Manual completo de TypeScript

JavaScript con tipos de 6 a 18 años · Sin necesidad de plataforma

Versión: manual autónomo · 3 módulos por edad
Para: alumnos, padres, ludotecas, centros educativos y tutores

Cómo usar este manual

TypeScript es **JavaScript con tipos**: el ordenador te avisa de errores **antes** de ejecutar el programa. Este manual funciona **sin plataforma online**.

Requisito previo

Haber completado al menos el **módulo A de JavaScript** (variables, funciones, arrays). Si no lo has hecho, empieza por el manual de JavaScript.

Estructura

Módulo	Edades	Duración orientativa	Contenido
A	6–12	8–10 sesiones × 30–45 min	Tipos básicos, funciones tipadas
B	12–14	10–12 sesiones × 45–60 min	Objetos, arrays, `type`, filtros
C	14–18	12–14 sesiones × 60 min	Interfaces, enums, genéricos, proyecto API

Para el adulto: No hace falta dominar TypeScript. Instala Node.js una vez, ejecuta los ejemplos con `npx tsx archivo.ts` y deja que el alumno escriba el código. Los errores de tipo en rojo son **aliados**: léelos en voz alta y preguntad juntos qué significan.

Qué necesitas

1. **Node.js 18 o superior** desde nodejs.org (incluye `npm` y `npx`).
2. Editor de texto: **VS Code** recomendado (extensión oficial TypeScript incluida).
3. Una carpeta de proyecto, por ejemplo `mi-typescript/`.

Configuración inicial (una sola vez)

Abre una terminal en la carpeta del proyecto:

```
npm init -y
npm install typescript tsx --save-dev
npx tsc --init
```

Crea un archivo `saludo.ts`:

```ts
let nombre: string = "Luna";
console.log("Hola,", nombre);
```

Ejecuta:

```
npx tsx saludo.ts
```

Deberías ver: `Hola, Luna`

> **Sin instalar en ludoteca:** typescriptlang.org/play permite probar código en el navegador. Copia los ejemplos allí si no puedes instalar Node.js.

Reglas de oro

1. **Un concepto por sesión.**
2. Si TypeScript marca error en rojo, **no ignores el mensaje**: es la lección.
3. Las soluciones están al final de cada módulo; intenta los ejercicios primero.

Módulo A · 6–12 años

Módulo A · Primeros tipos

Objetivos

- Explicar qué aportan los tipos respecto a JavaScript.
- Usar `string`, `number` y `boolean`.
- Escribir funciones con tipos en parámetros y retorno.

Lección A1 · ¿Qué es TypeScript?

Duración: 25 min

JavaScript deja guardar cualquier cosa en una variable. TypeScript **anota** qué tipo de dato esperamos:

```
// JavaScript (sin tipos)
let edad = 10;
edad = "diez";  // No da error hasta que algo falle

// TypeScript (con tipos)
let edadTs: number = 10;
// edadTs = "diez";  // ❌ Error: no es un número
```

Los tipos son como **etiquetas** en las cajas: «esta caja solo guarda números».

Práctica A1: Crea tres variables tipadas: tu nombre (`string`), tu edad (`number`) y si te gustan los videojuegos (`boolean`). Imprime las tres con `console.log` .

Lección A2 · Tipos básicos

Duración: 30 min

Tipo	Qué guarda	Ejemplo
`string`	Texto	`"Luna"`
`number`	Números	`10` , `3.14`
`boolean`	Verdadero/falso	`true` , `false`

```
let nombre: string = "Alex";
let puntos: number = 0;
let juegoActivo: boolean = true;

console.log(nombre, "tiene", puntos, "puntos");
console.log("¿Juego activo?", juegoActivo);

puntos = puntos + 15;
console.log("Nuevos puntos:", puntos);
```

Práctica A2: Variables para un animal favorito (nombre `string`), cuántas patas tiene (`number`) y si vuela (`boolean`). Imprime una frase completa.

