

Enlaces de interés:

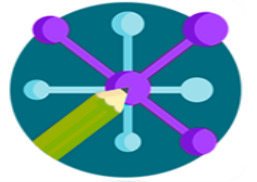
-Typescript playground

-<https://www.youtube.com/watch?v=rsYcRsmcPs8&t=0s>

-<https://github.com/Lemoncode/webinar-typescript>



ever
FUTURE



01 Qué es Angular

La **última tendencia** en el desarrollo web es la implementación de aplicaciones **web SPA**



SPA son las siglas de Single Page Application. Es un tipo de aplicación web donde todas las pantallas las muestra en la misma página, sin recargar el navegador.



1. Qué es Angular

Las **web SPA** son clientes completos implementados con **HTML, CSS y JavaScript** que se comunican con el **servidor web con API REST**

Existen **frameworks** especialmente diseñados para implementar **webs SPA**

Uno de los frameworks más usados es **Angular**

Pero, ¿qué es Angular?

- No es una biblioteca JavaScript. No hay funciones que podamos llamar y utilizar directamente.
- No es una biblioteca de manipulación DOM como jQuery. Pero utiliza el subconjunto de jQuery para la manipulación de DOM (llamado jqLite).
- AngularJS es un framework Javascript MVC creado por Google para crear aplicaciones web adecuadamente diseñadas y mantenibles de Tipo SPA.
- **Angular no tiene relación con AngularJS**, ya que tras su primera versión fue reescrito al completo.

1.

1.

¿Qué



Node JS nos proporcionará la infraestructura. **Chrome** las herramientas de desarrollo/debug. **TypeScript** como lenguaje de desarrollo. Y el **cliente Angular** para el trabajo con la consola.

1.



Plataforma para
ejecutar aplicaciones JS
fuera del navegador

¿Qué



Gestor de herramientas
de desarrollo y librerías
JavaScript (integrado
con node.js)



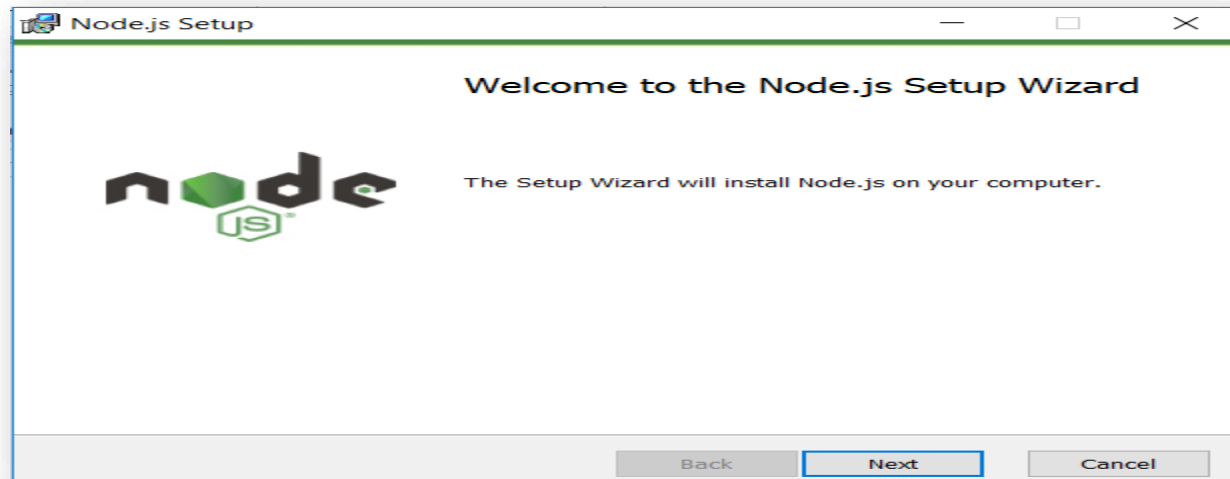
02 Instalación en Windows

NodeJS, TypeScript, Visual Studio code y Angular

Instalación de 

Accedemos a: <https://nodejs.org> y nos bajamos la 'Recommended For Most Users'.

Tras descargarla, la instalaremos.



Pulsamos 'siguiente' en todas las ocasiones y por último 'Instalar'.

NodeJS, TypeScript, Visual Studio code y Angular

Instalación 

Una vez finalizada la instalación, vamos a comprobar que se ha instalado correctamente, para ello abriremos la consola y ejecutaremos 'node -v':

```
Símbolo del sistema
Microsoft Windows [Versión 10.0.16299.1686]
(c) 2017 Microsoft Corporation. Todos los derechos reservados.

C:\WINDOWS\system32> node -v
v12.16.1
```

Tras ello lanzamos las actualizaciones con 'npm install npm@latest -g'.

Y tras finalizar comprobamos la versión de npm instalada con 'npm -v':

```
+ npm@6.14.2
added 435 packages from 866 contributors in 54.125s

C:\WINDOWS\system32> npm -v
6.14.2
```

NodeJS, TypeScript, Visual Studio code y Angular

Instalación de ^{TypeScript}

TypeScript es el lenguaje que vamos a utilizar de base en nuestra programación, para generar un código **JavaScript** más limpio con una correcta orientación a objetos.

Para instalarlo escribiremos en consola 'npm install -g typescript'.

```
C:\WINDOWS\system32> npm install -g typescript
C:\Users\jhernand\AppData\Roaming\npm\tsc -> C:\Users\jhernand\AppData\Roaming\npm\node_modules\typescript\bin\tsc
C:\Users\jhernand\AppData\Roaming\npm\tsserver -> C:\Users\jhernand\AppData\Roaming\npm\node_modules\typescript\bin\tsserver
+ typescript@3.8.3
added 1 package from 1 contributor in 13.753s
```

Comprobamos la versión con 'tsc -v'

