Índice

[**JavaScript: orientado a objetos, basado en prototipos** 1](#_Toc118904722)

[**Qué es programación orientada a objetos** 2](#_Toc118904723)

[**Qué es un objeto en JavaScript** 3](#_Toc118904724)

[**Prototipo** 3](#_Toc118904725)

[**Objeto** 4](#_Toc118904726)

[**Atributos** 4](#_Toc118904727)

[**Métodos** 4](#_Toc118904728)

[**Atributo \_\_proto\_\_** 4](#_Toc118904729)

[**Clases en JavaScript** 5](#_Toc118904730)

[**Ventajas de la programación orientada a objetos** 7](#_Toc118904731)

[**Qué es la abstracción** 8](#_Toc118904732)

[**Ventajas de la Abstracción** 8](#_Toc118904733)

[**Los 4 pilares de la programación orientada a objetos** 9](#_Toc118904734)

[**Qué es el encapsulamiento** 10](#_Toc118904735)

[**Encapsulamiento en JavaScript** 10](#_Toc118904736)

[**Formas de aplicar encapsulamiento en JavaScript** 10](#_Toc118904737)

[**Modificadores de accesos** 11](#_Toc118904738)

[**Getters y setters en JavaScript** 11](#_Toc118904739)

[**¿Cuál sería la ventaja de usar esto?** 11](#_Toc118904740)

[**Módulos de ECMAScript 6** 12](#_Toc118904741)

[**¿Como Exportar?** 13](#_Toc118904742)

[**¿Como Importar?** 13](#_Toc118904743)

[**¿Como aplicar el módulo en HTML?** 13](#_Toc118904744)

[**Herencia** 14](#_Toc118904745)

[**Qué es el polimorfismo** 14](#_Toc118904746)

[**Tipos** 15](#_Toc118904747)

# **JavaScript: orientado a objetos, basado en prototipos**

**Lenguaje basado en clases**

Se basan en el concepto de **entidades** o **clases**

* **Clases**: Define las propiedades que caracterizan un determinado conjunto de objetos. Una clase es una entidad abstracta.

Ejemplo: La clase empleado define a un conjunto de empleados.

* **Instancias**: Instanciación de una clase, de uno de sus miembros. Una instancia tiene exactamente las mismas propiedades de su clase padre, ni una más ni una menos.

Ejemplo: Victoria es una instancia de la clase empleados.

**Lenguaje basado en prototipos**

No hace las mismas distinciones que los lenguajes basados en clases, simplemente tiene **objetos**.

* Toma el concepto de objeto prototípico. Un objeto se usa como plantilla a partir del cual se obtiene un conjunto inicial de propiedades de un nuevo objeto.
* Cualquier objeto puede especificar sus propias propiedades.
* Cualquier objeto puede ser usado como prototípico de otro objeto.

# **Qué es programación orientada a objetos**

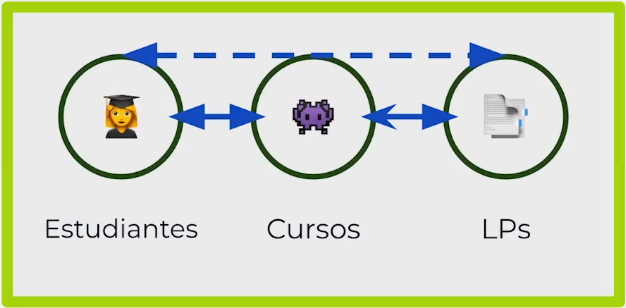
**Paradigmas**

Son formas o caminos que podemos seguir para programar las aplicaciones

Tipos de paradigmas

* Estructural.
* Orientado a objetos.
* Funcional.

Una de los principales objetivos del cual se creó la programación orientada a objetos es orden por ejemplo si se quiere crear un sistema como platzi se necesita:

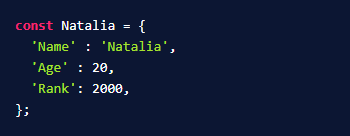
* Objetos.
  + Uno de los problemas que vino a resolver es el **ORDEN**.
  + Cada elemento de nuestro proyecto es un **objeto** y todo lo que esté relacionado con nuestro objeto debe vivir dentro de él.
  + Nos ayuda a **reutilizar** **código**.
* Estudiantes.
* Cursos.
* LPa.

**La programación orientada a objetos** también nos ayuda a reutilizar código a esto se le conoce como clases las son bloques de código donde podemos definir métodos o atributos.