Lección A3 · Funciones tipadas

Duración: 35 min

Una función puede decir **qué recibe** y **qué devuelve**:

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

function saludar(nombre: string): string {
  return "Hola, " + nombre + "!";
}

console.log(sumar(4, 5));
console.log(saludar("Luna"));
```

- Después de cada parámetro: `: tipo`
- Después de los paréntesis: `: tipoDeRetorno`

Práctica A3: Función `doble(n: number): number` que devuelva el doble. Función `esMayorDeEdad(edad: number): boolean` que devuelva `true` si edad ≥ 18.

Lección A4 · Arrays tipados

Duración: 35 min

Un array guarda una lista. El tipo indica **qué** contiene:

```
const colores: string[] = ["rojo", "azul", "verde"];
const notas: number[] = [8, 9, 7, 10];

console.log("Primer color:", colores[0]);
console.log("Número de notas:", notas.length);

for (let i = 0; i < colores.length; i++) {
  console.log(i + 1, colores[i]);
}
```

También puedes escribir: `Array<string>` en lugar de `string[]`.

Práctica A4: Array `string[]` con tres frutas. Recórrelo con un `for` e imprime cada una.

Lección A5 · Ficha de personaje

Duración: 40 min

Juntamos tipos en un mini programa:

```
let nombrePersonaje: string = "Sir Lancelot";
let nivel: number = 5;
let esMago: boolean = false;
let habilidades: string[] = ["espada", "escudo", "correr"];

function describir(nombre: string, nivel: number): string {
  return nombre + " es nivel " + nivel;
}

console.log(describir(nombrePersonaje, nivel));
console.log("¿Es mago?", esMago);
console.log("Habilidades:", habilidades.join(", "));
```

Práctica A5: Cambia los valores por un personaje inventado. Añade una habilidad más al array.

Proyecto A · Tarjeta de jugador tipada

Crea un programa que defina nombre, puntos de vida (`number`), está vivo (`boolean`) y tres objetos del inventario (`string[]`). Incluye una función `resumen(nombre: string, vida: number): string` que devuelva una frase tipo «Alex tiene 100 puntos de vida».

Soluciones módulo A

Práctica A1:

```
let nombre: string = "Luna";
let edad: number = 11;
let leGustanJuegos: boolean = true;
console.log(nombre, edad, leGustanJuegos);
```

Práctica A3:

```ts
function doble(n: number): number {
  return n * 2;
}

function esMayorDeEdad(edad: number): boolean {
  return edad ⩾ 18;
}

console.log(doble(7));
console.log(esMayorDeEdad(15));
```

Proyecto A (ejemplo):

```ts
let nombre: string = "Alex";
let vida: number = 100;
let vivo: boolean = true;
let inventario: string[] = ["espada", "poción", "mapa"];

function resumen(nombre: string, vida: number): string {
  return nombre + " tiene " + vida + " puntos de vida";
}

console.log(resumen(nombre, vida));
console.log("¿Vivo?", vivo);
console.log("Inventario:", inventario.join(", "));
```

Módulo B · 12–14 años

Módulo B · Objetos y colecciones

Objetivos

- Definir formas de objetos con `type`.
- Trabajar con arrays de objetos.
- Usar `filter` y `map` con tipos.
- Introducir tipos unión (`string | number`).

Lección B1 · El alias `type`

Duración: 40 min

Un `type` describe la **forma** de un objeto:

```ts
type Item = {
  nombre: string;
  cantidad: number;
};

const poción: Item = { nombre: "Poción roja", cantidad: 3 };
const llave: Item = { nombre: "Llave dorada", cantidad: 1 };

console.log(poción.nombre, "×", poción.cantidad);
```

Si falta una propiedad o el tipo no coincide, TypeScript avisa.

Práctica B1: Define `type Libro` con `titulo: string` y `paginas: number`. Crea dos libros e imprímelos.

Lección B2 · Arrays de objetos

Duración: 45 min

```ts
type Item = {
  nombre: string;
  cantidad: number;
};

const inventario: Item[] = [
  { nombre: "Poción", cantidad: 3 },
  { nombre: "Llave", cantidad: 1 },
  { nombre: "Moneda", cantidad: 50 },
];

function totalItems(items: Item[]): number {
  let total = 0;
  for (const item of items) {
    total += item.cantidad;
  }
  return total;
}

console.log("Total objetos:", totalItems(inventario));

for (const item of inventario) {
  console.log("-", item.nombre, ":", item.cantidad);
}
```

Práctica B2: Añade un cuarto ítem al inventario. Escribe una función `buscarPorNombre(items: Item[], nombre: string): Item | undefined`.

Lección B3 · `filter` y `map`

Duración: 45 min

