

WOMEN WHO
CODE®
/medellín

FROM HERO TO SUPERHERO

BACKEND CON NODEJS

AGRADECER A...

Nuestro patrocinador y tutores voluntarios

PATROCINADOR

softserve

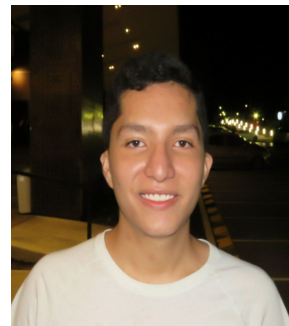
TUTORES VOLUNTARIOS



Edwin



Habid



Jean



Jose

TUTORES VOLUNTARIOS



**Edwin
Otalvaro**

Soy ingeniero de Sistemas con más de 7 años de experiencia en desarrollo de aplicaciones web, actualmente trabajo para Microsoft como Senior Software Engineer en la plataforma Azure.

En mi tiempo libre me gusta salir en bicicleta, jugar videojuegos o tocar algún instrumento. Se tocar la guitarra y en este momento estoy en clases de piano y violin.

NUESTROS VALORES

01 **PUNTUALIDAD**
El tiempo de todos es oro

02 **ORDEN**
Dejar todo mejor de lo que encontramos: limpio y ordenado.

03 **RESPETO**
Como invitados respetamos las reglas de nuestros anfitriones, en este caso Softserve

04 **COLABORACIÓN**
Apoyarnos y ayudarnos para terminar como un solo equipo

VIVIR LA CULTURA WWCODE

Que tus actos hablen más que tus palabras

01

Theoretical Lesson

Typed programming
languages

02

Features of the Topic

Strongly typed
Loosely typed

03

TIPS

Types
Interfaces

04

PRACTICAL EXERCISES

Express types

TABLE OF CONTENTS

GLOSARIO

Repasemos términos que usaremos antes de
iniciar

Glosario

Compilar

01

Compilar es el proceso de **transformar** un **programa** informático escrito en un lenguaje en un conjunto de instrucciones en **otro formato** o **lenguaje**

Runtime

02

Intervalo de **tiempo** en el que un **programa** de **computadora** se **ejecuta** en un sistema operativo

Statically typed

03

El programa valida los tipos y busca errores de tipos durante la compilación

Dynamically typed

04

El programa valida los tipos y busca errores de tipos durante la ejecución del programa (runtime)

Iniciemos

¿Qué son los tipos en un programa de software?

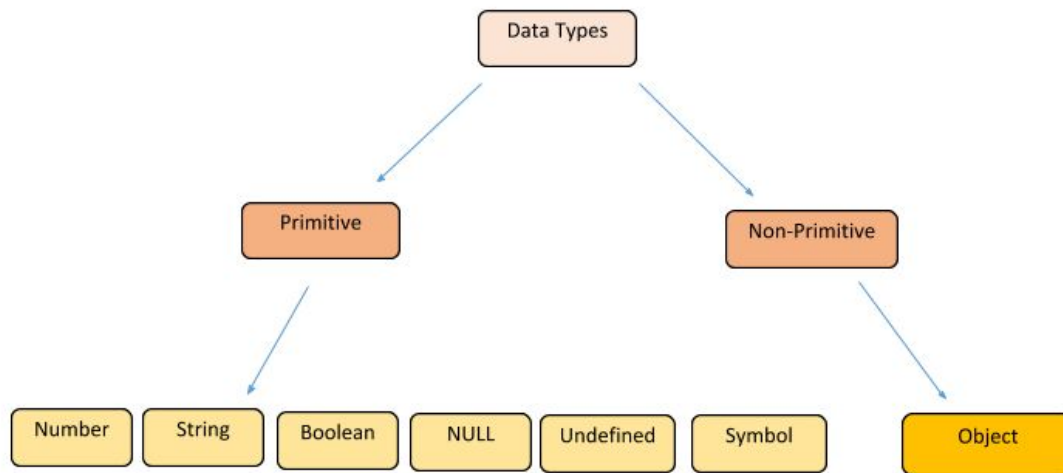
Tipos

Los tipos permiten al programador especificar y controlar qué tipo de datos pueden ser almacenados y manipulados en un programa.

Los tipos proporcionan información sobre la **naturaleza** de los **datos** y cómo deben ser interpretados y utilizados.