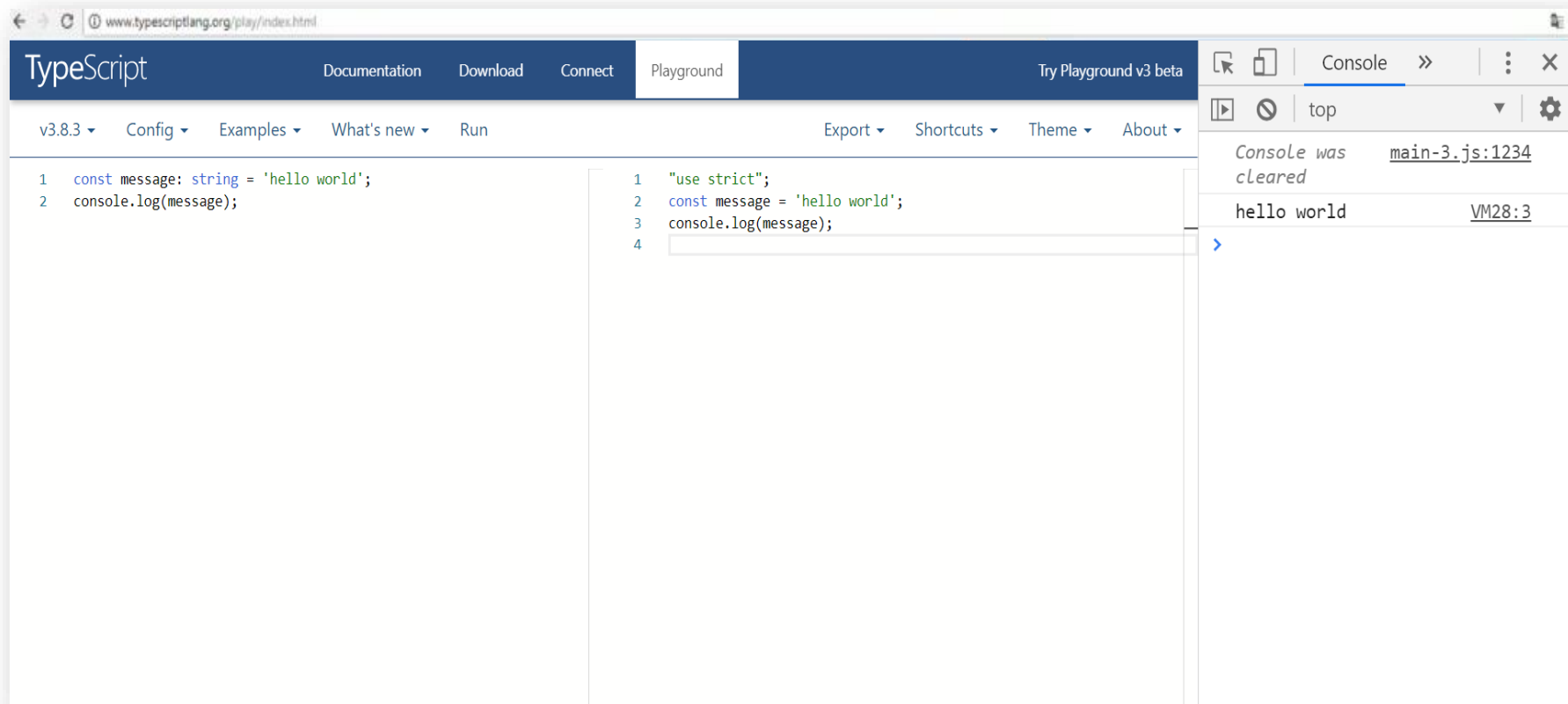


```
C:\WINDOWS\system32> tsc -v
Version 3.8.3
```

NodeJS, TypeScript, Visual Studio code y Angular

Instalación de TypeScript

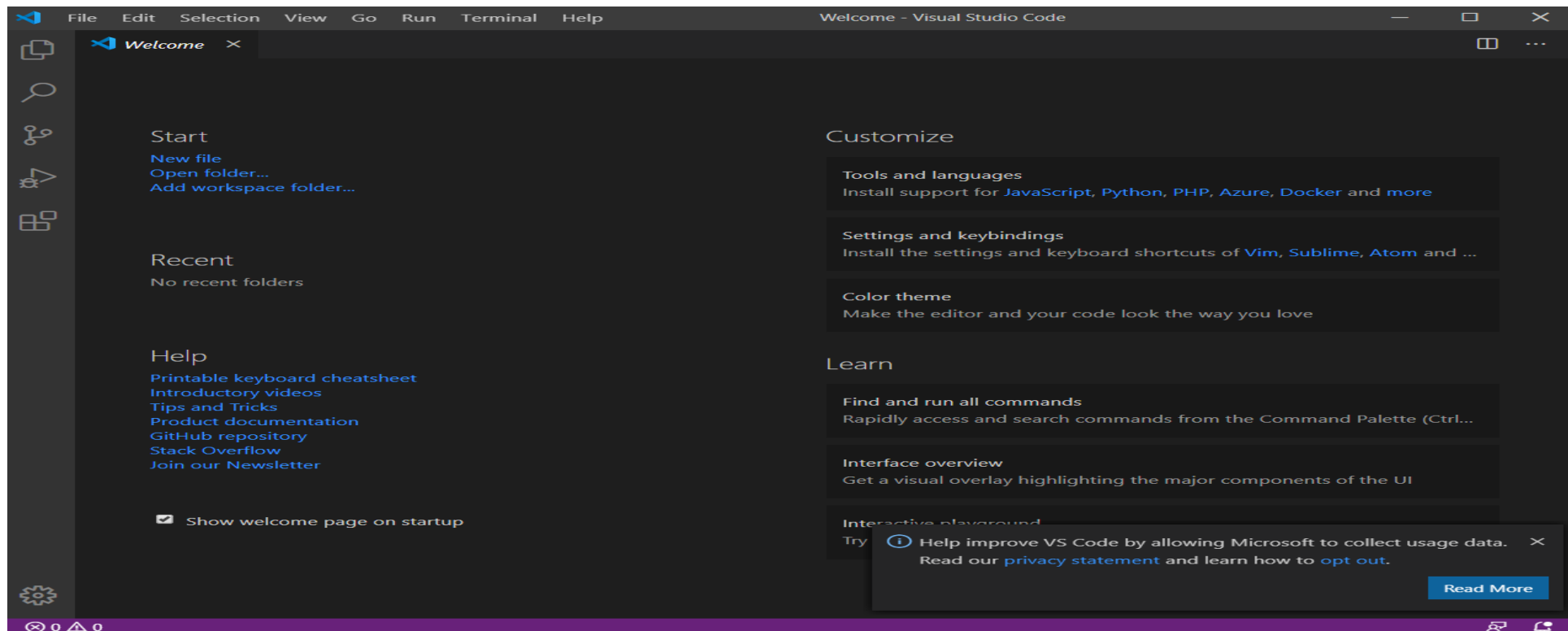
Desde el enlace <https://www.typescriptlang.org/play/index.html> vamos a poder escribir y probar nuestro código **TypeScript**:



NodeJS, TypeScript, Visual Studio code y Angular

Instalación de  Visual Studio

Desde el enlace <https://code.visualstudio.com/> nos descargaremos el IDE de desarrollo.