* **Clase**: Representación de **métodos** y **atributos** que representan a una **entidad**.
* **Objeto**: Instancia de una clase.
* **Herencia**: Crear nuevas clases con características similares a partir de otras.
* **Encapsulamiento**: Esconder datos que solo pueden ser accedidos por la clase.
* **Polimorfismo**: Un objeto puede comportarse de muchas formas, ejemplo: En herencia, las clases hijas pueden comportarse como su clase padre y viceversa.

# **Qué es un objeto en JavaScript**

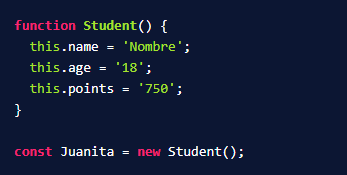
Un **objeto** en JavaScript **es una instancia del prototipo Object** que es creado de forma “nativa” por Js.



## **Prototipo**

**Es una estructura de código a partir de la cual se crean objetos**, ya que guarda los atributos y métodos que luego podrán ser heredados por sus instancias.

Podemos pensarlo como un “molde” de objetos.



## **Objeto**

Son estructuras de datos formadas por **métodos** y **atributos**, los cuales hereda de su prototipo **padre**.

De modo que los objetos son **instancias de ese prototipo**, particularmente cuando dicho prototipo fue creado por el desarrollador (en caso contrario se llaman **objetos literales**).

## **Atributos**

Dentro de los objetos se pueden guardar **atributos** para guardar en ellos la información que se les asocia. Así, toda la información del objeto se guarda en sí mismo.

## **Métodos**

Dentro de los **objetos** también pueden guardarse **métodos**, los cuales son funciones que, entre otras cosas, permiten actualizar la información guardada en los atributos de forma segura.

## **Atributo \_\_proto\_\_**

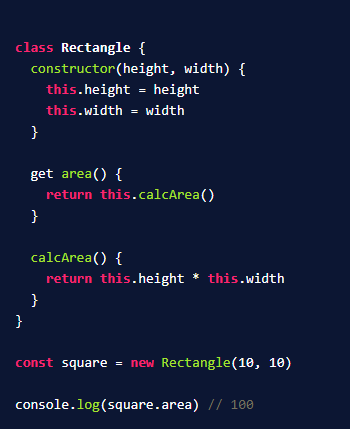
Es el nombre que se le da al atributo donde se guardan los **métodos** que las estructuras de datos tienen por defecto en JavaScript.

El atributo **\_\_proto\_\_** se hereda a partir de los prototipos por defecto de JavaScript para cada estructura de datos en particular, por ejemplo, de los prototipos **Object** y **Array**.

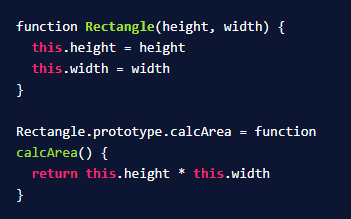
El atributo **\_\_proto\_\_** también se hereda a los objetos, ya que éstos son a la vez instancias de algún prototipo creado por el desarrollador y del prototipo **Object**.

# **Clases en JavaScript**

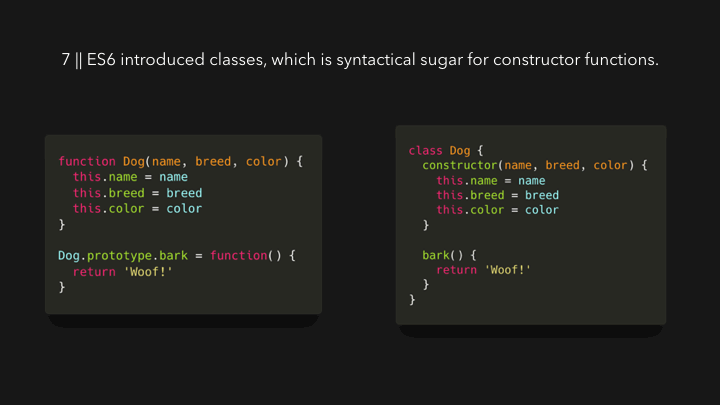
En ES6, ahora podemos usar la palabra clave **Class**, así como los métodos mencionados anteriormente para manipular el **prototype**. Las clases de JavaScript son atractivas para los desarrolladores con experiencia en programación orientada a objetos, pero esencialmente hace lo mismo que el prototype. En otras palabras, es un azúcar sintáctico.



