|  |  |
| --- | --- |
| angular | **Documentación Curso Angular**  **Instructor: DevTalles.**  **ADSO**  **ID:2873711**  **27/01/2025**  **Centro de Procesos Industriales**  **Regional Caldas** |

Contenido

[**instalaciones necesarias** 3](#_Toc220427296)

[Instalaciones Recomendadas. 3](#_Toc220427297)

[Descargar Node.js versión 14 a 16 para Angular 14 . 4](#_Toc220427298)

[Extensiones. 7](#_Toc220427299)

[**Instalación de Angular 14** 8](#_Toc220427300)

[**Conceptos Que es TypeScript** 10](#_Toc220427301)

[¿Por qué Angular Usa TS? 12](#_Toc220427302)

[Mitos y Realidades de Angular 13](#_Toc220427303)

[**Base de TypeScript (Sección 3)** 14](#_Toc220427304)

[Error Instalar versión vieja de npm: 16](#_Toc220427305)

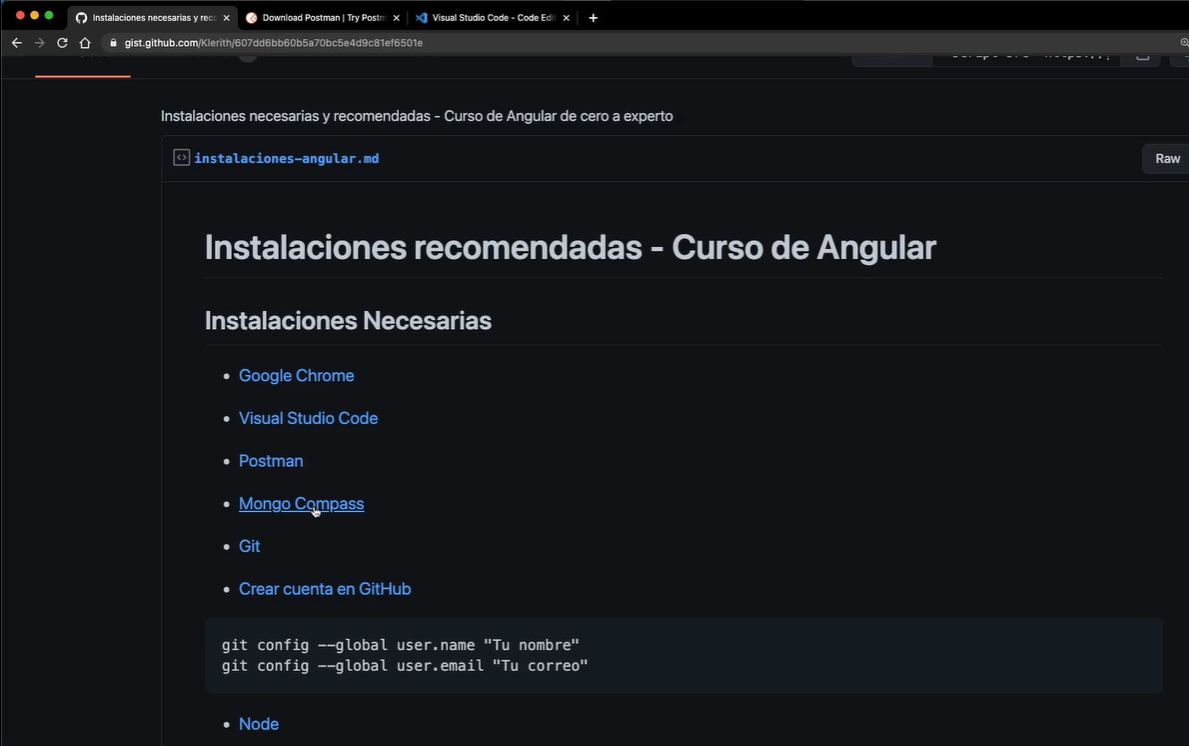
[Error en la ejecución de npm start, instalar versión vieja de TypeScript: 18](#_Toc220427306)

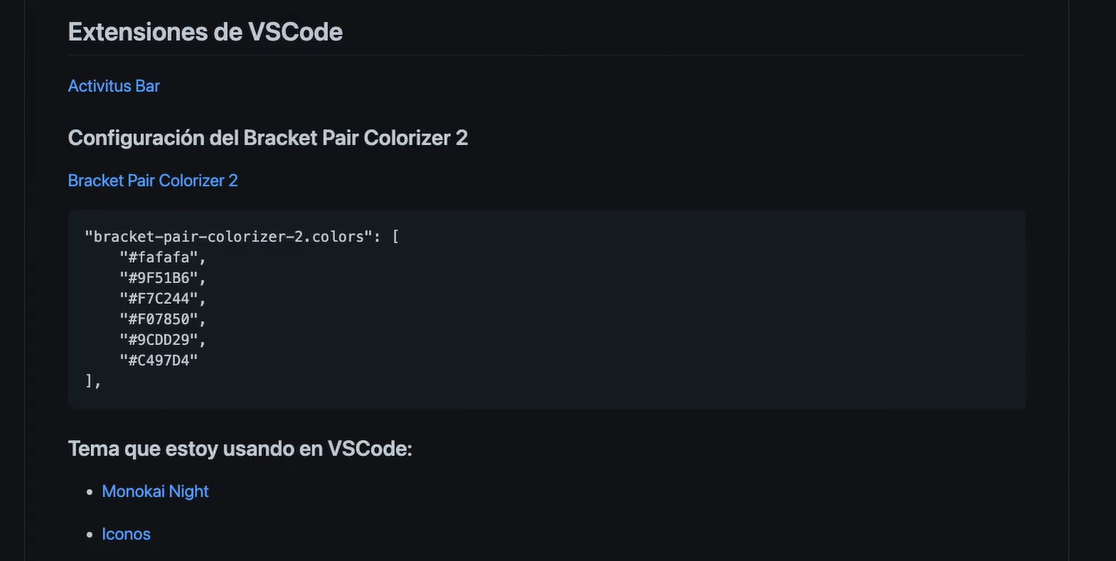
[Error en la ejecución de npm start, instalar versión vieja de TypeScript: 24](#_Toc220427307)

[Continuación del curso : 31](#_Toc220427308)

# **instalaciones necesarias**

### Instalaciones Recomendadas.

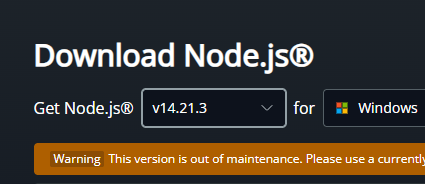




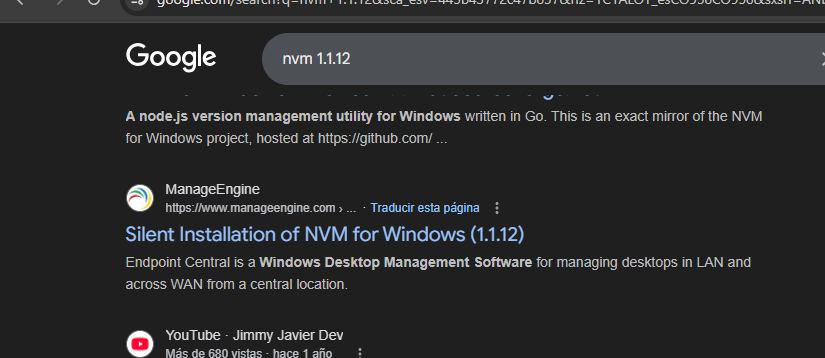


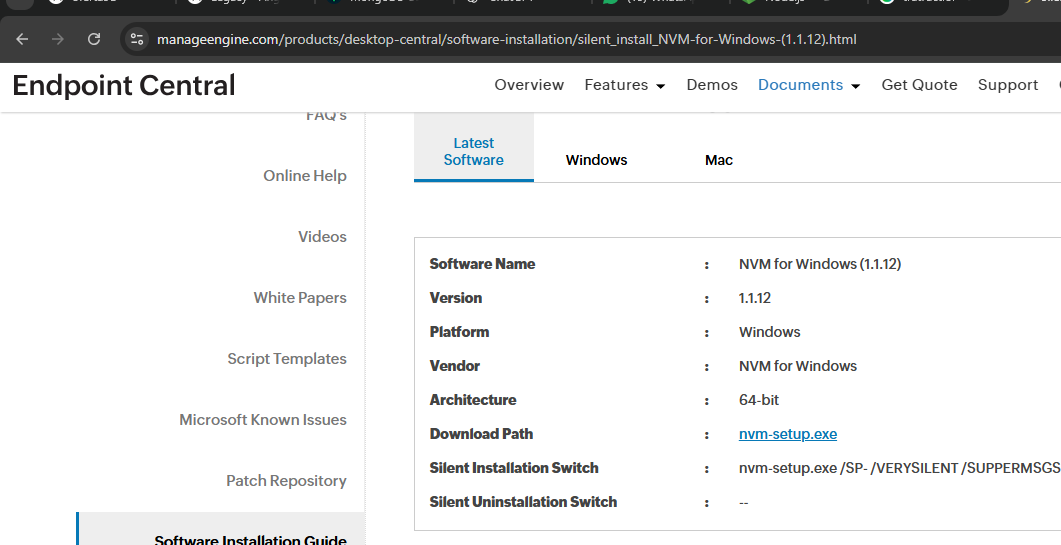
### Descargar Node.js versión 14 a 16 para Angular 14 .

