



DESARROLLO WEB EN ENTORNO CLIENTE

# UD4 –OBJETOS

---

CICLO FORMATIVO DE GRADO SUPERIOR EN DESARROLLO DE  
APLICACIONES WEB

I.E.S. HERMENEGILDO LANZ – 2022/2023

PROFESORA: VANESA ESPÍN

[vespin@ieshlanz.es](mailto:vespin@ieshlanz.es)

# Índice

---

1. Estructuras: Arrays, Conjuntos y Mapas
2. Funciones
- 3. Objetos**
  - a. Introducción a la POO
  - b. Orientación a Objetos en Javascript
  - c. Uso de objetos
  - d. Creación de objetos literales
  - e. Uso avanzado de objetos:
    - Constructores
    - InstanceOf
    - Prototipos, herencia y clases
    - JSON

# Programación Orientada a Objetos (POO) recordatorio

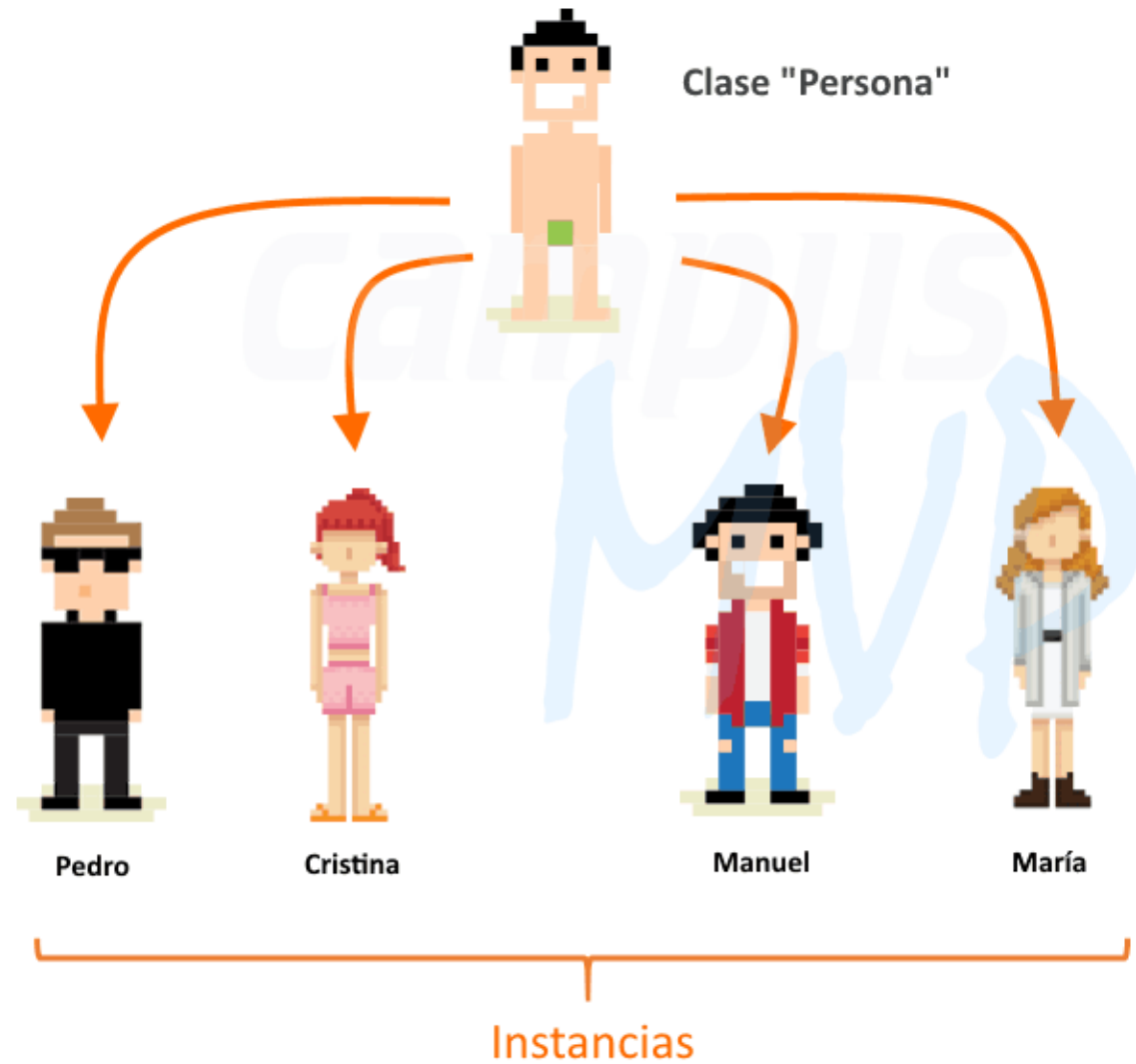
---

**Modelo de programación donde cada objeto se programa de forma independiente y los objetos pueden mandarse mensajes entre sí.**

