TypeScript

[TypeScript 2](#_Toc189844632)

[Introducción a TypeScript 2](#_Toc189844633)

[Caracteristicas 2](#_Toc189844634)

[Declaración de Variables 3](#_Toc189844635)

[Ejemplos de Declaración 3](#_Toc189844636)

[Inferencia de Tipos 3](#_Toc189844637)

[Funciones 3](#_Toc189844638)

[Funciones con Tipos 3](#_Toc189844639)

[Funciones que Retornan Valores 3](#_Toc189844640)

[Parámetros de Tipo any o unknown 3](#_Toc189844641)

[Funciones que Nunca Terminan 4](#_Toc189844642)

[Objetos y Tipos Alias 4](#_Toc189844643)

[Type Alias 4](#_Toc189844644)

[Propiedades Opcionales 4](#_Toc189844645)

[Creación de Objetos 4](#_Toc189844646)

[Funciones Anónimas y Callbacks 4](#_Toc189844647)

[Arrays e Iteraciones 5](#_Toc189844648)

[Inmutabilidad 5](#_Toc189844649)

[Optional Chaining 5](#_Toc189844650)

[Tipos de Uniones y Combinaciones 5](#_Toc189844651)

[Uniones con & 5](#_Toc189844652)

[Uniones con | 5](#_Toc189844653)

[Tipos de Objetos y Propiedades Opcionales 6](#_Toc189844654)

[Tipos de Arrays 6](#_Toc189844655)

[Tipos de Funciones 6](#_Toc189844656)

[Enumeraciones 6](#_Toc189844657)

[Operadores para Tipado y Comprobación de Objetos 6](#_Toc189844658)

[Uso del Operador typeof 6](#_Toc189844659)

[Uso del Operador instanceof 6](#_Toc189844660)

[Retorno de Tipo con typeof 7](#_Toc189844661)

[Arrays en TypeScript 7](#_Toc189844662)

[Matrices en TypeScript 7](#_Toc189844663)

[Tuplas en TypeScript 8](#_Toc189844664)

[Tipos Básicos 8](#_Toc189844665)

[Tipo any 8](#_Toc189844666)

[Tipo void 8](#_Toc189844667)

[Tipo unknown 8](#_Toc189844668)

[Tipo never 9](#_Toc189844669)

[Tipos e Interfaces 9](#_Toc189844670)

[Diferencias entre type e interface 9](#_Toc189844671)

[Uso de readonly 9](#_Toc189844672)

[Literales de Plantilla 9](#_Toc189844673)

[Tipos Indexados 10](#_Toc189844674)

[Tuplas 10](#_Toc189844675)

[Enumeraciones 10](#_Toc189844676)

[Aserciones de Tipo 10](#_Toc189844677)

[Comprobación de Tipos 10](#_Toc189844678)

[instanceof 10](#_Toc189844679)

[typeof 11](#_Toc189844680)

[Interfaces Avanzadas 11](#_Toc189844681)

[Anidamiento de Interfaces 11](#_Toc189844682)

[Extensión de Interfaces 11](#_Toc189844683)

[Narrowing (Discriminación de Tipos) 11](#_Toc189844684)

[Type Guards 11](#_Toc189844685)

[Clases 12](#_Toc189844686)

# **TypeScript**

## Introducción a TypeScript

TypeScript es un superconjunto de JavaScript que añade tipos estáticos opcionales, lo que permite un desarrollo más robusto y mantenible. A continuación, te presentamos conceptos básicos y ejemplos de código para empezar a aprender TypeScript.

### Caracteristicas

• Ahorro de documentación

• Tipado estático (vs al tipado dinámico de JavaScript)

• No funciona en tiempo entonces de ejecución

• No va funcionar en el navegador

• Añade seguridad, robustez

• Se va a escribir más código que en JavaScript

**Nota**: editor online: <https://www.typescriptlang.org/play>

**Nota**: debemos de evitar escribir los tipos

## Declaración de Variables

En TypeScript se pueden declarar variables utilizando let, const, y var, al igual que en JavaScript.

### Ejemplos de Declaración

let anyValue: any = 'hola'; // Permite cualquier tipo de dato.  
let otherValue: unknown = 'adios'; // Permite cualquier tipo, pero es más seguro que `any`.  
const persona = "hola"; // Constante que no puede ser reasignada.