```ts
type Tarea = {
  titulo: string;
  hecha: boolean;
};

const tareas: Tarea[] = [
  { titulo: "Estudiar TypeScript", hecha: false },
  { titulo: "Hacer ejercicio", hecha: true },
  { titulo: "Leer un capítulo", hecha: false },
];

function pendientes(lista: Tarea[]): Tarea[] {
  return lista.filter((t) => !t.hecha);
}

function titulos(lista: Tarea[]): string[] {
  return lista.map((t) => t.titulo);
}

console.log("Pendientes:", pendientes(tareas).length);
console.log("Títulos:", titulos(tareas));
```

Práctica B3: Función `marcarHecha(lista: Tarea[], indice: number): Tarea[]` que devuelva una copia con esa tarea marcada como hecha.

Lección B4 · Tipos unión

Duración: 40 min

A veces un valor puede ser de **varios** tipos:

```ts
type Id = string | number;

function mostrarId(id: Id): void {
  console.log("ID:", id, "tipo:", typeof id);
}

mostrarId("user-42");
mostrarId(42);

type ResultadoNumero = number | null;

function dividir(a: number, b: number): ResultadoNumero {
  if (b === 0) return null;
  return a / b;
}

const r = dividir(10, 2);
if (r !== null) {
  console.log("Resultado:", r);
}
```

`void` significa que la función no devuelve nada útil.

Práctica B4: Tipo `TextoONumero = string | number`. Función que reciba ese tipo e imprima si es texto o número.

Lección B5 · Tuplas (introducción)

Duración: 35 min

Una tupla es un array con **posiciones fijas** y tipos concretos:

```ts
type Coordenada = [number, number];

const origen: Coordenada = [0, 0];
const punto: Coordenada = [10, 25];

function distanciaAlOrigen(c: Coordenada): number {
  const [x, y] = c;
  return Math.sqrt(x * x + y * y);
}

console.log("Distancia:", distanciaAlOrigen(punto));
```

Práctica B5: Tupla `PersonaCorta = [string, number]` para nombre y edad. Crea dos personas e imprímelas.

Proyecto B · Lista de tareas tipada

Programa con tipo `Tarea`, array de al menos 5 tareas, y funciones:

1. `contarPendientes(tareas: Tarea[]): number`
2. `listarPendientes(tareas: Tarea[]): string[]` (solo títulos)
3. `completarTodas(tareas: Tarea[]): Tarea[]`

Muestra un informe en consola.

Soluciones módulo B

Práctica B2 (buscar):

```
function buscarPorNombre(items: Item[], nombre: string): Item | undefined {
  for (const item of items) {
    if (item.nombre === nombre) return item;
  }
  return undefined;
}
```

Proyecto B (ejemplo):

```ts
type Tarea = { titulo: string; hecha: boolean };

const tareas: Tarea[] = [
  { titulo: "Módulo A", hecha: true },
  { titulo: "Módulo B", hecha: false },
  { titulo: "Proyecto final", hecha: false },
  { titulo: "Repasar tipos", hecha: false },
  { titulo: "Jugar 30 min", hecha: true },
];

function contarPendientes(tareas: Tarea[]): number {
  return tareas.filter((t) => !t.hecha).length;
}

function listarPendientes(tareas: Tarea[]): string[] {
  return tareas.filter((t) => !t.hecha).map((t) => t.titulo);
}

function completarTodas(tareas: Tarea[]): Tarea[] {
  return tareas.map((t) => ({ ...t, hecha: true }));
}

console.log("Pendientes:", contarPendientes(tareas));
console.log("Lista:", listarPendientes(tareas));
console.log("Tras completar:", completarTodas(tareas));
```

Módulo C · 14–18 años

Módulo C · Código profesional

Objetivos

- Usar `interface` y `enum`.
- Campos opcionales y `readonly`.
- Patrón `Resultado<T>` con genéricos.
- Proyecto: mini API de notas tipada.

Lección C1 · Interfaces

Duración: 50 min

Una `interface` describe un contrato para objetos (muy similar a `type`):

