Clase 3: Typescript

Sitio:	Centro de E-Learning - UTN.BA	Imprimido por:	Nilo Crespi
Curso:	Curso de Backend Developer - Turno Noche	Día:	Friday, 23 de January de 2026, 10:15
Libro:	Clase 3: Typescript		

Descripción

Objetivos de la clase:

- Comprender las características principales de TypeScript
- Identificar las diferencias entre TypeScript y JavaScript
- Practicar la sintaxis básica de TS a través de ejemplos y ejercicios.

Tabla de contenidos

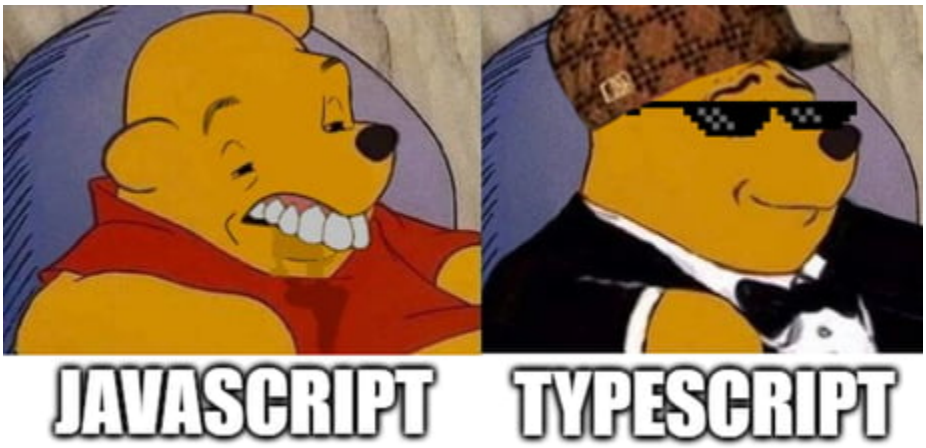
1. Introducción

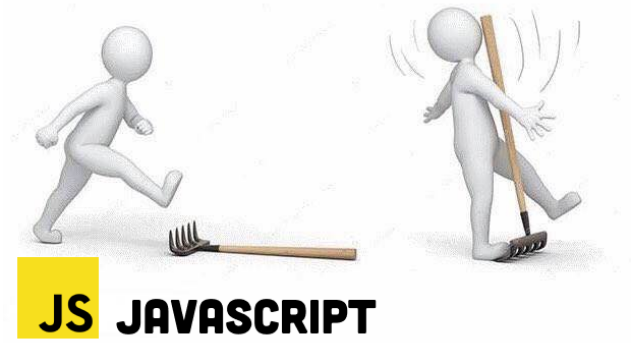
TypeScript es un lenguaje de programación de código abierto desarrollado por Microsoft que extiende las capacidades de JavaScript al agregar tipado estático, interfaces y características avanzadas de programación orientada a objetos. Su principal ventaja es que permite detectar errores en tiempo de compilación, mejorando la calidad y seguridad del código. Además, TypeScript es compatible con JavaScript, lo que significa que cualquier código TS puede ser convertido y ejecutado en entornos que admitan JavaScript, como navegadores y servidores.

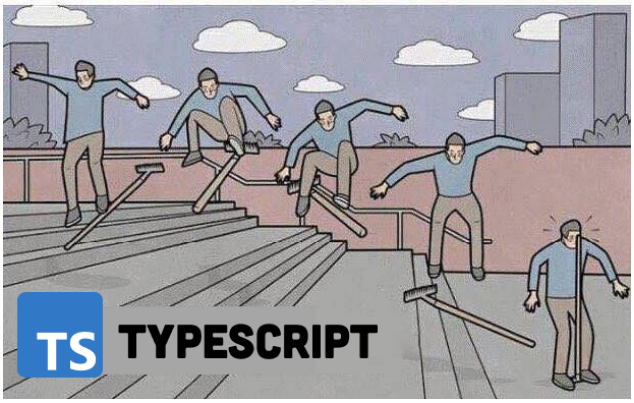
Aprender TypeScript es fundamental para desarrollar aplicaciones más escalables y mantenibles, ya que facilita la estructuración del código y permite el uso de características modernas de ECMAScript. En esta clase, exploraremos las diferencias clave entre TypeScript y JavaScript, su sintaxis básica y su aplicación práctica a través de ejemplos. Al finalizar, comprenderás cómo aprovechar sus ventajas para mejorar la calidad de tus proyectos de desarrollo.

2. ¿Qué es typescript?

Es un lenguaje de programación libre y de código abierto desarrollado por Microsoft que se basa en JavaScript y que añade características adicionales a este último, como el tipado estático, interfaces, clases, entre otras. TS se puede compilar en JavaScript, lo que significa que cualquier código TS puede ser ejecutado en un navegador web o en cualquier plataforma que soporte JavaScript.







¿Por qué es importante aprender TS?