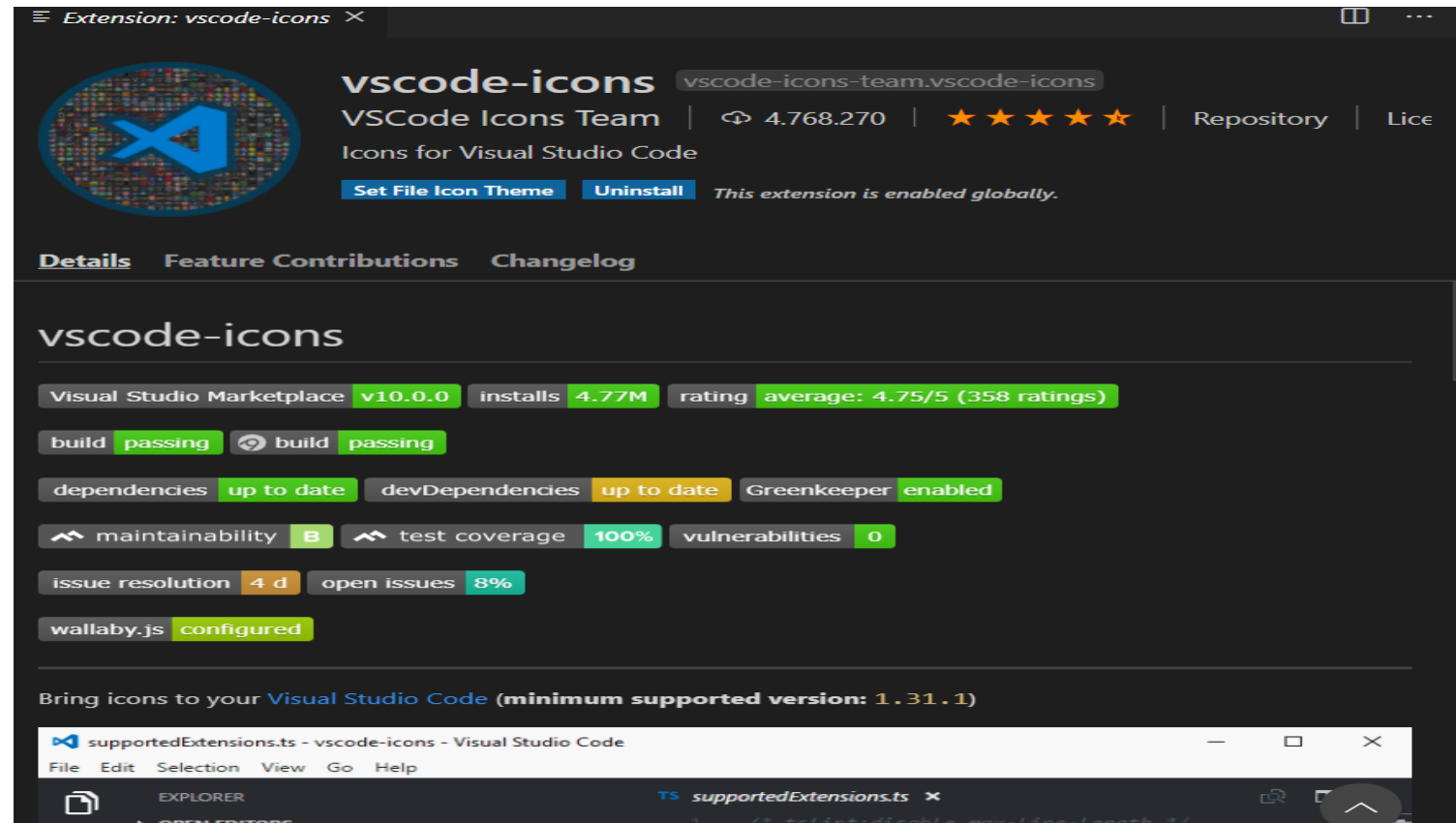


Para cambiar Visual Studio de idioma: Control+Shift+P → Configure Display Language

NodeJS, TypeScript, Visual Studio code y Angular

Instalación de Visual Studio

Posteriormente nos meteremos en la web y en la sección de extensiones y nos bajaremos la vscode-icons (también puede instalarse desde la sección de extensiones del IDE):



NodeJS, TypeScript, Visual Studio code y Angular

Instalación  ANGULAR CLI

La página de referencia es la siguiente: <https://cli.angular.io/>

Primeramente desde consola vamos a ejecutar **'npm install -g @angular/cli'**

```
C:\WINDOWS\system32> npm install -g @angular/cli
npm WARN deprecated mkdirp@0.5.3: Legacy versions of mkdirp are no longer supported. Please update to mkdirp 1.x. (Note that the
API surface has changed to use Promises in 1.x.)
npm WARN deprecated request@2.88.2: request has been deprecated, see https://github.com/request/request/issues/3142
C:\Users\jhernand\AppData\Roaming\npm\ng -> C:\Users\jhernand\AppData\Roaming\npm\node_modules\@angular\cli\bin\ng

> @angular/cli@9.0.6 postinstall C:\Users\jhernand\AppData\Roaming\npm\node_modules\@angular\cli
> node ./bin/postinstall/script.js

? Would you like to share anonymous usage data with the Angular Team at Google under
Google's Privacy Policy at https://policies.google.com/privacy? For more details and
how to change this setting, see http://angular.io/analytics. Yes

Thank you for sharing anonymous usage data. If you change your mind, the following
command will disable this feature entirely:

  ng analytics off

+ @angular/cli@9.0.6
added 260 packages from 205 contributors in 149.97s
```

NodeJS, TypeScript, Visual Studio code y Angular

Instalación  ANGULAR CLI

Tras ello validaremos la instalación con 'ng v':

```
C:\WINDOWS\system32>ng v

Angular CLI

Angular CLI: 9.0.6
Node: 12.16.1
OS: win32 x64

Angular:
...
Ivy Workspace:

Package                           Version
-----
@angular-devkit/architect         0.900.6
@angular-devkit/core              9.0.6
@angular-devkit/schematics        9.0.6
@schematics/angular              9.0.6
@schematics/update                0.900.6
rxjs                             6.5.3
```




03 TypeScript

¿Qué es TypeScript?

TypeScript viene a solucionar muchos de los problemas que trae consigo los grandes desarrollos basados en **JavaScript**.

3.

- TypeScript es un lenguaje que nos permite tener una orientación a objetos más limpia y potente en JavaScript, además añade un tipado fuerte y es el lenguaje que utilizamos para programar aplicaciones web con Angular que es uno de los frameworks más populares para desarrollar aplicaciones modernas y escalables en el lado del cliente.
- TypeScript es un lenguaje libre desarrollado por Microsoft y es un superconjunto de JavaScript que gracias a las ventajas que ofrece está siendo cada vez más utilizado.
- TypeScript es un lenguaje “compilado” → nosotros escribimos código TypeScript y el compilador (transpilador) lo traduce a código JavaScript que el navegador podrá interpretar.