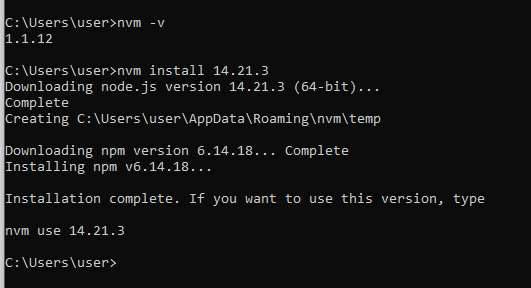
Necesitamos descargar la Versión de node:



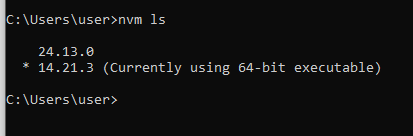
Para descargarla necesitamos el NVM (Node Version Manager) es una herramienta de línea de comandos esencial para desarrolladores que permite instalar, gestionar y cambiar fácilmente entre múltiples versiones de Node.js en una misma máquina.







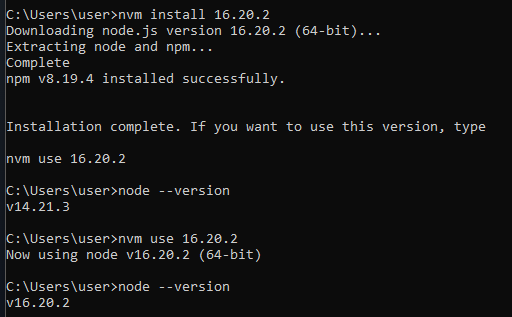
Comando para ver la lista que tenemos de node descargada:



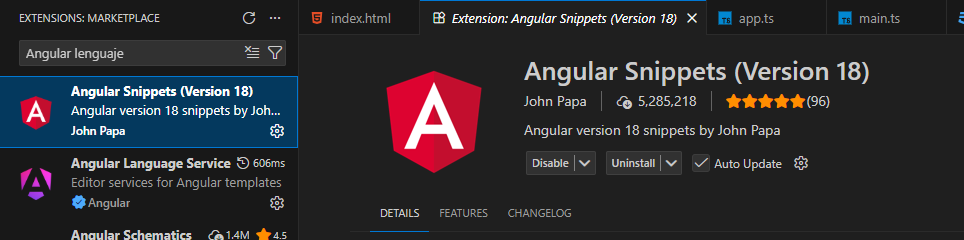
Para seleccionar nuestro node a usar :

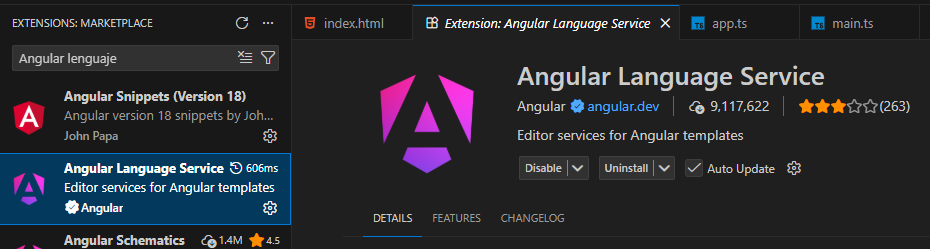


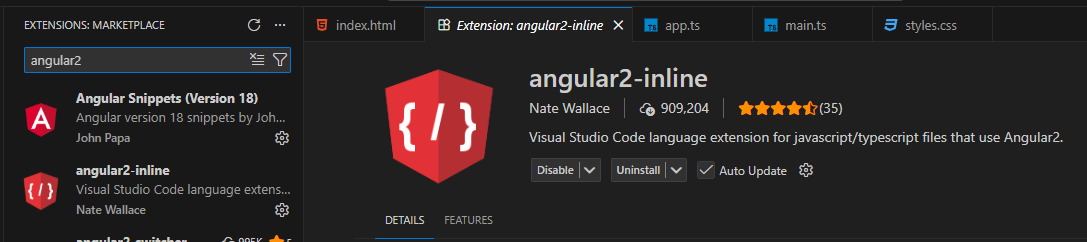
Mejor instalar versión recomendada 16.20.2:

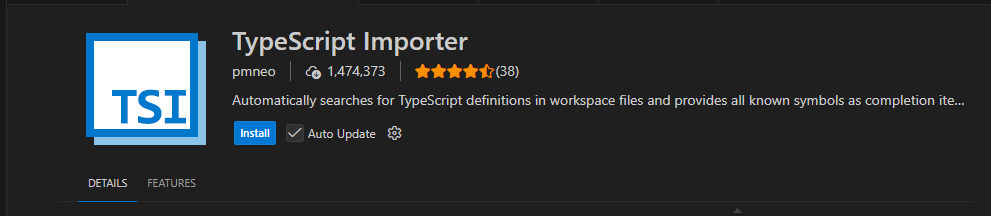


### Extensiones.

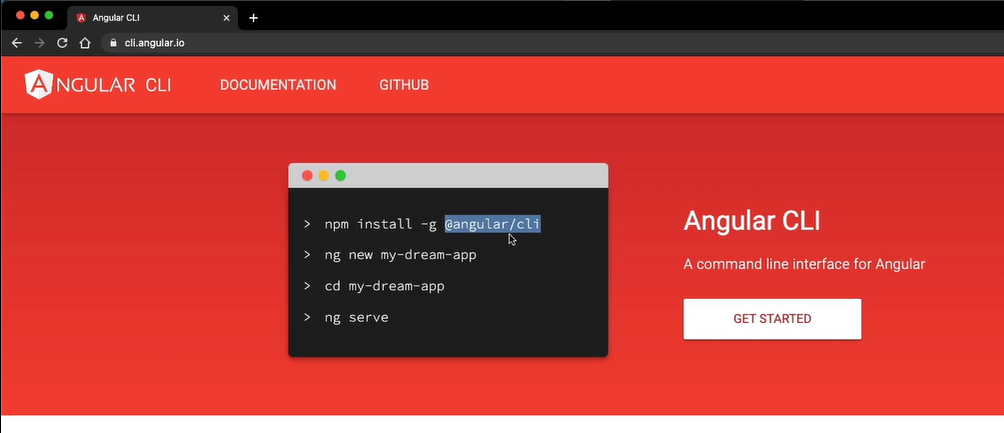


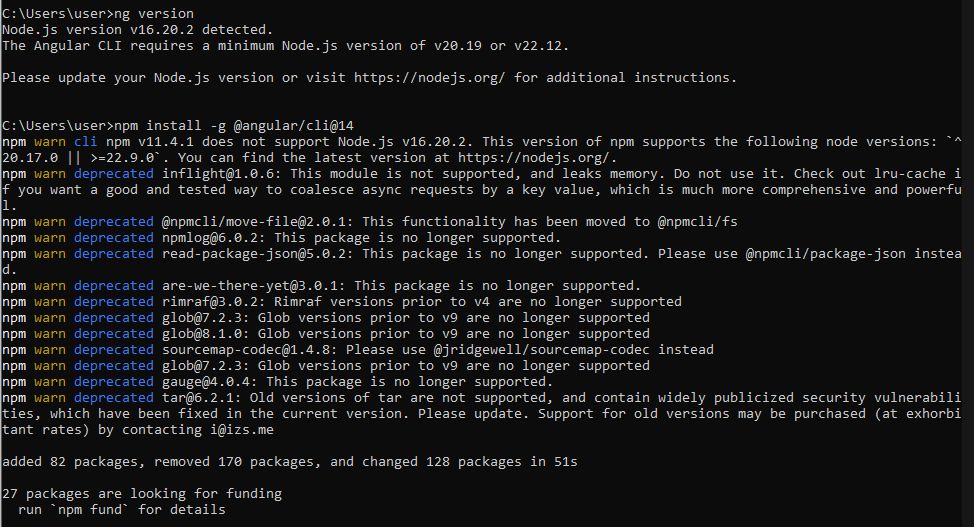






# **Instalación de Angular 14**





Recordemos pararnos en la versión node 16 .20.2 y luego instalamos con el comando.