Tipos en Javascript



Javascript - typeof

typeof es un operador JavaScript que al ser llamado sobre una variable, devuelve el tipo de dato que dicha variable contiene

```
typeof 42;      // 'number'
typeof 3.14;    // 'number'
typeof NaN;     // 'number'

typeof 'Berry'; // 'string'
typeof true;    // 'boolean'
typeof false;   // 'boolean'
```

Javascript - variables

Javascript tiene tipos, pero al crear una variable podemos asignarle cualquier tipo



Dynamically typed

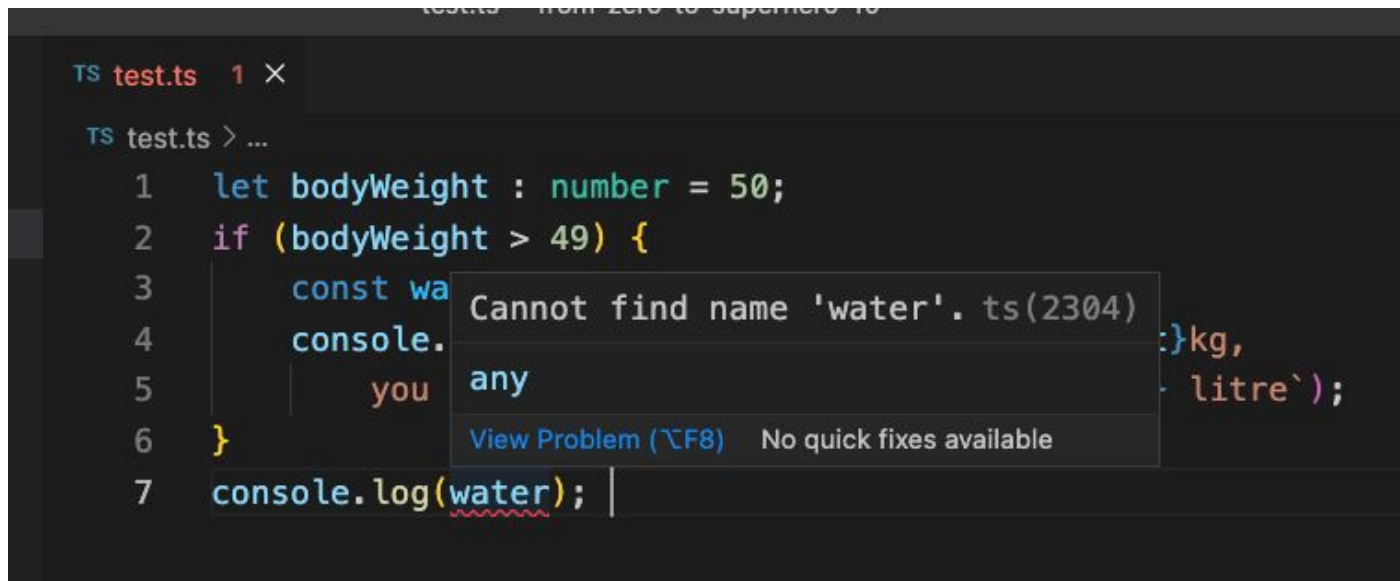
Al no tener un tipo al definir la variable significa que Javascript es **Dynamically typed**. Los tipos se validan en tiempo de ejecución



```
1  const bodyWeight = 50;
2  if (bodyWeight > 49) {
3      const water = 1.4;
4      console.log(`For body weight of ${bodyWeight}kg,
5          you should drink water at least ${water} litre`);
6  }
7  console.log(water);
```

Statically typed

Los lenguajes con tipado estático validan los tipos y los posibles errores en tiempo de compilación



The screenshot shows a code editor with a TypeScript file named `test.ts`. The code is as follows:

```
1 let bodyWeight : number = 50;
2 if (bodyWeight > 49) {
3     const wa
4     console.
5     you
6 }
7 console.log(water);
```

A tooltip is displayed over the word `water` on line 7, indicating a TypeScript error: `Cannot find name 'water'. ts(2304)`. The tooltip also shows the `any` type and a button to `View Problem (\F8)` with the note `No quick fixes available`.

Weak vs Strong typing

Los lenguajes de programación con tipado débil son flexibles al momento de asignar o operar sobre los datos

Los lenguajes de programación con tipado fuerte son estrictos al momento de asignar o operar sobre los datos