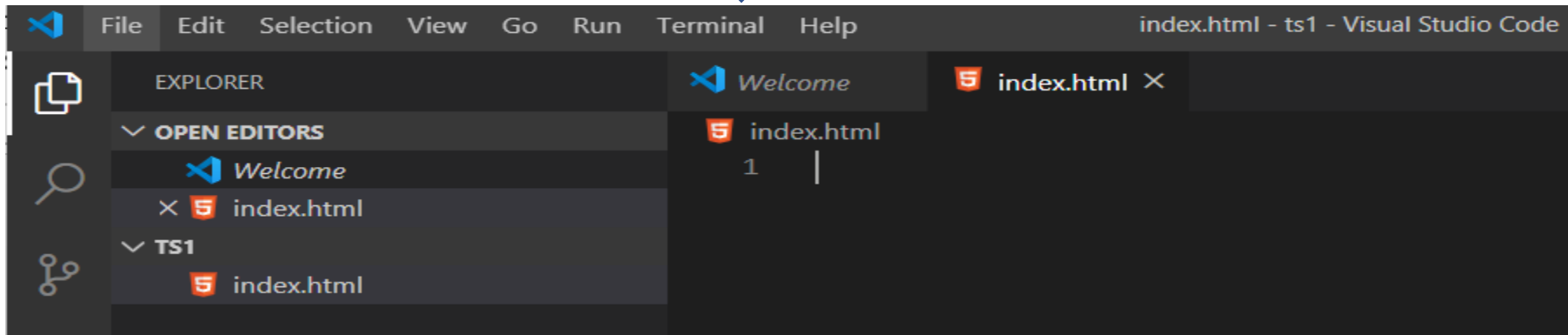
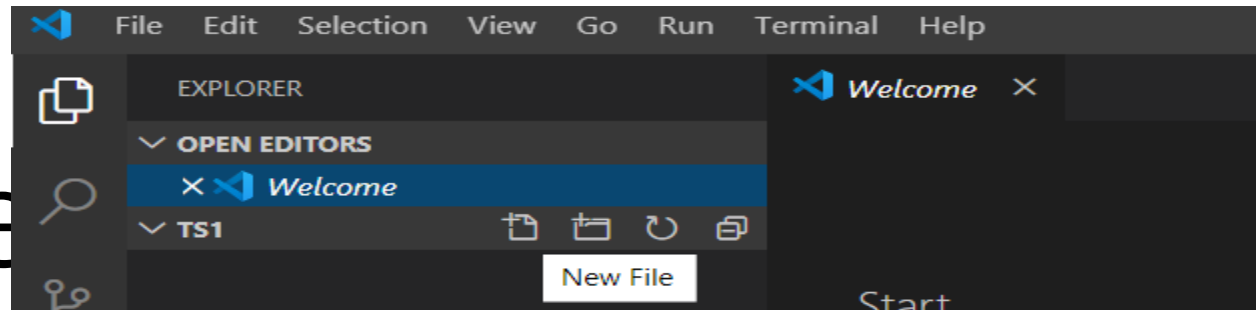
Transpilar → convertir código TypeScript en código JavaScript

Mi primer 'Hola mundo' con TypeScript

Primeramente vamos a crear un directorio de trabajo: `c:\dev\workspace\username\ts1`

Tras ello crearemos un fichero el fichero **'index.html'**

3. Type



Mi primer 'Hola mundo' con TypeScript

El contenido inicial que le daremos será el siguiente:

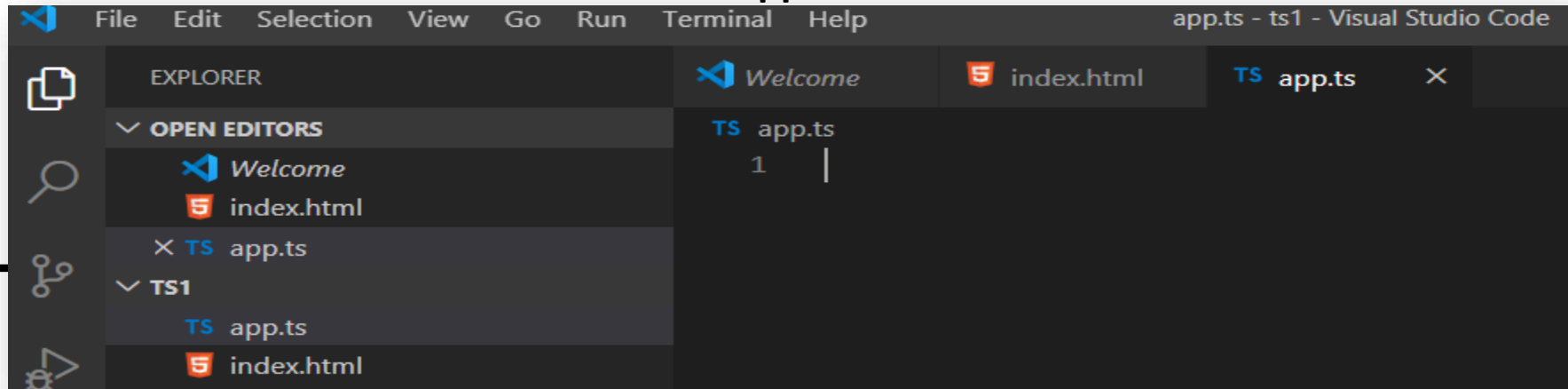
3. Type

```
5 index.html > ...  
1  <html>  
2    <head>  
3      <meta charset="UTF-8">  
4      <title>Mi primer TypeScript</title>  
5    </head>  
6    <body>  
7      <script src="app.js"></script>  
8    </body>  
9  </html>
```

Mi primer 'Hola mundo' con TypeScript

Tras ello crearemos un archivo llamado **'app.ts'**:

3.



El contenido será:

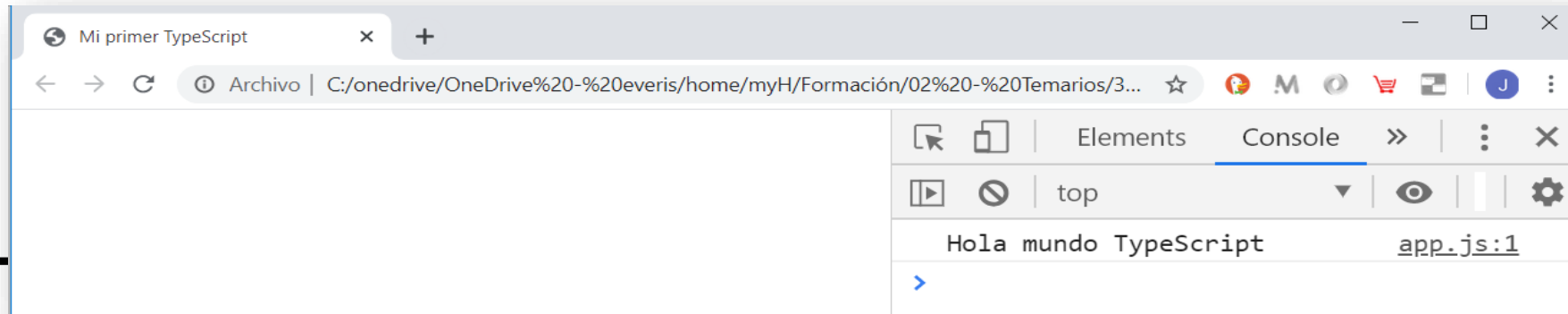
```
TS app.ts
1 console.log("Hola mundo TypeScript");|
```

Tras ello habilitamos el terminal de comandos desde: **menú 'View – Terminal'**

Una vez abierto escribiremos: **tsc app.ts** para que se nos genere automáticamente el archivo **app.js**