**Nota:** Aunque any permite flexibilidad, es preferible evitarlo en favor de unknown, que fuerza validaciones antes de su uso.

## Inferencia de Tipos

TypeScript puede inferir automáticamente el tipo de una variable basado en su valor inicial.

let a = 10; // Infirió como number.  
let b = "20"; // Infirió como string.  
  
console.log(a + b); // Concatenación: resultado "1020"

## Funciones

### Funciones con Tipos

Es posible definir funciones con parámetros y valores de retorno tipados.

function saludar(name: string) {  
 console.log(`Hola ${name}!`);  
}  
  
saludar("Juan"); // Correcto  
// saludar(123); // Incorrecto: El argumento debe ser un string

### Funciones que Retornan Valores

function saludar3({ name, age }: { name: string; age: number }): { name: string; age: number } {  
 console.log(`Hola ${name}! Tu edad es ${age}`);  
 return { name, age };  
}

### Parámetros de Tipo any o unknown

Aunque any permite flexibilidad, es recomendable usar unknown siempre que sea posible.

function handleUnknown(value: unknown) {  
 if (typeof value === "string") {  
 console.log(`Valor string: ${value}`);  
 }  
}

### Funciones que Nunca Terminan

function throwError(message: string): never {  
 throw new Error(message);  
}

## Objetos y Tipos Alias

Puedes definir estructuras de objetos mediante type o interface.

### Type Alias

type Hero = {  
 name: string;  
 age: number;  
 city: string;  
};  
  
let hero: Hero = {  
 name: 'Ironman',  
 age: 45,  
 city: 'New York',  
};

### Propiedades Opcionales

type Hero2 = {  
 id?: string; // Propiedad opcional  
 name: string;  
 age: number;  
 city?: string;  
};

### Creación de Objetos

function createHero(name: string, age: number, city: string): Hero {  
 return { name, age, city };  
}  
const thor = createHero('Thor', 1000, 'Asgard');

## Funciones Anónimas y Callbacks

const sayHiFromFunction = (fn: (name: string) => void) => {  
 fn("Juan");  
};  
  
sayHiFromFunction((name) => {  
 console.log(`Hola ${name}!`);  
});

## Arrays e Iteraciones

const avengers = ['Hulk', 'Thor', 'Ironman'];  
  
avengers.forEach((avenger) => {  
 console.log(avenger);  
});

## Inmutabilidad

Puedes hacer objetos inmutables utilizando Object.freeze().

const persona2 = {  
 nombre: 'Juan',  
 edad: 30,  
 ciudad: 'New York',  
};  
  
Object.freeze(persona2);  
persona2.nombre = 'Pedro'; // Error: No se puede modificar

## Optional Chaining

Permite acceder a propiedades opcionales sin causar errores.

thor3.id?.toString(); // Si `id` existe, lo convierte a string

## Tipos de Uniones y Combinaciones

### Uniones con &

Permite combinar las propiedades de varios tipos.

type A = { name: string, age: number };  
type B = { age: number, job: string };  
  
type IntersectionType = A & B;  
  
let person: IntersectionType = { name: 'John', age: 30, job: 'Developer' };  
console.log(person);

### Uniones con |

Permite definir tipos que pueden ser de varios tipos posibles.

type HeroId = `${string}-${string}`;  
type HeroPower = 'powerful' | 'aggressive' | 'healer';  
  
let ann: number | string;  
ann = 'Hero A';  
ann = 1234;

### Tipos de Objetos y Propiedades Opcionales

type Hero1 = { id: HeroId, power: HeroPower };  
type OptionalHero = { id?: HeroId, power?: HeroPower };  
  
let heroA: Hero1 = { id: 'Hero A-1', power: 'powerful' };  
let heroB: Hero1 = { id: 'Hero B-2', power: 'aggressive' };  
let optionalHeroA: OptionalHero = { id: 'Hero A-1', power: 'powerful' };  
let optionalHeroB: OptionalHero = { id: 'Hero B-2' };

### Tipos de Arrays

type HeroArray = Hero1[];  
let heroes: HeroArray = [heroA, heroB];