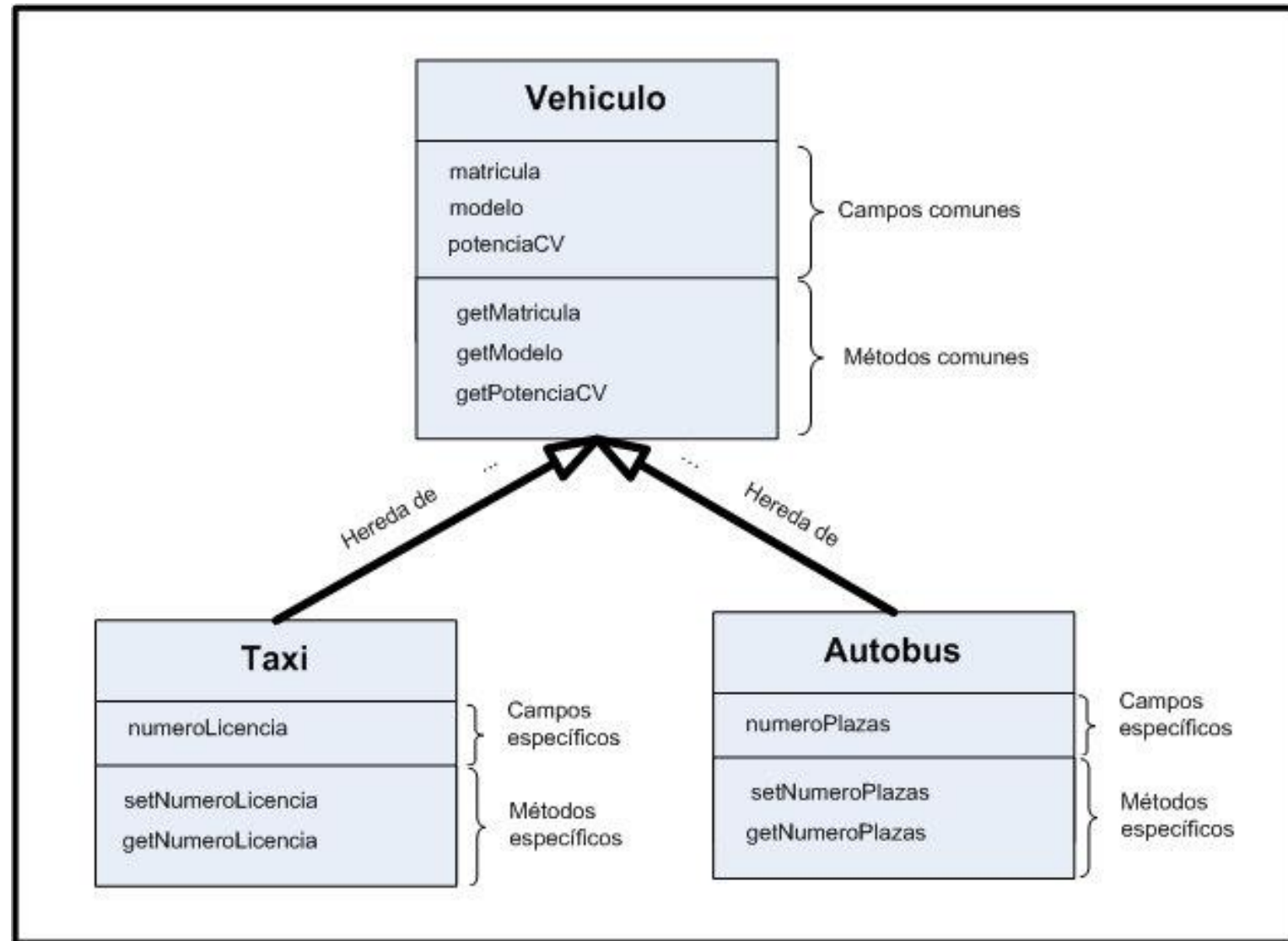
- **Objeto**: estructura que aglutina datos (atributos o propiedades) y acciones (métodos).  
*Objeto vehículo: atributo-matrícula métodos-acelerar y frenar*
- La definición de un tipo de objeto es una **Clase**.
- Características de la POO:
  - **Abstracción**: puedo utilizar un objeto sin conocer su funcionamiento interno gracias a su interfaz
  - **Encapsulamiento**: puedo ocultar métodos y propiedades al exterior del objeto
  - **Polimorfismo**: diferentes tipos de objetos pueden tener métodos con el mismo nombre pero que actúen de forma distinta.
  - **Herencia**: a partir de una clase base se puede definir una clase derivada que herede sus métodos y propiedades. Por ejemplo: la clase coche se define a partir de la clase vehículo por lo que también puede acelerar y frenar

