Introducción a Typescript

¿Qué es TypeScript?

TypeScript es un lenguaje de programación desarrollado por Microsoft que se construye sobre JavaScript. Fue lanzado en 2012 y se ha vuelto muy popular en los últimos años.

Principales características

1. Tipado estático opcional - TypeScript permite definir tipos de datos para variables, parámetros y retornos de funciones. Esto brinda más seguridad y evita ciertos errores comunes en JavaScript.

2. Soporte para programación orientada a objetos - TypeScript soporta clases, interfaces, herencia, etc. Esto facilita la construcción de aplicaciones grandes y escalables.

3. Compila a JavaScript simple - El código TypeScript se compila a JavaScript que se ejecuta en cualquier navegador o entorno de JavaScript.

4. Compatibilidad con ECMAScript más nuevo - TypeScript soporta las últimas características de ECMAScript como async/await, etc.

5. Gran adopción - En 2023, TypeScript se ha vuelto el lenguaje preferido para muchos desarrolladores de JavaScript y es ampliamente utilizado en aplicaciones Angular, React, Vue.js y Node.js.

6. Comunidad activa - Hay una gran comunidad alrededor de TypeScript con recursos y soporte disponibles.

Crear el primer proyecto

1. crea una carpeta, ingresa a ella y abre el editor de código de su preferencia.

```
D:\proyectoTypescript>mkdir proyecto1
```

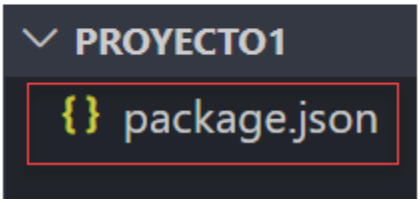
```
D:\proyectoTypescript>cd proyecto1

D:\proyectoTypescript\proyecto1> code .
```

2. Inicializa npm:

```
PS D:\proyectoTypescript\proyecto1> npm init -y
```

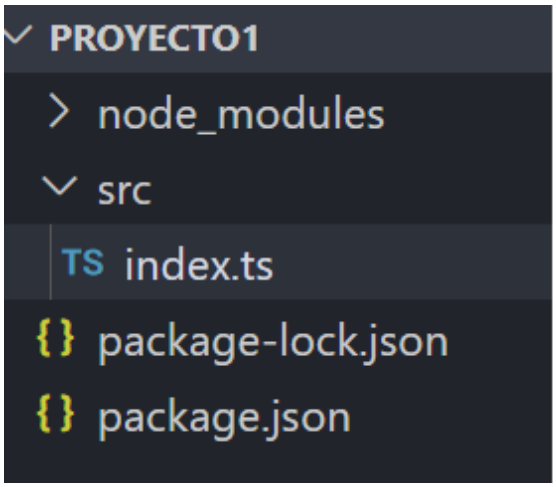
El archivo package.json es un archivo importante en los proyectos de Node.js. Contiene metadata sobre el proyecto y maneja las dependencias del proyecto.



3. Instala ts-node como dependencia de desarrollo:

```
PS D:\proyectoTypescript\proyecto1> npm install --save-dev ts-node
```

ts-node es un paquete de Node.js que permite ejecutar código TypeScript directamente sin la necesidad de compilarlo a JavaScript.



4. Crear una carpeta src, luego un archivo con la extension .ts

```
src > TS index.ts > ...
1   const message: string = "Hello World";
2
3   console.log(message);
```

