



Clase 03. ANGULAR

Typescript

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





OBJETIVOS DE LA CLASE

- Conocer y aplicar Typescript.
- Definir variables y sistema de módulos.
- Realizar codificaciones en un ejemplo de Typescript que utilice clases, tipado e interfaces.

MAPA DE CONCEPTOS

MAPA DE CONCEPTOS CLASE 3

¡Para
recordar!



CRONOGRAMA DEL CURSO

Clase 2



Componentes y Elementos de un proyecto Angular



Flujo de desarrollo Angular



Componentes



Ejemplo en vivo



Layouts

Clase 3



Typescript



Interfaces



Funciones y tipado genérico



Ejemplo en vivo

Clase 4



Componentes y Elementos de un proyecto Angular



Componente y directivas estructurales



Componente, directiva ngClass y Pipe Angular



Ejemplo en vivo



Lista de alumnos

ANTES DE EMPEZAR...
DEFINAMOS NODE.JS



NODEJS

¿QUÉ ES NODEJS?

Es considerada una tecnología de servidor que hace aportes a un proyecto javascript aunque sea con tecnología **Front-End**.

Repasemos, ¿por qué NodeJS?



¿POR QUÉ NODEJS?

Uno de los requisitos que tenemos a la hora de configurar nuestras herramientas para el trabajo con Angular es la instalación del **Runtime de NodeJS** 🤔 Esto no significa que usaremos NodeJs, pero sí utilizaremos una herramienta que viene con su instalación: **npm**.



¿POR QUÉ NODEJS?

ANGULAR-CLI utiliza muchas configuraciones que dependen internamente de **node** y **npm** que se emplean en el entorno de desarrollo y no se trasladan al proyecto compilado 🚀.

Un proyecto ANGULAR finalizado **no se encuentra limitado** a la plataforma de node para funcionar, sino que puede hacerlo con cualquier servidor web que soporte **html**, **css** y **javascript**.

PROYECTOS CON NODE



Configuración



Codificación



Ejecución entorno de desarrollo



Pruebas / Test



Preparación de
publicación



Creación del proyecto



CREACIÓN DEL PROYECTO

Consiste en la creación de un archivo `package.json` que es un json y que se puede generar manualmente, pero npm nos ofrece la posibilidad hacerlo mediante los siguientes comandos:



```
npm init
```

Nos va preguntando por valores que se irán cargando en el archivo `package.json`



```
npm init --yes
```

Responde que **sí** a todas las preguntas que nos va haciendo la generación del `packaga.json`



CONFIGURACIÓN

Ejemplo
en vivo



```
package.json x
package.json > [ ] keywords
1 {
2   "name": "jsproject",
3   "version": "0.0.1",
4   "description": "Este es mi primer proyecto javascript creado con npm ",
5   "main": "index.js",
6   "scripts": {
7     "start": "node src/index"
8   },
9   "keywords": [
10    "javascript",
11    "node",
12    "commonjs"
13  ],
14   "author": "@Profesor CoderHouse",
15   "license": "MIT"
16 }
17
```

Es el manifiesto de nuestro proyecto. Además, administra los paquetes y maneja la publicación del proyecto.

CODER HOUSE



CODIFICACIÓN

Ejemplo
en vivo



Con saber programar en javascript, solo vamos a necesitar crear
un archivo con extensión **.js**

```
console.log("--- Aplicacion de consola ---");  
console.log('Nuestra primera aplicacion puro javascript');  
console.log("*****");
```



EJECUCIÓN EN DESARROLLO

Mientras estamos en tiempo de desarrollo necesitamos ir probando nuestro código

Comando de Node



\$ node index.js

Configuracion del package.json



\$ npm start



PRUEBAS / TEST

Las **pruebas** no forman parte de las funcionalidades nativas del entorno de node. Mientras que las alternativas de **test** son:

- Generación nativa de pruebas (poco utilizada).
- Mediante la instalación de librerías externas.



PREPARACIÓN DE PUBLICACIÓN

Javascript es interpretado según el entorno (browser o runtime), por esto no requiere de un proceso de compilación. Sin embargo, es muy común atravesar un proceso de empaquetado o conversión a código estándar **EcmaScript** compatible.

Este proceso -en general- se lleva adelante mediante librerías especializadas debidamente configuradas para esta tarea.