# Instancias

En la POO los objetos son instancias de una clase



# Herencia



# Orientación a objetos en Javascript

---

- Hay lenguajes que son puramente orientados a objetos, siguiendo a rajatabla las características citadas anteriormente. Otros lenguajes son **más flexibles**.
- Javascript trabaja con objetos de forma diferente a Java o C++ pero sin duda **los objetos son parte fundamental** de Javascript y son su principal paradigma.
- Gran diferencia: en Javascript puedo **implementar objetos sin crear clases** que los definan.

[https://www.w3schools.com/js/js\\_objects.asp](https://www.w3schools.com/js/js_objects.asp)

# Acceso a propiedades y métodos de objetos

---

- Acceso a propiedad:
  - *objeto.propiedad* o *objeto["propiedad"]*
  - *coche.color* o *coche["color"]*
- Cambiar valor de propiedad:
  - *objeto.propiedad=valor* o *objeto["propiedad"]=valor*
  - *coche.color="Rojo"* o *coche["color"]="Rojo"*
- Para usar un método:
  - *objeto.método(parámetros)* o *objeto["método"](parámetros)*
  - *coche.acelerar(25)* o *coche["acelerar"](25)*

[https://www.w3schools.com/js/js\\_objects.asp](https://www.w3schools.com/js/js_objects.asp)

# Objetos literales (o instancias directas)

- Son el tipo de objeto más sencillo Javascript. Se definen sus propiedades y métodos sobre la marcha

```
const person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
```

```
const person = {  
  firstName: "John",  
  lastName: "Doe",  
  age: 50,  
  eyeColor: "blue"  
};
```

Property	Property Value
firstName	John
lastName	Doe
age	50
eyeColor	blue

```
console.log(person.firstName + " " + person.lastName); //John Doe
```



# Objetos literales con métodos

- Los métodos son funciones definidas en el objeto como propiedades
- Podemos hacer referencia al propio objeto con *this*

```
const person = {  
  firstName: "John",  
  lastName : "Doe",  
  id       : 5566,  
  nombrar : function() {  
    return this.firstName + " " + this.lastName;  
  }  
};
```

```
console.log(person.nombrar()); //John Doe
```

- Si llamamos al método sin los paréntesis devuelve su definición:

```
console.log(person.nombrar);
```

```
f () {  
  return this.firstName + " " + this.lastName;  
}
```

## Inciso: Uso the THIS

# What is **this**?

In JavaScript, the `this` keyword refers to an **object**.

**Which** object depends on how `this` is being invoked (used or called).

The `this` keyword refers to different objects depending on how it is used:

In an object method, `this` refers to the **object**.

Alone, `this` refers to the **global object**.

In a function, `this` refers to the **global object**.

In a function, in strict mode, `this` is `undefined`.

In an event, `this` refers to the **element** that received the event.

Methods like `call()`, `apply()`, and `bind()` can refer `this` to **any object**.

Para saber más sobre el uso de this:

[https://www.w3schools.com/js/js\\_this.asp](https://www.w3schools.com/js/js_this.asp)

# Propiedades calculadas

---

Puedo usar variables como propiedades:

```
let fruta=prompt ("Elige una fruta"); //El usuario introduce manzana
let bolsa = {
    [fruta]: 5, // El nombre de la propiedad se obtiene de la var fruta
};
alert( bolsa.manzana ); // 5 si fruta es manzana
```

Equivalente a:

```
let fruta = prompt("Elige una fruta", "manzana");
let bolsa = {};
// Toma el nombre de la propiedad de la variable fruta
bolsa[fruta] = 5;
```