Esto es básicamente lo mismo que:



Más ejemplos



Los métodos **Getter** y **Setter** en las clases vinculan una propiedad Object a una función que será llamada cuando se busque esa propiedad. Esto es solo azúcar sintáctico para ayudar a que sea más fácil buscar o establecer propiedades.

# **Ventajas de la programación orientada a objetos**

* **Reusabilidad**: Cuando hemos diseñado adecuadamente las clases, se pueden usar en distintas partes del programa y en numerosos proyectos.
* **Mantenibilidad**: Debido a las sencilleces para abstraer el problema, los programas orientados a objetos son más sencillos de leer y comprender, pues nos permiten ocultar detalles de implementación dejando visibles sólo aquellos detalles más relevantes.
* **Modificabilidad**: La facilidad de añadir, suprimir o modificar nuevos objetos nos permite hacer modificaciones de una forma muy sencilla.
* **Fiabilidad**: Al dividir el problema en partes más pequeñas podemos probarlas de manera independiente y aislar mucho más fácilmente los posibles errores que puedan surgir.

La programación orientada a objetos presenta también algunas desventajas como pueden ser:

* Cambio en la forma de pensar de la programación tradicional a la orientada a objetos.
* La ejecución de programas orientados a objetos es más lenta.
* La necesidad de utilizar bibliotecas de clases obliga a su aprendizaje y entrenamiento.

# **Qué es la abstracción**

**Es una manera de reducir la complejidad y permitir un diseño e implementación más eficientes en sistemas de software complejos**.

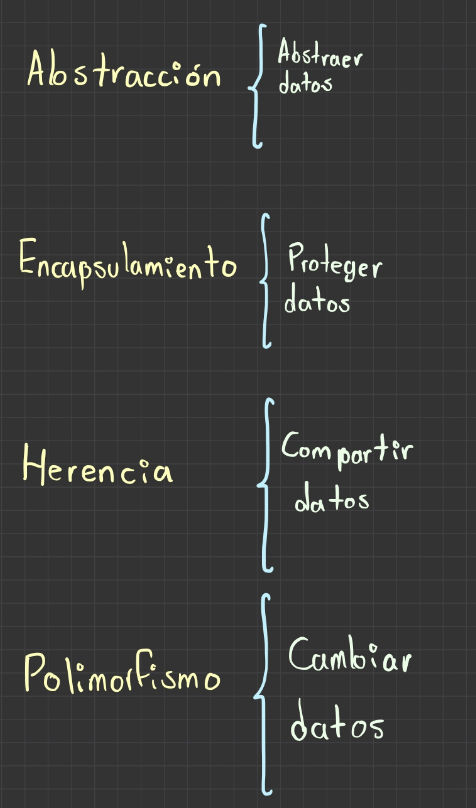
Básicamente, la abstracción en programación consiste en “**ocultar**” (abstraer) la lógica de algo que no es necesario mostrar, como, por ejemplo, algunos cálculos complejos o accesos a la base de datos, de la vista del programador.

* La abstracción consiste en abstraer los datos de un objeto para crear su molde, prototipo o clase. Para posteriormente crear instancias de dichos prototipos y nos ahorremos mucho tiempo
* Cada elemento de nuestro proyecto debe poderse abstraer para no repetir el código

## **Ventajas de la Abstracción**

* Evita duplicar código y aumenta la reusabilidad.
* Se puede cambiar la implementación interna de la clase de forma independiente sin afectar al usuario.
* Ayuda a aumentar la seguridad de la aplicación o programa, ya que solo los detalles importantes son proporcionados al usuario.

## **Los 4 pilares de la programación orientada a objetos**



# **Qué es el encapsulamiento**

Es guardar, proteger, guardar o limitar el acceso de ciertos atributos y/o propiedades en nuestros prototipos y objetos.

Cuando hablamos de **encapsulamiento** hablamos de:

* **Esconder** métodos y atributos.
* **No permitir** la alteración de métodos y atributos.

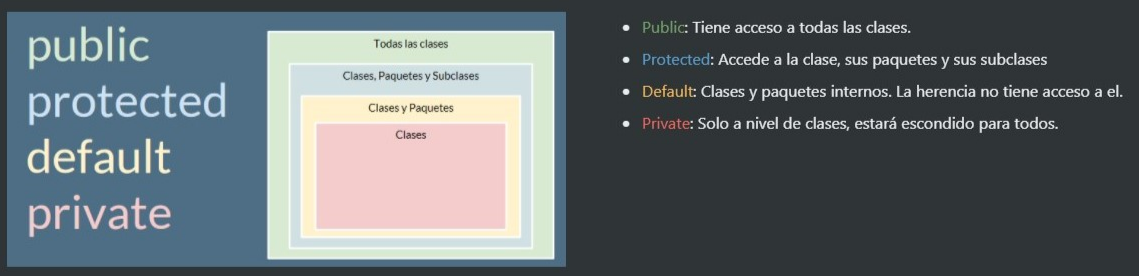
## **Encapsulamiento en JavaScript**

* No permitir la alteración de métodos y atributos.