```ts
interface Alumno {
  id: string;
  nombre: string;
  edad: number;
  email?: string;  // opcional
}

const alumno: Alumno = {
  id: "a-001",
  nombre: "María",
  edad: 16,
};

function presentar(a: Alumno): string {
  const correo = a.email ?? "sin email";
  return `${a.nombre} (${a.id}) — ${correo}`;
}

console.log(presentar(alumno));
```

`email?` puede faltar. `??` usa un valor por defecto si es `null` o `undefined`.

Práctica C1: Interface `Libro` con `isbn`, `titulo`, `autor` y `prestado?: boolean`. Crea dos libros.

Lección C2 · Enums

Duración: 45 min

Un `enum` agrupa constantes con nombre:

```ts
enum Rol {
  Alumno = "ALUMNO",
  Tutor = "TUTOR",
  Admin = "ADMIN",
}

enum EstadoMision {
  Pendiente,
  EnProgreso,
  Completada,
}

interface Perfil {
  nombre: string;
  rol: Rol;
  xp: number;
}

const perfil: Perfil = {
  nombre: "Alex",
  rol: Rol.Alumno,
  xp: 240,
};

function puedeCertificar(p: Perfil): boolean {
  return p.rol === Rol.Alumno && p.xp >= 200;
}

console.log(puedeCertificar(perfil));
console.log(EstadoMision.Completada);
```

Práctica C2: Enum `Dificultad` (Facil, Media, Dificil) y tipo `Nivel` con `titulo` y `dificultad`. Crea tres niveles.

Lección C3 · `readonly` y buenas prácticas

Duración: 40 min

```ts
interface Configuracion {
  readonly apiUrl: string;
  readonly version: string;
  tema: "claro" | "oscuro";
}

const config: Configuracion = {
  apiUrl: "https://api.ejemplo.com",
  version: "1.0.0",
  tema: "oscuro",
};

// config.apiUrl = "otra";  // ❌ Error: es readonly
config.tema = "claro";  // ✅ OK

type Punto = Readonly<{ x: number; y: number }>;
const p: Punto = { x: 1, y: 2 };
```

Usa `readonly` para datos que no deben cambiar tras crearse.

Práctica C3: Interface `Usuario` con `id` readonly y `nombre` editable. Intenta cambiar `id` y observa el error.

Lección C4 · Genéricos y `Resultado<T>`

Duración: 55 min

Los **genéricos** permiten reutilizar tipos:

```ts
type Resultado<T> =
  | { ok: true; data: T }
  | { ok: false; error: string };

function dividir(a: number, b: number): Resultado<number> {
  if (b === 0) {
    return { ok: false, error: "División por cero" };
  }
  return { ok: true, data: a / b };
}

function buscarAlumno(id: string): Resultado<Alumno> {
  if (id === "a-001") {
    return {
      ok: true,
      data: { id: "a-001", nombre: "María", edad: 16 },
    };
  }
  return { ok: false, error: "Alumno no encontrado" };
}

const r = dividir(10, 2);
if (r.ok) {
  console.log("Resultado:", r.data);
} else {
  console.error(r.error);
}
```

Ventaja frente a `null`: el error tiene **mensaje** y TypeScript obliga a comprobar `ok`.

Práctica C4: Función `raizCuadrada(n: number): Resultado<number>` que falle si `n < 0`.

Lección C5 · Organizar un proyecto

Duración: 45 min

Estructura recomendada:

```
mi-notas/
  package.json
  tsconfig.json
  src/
    tipos.ts
    notas.ts
    main.ts
```

`src/tipos.ts`:

```ts
export enum EstadoNota {
  Aprobado = "APROBADO",
  Suspenso = "SUSPENSO",
}

export interface Nota {
  alumnoId: string;
  asignatura: string;
  valor: number;
}

export type Resultado<T> =
  | { ok: true; data: T }
  | { ok: false; error: string };
```

`src/main.ts`:

