UD 1. Introducción a TypeScript

1. Que es TypeScript

TypeScript es un **lenguaje de programación desarrollado por Microsoft** que se basa en **JavaScript**, pero agrega herramientas y características que hacen que el código sea **más seguro, ordenado y fácil de mantener**.

En pocas palabras, **TypeScript es una mejora o "superconjunto" de JavaScript**. Esto significa que todo el código válido en JavaScript también es válido en TypeScript, pero además puedes usar nuevas funciones que JavaScript no tiene de forma nativa.

Decir que TypeScript es un **superconjunto** significa que:

- Cualquier código JavaScript válido puede ejecutarse en TypeScript.
- Además, TypeScript añade nuevas funcionalidades, como **tipado estático**, **interfaces**, **clases** y **módulos**.

Código JavaScript

```
function saludar(nombre) {
  return "Hola " + nombre;
}
```

Código TypeScript

```
function saludar(nombre: string): string {
  return `Hola ${nombre}`;
}
```

Aquí, indicamos que nombre debe ser una cadena (string) y que la función devolverá también una cadena (string).

Si tratamos de pasarle un número o un valor incorrecto, TypeScript mostrará un **error antes de ejecutar el programa**, ayudando a prevenir fallos.

2. VENTAJAS FRENTE A JAVASCRIPT

2.1. Tipado estático

La diferencia más importante entre TypeScript y JavaScript es el **tipado estático**. En JavaScript, el tipo de dato de una variable puede cambiar en cualquier momento, lo que puede causar errores difíciles de detectar.

En cambio, TypeScript permite **definir los tipos de datos** desde el principio y verificar que se usen correctamente.

Código JavaScript (sin tipos)

```
let edad = 25;
edad = "veinticinco"; // Error lógico, pero JavaScript lo permite
```

Código TypeScript (con tipos)

```
let edad = 25;
edad = "veinticinco"; // Error detectado en tiempo de compilación
```

Esto ayuda a **prevenir errores antes de ejecutar el programa**, algo muy útil cuando el proyecto crece.

2.2. Detección temprana de errores

Gracias al compilador de TypeScript (tsc), los errores se detectan antes de ejecutar el código, en la fase de compilación.

Esto permite al desarrollador corregir errores de tipo, variables mal escritas o funciones mal llamadas sin tener que ejecutar la aplicación.

```
function sumar(a: number, b: number): number {
  return a + b;
}

sumar(5, "3"); // ❌ Error: se esperaba un número, no una cadena
```

En JavaScript, este error no se detectaría hasta que el programa intente ejecutarse, lo que podría causar fallos en tiempo real.

2.3. Mejora la mantenibilidad del código

El tipado, las interfaces y las clases hacen que el código TypeScript sea **más estructurado** y fácil de entender.

Esto es especialmente importante cuando varias personas trabajan en el mismo proyecto o cuando se revisa código después de mucho tiempo.

```
interface Persona {
  nombre: string;
  edad: number;
}

function mostrarDatos(persona: Persona) {
  console.log(` ${persona.nombre} tiene ${persona.edad} años.`);
}
```

2.4. Programación orientada a objetos (POO)

TypeScript permite usar **clases, interfaces, herencia y modificadores de acceso** (public, private, protected), lo que facilita aplicar principios de **programación orientada a objetos**, algo que en JavaScript es más limitado.

```
class Animal {
 constructor(public nombre: string) {}

 hablar(): void {
  console.log(` ${this.nombre} hace un sonido.`);
 }
}

class Perro extends Animal {
 hablar(): void {
  console.log(` ${this.nombre} ladra.`);
 }
}

const perro = new Perro("Firulais");
perro.hablar(); // Firulais ladra.
```

2.4. Compatibilidad total con JavaScript

Todo el código TypeScript se **transpila (convierte)** a JavaScript estándar.

Esto significa que:

- No necesitas cambiar tu entorno de ejecución.
- Puedes usar cualquier librería o framework basado en JavaScript.
- Puedes incorporar TypeScript poco a poco en proyectos JavaScript existentes.

```
tsc app.ts
```

Esto generará un archivo app.js que puede ejecutarse normalmente en el navegador o en Node.js.

3. INSTALACIÓN Y CONFIGURACIÓN DEL COMPILADOR

3.1. Instalación del compilador TypeScript

Una vez que tengas Node.js, puedes instalar TypeScript de dos maneras: **globalmente** o **localmente** dentro de un proyecto.

A) Instalación global => Permite usar el comando tsc en cualquier carpeta de tu computadora

```
npm install -g typescript
```

B) Instalación local => Si prefieres usar TypeScript solo dentro de un proyecto específico

```
npm init -y

npm install typescript --save-dev
```

Ahora tendrás TypeScript instalado **solo para ese proyecto**.