TYPESCRIPT

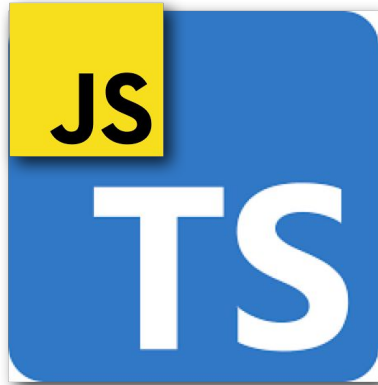
¿QUÉ ES TYPESCRIPT?



TYPESCRIPT

TypeScript es un lenguaje de programación de código abierto creado por Microsoft en el año 2012 que implementa mecanismos de programación **orientados a objetos**.

TYPESCRIPT



Se trata de un **superset de JavaScript** que extiende su sintaxis.

Al compilar, se genera código JavaScript ya que el navegador no puede interpretar TypeScript. Este proceso se conoce como ***transpilar*** que significa generar un código en un lenguaje específico a partir de otro.

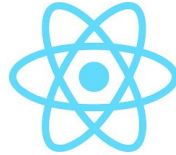


VENTAJAS DE TYPESCRIPT

- Extiende a Superset de Javascript.
- Ofrece tipado **estricto** y **flexible**.
- Mejora la legibilidad del código.
- Permite usar nuevas características.

USOS DE TYPESCRIPT

Es muy común encontrar typescript del lado del **FrontEnd**



Pero también en el **backend FrontEnd**



INSTALAR Y CONFIGURAR

Vamos a necesitar tener instalado nodejs y un editor de código:



El editor Visual Studio Code viene configurado para aprovechar al máximo TypeScript.

PROBANDO TYPESCRIPT

Observemos un ejemplo de flujo de trabajo con typescript



```
npm install -g typescript
```

Podemos crear un folder/carpeta y dentro un archivo .ts



```
Touch index.ts
```

Dentro del index.ts se puede escribir un código typescript válido



```
npx tsc index.ts --outFile dist/index.js
```



```
tsc index.ts --outFile dist/index.js
```

COMPILACIÓN



```
npx tsc index.ts
```

El proceso de compilación parte de un archivo **index.ts** al cual se le ejecuta el comando de compilación `tsc` y obtenemos un archivo **index.js**. Esto es necesario para que el código pueda ser interpretado mediante un browser o por el runtime de node.



```
npx tsc --watch index.ts
```

Sirve para observar los cambios que se producen en el archivo `index.ts`



EJEMPLO EN VIVO

Creamos un proyecto con typescript



ORGANIZACIÓN DE ARCHIVOS

```
ts-intro — adrgon@MacBook-Pro-  
[(base) → ~ cd dev  
[(base) → dev mkdir proyectos-typescript  
[(base) → dev cd proyectos-typescript  
[(base) → proyectos-typescript mkdir ts-intro  
[(base) → proyectos-typescript cd ts-intro
```

Organizamos archivos y carpetas



GENERACIÓN PACKAGE.JSON

```
ts-intro — adrgon@MacBook-Pro-de-Adrian — ..ript/ts-intro — -zsh — 105x34
```

```
(base) → ts-intro npm init
```

This utility will walk you through creating a package.json file.

It only covers the most common items, and tries to guess sensible defaults.

See `npm help init` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.

```
package name: (ts-intro)
```

```
version: (1.0.0)
```

```
description: Este es mi primer proyecto en typescript
```

```
entry point: (index.js) index.ts
```

```
test command:
```

```
git repository:
```

```
keywords: typescript, ts
```

```
author: Adrian L Gonzalez
```

```
license: (ISC) MIT
```



