

0Se sigue este curso → <https://www.udemy.com/course/angular-2-fernando-herrera/learn/lecture/6397656#overview>



INTRODUCCIÓN

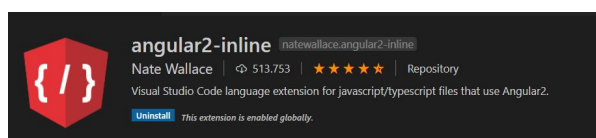
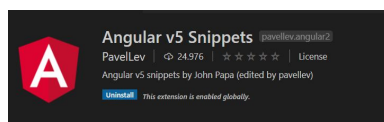
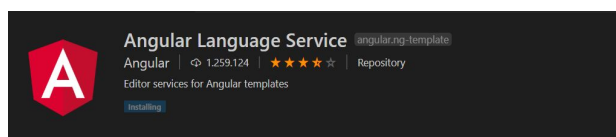
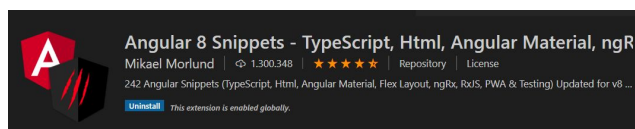
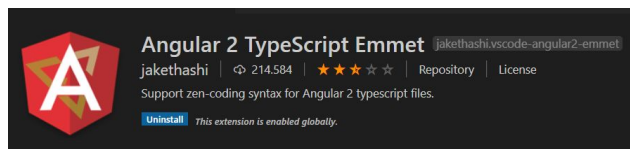
Para usar angular gastaremos las siguientes herramientas:

*TypeScript → Para detectar los errores en tiempo de escritura
Lo instalaremos con el comando en CMD → `npm install -g typescript`

*Node JS →

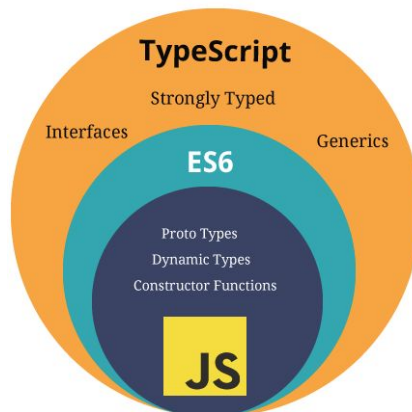
*Cliente Angular → `npm install -g @angular/cli`

Mis Snippets para Visual Studio



Typescript

A resumidas cuentas **TypeScript** es JS pero bien hecho y sin tanta entropía, podemos tener el control de código que nos daba java con algunas funciones añadidas, realmente TypeScript > **EcmaScript** > JS, solo que son ñapas que se han ido metiendo para dopar el código de mayor control



Comandos para compilar TypeScript

`tsc ts.ts` → Para compilar un ts a js

`tsc ts.ts -w` → Para que el programa escuche e ir compilando auto.

`tsc --init` → Para crear un archivo de configuración typescript. Desde ahí se compila todo el proyecto poniendo tsc aseca.

Remember de JS:

Var → Variable global

let → Variable local, a partir de ahora usar SIEMPRE LET

-function(**nombre:string**){} → Restricciones de parámetros en funciones para que no pite el código, nos avisa el propio intellisense que está mal si la cagamos.

-let nombre:**string**= "xsoms"; → Declaras en string fijo, realmente si ponemos una variable sin especificar nada ya lo pone solo, pero bueno... no está de más ponerlo.

-let nombre:**any**; → Es como hacerlo en JS puede adoptar cualquier valor y luego redeclararlo con otro

-let nombre:**number**; → Declara número

-let texto=**`Hola soy el número \${nombre}`**; → Typescript da esta nueva forma de crear líneas de string, concatenado con el acento invertido y \${}, podemos meter un ENTER y será igual a un \n

-let texto= **`\${getNombre()}`** → También podemos llamar funciones de JS desde ``

-function(**variable:string="por Defecto"**){} → Se dice que va a recibir una variable y si no de mete como parámetro se pondrá por defecto el valor asignado

-function(oka**?:string**){} → valor opcional, se puede invocar a la función sin meterlo, siempre es el último valor a declarar de todos los parámetros que pongamos

Desencriptación Objeto

Teniendo el objeto

```
let a={  
  atributo1="dato",  
  atributo2="dato"  
}
```

let{ nombre, clave, poder }= a; → Va buscando por nombre y lo desencripta, de tal forma que podamos usar la variable nombre fuera sin hacer alusión a a.

Si fuéramos a desencriptar un array de string por ej.

let a:string[]= ["dato1","dato2","dato3"];

let [parametro1, parametro2, parametro3]; → De esta forma no se enlazan por variables, sino por orden secuencial, parametro1 es "dato1"

Promesas(Ejecución función asíncrona)

```
let promesa= new Promise(function (resolve, reject){  
  console.log("Asíncrono realizado");  
  //Ejecutamos resolve() si queremos que acabe bien  
  //reject() si queremos que lance error  
})
```

```
promesa.then(function(){  
  console.log("Listo");  
}, function(){  
  console.log("Ejecutar si sale mal");  
});
```

→ La función then se ejecuta una vez acabe el proceso, se ejecuta la primera función si sale bien usando resolve y se ejecuta la segunda función y se ejecuta el reject().