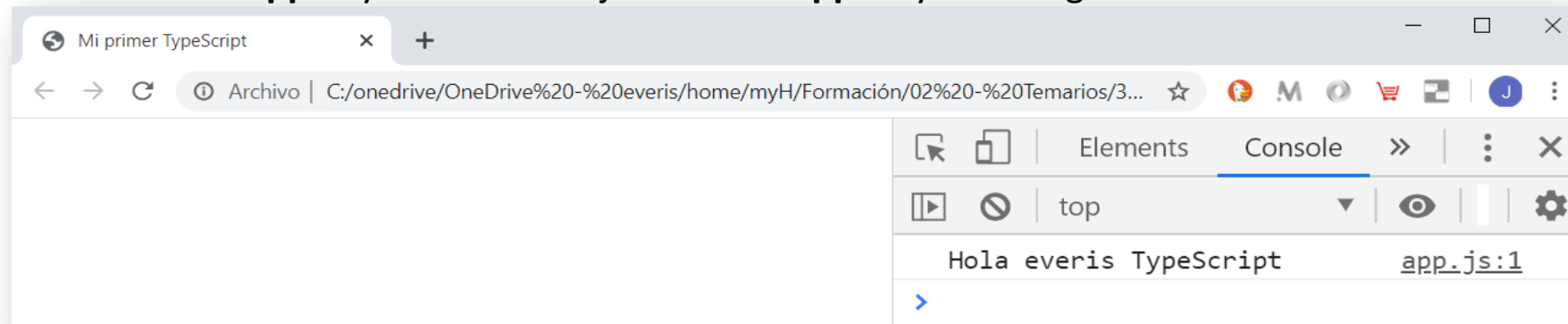
Mi primer 'Hola mundo' con TypeScript

Ahora si abrimos el fichero **index.html** con la opción de consola activada, veremos:



3. -typescript

Si queremos cambiar por ejemplo el mensaje a '**Hola everis TypeScript**', necesitamos cambiarlo en el fichero **app.ts** y volvemos a ejecutar '**tsc app.ts**' y al recargar el html veremos:



Mi primer 'Hola mundo' con TypeScript

Para no tener que estar ejecutando 'tsc' por cada cambio que hagamos, vamos a dejar ahora un escuchador abierto para que coja cualquier cambio que hagamos. Para ello ejecutaremos en consola '**tsc -w app.ts**'

Y cualquier cambio que hagamos en nuestro fichero app.ts, nos generará automáticamente la versión correspondiente del app.js

3. TypeScript

```
[13:30:10] File change detected. Starting incremental compilation...
```

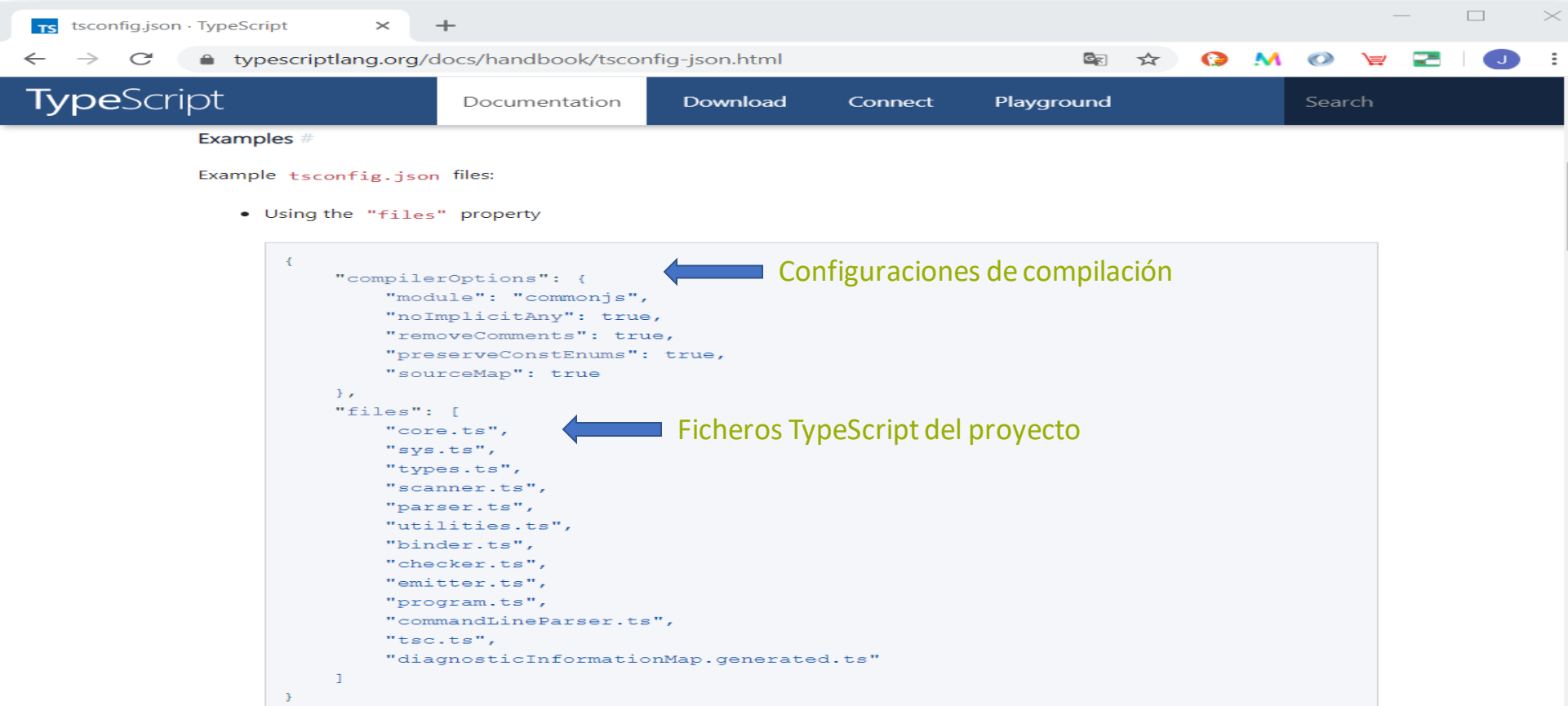
```
[13:30:10] Found 0 errors. Watching for file changes.
```


El fichero tsconfig.json

Vamos a ver un fichero especial: **tsconfig.json**.

Es el fichero de configuración de un proyecto **TypeScript**.

3.



The screenshot shows the TypeScript documentation page for `tsconfig.json`. The page title is "tsconfig.json · TypeScript". The URL is `typescriptlang.org/docs/handbook/tsconfig-json.html`. The navigation bar includes "TypeScript", "Documentation", "Download", "Connect", "Playground", and "Search". The main content area is titled "Examples #" and shows an example of `tsconfig.json` files. A list item "Using the 'files' property" is shown. Below it, a JSON configuration is displayed with two blue arrows pointing to specific parts: one to the `compilerOptions` object and another to the `files` array.

```
{
  "compilerOptions": {
    "module": "commonjs",
    "noImplicitAny": true,
    "removeComments": true,
    "preserveConstEnums": true,
    "sourceMap": true
  },
  "files": [
    "core.ts",
    "sys.ts",
    "types.ts",
    "scanner.ts",
    "parser.ts",
    "utilities.ts",
    "binder.ts",
    "checker.ts",
    "emitter.ts",
    "program.ts",
    "commandLineParser.ts",
    "tsc.ts",
    "diagnosticInformationMap.generated.ts"
  ]
}
```

Configuraciones de compilación

Ficheros TypeScript del proyecto

El fichero tsconfig.json

Ahora vamos a generar nuestro **tsconfig.json** ejecutando en el terminal '**tsc --init**':