### Tipos de Funciones

type HeroFunction = (id: HeroId) => Hero1;  
  
let getHeroById: HeroFunction = (id) => {  
 return heroA;  
};

### Enumeraciones

enum HeroPowerType {  
 POWERFUL = 'powerful',  
 AGGRESSIVE = 'aggressive',  
 HEALER = 'healer',  
}  
  
let heroPower: HeroPowerType = HeroPowerType.POWERFUL;

## Operadores para Tipado y Comprobación de Objetos

### Uso del Operador typeof

Obtiene el tipo de un objeto o variable.

let personita: { name: string; age: number } = { name: 'John', age: 30 };  
console.log('Type of person:', typeof personita); // Output: object

**Nota:** typeof devuelve el tipo primitivo de una variable, como string, number, boolean, u object.

### Uso del Operador instanceof

Verifica si un objeto hereda de una clase.

class Animal {  
 makeSound() {  
 console.log('Animal makes a sound');  
 }  
}  
  
class Dog extends Animal {  
 makeSound() {  
 console.log('Dog barks');  
 }   
}  
  
const dog = new Dog();  
console.log('Is dog an instance of Animal?', dog instanceof Animal); // Output: true

**Nota:** instanceof es útil para comprobar relaciones de herencia entre objetos.

### Retorno de Tipo con typeof

function getPersonType(person: { name: string; age: number }): string {  
 return typeof person;   
}  
console.log('Type of person returned by getPersonType:', getPersonType(personita)); // Output: object

## Arrays en TypeScript

const languager: string[] = []; // Array de tipo string  
languager.push('JavaScript'); // Bien  
languager.push('Python'); // Bien  
// languager.push(1); // Error: no se puede agregar un número a un array de strings  
  
const mixedArray: (string | number)[] = []; // Array de tipo string o número  
mixedArray.push('JavaScript'); // Bien  
mixedArray.push(1); // Bien

### Matrices en TypeScript

const matrix: number[][] = [  
 [1, 2, 3],  
 [4, 5, 6],  
 [7, 8, 9]  
];  
  
const matrix2: string[][] = [  
 ['a', 'b', 'c'],  
 ['d', 'e', 'f'],  
 ['g', 'h', 'i']  
];  
  
// Tipado de celdas para el tablero de 3 en raya  
type celda = 'X' | 'O' | '';   
type Game = [  
 [celda, celda, celda],  
 [celda, celda, celda],  
 [celda, celda, celda]  
];  
const tablero: Game = [  
 ['X', 'O', ''],  
 ['', 'X', 'O'],  
 ['O', '', 'X']  
];

**Nota:** Las matrices multidimensionales son útiles para representar estructuras como tableros de juegos o mapas.

## Tuplas en TypeScript

const personal: [string, number] = ['John', 30]; // Bien  
  
const mixedTuple: [string, number | string] = ['John', 30]; // Bien  
  
function getPersonDetails(): [string, number] {  
 return ['John', 30];  
}

**Nota:** Las tuplas son listas con una longitud fija donde cada posición puede tener un tipo diferente.

## Tipos Básicos

### Tipo any

Permite asignar valores de cualquier tipo, pero su uso se considera una mala práctica.

let anyVariable: any = 10; // Puede almacenar cualquier tipo de valor  
anyVariable = "Hola";  
anyVariable = true;  
anyVariable = {};  
anyVariable = [];

### Tipo void

Indica que una función no devuelve ningún valor.

function noReturnFunction(): void {  
 console.log("Esto no devuelve nada");  
}

### Tipo unknown

Permite almacenar cualquier tipo de valor, pero es más seguro que any porque requiere comprobaciones antes de usarlo.

let unknownVariable: unknown = 10;  
unknownVariable = "Hola";  
unknownVariable = true;  
unknownVariable = {};  
unknownVariable = [];

### Tipo never

Indica que una función nunca termina de manera correcta, ya que lanza una excepción o entra en un bucle infinito.

function error(message: string): never {  
 throw new Error(message);  
}

## Tipos e Interfaces

### Diferencias entre type e interface

interface Hombre {  
 nombre: string;  
 edad: number;  
}  
  
type Masculino = {  
 nombre: string;  
 edad: number;  
};  
  
function getHombre(nombre: string, edad: number): Hombre {  
 return { nombre, edad };  
}  
  
function getMasculino(nombre: string, edad: number): Masculino {  
 return { nombre, edad };  
}