Y ya queda instalado:



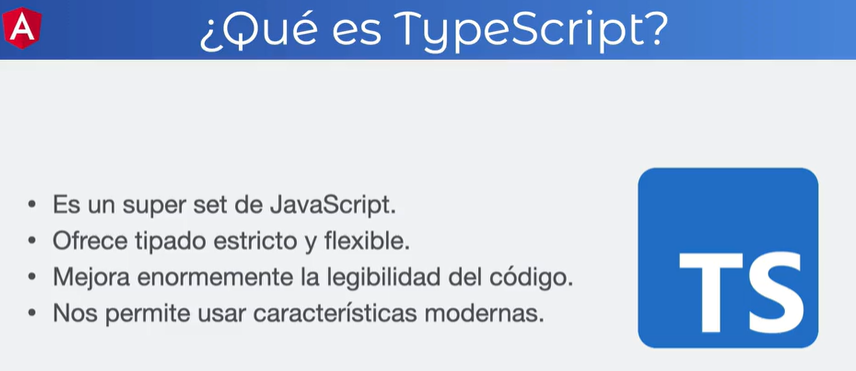
# **Conceptos Que es TypeScript**





Microsopft es el que mantiene TS







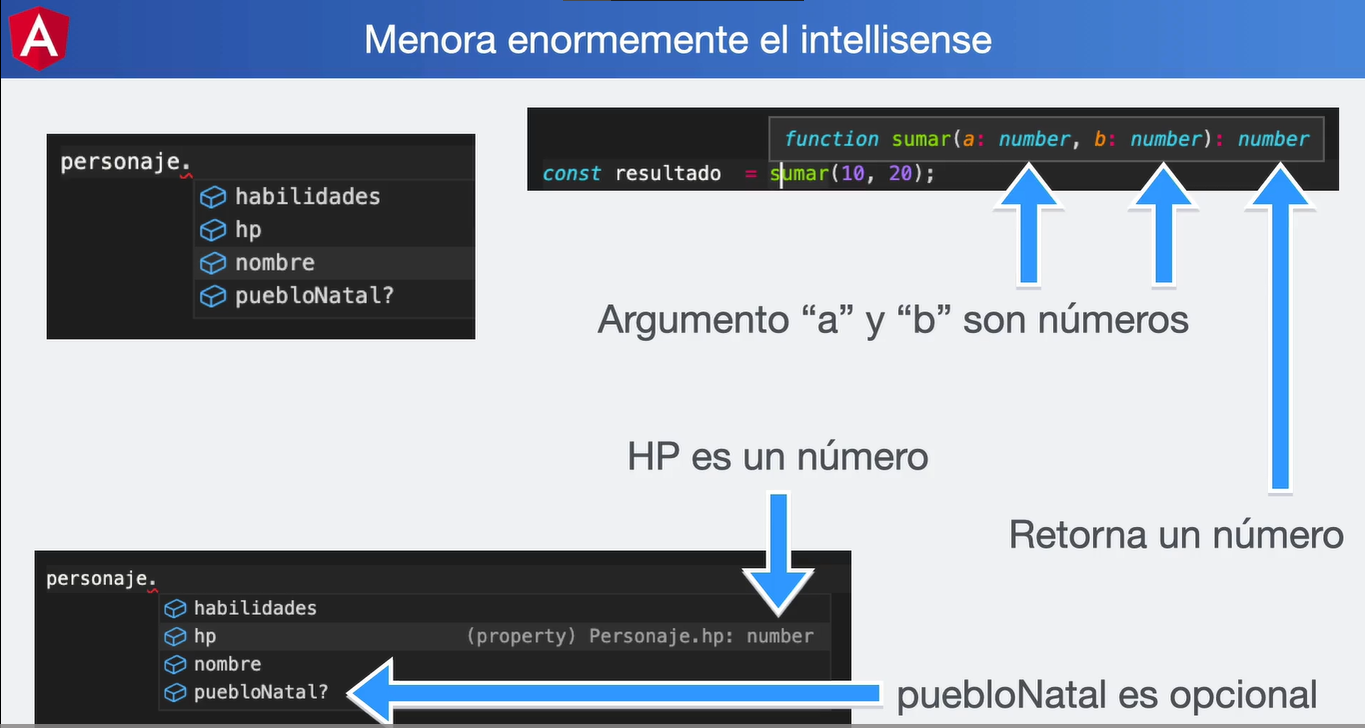
Esto JS no lo puede responder por sí solo, donde puede traer problemas y practicas.

Con TS:



Aquí es la diferencia.

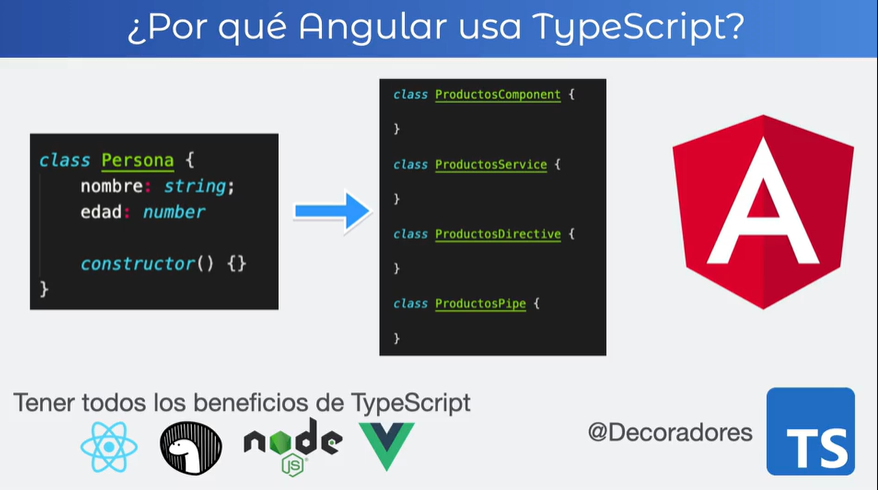
Otra característica de TypeScript:



Aquí e sla importancia de que TS pone tipado todo

### ¿Por qué Angular Usa TS?

Por Angular trabaja con Clases de TypeScript lo cual las clases de TS son las mismas de Angular





### Mitos y Realidades de Angular

Por qué angular usa Ts ?

Aquí volvemos con el tema de de que una tecnología es mejor que otra , pero cada una tiene su función a lo que quiere usar.

# **Base de TypeScript (Sección 3)**

Temas puntuales de la sección

¿Qué veremos en esta sección?

Este es un breve listado de los temas fundamentales:

Introducción a TypeScript

Tipos básicos

Objetos, arreglos e interfaces

Funciones y sus argumentos

Desestructuración de arreglos y objetos

Importaciones y exportaciones

Clases, constructores

Tipos genéricos

Decoradores

Encadenamiento opcional

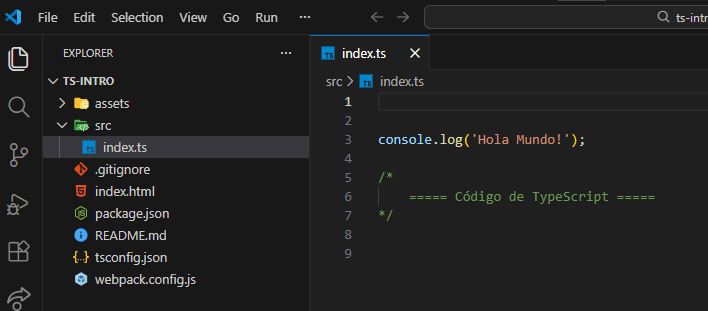
La idea de esta sección no es hacerlos expertos en TypeScript, pero sí irnos acostumbrando a la sintaxis y el tipado estricto de datos.

Después de la sección, podrán ver con otros ojos TypeScript y enfocarnos en todo lo que nos puede brindar, en lugar de las barreras que nos puede poner.

Lo descargamos desde el curso y seguimos las indicaciones :

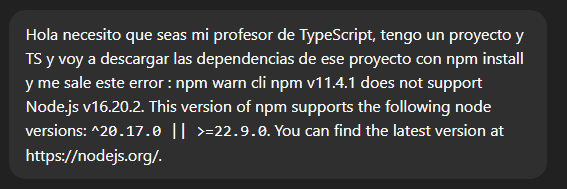


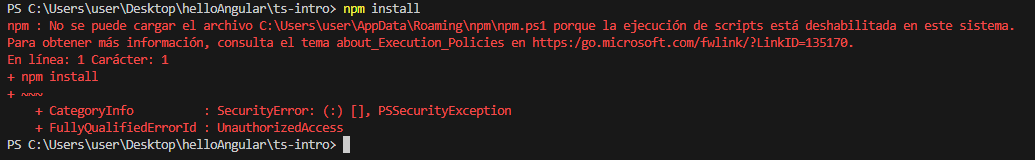
Lo nombramos como :

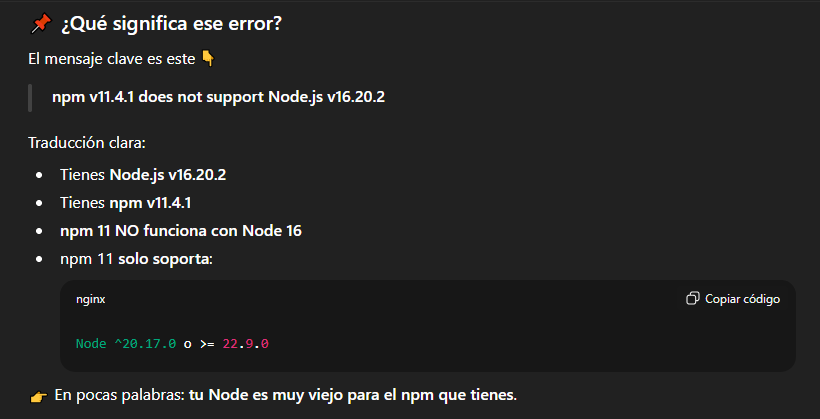


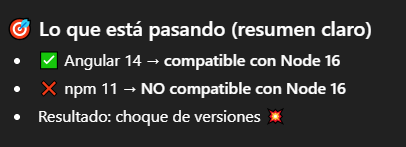
Instalamos todas las dependencias que nuestro proyecto TS necesita:

