TYPESCRIPT

Debido al crecimiento en el uso de JavaScript, se comenzó a desarrollar nuevas tecnologías y frameworks que permitieran optimizar el desarrollo de aplicaciones, sobre todo para aquellas cuyo volumen de código es muy grande, agilizando así el proceso de desarrollo y facilitando el posteriori mantenimiento de los sistemas.

En el caso de otros lenguajes de programación es más frecuente encontrar grandes entornos de desarrollo utilizados para la implementación de proyectos, este tipo de entornos suelen incorporar diferentes utilidades, que permiten entre otras ventajas reducir el número de errores a la hora de programar. Pues bien, para el desarrollo con JavaScript esto no existía.

Gracias a la aparición de TypeScript, un lenguaje de programación de código abierto que incorpora nuevas funcionalidades a JavaScript, se ha conseguido simplificar el desarrollo de aplicaciones utilizando este lenguaje.

Estas funcionalidades se pueden incorporar a un entorno de desarrollo específico de proyectos implementados con JavaScript, evitando ciertos errores comunes que aparecen cuando se programa sobre este lenguaje. Algunos de los más habituales suelen ser que una variable no se haya definido previamente antes de ser utilizada. Antes de la aparición de TypeScript estos errores no eran visibles hasta que la aplicación no estaba siendo ejecutada.

TypeScript es JavaScript pero añadiendo tipos de variables y tipos de métodos, entre otros.

El proceso de instalación de TypeScript es sencillo, desde la línea de comandos del terminal se introducen la siguiente instrucción:

npm --install -g typescript

Una de las principales ventajas que aporta el uso de TypeScript es que permite utilizar las nuevas funciones, como por ejemplo imports o exports.

1. Instalación y Configuración:

TypeScript se instala globalmente para que puedas acceder al compilador tsc desde cualquier ubicación en tu sistema. Utiliza -g para que sea una instalación global de todo el sistema.

npm install -g typescript

2. Creación del Archivo tsconfig.json:

npx tsc --init

Crea el archivo tsconfig.json con el siguiente contenido:

```
{
  "compilerOptions": {
    "target": "es5",
```

```
    "module": "commonjs",
    "strict": true
  },
  "include": ["src/**/*.ts"],
  "exclude": ["node_modules"]
}
```

Ejercicio 1: Crea un primer archivo .ts que devuelve el mensaje HolaMundo.

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

const mensaje = saludar("Mundo");
console.log(mensaje);
```

2. Tipos Básicos en TypeScript:

En TypeScript, hay varios tipos de elementos HTML representados por las interfaces del DOM.

1. **HTMLInputElement:** Representa un elemento de entrada HTML, como un campo de texto (<input type="text">), un botón de radio (<input type="radio">), un botón de checkbox (<input type="checkbox">), entre otros.

```
// Obtener un input de texto por su ID
const inputText: HTMLInputElement = document.getElementById('input-text') as HTMLInputElement;

// Acceder al valor del input
const valorInputText: string = inputText.value;
```

2. HTMLButtonElement: Representa un elemento de botón HTML (<button>).

```
// Obtener un botón por su ID
const boton: HTMLButtonElement = document.getElementById('boton') as HTMLButtonElement;

// Agregar un evento de clic al botón
boton.addEventListener('click', () => {
    console.log('Botón clickeado');
});
```

3. HTMLFormElement: Representa un elemento de formulario HTML (<form>).

```
// Obtener un formulario por su ID
const formulario: HTMLFormElement = document.getElementById('formulario') as HTMLFormElement;
```

```
// Agregar un evento de envío al formulario
formulario.addEventListener('submit', (event) => {
    event.preventDefault(); // Prevenir el envío del formulario
    console.log('Formulario enviado');
});
```

4. HTMLSelectElement: Representa un elemento de lista desplegable HTML (<select>).

```
// Obtener un elemento select por su ID
const select: HTMLSelectElement = document.getElementById('select') as HTMLSelectElement;

// Obtener el valor seleccionado del select
const valorSeleccionado: string = select.value;
```

5. HTMLTextAreaElement: Representa un elemento de área de texto HTML (<textarea>).

```
// Obtener un textarea por su ID
const textarea: HTMLTextAreaElement = document.getElementById('textarea') as HTMLTextAreaElement;

// Obtener el valor del textarea
const valorTextarea: string = textarea.value;
```

6. HTMLDivElement: Representa un elemento de división HTML ( ), utilizado comúnmente para dividir y estructurar el contenido de la página.

```
// Obtener un div por su ID
const div: HTMLDivElement = document.getElementById('div') as HTMLDivElement;

// Cambiar el contenido del div
div.textContent = 'Contenido del div modificado';
```

a. Números, Cadenas y Booleanos:

Números:

```
let edad: number = 25;
let precio: number = 10.5;
```

Cadenas:

```
let nombre: string = "Juan";
let saludo: string = `Hola, ${nombre}!`;
```

Booleanos:

```
let activo: boolean = true;
let inactivo: boolean = false;
```

Ejercicio 2: Diseña un formulario que recoge el nombre de un usuario, y la edad y devuelve un saludo. Usando typescript.

b. Arrays y Tuplas:

Arrays:

En TypeScript, al igual que en JavaScript, puedes crear arrays de varias formas y trabajar con ellos de manera similar. Sin embargo, TypeScript agrega tipado estático a los arrays, lo que significa que puedes declarar el tipo de datos que contendrá el array.

1. Declaración de un array:

```
// Declarar un array de números
let numeros: number[] = [1, 2, 3, 4, 5];

// Declarar un array de strings
let palabras: string[] = ['hola', 'mundo'];
```

2. Acceso a elementos:

```
let frutas: string[] = ['manzana', 'banana', 'naranja'];

console.log(frutas[0]); // Output: 'manzana'
console.log(frutas.length); // Output: 3
```

3. Agregar elementos:

```
let colores: string[] = ['rojo', 'verde'];

colores.push('azul');
console.log(colores); // Output: ['rojo', 'verde', 'azul']
```

4. Iterar sobre un array:

```
let numeros: number[] = [1, 2, 3, 4, 5];

numeros.forEach(numero => {
    console.log(numero);
});
```

5. Tipos de datos mixtos:

```
let mixto: (string | number)[] = ['hola', 42, 'mundo', 7];

mixto.forEach(elemento => {
    console.log(typeof elemento);
});
```

Ejercicio 3. Diseña un gestor de tareas (simple) que reciba en una caja de formulario la nueva tarea a integrar.

Tuplas:

En TypeScript, una tupla es una estructura de datos que te permite expresar un array un número fijo de elementos, donde cada elemento puede tener un tipo diferente. A diferencia de los arrays, en las tuplas, el tipo de dato y la longitud del arreglo están definidos de antemano.

```
let coordenadas: [number, number] = [3, 4];

// Declarar una tupla con dos elementos: un string y un número
let tupla: [string, number] = ['hola', 42];
// Acceder a elementos de la tupla
console.log(tupla[0]); // Output: 'hola'
console.log(tupla[1]); // Output: 42

// Asignar nuevos valores a la tupla
tupla[0] = 'adiós';
tupla[1] = 100;

// Acceder a la longitud de la tupla
console.log(tupla.length); // Output: 2

// Iterar sobre una tupla
tupla.forEach(elemento => {
    console.log(elemento);
});
```

Ejercicio 4. Crear un formulario que permita a los usuarios ingresar información sobre sus películas favoritas, incluyendo el título, el año de lanzamiento y el género. Al enviar el formulario, mostrar la información ingresada en una lista debajo del formulario.