3.

```
tsconfig.json > ...
1 {
2   "compilerOptions": {
3     /* Basic Options */
4     // "incremental": true,           /* Enable incremental compilation */
5     "target": "es5",                /* Specify ECMAScript target version: 'ES3' (default), 'ES5', 'ES2015', 'ES2016', 'ES2017',
6     "module": "commonjs",           /* Specify module code generation: 'none', 'commonjs', 'amd', 'system', 'umd', 'es2015', 'e
7     // "lib": [],                    /* Specify library files to be included in the compilation. */
8     // "allowJs": true,              /* Allow javascript files to be compiled. */
9     // "checkJs": true,              /* Report errors in .js files. */
10    // "jsx": "preserve",             /* Specify JSX code generation: 'preserve', 'react-native', or 'react'. */
11    // "declaration": true,           /* Generates corresponding '.d.ts' file. */
12    // "declarationMap": true,        /* Generates a sourcemap for each corresponding '.d.ts' file. */
13    // "sourceMap": true,             /* Generates corresponding '.map' file. */
14    // "outFile": "./",               /* Concatenate and emit output to single file. */
15    // "outDir": "./",               /* Redirect output structure to the directory. */
16    // "rootDir": "./",              /* Specify the root directory of input files. Use to control the output directory structure
17    // "composite": true,             /* Enable project compilation */
18    // "tsBuildInfoFile": "./",       /* Specify file to store incremental compilation information */
19    // "removeComments": true,        /* Do not emit comments to output. */
20    // "noEmit": true,                /* Do not emit outputs. */
21    // "importHelpers": true,         /* Import emit helpers from 'tslib'. */
22    // "downlevelIteration": true,    /* Provide full support for iterables in 'for-of', spread, and destructuring when targeting
23    // "isolatedModules": true,        /* Transpile each file as a separate module (similar to 'ts.transpileModule'). */
24  }
```

3. Type

let / var / const

En JavaScript podemos declarar las variables con **var** y con **let**. El primero tiene ámbito de bloque, el segundo no.

var

```
var foo = 123;  
if (true) {  
    var foo = 456;  
}  
console.log(foo); // 456
```

let

```
let foo = 123;  
if (true) {  
    let foo = 456;  
}  
console.log(foo); // 123
```

let / var /const

const nos va a permitir definir variables inmutables:

```
const foo = 123;  
foo = 456; // NO permitido
```

Las constantes también admiten objetos literales como por ejemplo:

3. Ty

```
const foo = { bar: 123 };  
foo = { bar: 456 }; // ERROR no se permite la modificación de objeto
```

Pero sí se puede modificar el contenido de las variables que contiene el objeto:

```
const foo = { bar: 123 };  
foo.bar = 456; // Permitido  
console.log(foo); // { bar: 456 }
```

3.

Tipos de datos primitivos

Boolean

true o false

```
let isDone: boolean = false;
```

Number

Datos numéricos

```
let decimal: number = 6;
```

```
let hex: number = 0xf00d;
```

```
let binary: number = 0b1010;
```

```
let octal: number = 0o744
```

String

Cadenas de caracteres y/o textos

```
let color: string = "blue"; //  
color = 'red';
```

También se pueden utilizar *"Templates"* plantillas para concatenar strings como por ejemplo:

```
let fullName: string = `Bob Bobbington`;  
let age: number = 37;  
let sentence: string = `Hello, my name is ${ fullName }. I'll be ${ age + 1 } years old next month.`
```

Para poder utilizar esta sintaxis los string deben estar contenidos entre ```.

Este tipo de sintaxis es el equivalente a:

```
let sentence: string = "Hello, my name is " + fullName + ". I'll be " + (age + 1)  
+ " years old next month."
```

Tipos de datos primitivos

Arrays, sino se les especifica tipo son **ANY**

ANY quiere decir que admite 'cualquier tipo' de dato

```
let list: number[] = [1, 2, 3];
```

Con esta sintaxis se puede especificar qué tipo de datos debe haber en el array

```
let list: Array<number> = [1, 2, 3];
```

3.

Undefined

Es cuando un objeto o variable existe pero no tiene un valor. Si nuestro código interactúa con alguna API podemos recibir null como respuesta, para evaluar esas respuestas es mejor utilizar `==` en vez de `===`

```
// ----- ejemplo.ts -----  
console.log(undefined == undefined); // true  
console.log(null == undefined); // true  
console.log(0 == undefined); // false  
console.log('' == undefined); // false  
console.log(false == undefined); // false
```

3. T

Tipos de datos primitivos

Any

Puede ser cualquier tipo de objeto de javascript

```
let notSure: any = 4;  
notSure = "maybe a string instead"; // typeof = string  
notSure = false; // typeof = boolean
```

```
let notSure: any = 4;  
notSure.toFixed(); // OK, toFixed existe, pero no es comprobado por el compilador  
let prettySure: Object = 4;  
prettySure.toFixed(); // Error: La propiedad 'toFixed' no existe en un 'Object'.
```

```
let list: any[] = [1, true, "free"];  
list[1] = 100;
```


3.

For in

For in es una característica que ya tenía javascript ,y no ha sido mejorada en TypeScript, mediante la cual puedes acceder y recorrer objetos y arrays y obtener tanto los índices

For in accediendo al valor de una variable dentro de un objeto:

TypeScript

```
let list = {a: 1, b: 2, c: 3};

for (let i in list) {
  console.log (i); // a, b, c
}
```

3.

For Of

For of es una característica nueva de ES6 con la **cual puedes acceder y recorrer arrays y strings obteniendo su valor**, es decir, no puede recorrer objetos. Aunque se podrían recorrer objetos en el caso de que estos fueran creados por clases que implementen `Symbol.iterator`. `for ... of` también tiene un peor rendimiento en comparación con el `for...in` ya que al compilarlo a JS crea más variables y hace más comprobaciones.

For of accediendo al valor de una variable dentro de un array:

TypeScript

```
let list = ["a", "b", "c"];

for (let b of list) {
    console.log(b); // a, b, c
}
```



```
let list = {a: 1, b: 2, c: 3};

for (let i in list) {
    console.log (i); // a, b, c
}

let list2 = [1, 2, 3];

for (let i of list2) {
    console.log (i); // 1, 2, 3
}
```