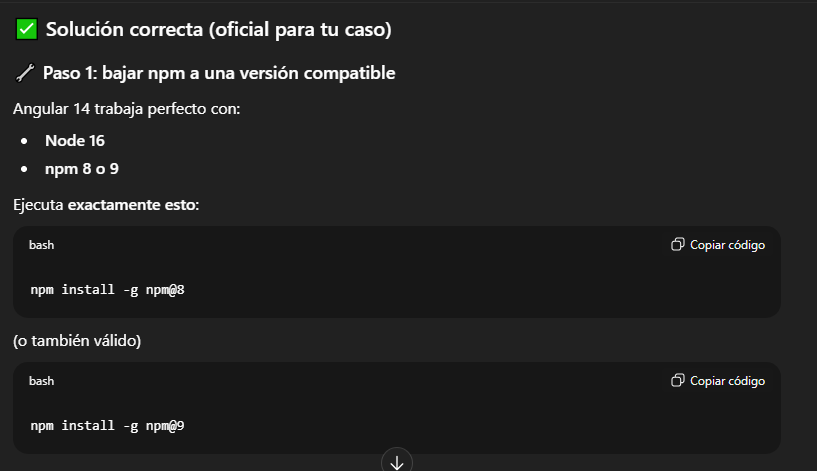
### Error Instalar versión vieja de npm:

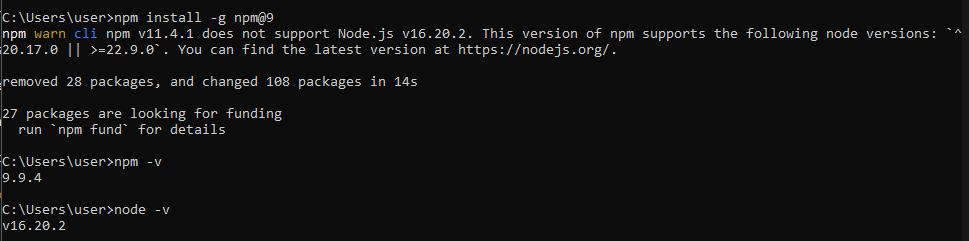


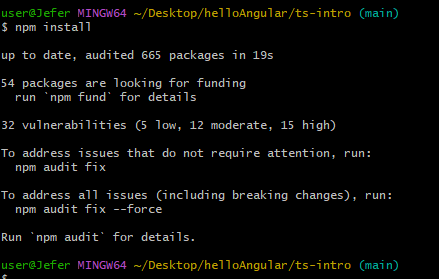








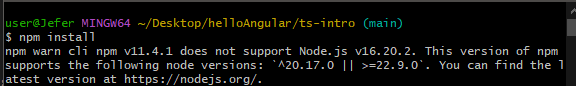




Y listo ,

**Recordemos que el npm es para instalar todas las dependecias que nuestro proyecto necesita para funcionar**

Para iniciar proyecto :

****

### Error en la ejecución de npm start, instalar versión vieja de TypeScript:

****

****

****

****

****

****

****

****

Siguiente error:







Error final de minipath:







### Error en la ejecución de npm start, instalar versión vieja de TypeScript:

****

****

****

****

****

****

****

****

Siguiente error:







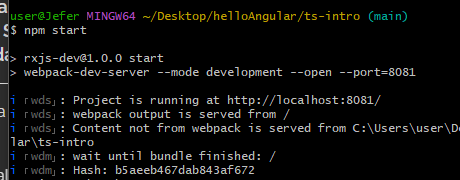
Error final de minipath:

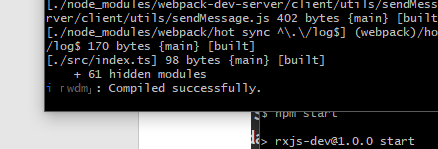


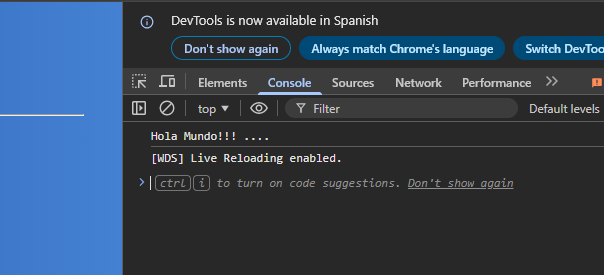




### Continuación del curso :



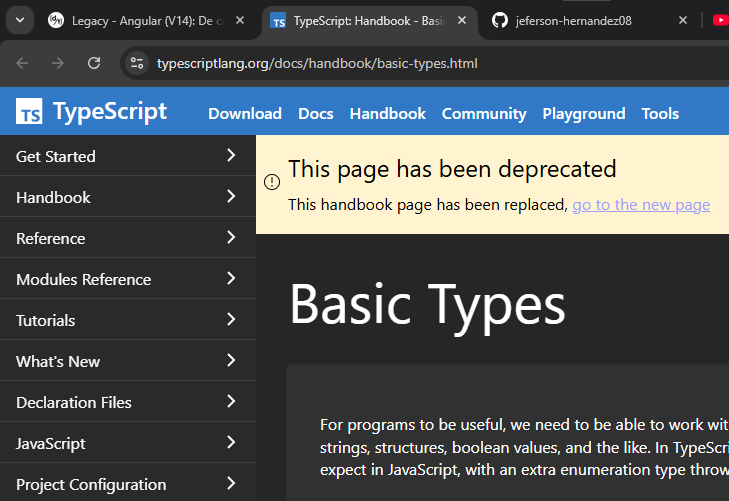




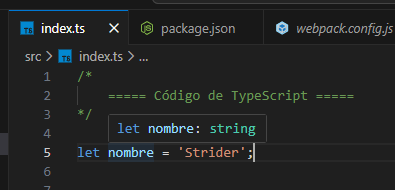
Debe compilar OK

### Tipos Básicos y Conceptos generales:

Recordar siempre documentación

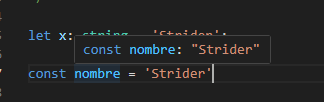


Podemos ver mas la difrecia de JS a TS :

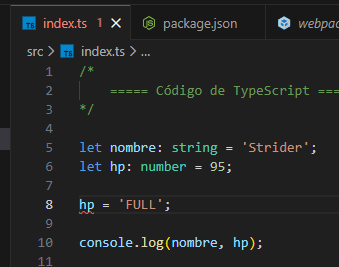


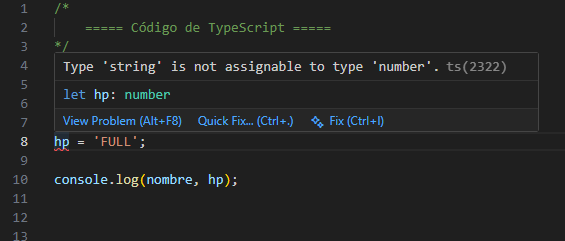
Los dos puntos después de una variable indican que tipo de variable es, por que TS indica o infiere el tipo de dato.

Pero con const en este casi siempre va ser Strider por que const es un tipo de variable que nunca va a cambiar:



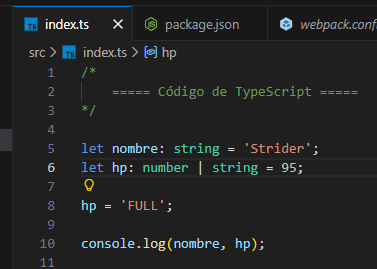
En este caso :

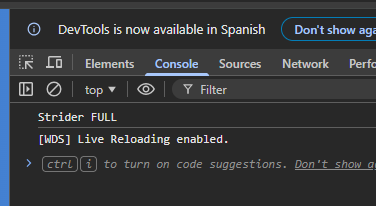




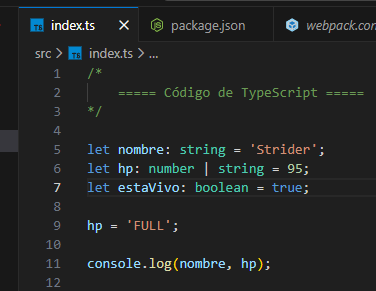
Nos sale error por TS nos indica que hp no es String.

