# Tipos en TS

Antonio Espín Herranz

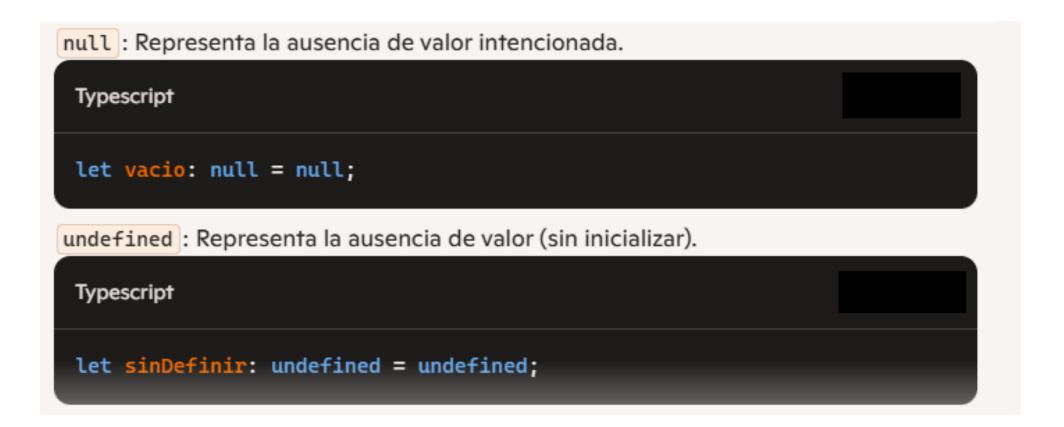
#### Contenidos

- Primitivos
- Undefined y null
- Arrays, Tuplas, Object
- Especiales: Any, void, unknow, never
- Tipos Avanzados: union, intersection, literal, enum
- Definición de variables y constantes:
  - var, let, const

#### **Primitivos**

```
string : Representa texto.
 Typescript
 let saludo: string = "Hola, mundo";
number: Representa números (enteros o decimales).
 Typescript
 let edad: number = 30;
boolean : Representa valores de verdadero o falso.
 Typescript
 let esActivo: boolean = true;
```

#### **Primitivos**



#### **Primitivos**



#### Especiales

any: Puede representar cualquier tipo. Úsalo con precaución porque pierde las ventajas del tipado. Typescript let cualquierCosa: any = 42; unknown: Similar a any, pero requiere que verifiques el tipo antes de usarlo. Typescript let algo: unknown = "Hola"; if (typeof algo === "string") { console.log(algo.toUpperCase());

# Especiales

```
void: Indica que una función no devuelve ningún valor.
 Typescript
function saludar(): void {
   console.log("Hola");
never: Indica que algo nunca debe suceder, como funciones que arrojan errores.
  Typescript
  function error(mensaje: string): never {
    throw new Error(mensaje);
```

# Arrays, tuplas y object



#### union, intersection, literal, enum

```
union: Permite que una variable sea de varios tipos.
 Typescript
 let id: string | number = 123;
intersection: Combina varios tipos en uno.
 Typescript
 type A = { a: string };
 type B = { b: number };
 let combinado: A & B = { a: "Texto", b: 42 };
```

## union, intersection, literal, enum

```
literal: Permite definir un valor específico como tipo.
 Typescript
 let estado: "activo" | "inactivo" = "activo";
enum: Define un conjunto de valores constantes.
 Typescript
 enum Color {
  Rojo,
  Verde,
  Azul,
 let favorito: Color = Color.Verde;
```

- Definición de variables: var y let
- Constantes: const
- var: Evitar su uso, ya que tiene comportamientos impredecibles debido a su ámbito de función/global.
- **let**: Utilizarlo para variables que puedan cambiar su valor durante la ejecución.
- **const**: Usarlo siempre que sea posible para valores que no cambiarán su referencia.

```
1. Alcance (Scope)
var (Ámbito de función o global)
  Typescript
  function ejemploVar() {
   if (true) {
      var x: number = 10; // Declarada con `var`
      console.log(x); // Funciona dentro del bloque
    console.log(x); // También funciona fuera del bloque
  ejemploVar();
  // Salida:
  // 10
  // 10
```

```
let (Ámbito de bloque)
 Typescript
 function ejemploLet() {
   if (true) {
     let y: string = "Hola"; // Declarada con `let`
     console.log(y); // Funciona dentro del bloque
   // console.log(y); // Error: `y` no está definido fuera del bloque
 ejemploLet();
```

```
const (Ámbito de bloque)
 Typescript
 function ejemploConst() {
   if (true) {
     const z: boolean = true; // Declarada con `const`
     console.log(z); // Funciona dentro del bloque
   // console.log(z); // Error: `z` no está definido fuera del bloque
 ejemploConst();
```



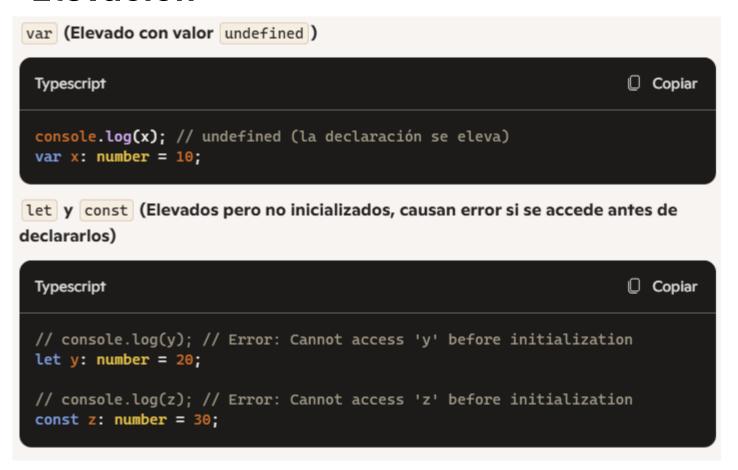
const (No permite reasignación, pero el contenido puede ser mutable en objetos o
arrays)

Typescript

const c: string = "Fijo";
// c = "Error"; // Error: No se puede reasignar

const persona: { nombre: string; edad: number } = { nombre: "Antonio", edad:
persona.nombre = "María"; // Cambiar propiedades es válido
console.log(persona); // { nombre: "María", edad: 30 }

#### Elevación



# var, let, const: Resumen

Característica	var	let	const
Alcance	Función o global	Bloque	Bloque
Reasignación	Permitido	Permitido	No permitido
Redeclaración	Permitido	No permitido	No permitido
Inicialización	Opcional (undefined)	Opcional	Obligatoria
Mutabilidad	Mutable	Mutable	Referencia fija, pero contenido mutable
Hoisting	Sí, con valor undefined	Sí, pero no inicializado	Sí, pero no inicializado