# Patrón Módulo

Es el patron de diseño mas comun que existe en Javascript ya que es compatible con casi todas las versiones. Permite encapsular y proteger el codigo.

Consiste en colocar nuestro codigo js dentro de una funcion anonima autoinvocable. Utilizar ‘use strict’

y lo que pongamos en el return de esta funcion va a ser publico.

# [javascript-minifier.com](https://javascript-minifier.com/)

Para ofuscar y minimizar nuestro codigo. Tambien existen extenciones en VsCode. Nos permite crear un archivo cuya nomenclatura deberia ser por ejemplo: juego-min.js y esto significaria que ya está listo para ser codigo productivo

# [Underscore.js (underscorejs.org)](https://underscorejs.org/)

Libreria js con varias funcionalidades:

# Clases

Viene a reemplazar el prototype y la manera de trabajar con funciones simulando las clases

## Sets, Gets y Propiedades, sets y metodos estaticos

*class* Persona {

    static \_conteo = 0;

    static *get* conteo() {

        return Persona.\_conteo + ' instancias';

    }

    static mensaje() {

        console.log( this.nombre ); // undefined

        console.log('Hola a todos, soy un método stático');

    }

    nombre = '';

    codigo = '';

    frase  = '';

    comida = '';

*constructor*( *nombre* = 'Sin nombre', *codigo* = 'Sin código', *frase* = 'Sin frase') {

        this.nombre = nombre;

        this.codigo = codigo;

        this.frase  = frase;

        Persona.\_conteo++;

    }

*set* setComidaFavorita( *comida* ) {

        this.comida = comida.toUpperCase();

    }

*get* getComidaFavorita() {

        return `La comida favorita de ${ this.nombre } es ${ this.comida }`;

    }

    quienSoy() {

        console.log(`Soy ${ this.nombre } y mi identidad es ${ this.codigo }`);

    }

    miFrase() {

        this.quienSoy();

        console.log(`${ this.codigo} dice: ${ this.frase }`);

    }

}

*const* spiderman = new Persona( 'Peter Parker', 'Spiderman', 'Soy tu amigable vecino Spiderman' );

*const* ironman   = new Persona( 'Tony Stark', 'Ironman', 'Yo soy Ironman' );

// console.log( ironman );

spiderman.miFrase();

// ironman.miFrase();

spiderman.setComidaFavorita = 'El pie de cereza de la tía May';

// spiderman.comida = 'Duende Verde';

// spiderman.nemesis = 'Duende Verde';// es es aceptado por JS y crea la propiedad aunque no esté en su definicion

// console.log( spiderman.getComidaFavorita );

// console.log( spiderman );

// Persona.\_conteo = 2;

console.log('Conteo stático', Persona.\_conteo );

console.log( Persona.conteo );

Persona.mensaje();

Persona.propiedadExterna = 'Hola Mundo';//esto es permitido pero no recomendado

console.log( Persona.propiedadExterna );

console.log( Persona );

Mas en ES6 y en el codigo fuente de JSModerno. Herencia, subclases

# Propiedades Privadas

Al momento de realizar el curso todavia no está globalmente aceptado por eso hay que revisar este link

[JavaScript classes: Private class fields | Can I use... Support tables for HTML5, CSS3, etc](https://caniuse.com/mdn-javascript_classes_private_class_fields)

Todavia no estan aceptados los metodos privados

*class* Rectangulo {

    #area = 0;

*constructor*(*base* = 0, *altura* = 0) {

        this.base   = base;

        this.altura = altura;

        this.#area = base \* altura;

    }

    calgularArea() {

        console.log( this.#area \* 2 );

    }

}

*const* rectangulo = new Rectangulo(10, 15);

// rectangulo.#area  = 100;

rectangulo.calgularArea();

console.log(rectangulo);

# Singleton

Es una instancia unica de una clase de manera global, util para compartir informacion a lo largo de la app, con BD, etc…

*class* Singleton {

    static instancia; // undefined

    nombre = '';

*constructor*( *nombre* = '' ) {

        if ( !!Singleton.instancia ) {

            return Singleton.instancia;

        }

        Singleton.instancia = this;

        this.nombre = nombre;

    }

}

*const* instancia1 = new Singleton('Ironman');

*const* instancia2 = new Singleton('Spiderman');

*const* instancia3 = new Singleton('BlackPanther');

console.log(`Nombre en la instancia1 es: ${ instancia1.nombre }`);

console.log(`Nombre en la instancia2 es: ${ instancia2.nombre }`);

console.log(`Nombre en la instancia3 es: ${ instancia3.nombre }`);

# Multiples Constructores

A diferencia de otros lenguajes JS no lo soporta pero se puede simular creando unmetodo estatico, es util siq queremos crear instancias de diferentes formas:

*class* Persona {

    static porObjeto({ *nombre*, *apellido*, *pais* }) {

        return new Persona( nombre, apellido, pais );

    }

*constructor*( *nombre*, *apellido*, *pais* ) {

        this.nombre   = nombre;

        this.apellido = apellido;

        this.pais     = pais;

    }

    getInfo() {

        console.log(`info: ${ this.nombre }, ${ this.apellido }, ${ this.pais }`);

    }

}

*const* nombre1   = 'Melissa',

      apellido1 = 'Flores',

      pais1     = 'Honduras';

*const* fher = {

    nombre:   'Fernando',

    apellido: 'Herrera',

    pais:     'Costa Rica'

}

*const* persona1 = new Persona( nombre1, apellido1, pais1 );

*const* persona2 = Persona.porObjeto( fher );

persona1.getInfo();

persona2.getInfo();

# Bundlers o empaquetador

Son empaquetadores de modulos. Nos ayudan a desarrollar una aplicación de JavaScript moderna, es decir, con módulos, separando archivos, que cada archivo tenga una responsabilidad única, que nosotros podamos importar diferentes tipos de tecnologías que podamos usar, sass o diferentes tipos de estilo de CSS y todo combinarlo, ejecutarlo, minimizar y ofuscar el codigo y crear un resultado final que es nuestra aplicación, entre comillas, lista para producción.

Genera el build de producción: tenemos archivos JavaScript, tenemos archivos de Handlebars, sass, png, ect. Se toman todos estos archivos, se pasan por el Bundler y genera el build de producción.

También hay otros pasos extras que podemos hacer, como por ejemplo pasarlo por Babbel, el cual nos permite a escribir código actual de JavaScript y transformarlo a código de cualquier versión que nosotros queramos de JavaScript. Tranformando un estandar a otro

## Webpack

Webpack es un bundler muy popular que se utilizó por muchos años, ademas de todo lo nombrado anteriormente tambien tiene pluggins para que tengamos servidores en tiempo real, para que cuando se haga un cambio, mi aplicación automáticamente se recargue y se muestren los cambios en pantalla de nuevo. Gestiona dependencias, Monta servidores de desarrollo y pruebas. Uno de los principales problemas actuales de WP es su velocidad impide que podamos generar un código o desarrollar código más rápidamente. Su configuracion inicial no es simple y puede ser complicado detectar problemas cuando un paquete falla.

Node

## Vite

Es una herramienta Similar a WP pero con mas velocidad. Ademas ya viene con un montón de código pre generado para permitir rápidamente no tener que hacer toda la configuración manual que tendríamos que hacer con Web Pack

## TurboPack

Es el sucesor de Webpack aunque todavia está en pruebas (escrito en RUST)