Pero si queremos que dejarlo de las dos maneral le hacer con un o |:

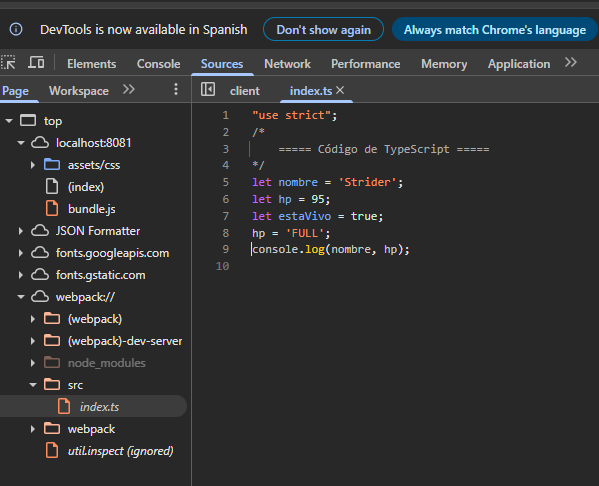




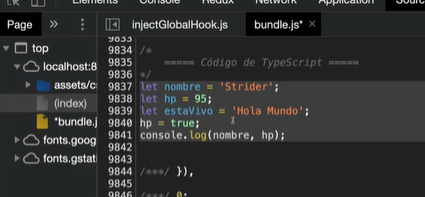
Tipos de datos:



Si damos click en index.ts podemos ver que nos lelva a onde se imprime y podemos ver que no hay nada de TS, por que lo escribamos de TS al navegador web se convierte en JS por que el navegador web no soporta o no entiende TS, lo que TS es traducirlo a JS , donde en consola no vemos el tipado:



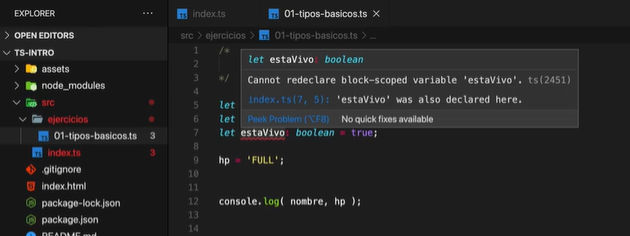
Donde cambiamos estos datos desde consola y nos funciona por que ya es JS:



Aquí en esta consola no vamos a ver código en TS por que el objetivo de TS es que tomemos las caractericas nuevas de TS y JS y la traduzca.

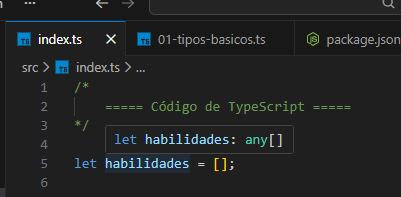
Conclusión en VScode vemos el tipo TS y en Consola vemos JS

Finalmente podemos ver que si creamos dos archivos con las variables iguales nos sale error por el scope global, esto es lo que tiene TS:



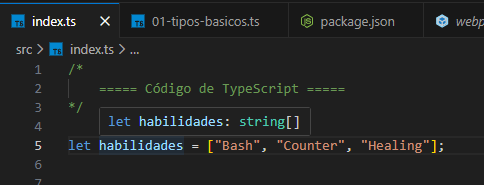
Esto es porque ponemos todas las variables en el scope global, borramos un archivo para dejar únicas variables y no choquen las variables

### Objetos, Arreglos e Interfaces:



Recordemos que : es el tipo y podemos ver que es de tipo any, es tipo especial de TS que le dice que podemos meter ahí lo que se nos de la gana , any es lo que sea.

Ponemos array tipo string:

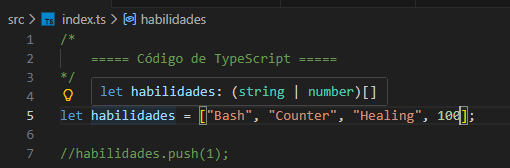


TS nos indica que habilidades es de tipo String.

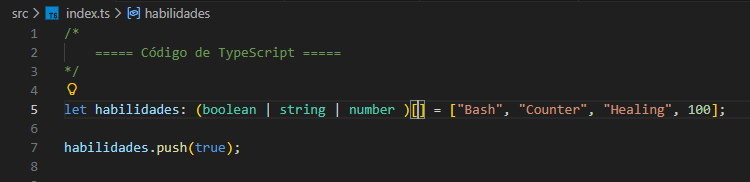
Si agregamos un datos de tipo number al array nos sale error :



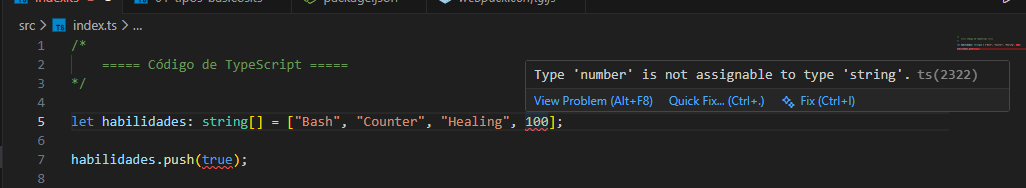
Si le agregamos un dato number TS nos indica string o number automáticamente:



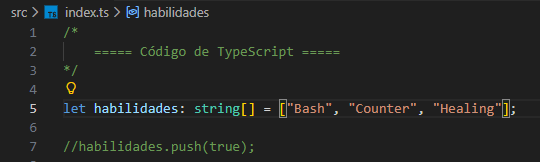
Si quremos que nuestro array tenga todo tipo de dato lo tipamos:



Pero si queremos que nuestro array de habilidades solo soporte strings lo tipamos:

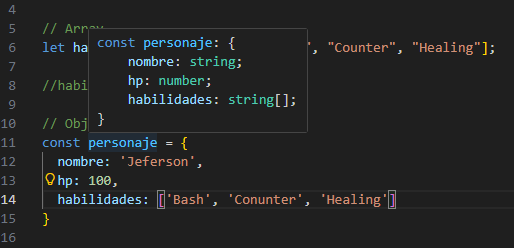


Obviamente sale error, donde lke debemos poner el tipo,, es buena practica :



Objetos:

Onjeto en JS



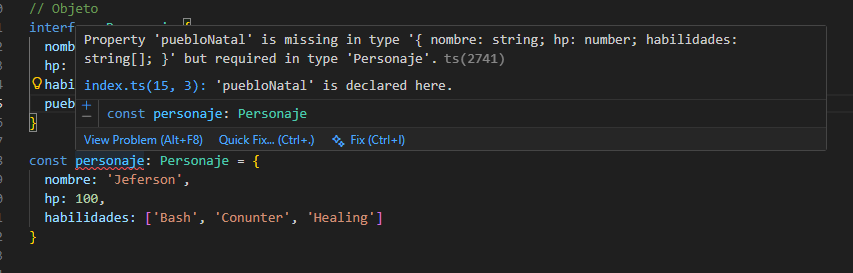
Una manera de tiparlo es escribit any ,



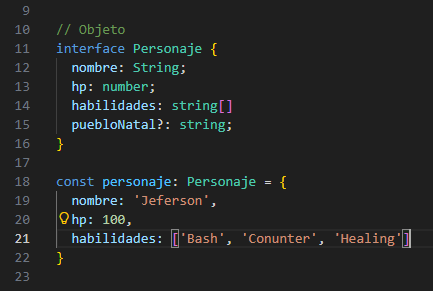
Pero queremos hacerlo con interface.

Interface: Una interface es una serie de llaves para asegurarnos que nuestro código sea como nosotros queramos o que un objeto sea como nosotros queramos.

Aquí TS como es tipado nos obliga a escribir el código:

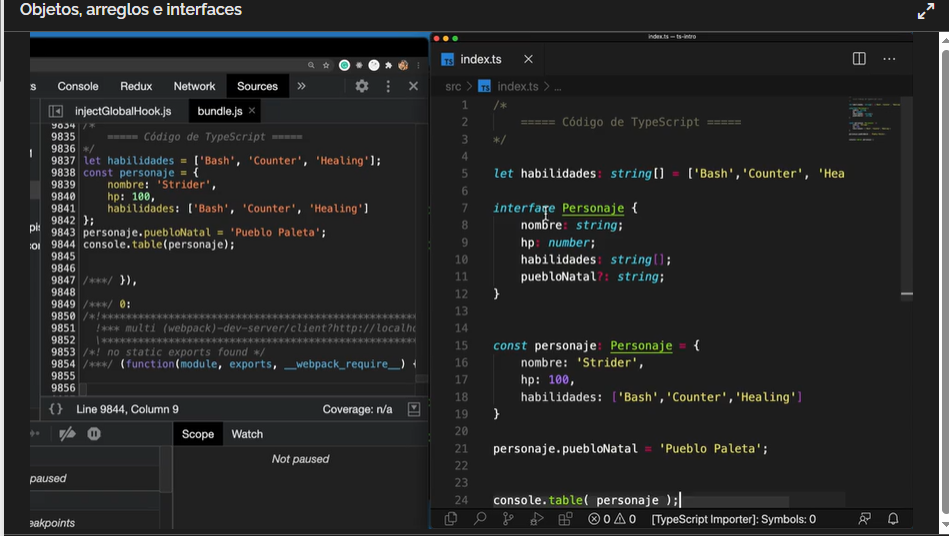


Con ?: , para indicar que es opcional la variable.



console.table(personaje); : Lo que hace es imprimir el objeto en consola.