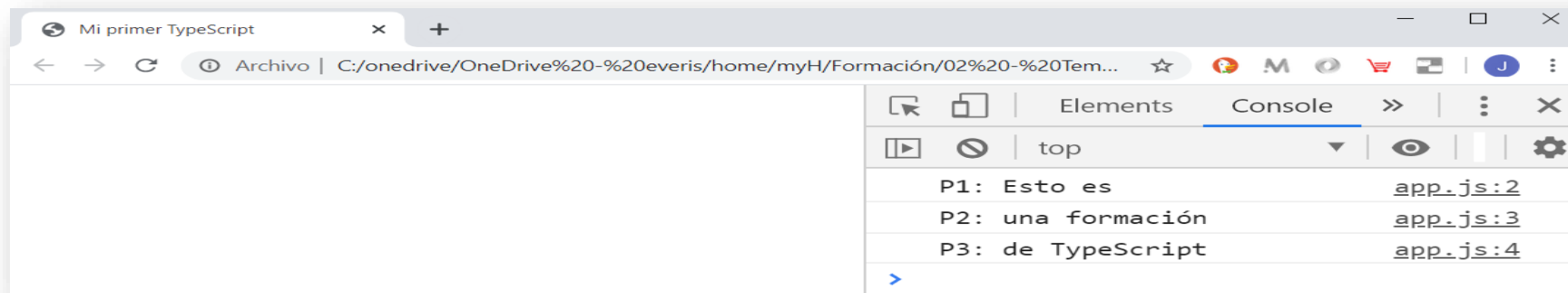
Parámetros obligatorios por defecto y opcionales

Vamos a empezar definiéndonos una función en el app.ts:

3. TS

```
index.html TS app.ts X
TS app.ts > ...
1 function testFunTypeScript (param1: String, param2: String, param3: String) {
2     console.log (" P1: " + param1);
3     console.log (" P2: " + param2);
4     console.log (" P3: " + param3);
5 }
6
7 testFunTypeScript ("Esto es", "una formación", "de TypeScript");
```

Al actualizar el fichero 'index.html':



Parámetros obligatorios por defecto y opcionales

Si intentamos llamar a nuestra función pasándole un único parámetro nos dará error:

```
testFunTypeScript ("Esto es");
```



```
app.ts:9:1 - error TS2554: Expected 3 arguments, but got 1.  
9 testFunTypeScript ("Esto es");  
  ~~~~~  
  
app.ts:1:45  
1 function testFunTypeScript (param1: String, param2: String, param3: String) {  
  ~~~~~  
  An argument for 'param2' was not provided.  
  
[14:00:40] Found 1 error. Watching for file changes.
```

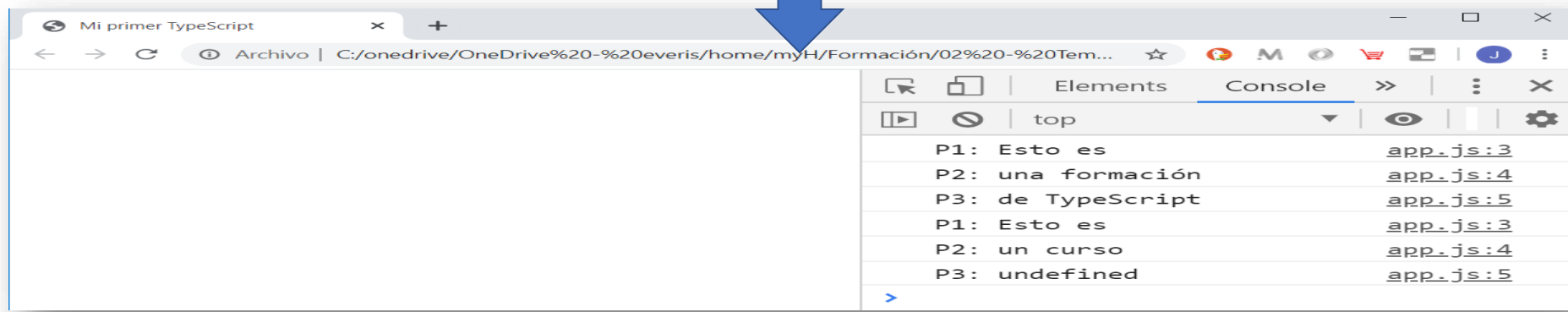
3. Types

Parámetros obligatorios por defecto y opcionales

Ahora vamos a hacer la misma llamada pero antes vamos a modificar nuestra función indicando un valor por defecto para el segundo parámetro (en caso de no recibirlo) y marcando el tercer parámetro como opcional:

3.

```
function testFunTypeScript (param1: String, param2: String = "un curso", param3?: String) {  
    console.log (" P1: " + param1);  
    console.log (" P2: " + param2);  
    console.log (" P3: " + param3);  
}  
  
testFunTypeScript ("Esto es", "una formación", "de TypeScript");  
  
testFunTypeScript ("Esto es");
```



Arrow functions



3. TypeScript

```
function testSumaDosNumeros(param1: number, param2: number) {  
    return param1 + param2;  
}  
  
console.log("1. Suma de 5 y 55 = "+ testSumaDosNumeros (5,55) ); // 60
```



```
var testSumaDosNumeros_v2 = (param1: number, param2: number) => param1 + param2;  
  
console.log("2. Suma de 5 y 55 = "+ testSumaDosNumeros_v2 (5,55) ); // 60
```

Declaración de promesas en TypeScript

Desde **TypeScript** vamos a poder definir las **promesas** que tan imprescindibles se han vuelto en el contexto **JavaScript** con funciones dependientes de la ejecución de otras.

Lo único que nos cambiará respecto **JavaScript** tradicional será la nomenclatura en la definición de la función interna de la **promise**:

3. Typ

```
let mipromesa = new Promise(function (resolve: any, reject: any) {  
    resolve();  
});  
  
mipromesa.then(function () {  
    console.log('la operacion de la promesa a finalizado con exito');  
}, function () {  
    console.log('la operacion de la promesa a finalizado con error');  
});
```


Programación Orientada a Objetos en TypeScript

Modificadores de acceso:

Private: Cuando un método o atributo (variable) es declarada como private, su uso queda restringido al interior de la misma clase, no siendo visible para el resto.

3. **Protected:** Un método o atributo definido como protected es visible para las clases que se encuentren en el mismo paquete y para cualquier subclase de esta aunque este en otro paquete. Este modificador es utilizado normalmente para Herencias, así que lo estudiaremos más a fondo cuando lleguemos a las Herencias.

Public: El modificador public ofrece la máxima visibilidad, una variable, método o clase con modificador public será visible desde cualquier clase, aunque estén en paquetes distintos

Por defecto las propiedades como modificador de acceso son publicas en typescript

3.

Programación Orientada a Objetos en TypeScript

```
1  class Programa{
2      public nombre: string;
3      public version: number;
4
5      getNombre(){
6          return this.nombre;
7      }
8
9      setNombre(nombre:string){
10         this.nombre = nombre;
11     }
12
13     getVersion(){
14         return this.version;
15     }
16
17     setVersion(version:number){
18         this.version = version;
19     }
20 }
```