## **Formas de aplicar encapsulamiento en JavaScript**

* Getters y Setters.
* Namespaces.
* Object.defineProperties.
* Módulo de ES6.

## **Modificadores de accesos**

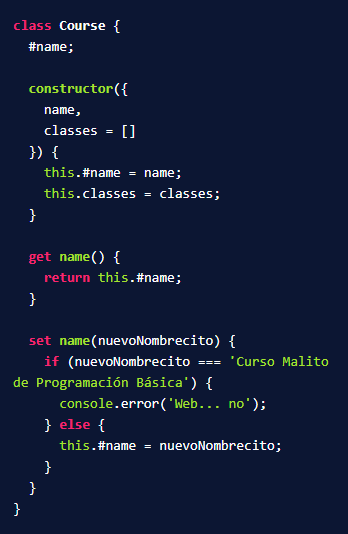


# **Getters y setters en JavaScript**

En ES2020 se introdujo la sintaxis campos privados en las clases. Se hace uso de un numeral **#** como prefijo del nombre de la variable.

## **¿Cuál sería la ventaja de usar esto?**

No existe la posibilidad de que alguien modifique la variable privada desde la instancia a menos de que use el **setter**.



# **Módulos de ECMAScript 6**

El uso de los módulos en JavaScript depende de las declaraciones **Export** e **Import**.

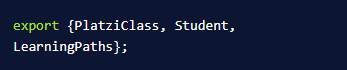
Para modular, solo basta que el archivo tenga la extensión **.js**, pero el motor V8 que es el que corre en los navegadores, recomienda usar la extensión **.mjs**

* Esto es útil porque queda claro cuál archivo es modulo y cual no.
* Asegura que los archivos de modulo sean tratados como tal por la extensión Babel o Node.js

## **¿Como Exportar?**

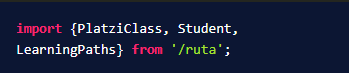
Hay dos formas de exportar:

* Colocar en frente de cada elemento la palabra export.
* Exportar en una sola declaración al final del archivo modulo las características que se quieren exportar, ejemplo:

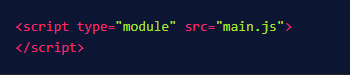


## **¿Como Importar?**

La importación obviamente se hace en el script que queremos usar dichos elementos.

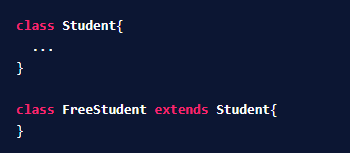


## **¿Como aplicar el módulo en HTML?**



# **Herencia**

* La herencia nos permite crear “moldes padre” para crear otros prototipos a partir de la **clase padre**.
* La herencia nos ayuda cuando queremos clasificar un grupo de prototipos los cuales queremos que tengan comportamientos bastante similares, pero cada uno con cierta particularidad.
* Al referirnos a herencia nos referimos a que obtenemos o heredamos los **atributos** y **métodos** de la **clase** **padre**.
  + Para poder heredar de una clase se utiliza la palabra reservada **extends**.



**Herencia** es un pilar de la programación orientada a objetos, este concepto nos ayuda a crear una clase la cual nos servirá para poder crear otras clases a partir de ésta.

# **Qué es el polimorfismo**

Permite a nuestras subclases **cambiar** o **anular** los comportamientos de los **métodos** y **atributos** del prototipo **hijo**, de la clase **padre**. Aunque herede las propiedades, **el polimorfismo permite cambiar su comportamiento**.

## **Tipos**

* **Polimorfismo de Sobrecarga**: Ocurre cuando existen métodos con el mismo nombre y funcionalidad similar en clases totalmente independientes entre ellas.
* **Polimorfismo Paramétrico**: El polimorfismo paramétrico es la capacidad para definir varias funciones utilizando el mismo nombre, pero usando parámetros diferentes (nombre y/o tipo).
* **Polimorfismo de Inclusión** (JS): La habilidad para redefinir por completo el método de una superclase en una subclase.