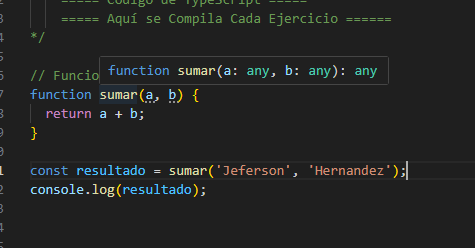


Podemos ver que el código JS Y TS en consola, click en index,js

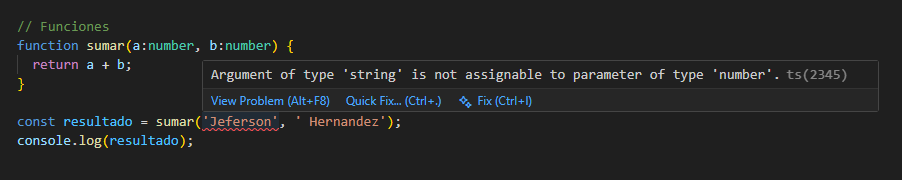
Podemos comparar como TS traduce a JS y no tiene interface en JS es diferente.

### Funciones básicas:

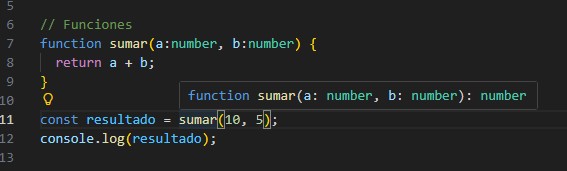
En función normal de JS recordemos mirar lo que dice en TS ese el truco:

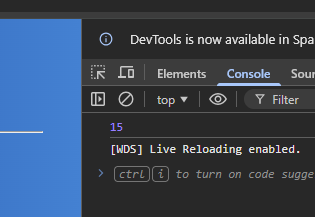


Lo tipamos :

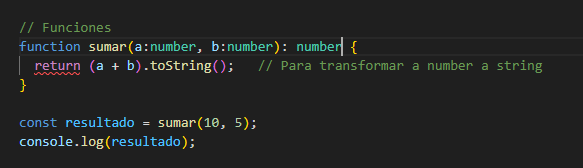


Lo corregimos y nos muestra el retorno de number, que sale un numero:





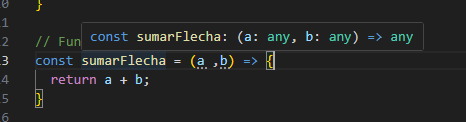
Podemos obligar TS al final de la función que sea el tipo de dato de salida:



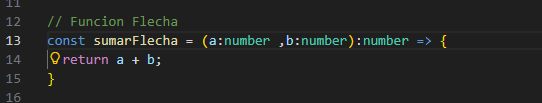
Sale error por que nos dice que retorna un tostring.

**Función Flecha:**

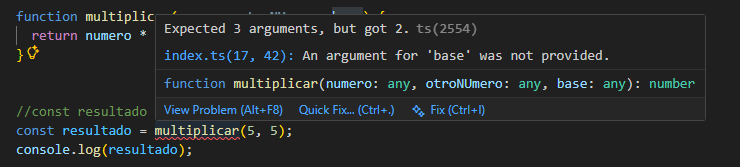
En función flechs JS veamos la documentación de TS:



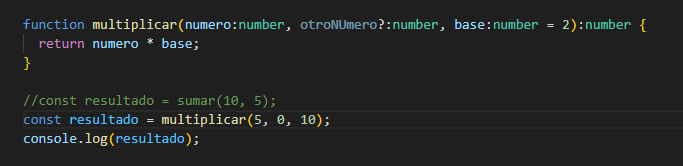
A TS:



En este otro ejemplo siempre observemos lo que nos dice TS:

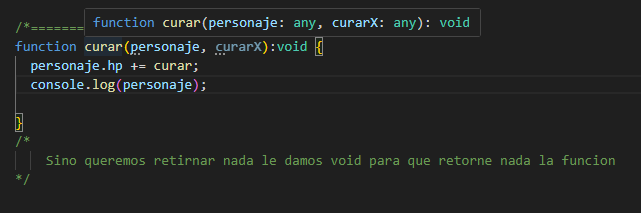


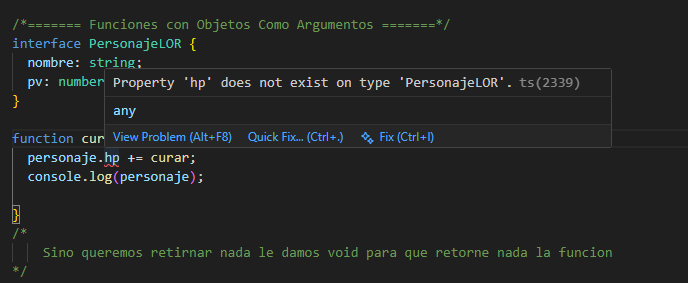
Indica que siempre poner los paramametros obligatorios tipado y el retorno también.



El primer dato es el número asignado, el segundo es opcional y el tercero es por defecto, retorna total de 50 por que le enviamos el argumento base 10 .

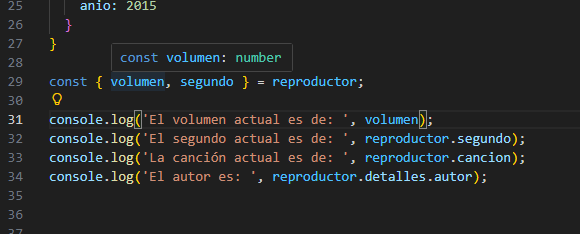
### Funciones con Objetos como Argumentos:





TS nos indica que variables podemos usar dentro de la interface que creamos.

### Desestructuración de Objetos:

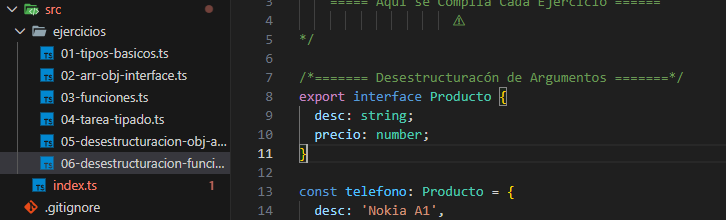


Aquí podemos ver que nos indica el tipo de dato.

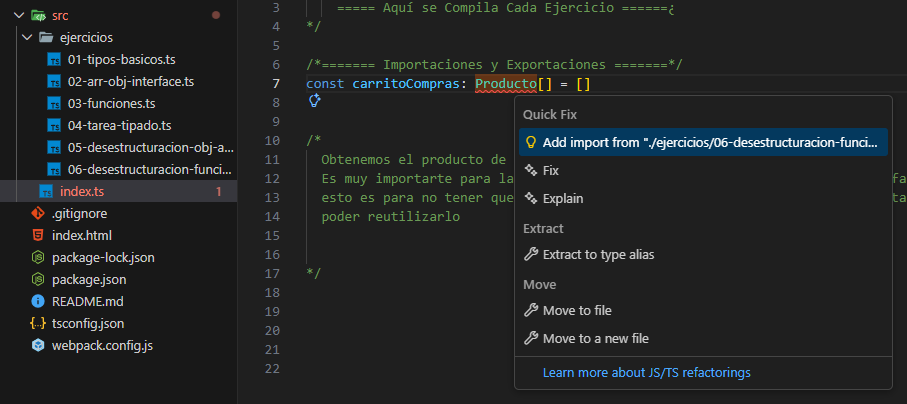
### Importaciones y Exportaciones:

Es muy importarte para la reutilización de código si deseamos tener la interface Producto, esto es para no tener que repetir código dos veces si no que podemos importar la interface Producto para poder reutilizarlo.