- **Mejora la calidad del código: TypeScript es un lenguaje de programación tipado estáticamente, lo que significa que los tipos de datos se declaran en tiempo de compilación. Esto ayuda a evitar errores comunes en tiempo de ejecución, lo que conduce a una mayor calidad del código y a la eliminación de errores que de otro modo serían difíciles de detectar.**

- **Es compatible con JavaScript: es una extensión de JS, lo que significa que el código puede ser compilado en JS y ejecutado en cualquier plataforma que soporte JavaScript. Esto hace que TypeScript sea una herramienta muy versátil.**

- **Permite el uso de características modernas de ECMAScript: admite las características más modernas de ECMAScript, lo que significa que los desarrolladores pueden escribir código moderno y usar las características más recientes de JavaScript antes de que estén disponibles en todos los navegadores.**

Algunas de las diferencias principales entre TypeScript y JavaScript son:

- **Tipado estático: es un lenguaje tipado estáticamente, mientras que JavaScript es tipado dinámicamente. Esto significa que en TS, los tipos de datos se declaran en tiempo de compilación, mientras que en JS, los tipos de datos se declaran en tiempo de ejecución.**

- **Interfaces y clases: admite la creación de interfaces y clases, lo que permite crear una estructura más organizada y escalable.**
- **Inferencia de tipos: TS puede inferir tipos de datos, lo que significa que no se tiene que declarar explícitamente los tipos de datos en todas las variables y funciones.**
- **Validación en tiempo de compilación: realiza una validación de tipos en tiempo de compilación, lo que ayuda a detectar errores antes de la ejecución del código.**

TypeScript y JavaScript

3. Configuración TypeScript

Configuración TypeScript

1. Instalar TypeScript

npm install -g typescript

2. Crear un archivo tsconfig.json :

tsc --init

3. Compilar:

tsc app.ts ó npx tsc main.ts

4. Creamos HTML y linkeamos

DOCUMENTACIÓN; https://www.typescriptlang.org/es/download

tsconfig.json

```
{
  "compilerOptions": {
    "target": "es5",
    "module": "commonjs",
    "outDir": "dist",
    "strict": true
  }
}
```

4. Variables

En TS, las variables pueden tener tipos explícitos que se especifican en la declaración, por lo que el compilador puede detectar errores de tipo en tiempo de compilación. Por ejemplo, se puede definir una variable de tipo cadena de la siguiente manera:

```
let mensaje: string = "Hola, mundo!";
```

Las variables en TypeScript y cómo se definen utilizando las palabras clave "let" y "const".

- **let:** Se utiliza para declarar variables que pueden cambiar de valor. Por ejemplo:

```
let edad: number = 25; edad = 26;
```

- **const:** Se utiliza para declarar variables que no pueden cambiar de valor. Por ejemplo:

```
const pi: number = 3.14;
```

Tipo de datos

TypeScript incluye varios tipos de datos que se pueden utilizar para declarar variables con tipos explícitos. Algunos de los tipos de datos disponibles en TypeScript son:

```
let a: any = "apple";
```

- **number: Representa valores numéricos, incluyendo números enteros y números de punto flotante.**

  Ejemplo:
  let edad: number = 30;

- **string: Representa cadenas de texto.**

Ejemplo:
let nombre: string = "Juan";

- **boolean: Representa valores booleanos true o false.**

  Ejemplo:

  let activo: boolean = true;

- **any: Representa cualquier tipo de valor. Si no se especifica un tipo de datos para una variable, TypeScript inferirá automáticamente el tipo any.**

  Ejemplo:
  let variable: any = "Hola mundo";

- **void: Representa la ausencia de un valor. Se utiliza principalmente para indicar que una función no devuelve ningún valor.**

  Ejemplo:
  function saludar(): void { console.log("Hola mundo"); }

- **null y undefined: Representan valores nulos y no definidos, respectivamente.**

  Ejemplo:
  let valor: null = null; o let valor2: undefined = undefined;

- **object: Representa cualquier tipo de objeto, incluyendo objetos personalizados y objetos construidos a partir de clases.**

  Ejemplo:
  let persona: { nombre: string, edad: number } = { nombre: "Juan", edad: 30 };

- **array: Representa un arreglo de valores de un tipo de datos específico.**

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

- **tuple: Representa una secuencia de valores de tipos de datos específicos, donde cada valor puede tener un tipo diferente.**

  Ejemplo:

let tupla: [string, number] = ["Juan", 30];