Promise() → Es una clase que ejecuta funciones en su interior, se ejecuta de manera asíncrona al resto del programa, se podría considerar un servicio

Clases Interfaces y sus importaciones

```
interface Objeto{  
  nombre:string,  
  edad:number  
} → Interfaz de TypeScript
```

```
class Coche{  
  nombre:string="toni";  
  edad:string= undefined;  
  
  constructor( nombre:string, equipo:string , nombreReal:string){  
    this.nombre= nombre;  
  } → Constructor de la clase
```

} → Clase en TypeScript con valores por defecto

Podemos importar las clases con import {} from "url"

```
import { Coche } from "../clases/Coche.class";
```

Y exportarlas si ponemos un export al declarar la clase

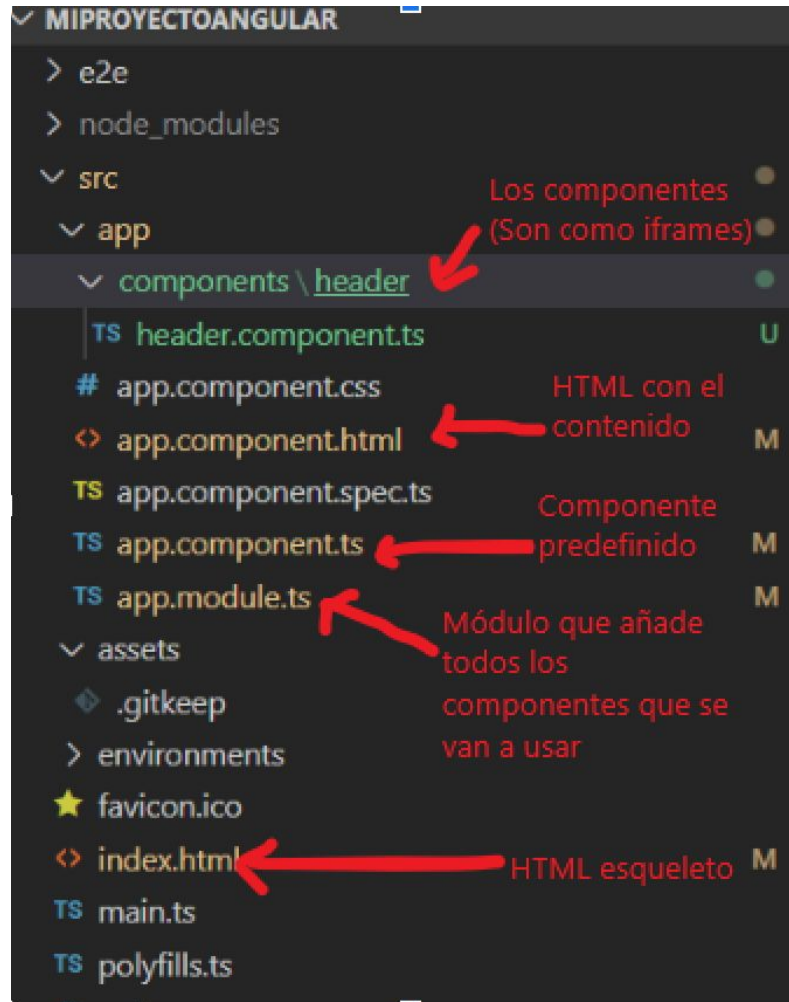
```
export class Coche{
```

ANGULAR

`ng new miProyectoAngular` → Para crear el proyecto

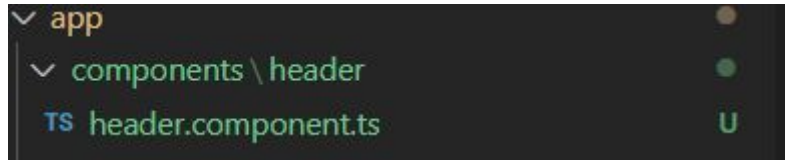
`ng serve -o` → Para lanzar el proyecto y abrir una pestaña nueva

Cuando creemos el proyecto se nos genera la siguiente estructura (los ficheros tocados son los más importantes)



Añadir componentes

Creamos un `.component.ts` como nomenclatura



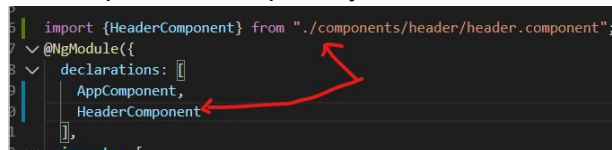
En la clase:

`import {Component} from '@angular/core';` → Para importar `@Component`

`@Component({`
 `selector: 'app-header',` → Es con lo que se invocará el componente en HTML
 `template: '<h1>Componente</h1>'` → Lo que imprimirá el html cuando se invoque
})

`export class HeaderComponent` → No hace falta llenar la clase, lo importante es `@Component`

Lo añadimos al `app.module`, para avisar que hay módulo nuevo



Ya lo podemos usar en HTML como:

```
<app-header></app-header>
```