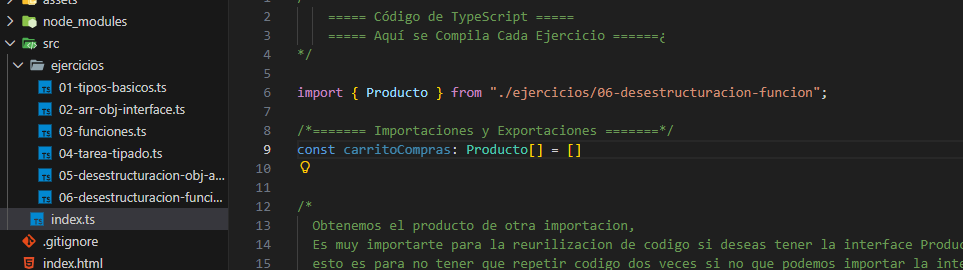
Del otro archivo anterior vamos a usar prodecto, para reutilizarlo damos export:



Luego en el otro archivo donde vamos a usar esta interface, nos paramos con el cursor en el error y damos ctrl + . para que se nos despliegue la importación para poderla importar la interface Producto:

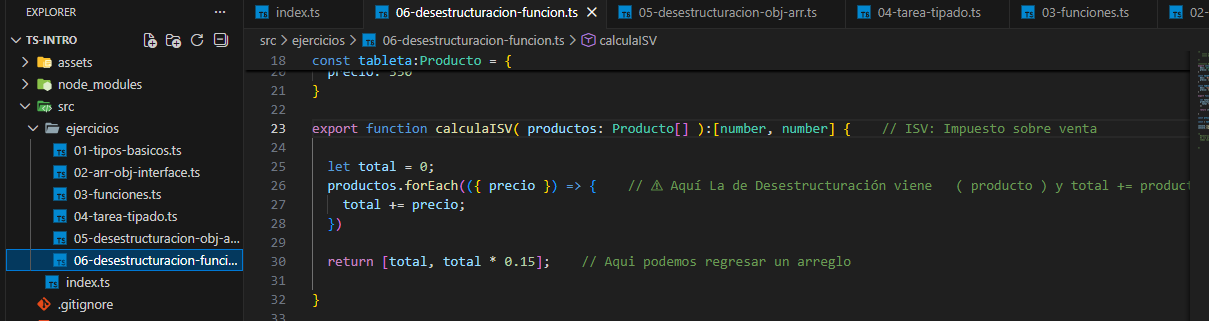


Lo elegimos y vemos que nos importa:

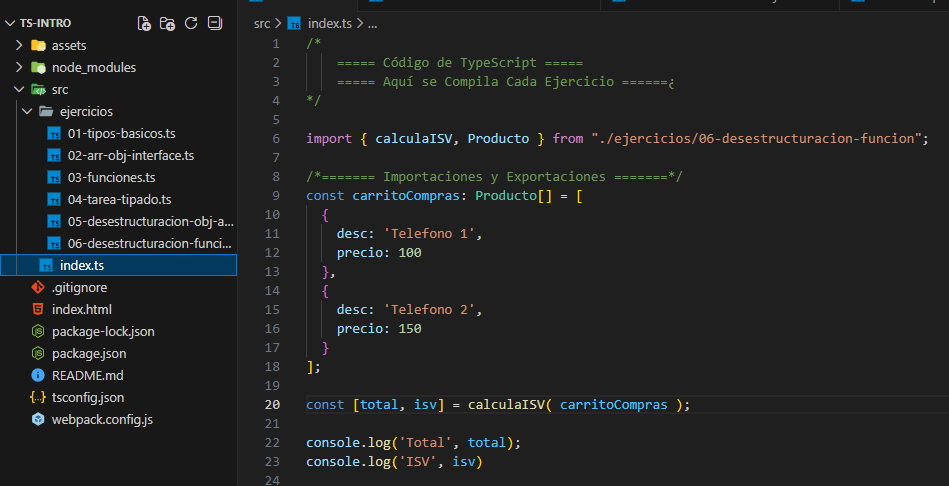


Podemos ver el objeto que ha sido exportado.

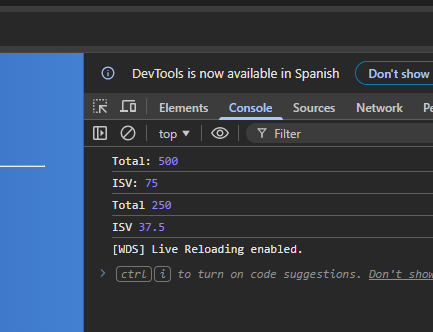
Así mismo con otro ejemplo de exportación con function calculaISV:



Y en el archivo de importación



Podemos ver estos ejemplos como se exporta y como se importa para reutilización de código.



Aparece en dos veces por que está ejecutando el otro archivo de exportación.

### Clases Básicas:

Son muy parecidas a las clases de JS.

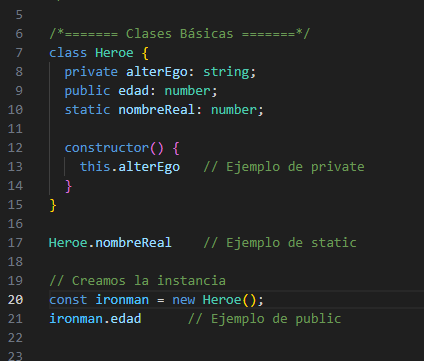
Diferencias entre private, public y static:

- Private: Significa que esa propiedad solo va ser visible dentro de esa clase Heroe

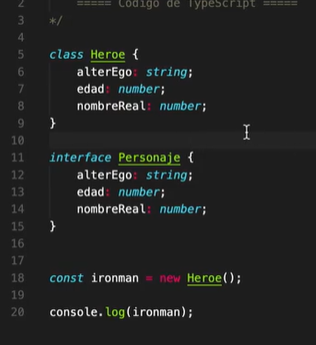
- Public: Significa que afuera de la clase Heroe alguien va poder ver esa propiedad.

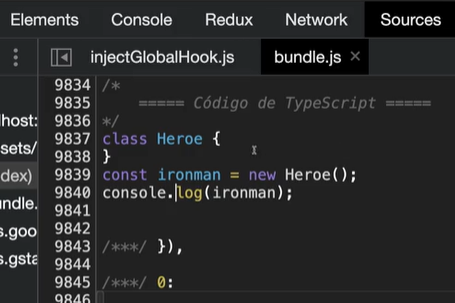
- static: Significa que podemos acceder al valor de esa propiedad sin crear una instancia de la clase

Podemos ver los ejemplos de cada una y su diferenciación:



La clase y una interface es lo mismo, pero la interface no aparece en consola con JavaScript

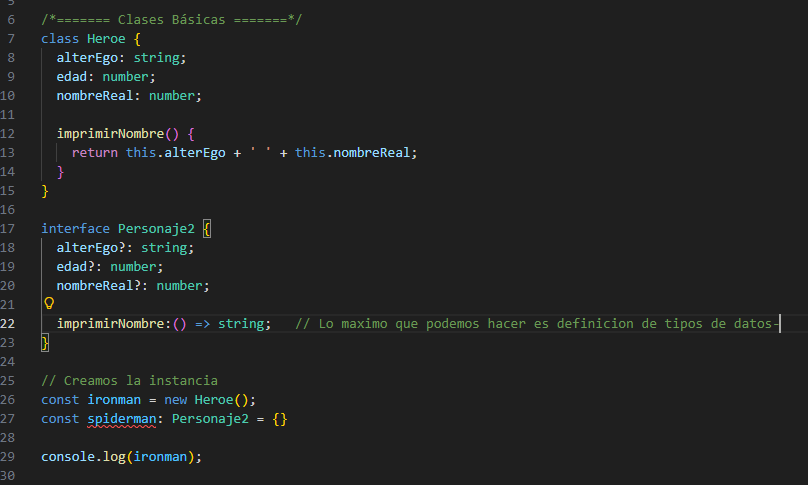




La diferencia es las clases nos sirven para crear instancias y en la Interface no podemos definir como quiero que funcione algún método o si queremos tener getters y seders.

Aquí en este podemos ver la diferencia entre clase e interface:

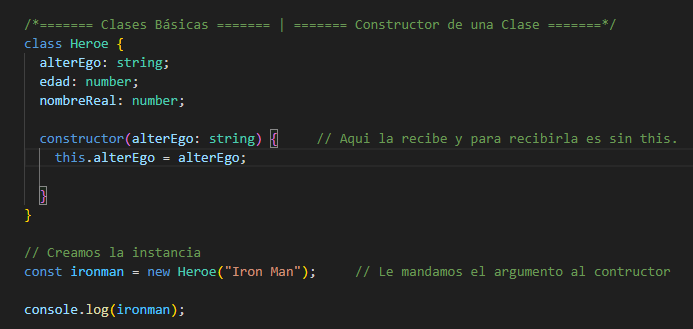
Por eso se dice que inteface es clase tonta.



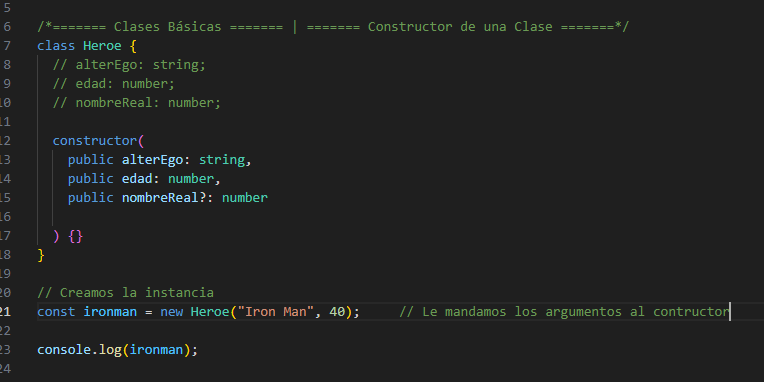
### Constructor de una Clase:

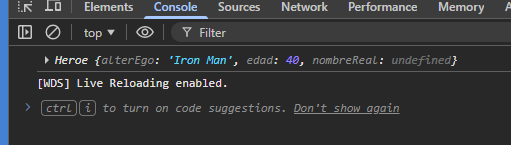
Hay dos formas de hacer el constructor de la clase,

Forma normal:



Forma corta:





Con public le decimos a TS que cree una propieda de una vez que es alterEgo y establezca la propiedad de la instancia a la propiedad de la clase .