# Recorrer las propiedades de un objeto

---

- Podemos usar **for..in** para obtener las propiedades de un objeto

```
//Recorremos las PROPIEDADES
for (let i in person){ //se recorre como un array, por eso corchetes
    //evitamos las funciones
    if (typeof person[i]!="function"){ //OJO no typeof person.i
        console.log(`${i} tiene el valor ${person[i]}`); //OJO no ${person.i}
    }
}
```

```
firstName tiene el valor John
lastName tiene el valor Doe
id tiene el valor 5566
```

# Recorrer las propiedades de un objeto

---

Además, para un objeto o:

- **Object.keys(o):** devuelve un array con todos los nombres de propiedades enumerables ("claves") propias (no en la cadena de prototipos) de un objeto o.
- **Object.getOwnPropertyNames(o):** devuelve un array con todos los nombres (enumerables o no) de las propiedades de un objeto o.

```
console.log(Object.getOwnPropertyNames(person));
```

Escribe: (4) ['firstName', 'lastName', 'id', 'nombrar']

# Borrar las propiedades de un objeto

---

- Podemos usar el operador **delete** para eliminar propiedades de objeto

```
// Create an object:
const person = {
  firstName: "John",
  lastName: "Doe",
  id: 5566,
  nombrar: function() {
    return this.firstName + " " + this.lastName;
  }
};

delete person.id;
console.log(person.id); //Escribe undefined
```



# USO AVANZADO DE OBJETOS

- CONSTRUCTORES
- INSTANCEOF
- PROTOTIPOS
- HERENCIA
- CLASES
- NOTACIÓN JSON

# Constructores

---

- Para crear varios objetos similares. Por ejemplo: múltiples usuarios
- Utilizamos el **constructor de funciones** y el operador ***new***.

## Función constructora

Es técnicamente una función normal con 2 convenciones:

1. Nombradas con la primera letra mayúscula.
2. Sólo deben ejecutarse con el operador "new".

```
function User(name) {  
    this.name = name;  
    this.isAdmin = false;  
}  
  
let user = new User("Jack");  
  
alert(user.name); // Jack  
alert(user.isAdmin); // false
```



# Métodos en constructores

---

- Al igual que con los objetos literales, usamos funciones para definir métodos dentro de la función constructora

```
function User(name) {  
  this.name = name;  
  
  this.sayHi = function() {  
    console.log( "Hola mi nombre es: " + this.name );  
  };  
}  
  
let john = new User("John");  
john.sayHi(); // Hola mi nombre es: John
```

# El operador instanceof

---

- Todos los objetos pertenecen al mismo tipo de datos: Object. Pero con el operador **instanceof** podemos comprobar a qué tipo de objeto pertenece una variable.
- Para obtener el tipo directamente podemos usar **constructor.name**

```
function Car(make, model, year) {  
  this.make = make;  
  this.model = model;  
  this.year = year;  
}  
var mycar = new Car('Eagle', 'Talon TSi', 1993);  
console.log(mycar instanceof Car); //Devuelve true  
console.log(mycar instanceof Object); //Devuelve true  
console.log(mycar.constructor.name); //Devuelve Car
```

Es menos seguro

# Prototipos

---

- En muchos lenguajes OO:
  - todo objeto pertenece a una clase, por lo que debemos crear un objeto a partir de una clase (en Javascript no).
  - La herencia se indica indicando que una clase hereda de otra.
- En Javascript, los objetos procedentes del mismo tipo de función constructora tienen un mismo **prototipo** con el que enlazan.
- El **prototipo** de un objeto es una serie de métodos y propiedades comunes con los objetos del mismo tipo.
- Podemos modificar el prototipo sobre la marcha y los objetos que se enlazan con él se actualizan inmediatamente ya que el enlace es dinámico.
- Usamos la propiedad `__proto__` para acceder al prototipo de un objeto.
- También podemos acceder con el método `Object.getPrototypeOf(objeto)`
- Objeto. También puedo acceder al prototipo de la función constructora con `Object.prototype`. Por ejemplo: `Car.prototype`.