3.2. Compilación básica de archivos TypeScript

Una vez instalado, puedes crear tu primer archivo TypeScript:

```
let mensaje: string = "¡Hola TypeScript!";
console.log(mensaje);
```

Para compilarlo, usa el comando:

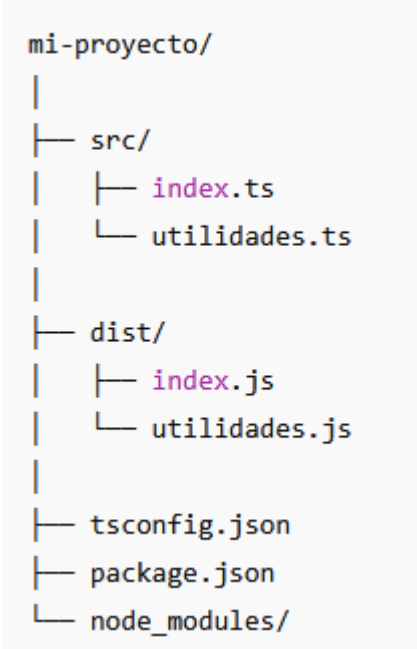
```
tsc hola.ts

node hola.js
```

Esto creará un archivo tsconfig.json en tu proyecto. Un ejemplo básico puede verse así:

```
{
  "compilerOptions": {
    "target": "ES6",        // Versión de JavaScript a generar
    "module": "commonjs",    // Sistema de módulos
    "rootDir": "./src",      // Carpeta donde están los archivos .ts
    "outDir": "./dist",      // Carpeta donde se guardarán los archivos .js compilados
    "strict": true,         // Activa comprobaciones estrictas de tipos
    "esModuleInterop": true   // Mejora compatibilidad con módulos JS
  }
}
```

3.3. Estructura básica de un proyecto TypeScript



src/: Carpeta con el código fuente TypeScript.

dist/: Carpeta con el código JavaScript compilado.

tsconfig.json: Configuración del compilador.

4. INTEGRACIÓN CON NODE.JS Y FRAMEWORKS

4.1. Integración con Node.js

Node.js es una plataforma que permite ejecutar JavaScript en el servidor. Con TypeScript, podemos aprovechar todas sus ventajas (tipado, clases, módulos, etc.) para crear aplicaciones backend seguras y escalables.

- Crea una carpeta para tu proyecto

```
mkdir proyecto-node
cd proyecto-node
```

- Inicializa un proyecto con Node.js

```
npm init -y
```

- Instala TypeScript y los tipos de Node

```
npm install typescript @types/node --save-dev
```

- Crea el archivo de configuración

```
npx tsc --init
```

- Configura el archivo tsconfig.json

```
{
  "compilerOptions": {
    "target": "ES6",
    "module": "commonjs",
    "outDir": "./dist",
    "rootDir": "./src",
    "strict": true
  }
}
```

- Ejemplo de aplicación simple en Node.js con TypeScript

```
import http from "http";

const servidor = http.createServer((req, res) => {
  res.writeHead(200, { "Content-Type": "text/plain" });
  res.end("Servidor Node.js con TypeScript funcionando");
});

servidor.listen(3000, () => {
  console.log("Servidor en ejecución en http://localhost:3000");
});
```

- Compila y ejecuta

```
npx tsc
node dist/app.js
```

4.2. Integración con frameworks front-end

A) Integración con Angular

Angular utiliza TypeScript **de forma nativa**.

De hecho, todo el código Angular está basado en TypeScript, lo que permite aprovechar características como clases, decoradores y tipado fuerte.

- Ejemplo de instalación

```
npm install -g @angular/cli
ng new mi-proyecto-angular
```

- Ejemplo de componente Angular (app.component.ts)

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

@Component({
 selector: 'app-root',
 template: `<h1>Hola desde Angular con TypeScript!</h1>`
})
export class AppComponent {
 titulo: string = "Mi primera app en Angular";
}
```

B) Integración con React

React no requiere TypeScript por defecto, pero puede configurarse fácilmente. Usar TypeScript en React ayuda a tipar las propiedades (props) y el estado de los componentes, reduciendo errores.

- Creación de un proyecto React con TypeScript:

```
npx create-react-app mi-app --template typescript
```

- Ejemplo de componente en React con TypeScript:

```
type Props = {
 nombre: string;
};

function Saludo({ nombre }: Props) {
 return <h2>Hola, {nombre} </h2>;
}
```

C) Integración con Vue.js

Vue también puede usar TypeScript, especialmente a partir de **Vue 3**, que tiene soporte oficial.

- Para crear un proyecto Vue con TypeScript:

```
npm init vue@latest
```

- Ejemplo de componente Vue con TypeScript (App.vue)

```
<script lang="ts">
import { defineComponent } from "vue";

export default defineComponent({
```

```
  data() {
    return {
      mensaje: "Hola desde Vue con TypeScript!"
    };
  }
});
</script>

<template>
  <h1>{{ mensaje }}</h1>
</template>
```