ABRIMOS EL EDITOR

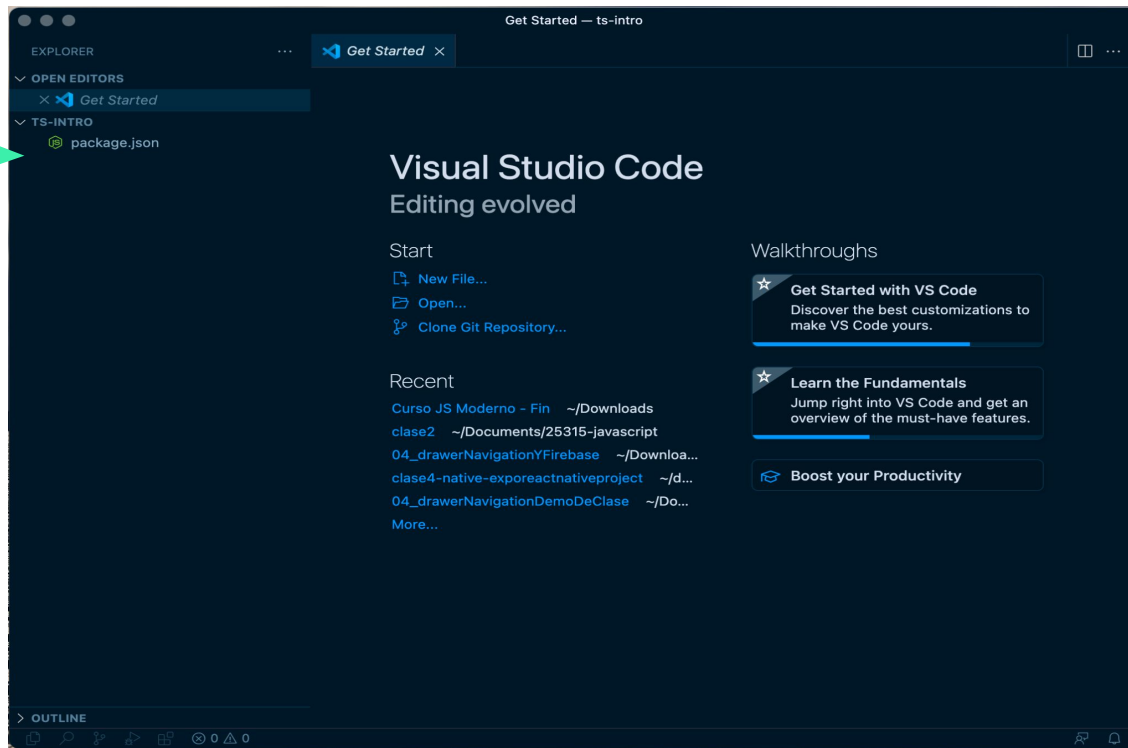
```
ts-intro -  
[ (base) → ts-intro code .  
(base) → ts-intro ]
```

Iniciamos Visual Studio Code. Es decir, con el comando **code** .
abrimos un editor en la carpeta donde estamos posicionados.



VISUAL STUDIO CODE

Archivo
package.json
generado con
npm init





ESCRITURA DE CÓDIGO

Archivo de
typescript



The screenshot shows the VS Code interface. The Explorer panel on the left shows a project structure with 'OPEN EDITORS' containing 'Get Started' and 'index.ts', and 'TS-INTRO' containing 'index.ts' and 'package.json'. The main editor area shows 'index.ts' with the following content:

```
1  /*
2  ██████████ ===== Código de TypeScript =====
3  */
4
```

The bottom panel shows the 'TERMINAL' with the following output:

```
(base) → ts-intro npm install typescript -g
added 1 package, and audited 2 packages in 1s

found 0 vulnerabilities
(base) → ts-intro
```

Código
fuente



Instalación
de paquete





ESCRITURA DE CÓDIGO

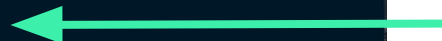
Archivo
compilado



The screenshot shows the VS Code interface with the following details:

- EXPLORER:** Shows a project named 'ts-intro' with files 'index.js' (JavaScript) and 'index.ts' (TypeScript). A green arrow points to 'index.js' with the label 'Archivo compilado'.
- EDITOR:** The 'index.ts' file is open, showing a TypeScript comment:

```
1 /*  
2 ===== Código de TypeScript =====  
3 */  
4
```
- TERMINAL:** Shows the command `ts-intro npx tsc index.ts` being executed, with the output `ts-intro` and a cursor.
- STATUS BAR:** At the bottom, it indicates the file is a TypeScript file using Prettier formatting.