# Ejemplo de acceso a prototipo

```
console.log(mycar.__proto__); //Devuelve su prototipo (El objeto COMPLETO)
console.log(Object.getPrototypeOf(mycar)); Equivalente
```

## ▼ Object ⓘ

- ▶ **constructor**: *f Car(make, model, year)*
- ▼ **[[Prototype]]**: Object
  - ▶ **constructor**: *f Object()*
  - ▶ **hasOwnProperty**: *f hasOwnProperty()*
  - ▶ **isPrototypeOf**: *f isPrototypeOf()*
  - ▶ **propertyIsEnumerable**: *f propertyIsEnumerable()*
  - ▶ **toLocaleString**: *f toLocaleString()*
  - ▶ **toString**: *f toString()*
  - ▶ **valueOf**: *f valueOf()*
  - ▶ **\_\_defineGetter\_\_**: *f \_\_defineGetter\_\_()*
  - ▶ **\_\_defineSetter\_\_**: *f \_\_defineSetter\_\_()*
  - ▶ **\_\_lookupGetter\_\_**: *f \_\_lookupGetter\_\_()*
  - ▶ **\_\_lookupSetter\_\_**: *f \_\_lookupSetter\_\_()*
  - ▶ **\_\_proto\_\_**: (...)
  - ▶ **get \_\_proto\_\_**: *f \_\_proto\_\_()*
  - ▶ **set \_\_proto\_\_**: *f \_\_proto\_\_()*

`console.log(mycar.valueOf());` //Puedo acceder a los métodos del prototipo (heredados de Object)

# Recorrer objeto y prototipo

---

```
function recorrerPropiedades(o) {  
    var objeto;  
    var result = [];  
  
    for(objeto = o; objeto !== null; objeto=Object.getPrototypeOf(objeto)) {  
        result = result.concat(  
            Object.getOwnPropertyNames(objeto)  
        );  
    }  
    return result;  
}  
  
console.log(recorrerPropiedades(mycar));
```

```
(16) ['make', 'model', 'year', 'constructor', 'constructor', '__defineGetter__', '__defineSetter__', 'hasOwnProperty', '__lookupGetter__', '__lookupSetter__', 'isPrototypeOf', 'propertyIsEnumerable', 'toString', 'valueOf', '__proto__', 'toLocaleString']
```

# Herencia en los prototipos

---

- ¿**Dónde se definen las propiedades y métodos heredados** de Object en el ejemplo anterior? ¿O se heredan todos?
- Solo se heredan los que empiezan con **Object.prototype**, y no los que empiezan sólo con Object. Así el prototipo es básicamente un repositorio (bucket) para almacenar propiedades y métodos que queremos que sean heredados por los objetos más abajo en la cadena del prototipo.
- Por eso **Object.prototype.watch()**, **Object.prototype.valueOf()**, etc., están disponibles para cualquier tipo de objeto que herede de Object.prototype, incluyendo nuevas instancias de objeto creadas desde el constructor.

# Modificar Prototipos

---

- Puedo modificar el prototipo de un objeto una vez definido

```
Car.prototype.acelera = function() { //Se añade de forma dinámica!  
    alert(this.model + ' quiere acelerar!');  
};  
mycar.acelera(); //Tengo nuevo método en Car!
```

- OJO!! Esto no funciona (porque this no apunta al ámbito de la función sino al ámbito global)

```
Car.prototype.fullName=this.make+" "+this.model;  
console.log(mycar.fullName); //undefined undefined
```

# Patrón común de la definición de Objetos

---

```
// Constructor con definición de propiedades

function Test(a, b, c, d) {
  // definición de propiedades
}

// Definición de primer método

Test.prototype.x = function() { ... };

// Definición de segundo método

Test.prototype.y = function() { ... };

// etc.
```



# Ejemplo de herencia CON PROTOTIPOS

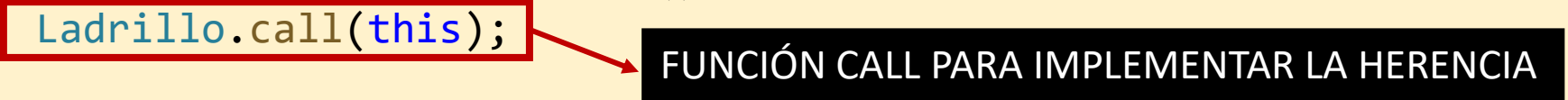
[https://developer.mozilla.org/es/docs/Learn/JavaScript/Objects/Classes\\_in\\_JavaScript](https://developer.mozilla.org/es/docs/Learn/JavaScript/Objects/Classes_in_JavaScript)

- Heredando de un constructor sin parámetros

```
//Objeto Ladrillo
function Ladrillo() {
  this.anchura = 10;
  this.altura = 20;
}

//Ladrillo que hereda anchura y altura y además añade opacidad y color
function LadrilloAzulCristal() {
  Ladrillo.call(this);

  this.opacidad = 0.5;
  this.color = 'blue';
}
```