```ts
import { Nota } from "./tipos.js";

const notas: Nota[] = [
  { alumnoId: "a-001", asignatura: "Matemáticas", valor: 8 },
];

console.log(notas);
```

En `tsconfig.json`, activa `"module": "NodeNext"` para imports con `.js`.

Ejecuta: `npx tsx src/main.ts`

Proyecto C · API de notas tipada

Modela con interfaces:

- `Alumno` (id, nombre)
- `Nota` (alumnoId, asignatura, valor 0–10)
- `enum EstadoNota` (Aprobado ≥ 5, Suspenso < 5)

Funciones:

1. `añadirNota(notas: Nota[], nota: Nota): Nota[]`
2. `mediaPorAlumno(notas: Nota[], alumnoId: string): Resultado<number>`

3. `alumnosAprobados(notas: Nota[], umbral: number): string[]` (ids únicos con media ≥ umbral)

Imprime un informe final en consola.

Soluciones módulo C

Práctica C4:

```ts
function raizCuadrada(n: number): Resultado<number> {
  if (n < 0) return { ok: false, error: "No hay raíz de negativos" };
  return { ok: true, data: Math.sqrt(n) };
}
```

Proyecto C (ejemplo simplificado):

```typescript
enum EstadoNota {
  Aprobado = "APROBADO",
  Suspenso = "SUSPENSO",
}

interface Alumno {
  id: string;
  nombre: string;
}

interface Nota {
  alumnoId: string;
  asignatura: string;
  valor: number;
}

type Resultado<T> =
  | { ok: true; data: T }
  | { ok: false; error: string };

const alumnos: Alumno[] = [
  { id: "a-001", nombre: "María" },
  { id: "a-002", nombre: "Luis" },
];

let notas: Nota[] = [
  { alumnoId: "a-001", asignatura: "Mates", valor: 7 },
  { alumnoId: "a-001", asignatura: "Lengua", valor: 9 },
  { alumnoId: "a-002", asignatura: "Mates", valor: 4 },
];

function añadirNota(notas: Nota[], nota: Nota): Nota[] {
  return [...notas, nota];
}

function mediaPorAlumno(notas: Nota[], alumnoId: string): Resultado<number> {
  const delAlumno = notas.filter((n) => n.alumnoId === alumnoId);
  if (delAlumno.length === 0) {
    return { ok: false, error: "Sin notas" };
  }
  const suma = delAlumno.reduce((acc, n) => acc + n.valor, 0);
  return { ok: true, data: suma / delAlumno.length };
}

function alumnosAprobados(notas: Nota[], umbral: number): string[] {
  const ids = [...new Set(notas.map((n) => n.alumnoId))];
  return ids.filter((id) => {
    const r = mediaPorAlumno(notas, id);
```

```
    return r.ok && r.data >= umbral;
  });
}

notas = añadirNota(notas, { alumnoId: "a-002", asignatura: "Lengua", valor: 6 });

for (const a of alumnos) {
  const r = mediaPorAlumno(notas, a.id);
  if (r.ok) {
    const estado = r.data >= 5 ? EstadoNota.Aprobado : EstadoNota.Suspenso;
    console.log(a.nombre, "→ media", r.data.toFixed(1), estado);
  }
}

console.log("Aprobados (media ≥ 5):", alumnosAprobados(notas, 5));
```

Apéndice · Errores frecuentes

Error	Causa	Solución
`Type 'string' is not assignable to type 'number'`	Tipo incorrecto	Corrige el valor o el tipo anotado
`Object is possibly 'undefined'`	Acceso sin comprobar	Usa `if` o `?.`
`Cannot find module`	Ruta de import mal	Revisa nombre y extensión `.js` en imports
`tsx: command not found`	No instalado tsx	`npm install tsx --save-dev`
Propiedad falta en objeto	No cumple `type` / `interface`	Añade la propiedad o márcala opcional con `?`

Apéndice · Glosario

Término	Significado
Tipo	Etiqueta que describe qué valores puede tener una variable
`type`	Alias para definir formas de datos
`interface`	Contrato para la forma de un objeto

`enum`	Conjunto de constantes con nombre
Genérico (`<T>`)	Tipo parámetro reutilizable
Unión (`A | B`)	Valor que puede ser de tipo A o B
`readonly`	Propiedad que no se puede modificar
`Resultado<T>`	Patrón éxito/error tipado

> **Opcional:** La plataforma Academia (`ts-starter` , Editor) refuerza este manual con ejercicios interactivos.