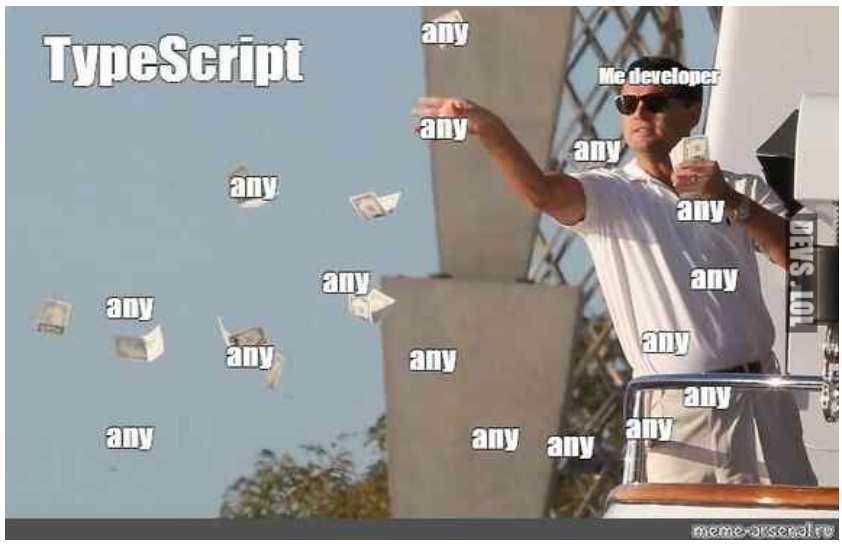
- **enum: Representa un conjunto de valores con nombre.**

  Ejemplo:
  enum DiasSemana { Lunes, Martes, Miercoles, Jueves, Viernes, Sabado, Domingo };

Al declarar variables con tipos explícitos, se puede mejorar la legibilidad y la seguridad del código al detectar errores en tiempo de compilación.

Tipo de datos: any

```
let a: any = "apple";
```



5. Funciones

En TS, una función se define de manera similar a como se hace en JavaScript, pero con la ventaja de que se puede especificar el tipo de los parámetros y del valor de retorno.

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

En este ejemplo, la función sum toma dos parámetros de tipo number y devuelve un valor de tipo number. La sintaxis para especificar los tipos de parámetros y valores de retorno es parametro: tipo.

También se puede especificar que un parámetro sea opcional o tenga un valor predeterminado utilizando el signo ? o = respectivamente.

Por ejemplo:

```
function greet(name: string, greeting?: string): string {
    if (greeting) {
    return `${greeting}, ${name}!`;
    } else {
    return `Hello, ${name}!`;
    }
}


console.log(greet("Harry"));
console.log(greet("Hermione", "Hola"));
```

En este ejemplo, la función greet tiene un parámetro opcional greeting y devuelve un valor de tipo string. Si se proporciona el parámetro greeting, se concatenará con el nombre para formar un saludo personalizado. Si no se proporciona, se utilizará un saludo predeterminado.

Además, permite definir funciones de flecha, que son una forma más compacta y concisa de definir funciones. Por ejemplo:

```
const multiply = (a: number, b: number): number => a * b;
```

let suma = (a , b) => { a + b]

En este ejemplo, se utiliza la sintaxis de flecha => para definir la función multiply que toma dos parámetros de tipo number y devuelve un valor de tipo number.

En TS se pueden definir funciones de la misma manera que en JavaScript, pero con la ventaja de que se pueden especificar los tipos de parámetros y valores de retorno.

6. Programación orientada a objetos (POO)

Es un paradigma de programación que se basa en el concepto de objetos, que contienen datos y métodos que manipulan esos datos.

Nociones básicas

- **Clases: En TypeScript, las clases se utilizan para definir objetos y sus comportamientos. Una clase es una plantilla para crear objetos que tienen propiedades y métodos comunes. Para definir una clase en TS, utilizamos la palabra clave class, seguida del nombre de la clase y sus propiedades y métodos.**
- **Objetos: Un objeto es una instancia de una clase, que tiene sus propias propiedades y métodos. En TS, se pueden crear objetos usando la palabra clave "new".**
- **Herencia: es un mecanismo que permite crear una nueva clase a partir de una clase existente, heredando sus propiedades y métodos. En TypeScript, se utiliza la palabra clave extends para crear una clase hija que hereda de una clase padre.**
- **Encapsulamiento: se refiere a la ocultación de la información interna de un objeto y la exposición sólo de la información necesaria para su uso externo. En TS, podemos utilizar los modificadores de acceso public, private y protected para definir la visibilidad de las propiedades y métodos de una clase.**

7. ¡A practicar!

Espacio de práctica

Definí las siguientes funciones:

- **que tome dos parámetros de tipo number y devuelva la suma de ambos.**
- **que tome un parámetro de tipo string y devuelva el número de caracteres que contiene.**
- **que tome un parámetro de tipo string y devuelva el mismo string pero con todas las letras en mayúscula.**
- **que tome un array de números y devuelva la suma de todos los elementos.**
- **que tome un array de objetos de tipo Persona y devuelva un nuevo arreglo con solo los nombres de cada persona.**

Interfaces y clases:

- **Tomar como referencia el ejemplo de la clase GestorDeUsuario que maneja usuarios, y utilizar la interfaz Producto para crear una clase GestorDeProducto. Esta clase deberá gestionar una lista de productos, mostrar los productos y calcular el precio total de todos ellos.**