# Ejemplo de herencia CON PROTOTIPOS

- Heredando de constructor con parámetros. Vamos a heredar de Persona

```
function Persona(nombrePila, apellido, edad, genero, intereses) {  
  this.nombre = {  
    nombrePila,  
    apellido  
  };  
  this.edad = edad;  
  this.genero = genero;  
  this.intereses = intereses;  
};  
  
Persona.prototype.saludo = function() {  
  alert('¡Hola! soy ' + this.nombre.nombrePila);  
};
```

Queremos implementar el objeto Profesor que herede de Persona, pero que incluya también:

- Nueva propiedad: materia (materia que imparte)
- Actualización del método saludo para que sea más formal

Código en: <https://github.com/mdn/learning-area/blob/main/javascript/oojs/introduction/oojs-class-further-exercises.html>

# Ejemplo de herencia - cont

```
function Profesor(nombrePila, apellido, edad, genero, intereses, materia) {  
    Person.call(this, nombrePila, apellido, edad, genero, intereses);
```

```
    this.materia = materia;  
}
```

```
//Que pasa con el método saludo?
```

```
console.log(Persona.prototype.saludo);
```

```
console.log(Profesor.prototype.saludo); //undefined, prototype se hereda vacío
```

```
//Uso la fc Object.create para construir el prototipo a partir del padre
```

```
Profesor.prototype = Object.create(Persona.prototype);
```

```
console.log(Profesor.prototype.saludo); //ahora sí tenemos el método saludo.
```

```
//Además debemos asegurar que el constructor de profesor es el suyo y no el de persona
```

```
Profesor.prototype.constructor = Profesor;
```

Creamos el objeto Profesor y añadimos su nueva propiedad, pero nos falta el saludo

# Ejemplo de herencia - cont

Creamos nuestro método saludo que sobrescribirá al de Persona

```
Profesor.prototype.saludo = function() {  
    var prefijo;  
  
    if (this.genero === 'h' || this.genero === 'H') {  
        prefijo = 'Sr.';  
    } else if (this.genero === 'm' || this.genero === 'M') {  
        prefijo = 'Sra.';  
    } else { prefijo = 'Sx.'; }  
  
    alert('Hola. Mi nombre es ' + prefijo + ' ' + this.nombre.apellido + ', y enseño ' +  
        this.materia + '.');  
};
```

```
var profeVanessa=new Profesor("vanessa","espin",20,"m",["musica", "lectura"], "DWECE");  
profeVanessa.saludo();
```

127.0.0.1:5500 dice

Hola. Mi nombre es Sra. espin, y enseño DWECE.

# Uso de clases

Código en:

<https://github.com/mdn/learning-area/blob/main/javascript/ojs/advanced/es2015-class-inheritance.html>

Las clases aparecen en ECMAScript 2015, son soportadas en todos los navegadores y su sintaxis es más parecida a lenguajes como Java y C++

# Sintaxis de definición de Clases

---

```
class Person {  
    constructor(nombrepila, apellido, edad, genero, intereses) {  
        this.nombre = {nombrepila,apellido};  
        this.edad = edad;  
        this.genero = genero;  
        this.intereses = intereses;  
    }  
  
    saludo() {  
        console.log(`Hola! Soy ${this.nombre.nombrepila}`);  
    };  
  
    despedida() {  
        console.log(`${this.nombre.nombrepila} se va. Ciao!`);  
    };  
}
```

# Ejemplo de Herencia con CLASES

---

```
class Profesor extends Persona {
  constructor(nombrepila, apellido, edad, genero, intereses, materia, curso) {
    super(nombrepila, apellido, edad, genero, intereses);

    // materia y curso son especificas de profesor
    this.materia = materia;
    this.curso = curso;
  }
}

let snape = new Profesor('Severus', 'Snape', 58, 'H', ['Pociones'], 'Artes oscuras', "2 DAW");
snape.saludo(); // Hi! I'm Severus.
snape.despedida(); // Severus has left the building. Bye for now.
snape.edad // 58
snape.materia; // Artes oscuras
```

# Javascript JSON

JSON es un formato de datos basado en texto que sigue la sintaxis de objeto de JavaScript, popularizado por Douglas Crockford. Aunque es muy parecido a la sintaxis de objeto literal de JavaScript, puede ser utilizado independientemente de JavaScript, y muchos entornos de programación poseen la capacidad de leer (convertir; parsear) y generar JSON.

LO ESTUDIAREMOS POR:

<https://developer.mozilla.org/es/docs/Learn/JavaScript/Objects/JSON>





*That's all Folks!*