Compilación

CARACTERÍSTICAS DEL LENGUAJE

```

class Persona {
  nombre: string;
  apellido: string;
  edad: number;

  constructor(n: string, a: string, e: number) {
    this.nombre = n;
    this.apellido = a;
    this.edad = e;
  }
}

CrearPersona() {
  //persona: Instancia de tipo Persona
  let persona: Persona = new Persona("Juan", "Perez", 39);

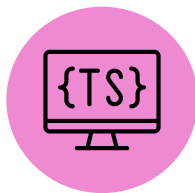
  console.log("La persona es:", persona);
}

```

Clases
Tipos de datos
Propiedades
Constructor
Métodos
Instancia

JAVASCRIPT
 +
 PROGRAMACIÓN
 ORIENTADA A OBJETOS

TYPESCRIPT



IMPORTANTE EN TYPESCRIPT

- Tipos básicos
- Objetos, Arrays, Interfaces
- Clases
- Tipos genéricos
- Decoradores

TYPADO

Typescript tiene la capacidad de trabajar con tipado de datos de dos formas:

- 1 Explícito: define una sintaxis para la creación de variables con tipo de dato
nombreVariable = Tipo De Dato
- 2 Inferido: puede deducir en función del valor
nombreVariable = Valor

TIPOS PRIMITIVOS

Number

Boolean

String

Array

Tuple

Enum

Any

Void

Null

Undefined

Never

Object

Referencia: [TypeScript](#)

TIPOS BÁSICOS

Ejemplo
en vivo



```
let nombre: string;  
let edad: number | string;  
let vive: boolean;  
  
hp = 'FULL';  
  
console.log(nombre, hp);
```


ARRAYS

Ejemplo
en vivo



Al igual que en Javascript, TypeScript permite definir Arrays para un conjunto de valores y se utilizan las notaciones de:

- **[]**

```
let habilidades: string[] = ['Bash', 'Counter', 'Healing'];
```

- **Array<tipo>**

```
let pictures: Array<string>;  
Pictures = ['Sunset', 'Vacation', 'Landscape']
```

INTERFACES

Ejemplo
en vivo



Las interfaces en TypeScript constituyen una forma poderosa de definir “contratos” para el código que vaya a implementarlas.

```
interface Personaje {  
    nombre: string;  
    hp: number;  
    habilidades: string[];  
    puebloNatal?: string;  
}
```



OBJETOS

```
const personaje: Personaje = {  
  nombre: 'Strider',  
  hp: 100,  
  habilidades: []  
}
```

object vs Object

Object es un tipo de dato no primitivo.

👁️ **Cuidado** al declarar a una variable con el tipo **object** porque no es lo mismo que crear un Object nativo de JS.



OBJETOS

¡Recuerda! 💡

Si tenemos un objeto declarado con el `object` de TS no podremos acceder a sus atributos mientras que si lo hacemos regularmente como en vanilla JS **sí podremos**.

CLASE

Ejemplo
en vivo



```
class Heroe extends PersonaNormal {  
    public alterEgo: string  
    constructor(  
        public edad: number,  
        public nombreReal: string  
    ) {  
        super( nombreReal, 'New York, USA' );  
    }  
}
```

Las clases y la POO, se pueden conectar las diferentes entidades y relacionarlas.

Es decir, una clase es la **abstracción de un conjunto de objetos.**



TIPOS GENÉRICOS

Es una utilidad de TypeScript muy relacionada con su sistema de tipado y no existe en Javascript.

Los genéricos son **plantillas de código** que pueden definir y reutilizar en todo el código base. Proporcionan una manera de indicar a las funciones, clases o interfaces qué tipo quiere usar al llamarlas.

Referencia: Definición de genéricos en TypeScript | [docs.microsoft.com](https://docs.microsoft.com/es-es/typescript/generics)

TIPOS GENÉRICOS

Ejemplo
en vivo



```
function queTipoSoy<T>(argumento: T) {  
    return argumento;  
}  
  
let soyString = queTipoSoy('Hola Mundo');  
let soyNumero = queTipoSoy( 100 );  
let soyArreglo = queTipoSoy( [1,2,3,4,5,6,7,8,9,10] );  
  
let soyExplicito = queTipoSoy<number>( 100 );
```

DECORADORES

¿QUÉ SON? 🤔

Son una característica exclusiva de typescript, por lo que cuando son compilados a es5, se crean funciones que expanden las clases de forma diferente.

El **objetivo** de los decoradores es **cambiar las clases cuando son definidas.**

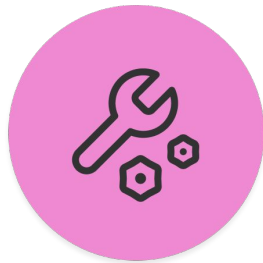
Para conocer más ejemplos de decoradores, ingresa [aquí](#).