← Definición de clase

← Métodos getters/setters

Herencia

↓

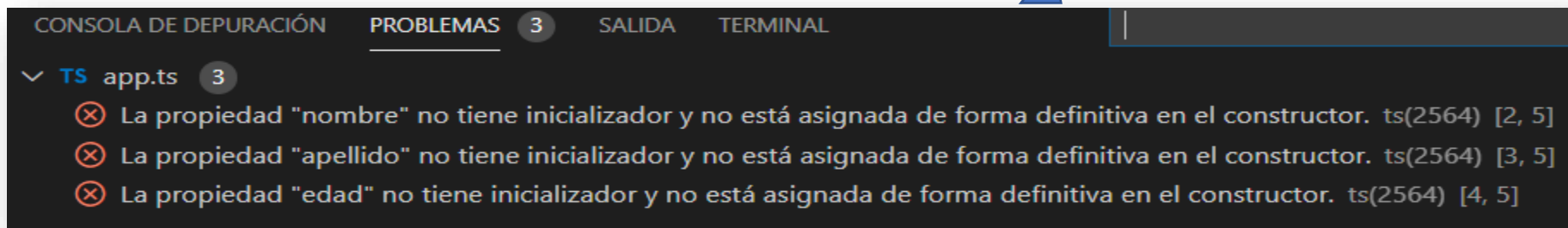
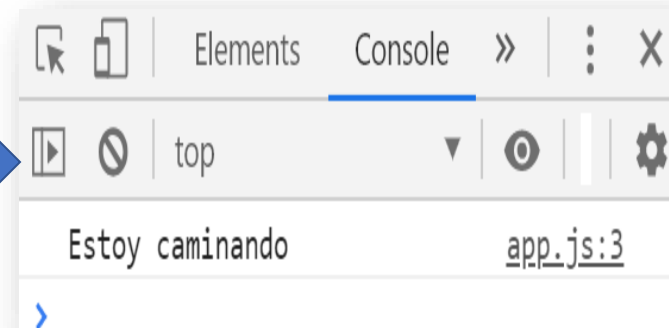
```
21
22 class EditorVideo extends Programa{
23     public timeline:number;
```

Programación Orientada a Objetos en TypeScript

Actividad: Vamos a escribir el siguiente código en el **app.ts** y lo probamos:

3.

```
TS app.ts > ...
1  class Persona {
2      nombre: string;
3      apellido: string;
4      edad: number;
5
6      caminar = () => console.log("Estoy caminando");
7  }
8
9  let objetoPersona = new Persona();
10 objetoPersona.caminar();
```



3. Typ

Programación Orientada a Objetos en TypeScript

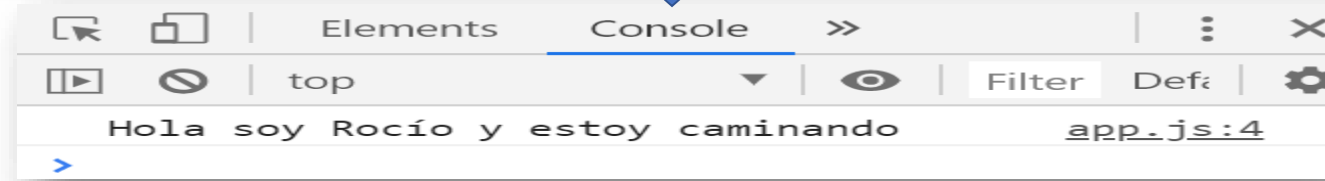
Actividad: Versionaremos nuestra clase de **app.ts** y la volvemos a probar:

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

  constructor(name:string, surname:string, age:number) {
    this.nombre = name;
    this.apellido = surname;
    this.edad = age;
  }

  caminar = () => console.log("Hola soy",this.nombre, "y estoy caminando");
}

let objetoPersona = new Persona("Rocío", "De la O", 23);
objetoPersona.caminar();
```



Nota: para este ejemplo ignoraremos el error generado en consola.

```
PS C:\Dev\workspace\jhernand\ts1> tsc Persona.ts
PS C:\Dev\workspace\jhernand\ts1> tsc app.ts
app.ts:3:25 - error TS2304: Cannot find name 'Persona'.

3 let objetoPersona = new Persona("Rocío", "De la O", 23);
                          ~~~~~~

Found 1 error.
```



Hola soy Rocío y estoy caminando

Programación Orientada a Objetos en TypeScript

Actividad: Ahora vamos a llevarnos la clase Persona a un nuevo fichero **Persona.ts**

3.

```
TS Persona.ts > Persona
1 class Persona {
2     nombre: string;
3     apellido: string;
4     edad: number;
5
6     constructor(name:string, surname:string, age:number) {
7         this.nombre = name;
8         this.apellido = surname;
9         this.edad = age;
10    }
11
12    caminar = () => console.log("Hola soy",this.nombre, "y estoy caminando");
13 }
```

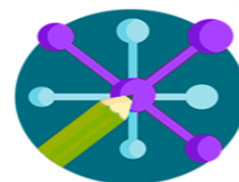
```
TS app.ts > ...
1 let objetoPersona = new Persona("Rocío", "De la O", 23);
2 objetoPersona.caminar();
```

index.html

```
5 index.html > ...
1 <html>
2     <head>
3         <meta charset="UTF-8">
4         <title>Mi primer TypeScript</title>
5     </head>
6     <body>
7         <script src="Persona.js"></script>
8         <script src="app.js"></script>
9     </body>
10 </html>
```



ever
FUTURE



04 Módulos y Decoradores

4.

Módulos

TypeScript nos va a permitir la facilidad de **exportar** nuestras clases para ser **importadas** y usadas en cualquier otra parte. Esto son lo que llamamos **módulos**. Para ello en línea con el ejemplo anterior:

```
TS Persona.ts > ...
1  export class Persona {
2      nombre: string;
3      apellido: string;
4      edad: number;
5
6      constructor(name:string, surname:string, age:number) {
7          this.nombre = name;
8          this.apellido = surname;
9          this.edad = age;
10     }
11
12     caminar = () => console.log("Hola soy",this.nombre, "y estoy caminando");
13 }
```

```
TS app.ts > ...
1  import {Persona} from "./Persona";
2
3  let objetoPersona = new Persona("Rocío", "De la O", 23);
4  objetoPersona.caminar();
```

```
index.html > ...
1  <html>
2      <head>
3          <meta charset="UTF-8">
4          <title>Mi primer TypeScript</title>
5      </head>
6      <body>
7          <script src="app.js"></script>
8      </body>
9  </html>
```

Decoradores

Básicamente es una implementación de un patrón de diseño de software que en sí sirve para extender una función mediante otra función, pero sin tocar aquella original, que se está extendiendo. El decorador recibe una función como argumento (aquella que se quiere decorar) y devuelve esa función con alguna funcionalidad adicional.

4. Las funciones decoradoras comienzan por una "@" y a continuación tienen un nombre. Ese nombre es el de aquello que queremos decorar, que ya tiene que existir previamente. Podríamos decorar una función, una propiedad de una clase, una clase, etc.

Podemos considerar un decorador como la forma de aumentar las funcionalidades a ciertos tipos.

Decoradores

Se crean con el objetivo de mejorar, ampliar, validar, etc lo que ya existe.

Los **decoradores** pueden ser de: **clases**, **propiedades**, **parámetros** y **métodos**.

Estamos asignándole
dinámicamente el
método 'saludo'

```
function Bienvenida(target: Function): void {  
    target.prototype.saludo = function(): void {  
        console.log('¡Hola!');  
    }  
}  
  
@Bienvenida  
class Saludar {  
    constructor() {  
        // Implementación va aquí...  
    }  
}  
  
var miSaludo = new Saludar();  
miSaludo.saludo(); // Consola mostrará '¡Hola!'
```