5. Ejecutar e archivo .ts

```
PS D:\proyectoTypescript\proyecto1> ts-node src/index.ts
Hello World
```

Tipo de datos:

```
1   // Boolean
2   let isDone: boolean = false;
3
4   //Number
5   let decimal: number = 6;
6   let hex: number = 0xf00d;
7   let binary: number = 0b1010;
8
9   // String
10  let color: string = "blue";
11  let sentence: string = `Hello, my name Arle`;
```

```ts
// Array
let list1: number[] = [1, 2, 3];
let list2: Array<number> = [1, 2, 3];

//Una tuple en TypeScript es un tipo de dato que representa una lista ordenada de elementos que pueden ser de diferentes tipos.

let x: [string, number];
x = ['hello', 10];

let usuario: [nombre: string, edad: number, estáActivo: boolean]

usuario = ["Juan", 30, true];
```

```ts
/* El tipo de dato never en TypeScript representa el tipo de valores que nunca ocurren.

Algunas características del tipo never:
- Es un subtipo de todos los demás tipos.
- No tiene valores en sí mismo.
- Representa el tipo de valor que nunca se alcanza durante la ejecución.
//funciones que lanzan excepciones
function error(msg: string): never {
    throw new Error(msg);
}
// funciones que nunca teriminan su ejecucion
function infiniteLoop(): never {
  while (true) {}
}
```

```ts
//Any
let notSure: any = 4;
notSure = "maybe a string instead";
notSure = false;

// Void
function warnUser(): void {
    console.log("This is my warning message");
}

// Object
function create(variable: object | null): void {

}
```

Ejemplo decisiones en TypeScript

```typescript
function suma(a: number, b: number){
    if (a <= 0 || b <= 0) {
      throw new Error("Los números deben ser mayores a 0");
    }
    return a + b;
  }

  const num1 = 5;
  const num2 = 10;

  const resultado = suma(num1, num2);

  console.log(resultado);
```

Ejemplo ternario en TypeScript

```typescript
let num1:number = 5;
let num2: number = 3;

// Operador ternario
let resultado:string = num1 > num2 ? "num1 es mayor" : "num2 es mayor o igual";

// Imprimir resultado
console.log(resultado);
```

Switch -case

```typescript
function showDay(day: string) {
    switch (day) {
      case "lunes":
        console.log("Es lunes");
        break;
      case "martes":
        console.log("Es martes");
        break;
      case "miércoles":
        console.log("Es miércoles");
        break;
      case "jueves":
        console.log("Es jueves");
        break;
      case "viernes":
        console.log("Es viernes");
        break;
      case "sábado":
        console.log("Es sábado");
        break;
      case "domingo":
        console.log("Es domingo");
        break;
      default:
        throw new Error("Día no válido");
    }
  }

  showDay("lunes"); // Imprime "Es lunes"
```

En TypeScript, los uniones son tipos que pueden tomar uno de dos o más tipos. Por ejemplo, la siguiente unión puede almacenar un número o una cadena:.

```ts
type MyUnion = number | string;

let x: MyUnion = 10;
x = "Hola";
```

```ts
// unión de tipos de objetos
type MyUnion = {
    name: string;
    age: number;
  } | {
    email: string;
    phone: string;
  };

  let x: MyUnion = { name: "Juan", age: 25 };
  x = { email: "juan@example.com", phone: "123-456-7890" };
```

Ciclo for

```ts
// Imprime los números del 1 al 5
for (let i:number = 1; i <= 5; i++) {
    console.log(i);
 }
```

for-each

```ts
// Imprime los elementos de un array
const numbers: Array<number> = [8, 6, 1, 4, 5];

for (const numero of numbers) {
  console.log(numero);
}
```

Ciclo while

```ts
// Imprime los números del 1 al 4
let i:number = 1;

while (i <= 4) {
  console.log(i);
  i++;
}
```

En TypeScript, las anotaciones son un mecanismo para especificar el tipo de datos de una variable, función o expresión. Las anotaciones se utilizan para mejorar la seguridad del código y facilitar la depuración.

readonly para parámetros opcionales: Esta anotación se utiliza para especificar que un parámetro opcional es de solo lectura.
never para tipos de retorno: Esta anotación se utiliza para especificar que una función nunca devuelve un valor.
readonly para propiedades de clases: Esta anotación se utiliza para especificar que una propiedad de una clase es de solo lectura.
optional para propiedades de clases: Esta anotación se utiliza para especificar que una propiedad de una clase es opcional

Readonly- solo lectura

```typescript
class Persona {
    readonly nombre: string;
    readonly edad: number;

    constructor(nombre: string, edad: number) {
      this.nombre = nombre;
      this.edad = edad;
    }

    saludar() {
      console.log(`Hola, soy ${this.nombre} y tengo ${this.edad} años.`);
    }
  }

  const persona = new Persona("Juan", 30);
  console.log(persona.nombre); // "Juan"
  console.log(persona.edad);   // 30
  persona.nombre = "teresa"; // no se puede cambiar porque es readonly
```