DECORADORES

Ejemplo
en vivo



```
function classDecorator<T extends { new (...args: any[]): {} }>(constructor: T) {  
    return class extends constructor {  
        newProperty = "new property";  
        hello = "override";  
    };  
}  
  
@classDecorator  
class MiSuperClase {  
    public miPropiedad: string = 'ABC123';  
    imprimir() { console.log('Hola Mundo') }  
}  
  
const miClase = new MiSuperClase();
```



APLICANDO TYPESCRIPT

En este desafío se espera que puedan codificar un ejemplo de Typescript que utilice clases, tipado e interfaces.

Tiempo: 10 minutos



1 - Transcribir en TypeScript

```
var nombre;  
nombre = "Miguelo";  
var edad;  
edad = 30;  
var PERSONAJE = {  
  nombre: nombre,  
  edad: edad  
};
```



- Usar let y const
- Usar tipado estricto

2 - Crear una Interface en TypeScript

```
var spiderman = {  
  nombre: "Peter parker",  
  poderes: ["trepar", "fuerza", "agilidad",  
    "telas de araña"]  
};
```



BREAK

¡5/10 MINUTOS Y VOLVEMOS!

VARIABLES



VARIABLES

Una variable es un **espacio reservado** de memoria que utilizaremos para trabajar con datos en un programa, ya sea para guardar datos como para leer esos datos guardados 📁.

VARIABLES



IMPORTANTE: todas las variables en typescript deben tener un **tipo de dato**, ya sea primitivo (number, string...) o un dato abstracto (Object, interface creada...)

```
let variable = nombre_variable:tipo = 'valor';
```

VARIABLES

Las variables deben definirse con los siguientes límites:

- No pueden tener **espacios**.
- No empezar con un **número**.
- No puede ser una **palabra reservada** (if, for, while...).

```
let variable = nombre variable:tipo = 'valor';  
let variable = 1variable:tipo = 'valor';  
let variable = if:tipo = 'valor';
```


VARIABLES

Typescript define unos tipos de datos primitivos para la declaración de las variables:

```
let booleano = variable:boolean = true | false;  
let numero = variable:number = 1;  
let string = variable:string = 'texto';  
let variable = variable:any = 'todos los tipos';  
let numArray = variable:number[] = [1,2,3];  
let varios = variable:number|string = 1 o 'texto';
```

CLASES E INTERFACES



CLASES

Podemos utilizar plantillas para la creación de objetos de datos según un modelo predefinido. Se utilizan para representar entidades o conceptos.

Define un conjunto de variables y métodos para operar con dichos datos 🙌.

Es **conveniente** declarar un constructor para realizar una correcta inicialización de las variables.



```
class Persona {  
    nombre: string;  
    edad: number;  
    constructor(nombre: string, edad: number) {  
        this.nombre = nombre;  
        this.edad = edad;  
    }  
    imprimir() { console.log(`Nombre:${this.nombre}`); }  
    let jose: Persona = new Persona('Jose', 30);  
    jose.imprimir();  
}
```

INTERFACES


¡Para
recordar!



Declara una serie de propiedades que deben ser implementados por una o más clases. Normalmente se utilizan para definir “**tipos**” que no tenemos disponibles por defecto con los primitivos de angular 😊.

INTERFACES

¿CÓMO SE DECLARAN?

Las interfaces en TypeScript se declaran de manera bastante similar a la de las clases, indicando la lista de propiedades y métodos que contendrán  Solo hay un detalle fundamental: **las propiedades no pueden tener valores y los métodos no pueden tener código para su implementación.**

INTERFACES

COMO TIPO

Ejemplo
en vivo



```
//interfaz como tipo  
  
interface persona {  
    nombre: string;  
    edad: number;  
    nacimiento: Date;  
    aficiones: string[]; //array de aficiones  
    casado?: boolean; //? Indica que no es obligatorio  
}  
  
let jose: persona; //variable con el nuevo tipo
```

INTERFACES

COMO IMPLEMENTACIÓN

Ejemplo
en vivo



```
interface sumergibleInterface {  
    tiempoMaxBajoElAgua: number;  
    profundidadMaxima: number;  
  
    repelerAgua(): void;  
}  
  
class relojSumergible implements sumergibleInterface {  
    tiempoMaxBajoElAgua = 1;  
    profundidadMaxima = 10;  
    repelerAgua() { console.log('El agua me resbala'); }  
}
```



Declaramos la clase **relojSumergible** que implementa la interfaz **sumergibleInterface**, lo que nos permite acceder a sus propiedades y métodos.