### Uso de readonly

Las propiedades declaradas como readonly no pueden ser modificadas.

type Heroe = {  
 readonly id?: string;  
 name: string;  
 age: number;  
 isActive?: boolean;  
};  
  
function createHeroe(heroe: Heroe): Heroe {  
 const { name, age } = heroe;  
 return {  
 id: crypto.randomUUID(),  
 name,  
 age,  
 isActive: true,  
 };  
}

## Literales de Plantilla

type Color = string;  
type ColorHex = `#${string}`;  
  
const hexa: Color = "ff0000";  
const hexa2: ColorHex = "#ff0000";

## Tipos Indexados

Permiten acceder a las propiedades de un tipo basándose en un índice.

type Person = {  
 name: string;  
 address: {  
 city: string;  
 country: string;  
 };  
};  
  
type City = Person['address']['city']; // string

## Tuplas

Listas de elementos con tipos definidos y una longitud fija.

type Persona = [string, number, boolean];  
const persona: Persona = ["Juan", 30, true];

## Enumeraciones

Proporcionan una forma de definir valores constantes con nombres significativos.

enum ErrorType {  
 NotFound = "notFound",  
 Internal = "internal",  
 Unknown = "unknown"  
}  
  
function mostrarMensaje(tipoDeError: ErrorType) {  
 if (tipoDeError === ErrorType.NotFound) {  
 console.log("El recurso no existe");  
 } else if (tipoDeError === ErrorType.Internal) {  
 console.log("Error interno del servidor");  
 } else {  
 console.log("Error desconocido");  
 }  
}  
  
mostrarMensaje(ErrorType.NotFound);

## Aserciones de Tipo

Permiten a TypeScript tratar una variable como un tipo específico.

const button = document.getElementById("button") as HTMLButtonElement;  
const canvas = document.getElementById("canvas") as HTMLCanvasElement;

## Comprobación de Tipos

### instanceof

Verifica si un objeto es una instancia de una clase.

class Personal {  
 constructor(public name: string) {}  
}  
  
const personal = new Personal("John");  
if (personal instanceof Personal) {  
 console.log("El objeto es una instancia de Personal");  
}

### typeof

Devuelve el tipo de una variable.

let myVariable: any = 10;  
console.log(typeof myVariable); // "number"

## Interfaces Avanzadas

### Anidamiento de Interfaces

interface Producto {  
 id: number;  
 nombre: string;  
 precio: number;  
}  
  
interface Carrito {  
 productos: Producto[];  
 total: number;  
}

### Extensión de Interfaces

interface Heroe extends Persona {  
 poderes: string[];  
}  
  
const superHeroe: Heroe = {  
 nombre: "Superman",  
 edad: 40,  
 altura: 1.90,  
 saludar: () => {  
 console.log(`Hola, soy Superman y tengo 40 años.`);  
 },  
 poderes: ["Invencibilidad", "Superfuerza"],  
};

## Narrowing (Discriminación de Tipos)

function mostrarLongitud(objeto: number | string): number {  
 if (typeof objeto === "string") {  
 return objeto.length;  
 } else {  
 return objeto.toString().length;  
 }  
}

### Type Guards

interface Mario {  
 company: "nintendo";  
 nombre: string;  
 saltar: () => void;  
}  
  
interface Sonic {  
 company: "sega";  
 nombre: string;  
 correr: () => void;  
}  
  
type Personaje = Mario | Sonic;  
  
function checkIsSonic(personaje: Personaje): personaje is Sonic {  
 return personaje.company === "sega";  
}  
  
function jugar(personaje: Personaje) {  
 if (checkIsSonic(personaje)) {  
 personaje.correr();  
 } else {  
 personaje.saltar();  
 }  
}

## Clases

class Avenger {  
 public nombre: string;  
 public equipo: string;  
 private vida: number;  
  
 constructor(nombre: string, equipo: string, vida: number) {  
 this.nombre = nombre;  
 this.equipo = equipo;  
 this.vida = vida;  
 }  
}  
  
let avenger1 = new Avenger("Thor", "Avengers", 100);