El ? es para dejarlo como no obligatorio si no opcional.

Esta es la manera de crear de una vez atributos de una clase desde el constructor de una vez.

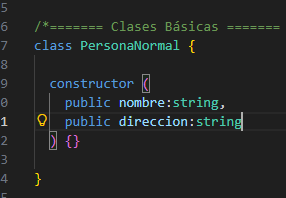
### Extender una Clase O Herencia:

Repasamos, para crear una clase normalmente:



Pero recordemos que con TS lo podemos resumir. A menos que sea estrictamente necesario

De manera resumida:

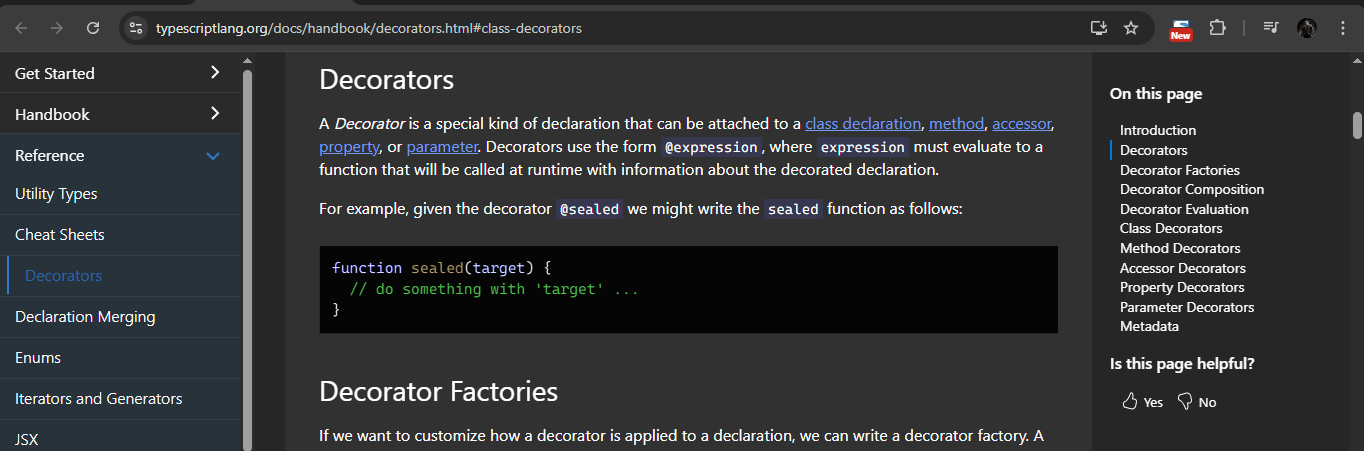


**Super: Debemos de tener la llamada del super , el super es para llamar al constructor del padre**

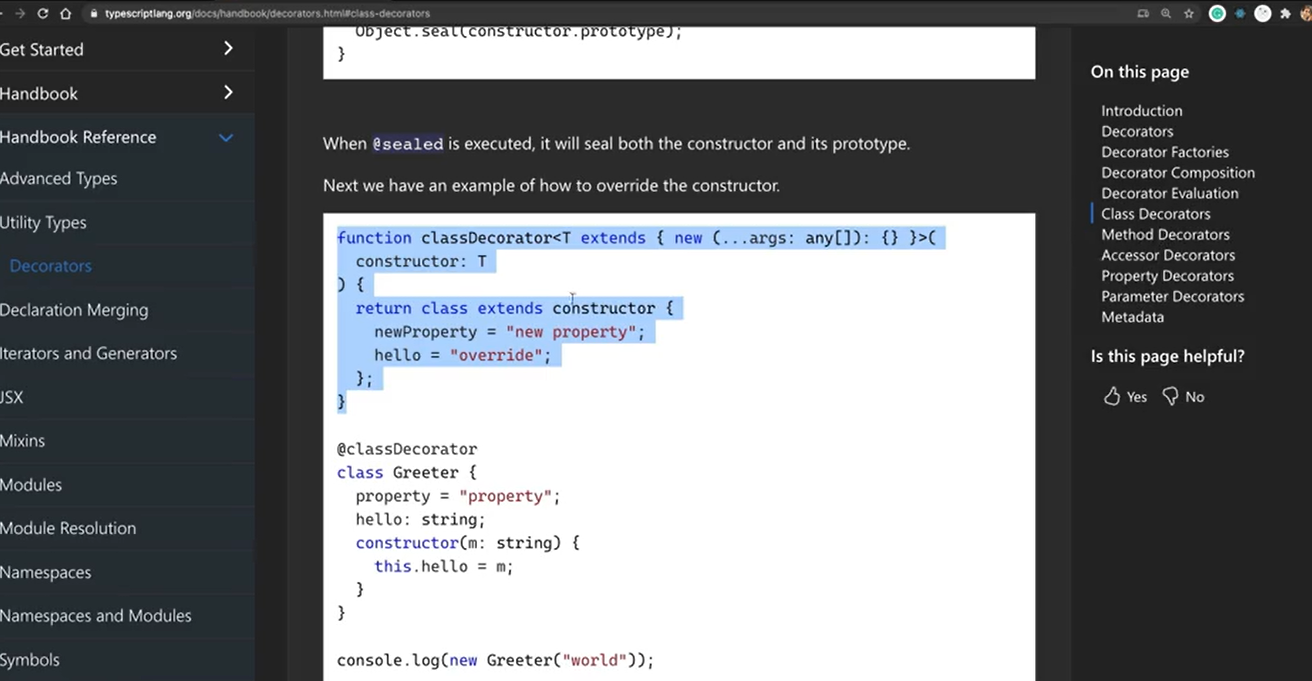
**Super llama al constructor del padre**

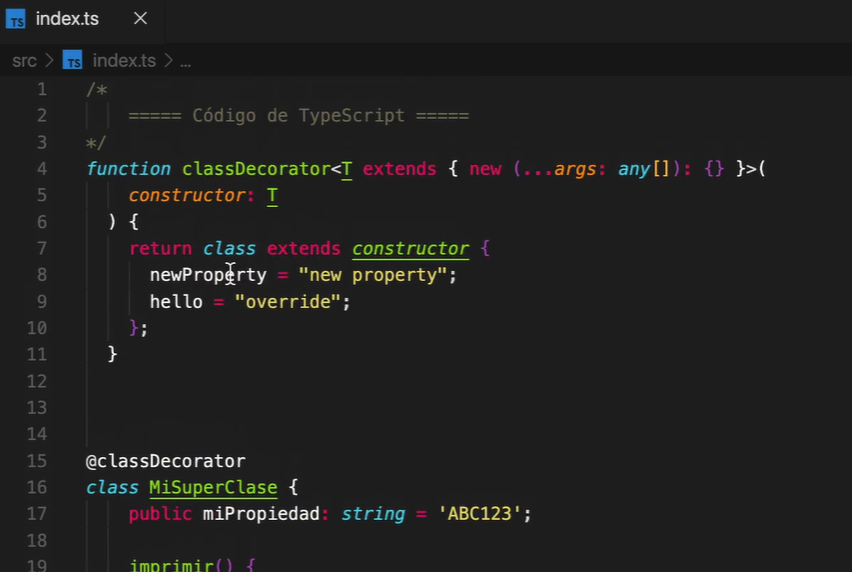
### Decoradores de Clases:

Topos de decoradores:

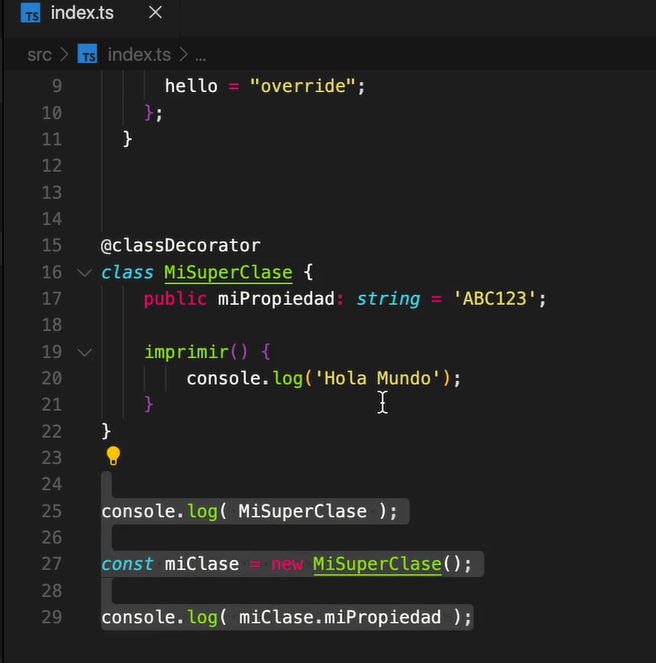


Decoradores en la documentación versión 14 , Video:





NOTA: En el video indica que es opcional escribir



NOTA: En el video indica que es opcional escribir