SISTEMA DE MÓDULOS: IMPORT Y EXPORT

MÓDULOS

¿QUÉ SON?

A partir de **ECMAScript 2015**, JavaScript tiene un concepto de **módulos**.

TypeScript lo comparte y se puede definir de la siguiente manera:

Un módulo puede contener clases, variables o funciones. Sólo son visibles en el ámbito del módulo a menos que usemos los siguientes términos:

IMPORT

Permite al módulo utilizar los componentes del módulo importado.

EXPORT

Permite que el contenido del módulo sea visible fuera del ámbito del mismo.

MÓDULOS

EXPORTACIÓN E IMPORTACIÓN

Ejemplo
en vivo



Las declaraciones (variable, const, función, clase, etc.) se pueden exportar para ser importadas en otro módulo.

Tenemos dos tipos de **exportación**:
nombrada y predeterminada.

Las **importaciones** nombradas permiten especificar el elemento a importar.

```
// persona.ts
export function saludo(name: string){
  console.log(`Buenos días ${name}!`);
}
export const nombreConstante: string = 'Jorge';
export const numerico: number = 0;
```

```
import {saludo, nombreConstante} from "./persona.ts";
saludo(nombreConstante); // Hola Jorge!
```

MÓDULOS

EXPORTACIÓN E IMPORTACIÓN

Ejemplo
en vivo



La exportación predeterminada indica qué partes se importan por defecto.

```
// persona.ts  
const edadDefault = 35;  
export default edadDefault;
```

```
import edadDefault from "./persona";  
console.log(edadDefault);
```

MÓDULOS

EXPORTACIÓN E IMPORTACIÓN

Ejemplo
en vivo



Otra funcionalidad en Typescript es **reexportar** declaraciones:

```
//operador.ts
interface Operator {
  eval(x: number, y: number): number;
}
export default Operator;
```

```
//suma.ts
import Operator from "./Operator";
export class Add implements Operator {
  eval(x: number, y: number): number {
    return x + y;
  }
}
```

MÓDULOS

EXPORTACIÓN E IMPORTACIÓN

Ejemplo
en vivo



```
//persona.ts
export interface persona {
    nombre: string;
    edad: number;
}

//tratamientoPersona.ts
import { persona } from './persona.ts';
let alba: persona = {nombre: 'Alba', edad: 35};
```



CLASES E INTERFACES

Crear interfaces y datos que las utilicen para mostrarlos en un componente

Tiempo: 10 minutos



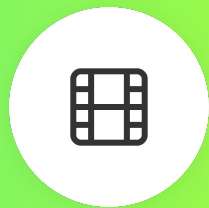
Haciendo uso de lo visto hasta el momento generar **2 archivos .ts** (**persona**, **datos_contacto**) para realizar el siguiente comportamiento:

- Persona.ts será una interface que contendrá la definición de datos personales (nombre, apellidos, edad y dirección).
- Datos_contacto importará la interface y la cargará con datos que se mostrarán por la consola.

Tiempo: 10 minutos

¿PREGUNTAS?





***¿QUIERES SABER MÁS? TE DEJAMOS
MATERIAL AMPLIADO DE LA CLASE***



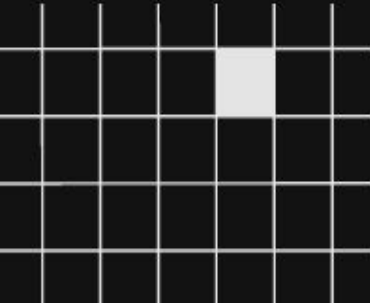
- [TypeScript in Visual Studio Code](#) | **CodeLens**
- [What is TypeScript?](#) | **TypeScript**

Disponible en nuestro repositorio.



¡MUCHAS GRACIAS!

Resumen de lo visto en clase hoy:

- Configuración de un proyecto
 - Tipado, clases, interfaces, decoradores.
 - Variables
 - Sistema de módulos.
- 



OPINA Y VALORA ESTA CLASE

#DEMOCRATIZANDO LA EDUCACIÓN

CODER HOUSE