Tipos en TS

Antonio Espín Herranz

Contenidos

- Declaraciones
- Buenas prácticas
- 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

Declaraciones

Para definir variables:

- var: en desuso en TS
 - Alcance equivalente al bloque de funciones en el que se definió.
 - Definida dentro de una función \rightarrow alcance local dentro de la función
 - Definida fuera de una función \rightarrow alcance global desde cualquier función.

let y const: es lo habitual

- Cuando una variable se declara usando let o const, su alcance es equivalente al bloque donde se declara.
- Hay que asignar un valor antes de usarla.
- Con const es obligatorio asignarla un valor al declararla y ya no se puede reasignar
- Con let si se puede cambiar el valor.

Buenas prácticas

No utilizar var

• De forma predeterminada utilizar **const**, pero si hay que reasignar el valor de la variable utilizaremos **let**.

- Definición:
 - let / const variable: type
 - let / const variable: type = valor
- Ejemplos:
 - const edad: number = 45

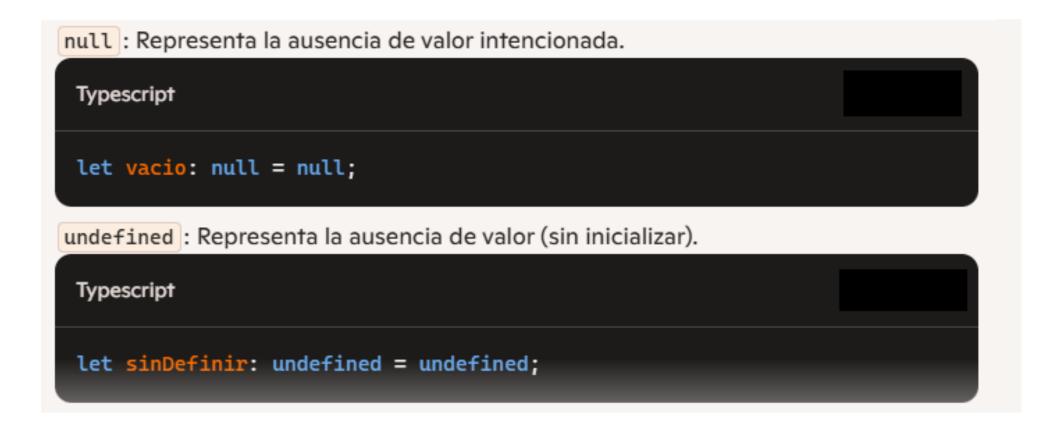
Definiciones

- Si tenemos dos ficheros ts con una variable repetida en ambos TS dará error, porque, aunque estén en ficheros separados se consideran en el mismo espacio global.
- Se puede utilizar namespaces o módulos (más adelante)
- O de momento modificar una opción del fichero de configuración: tsconfig.json con la opción: "isolatedModules": true
- // Utilizar un fichero de configuración definiciones: globals.d.ts
- declare const miVariableGlobal: string

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);
```

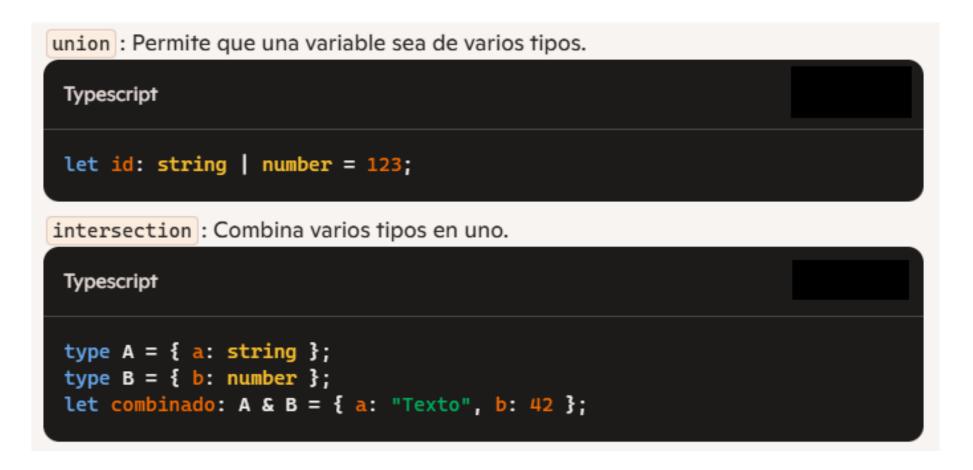
Arrays, tuplas y object



Sintaxis 2:

let/const variable: Array<type>

union, intersection, literal, enum



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();
```



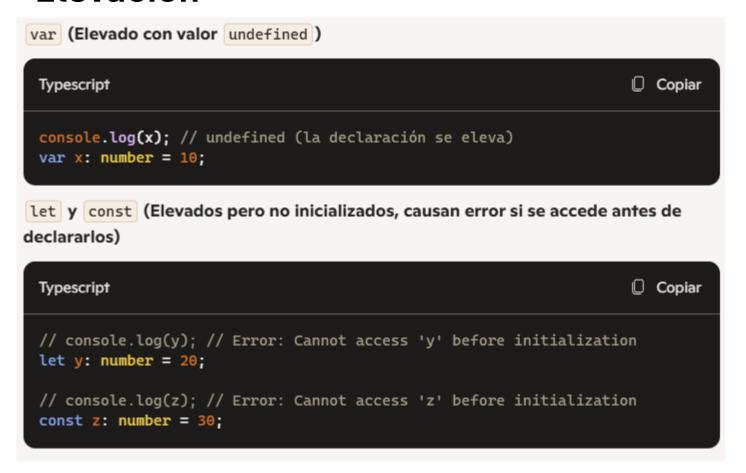
const (No permite reasignación, pero el contenido puede ser mutable en objetos o
arrays)

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