son llamados sin instanciar su clase y **no** pueden ser llamados mediante una instancia de clase. Los métodos estáticos son a menudo usados para crear funciones de utilidad para una aplicación.

```
class Punto {
  constructor ( x , y ){
    this.x = x;
    this.y = y;
  }
  static distancia ( a , b) {
    const dx = a.x - b.x;
    const dy = a.y - b.y;
    return Math.sqrt ( dx * dx + dy * dy );
  }
}
const p1 = new Punto(5, 5);
const p2 = new Punto(10, 10);
console.log (Punto.distancia(p1, p2)); // 7.0710678118654755
```

5. Getters y Setters

Los getters y setters le permiten definir objetos descriptores de acceso (propiedades calculadas).

```
<script>
<p id="demo"></p>
<script>
// Declare an object:
class person  {
  constructor(){
  this.firstName= "John",
  this.lastName="Doe",
  this.language= "NO"
  }
  get lang() {
    return this.language;
  }
  get fullName() {
    return this.firstName + " " + this.lastName;
  }
  set lang(value) {
    this.language = value;
  }
}
// Set a property using set:
person1=new person();
person1.lang = "en";
// Display data from the object:
document.getElementById("demo").innerHTML = person1.lang;
document.getElementById("demo").innerHTML += "<br>"+person1.fullName;
</script>
```

6. Subclases con extends

La palabra clave `extends` es usada en *declaraciones de clase* o *expresiones de clase* para crear una clase hija o derivada.

```
<script>
class Animal {
  constructor(nombre) {
    this.nombre = nombre;
  }
  hablar() {
    console.log(this.nombre + ' hace un ruido.');
  }
}
class Perro extends Animal {
  hablar() {
    console.log(this.nombre + ' ladra.');
  }
}
let miAnimal = new Animal("perro");
let miPerro=new Perro("chispa");
miAnimal.hablar()
miPerro.hablar();
</script>
```

7. Propiedades y métodos privados

Anteponer # al nombre de la propiedad o método

```
<html>
<h1>Creación de una clase con propiedad y método privados</h1>
<br>
<br>
<script>
class Gato {
  #frecuencia;
 constructor(nombre){
    this.nombre = nombre;
    //Creamos una variable privada
    this.#frecuencia = "moderada"
 }
 // Creamos un metodo privado
    #irAlBano() {
        console.log("El gato va al baño con frecuencia "+this.#frecuencia);
    }
 // Creamos método público para acceder al privado
    accesoMetodoPrivado(){
                this.#irAlBano();
                }
}
var Michin = new Gato("Michifu");
console.log(Michin.nombre)
console.log(Michin.frecuencia) //No tiene acceso
Michin.accesoMetodoPrivado()
</script>
</html>
```

8. 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 trambién 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
```