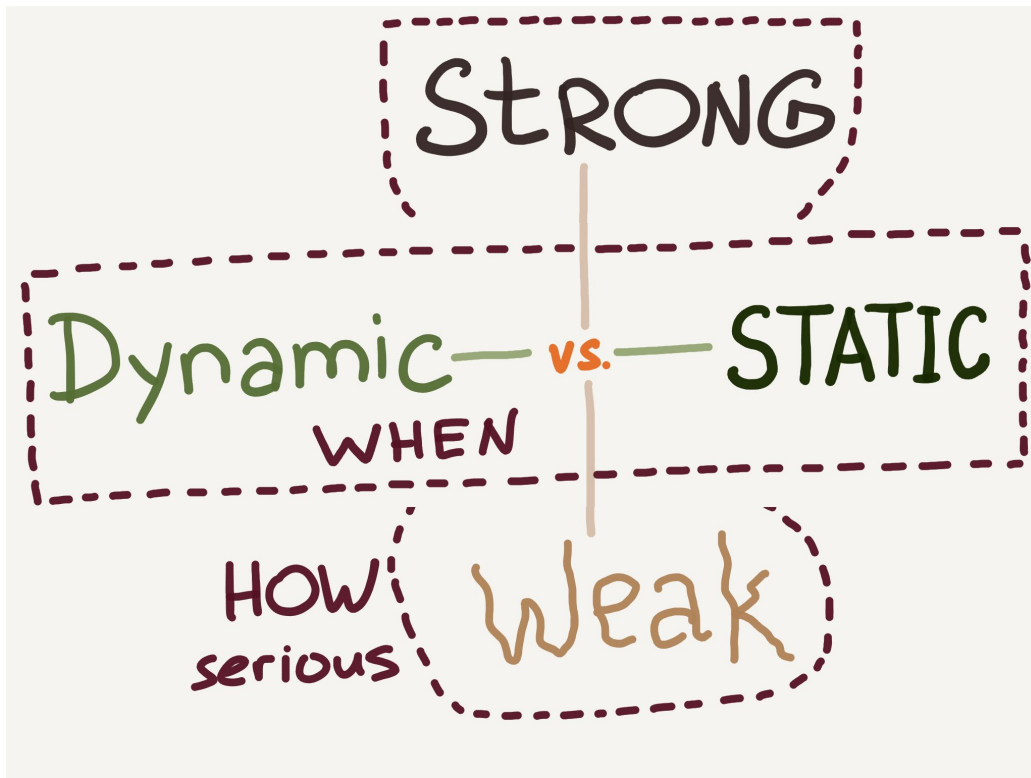


Javascript - Weak typing

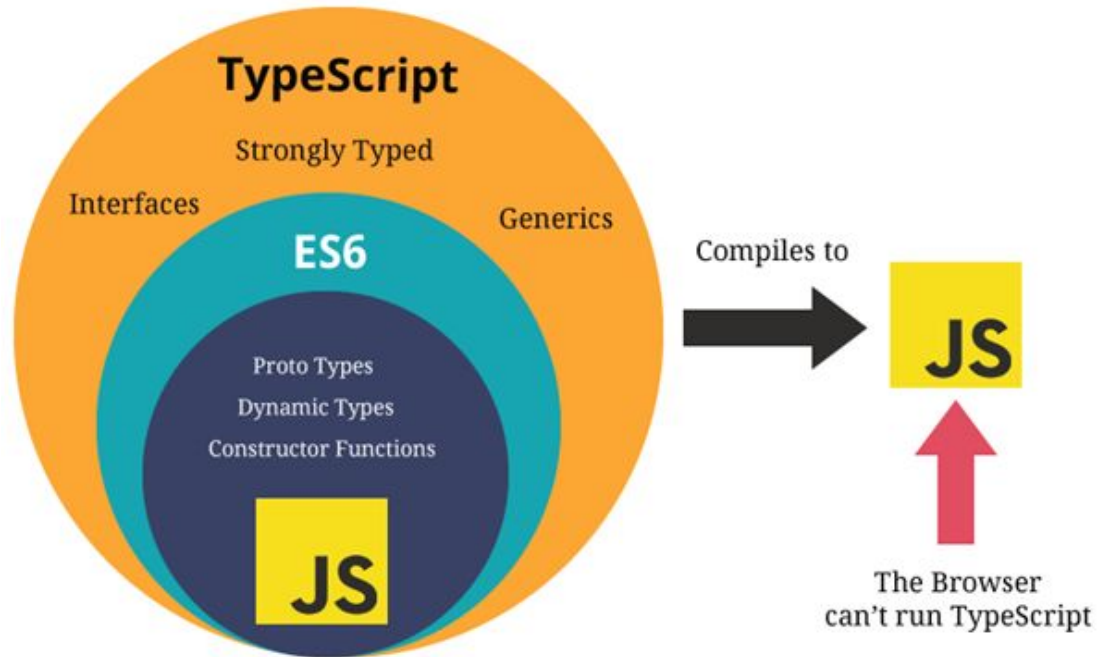
Javascript es un lenguaje de programación débilmente tipado. Ya vimos que tiene tipos (números, cadenas de texto, boolean, etc) pero nos permite operar entre ellos de forma flexible

```
4 + '7';    // '47'  
4 * '7';    // 28  
2 + true;   // 3  
false - 3;  // -3
```

Strong vs Weak - Dynamic - Static



TypeScript

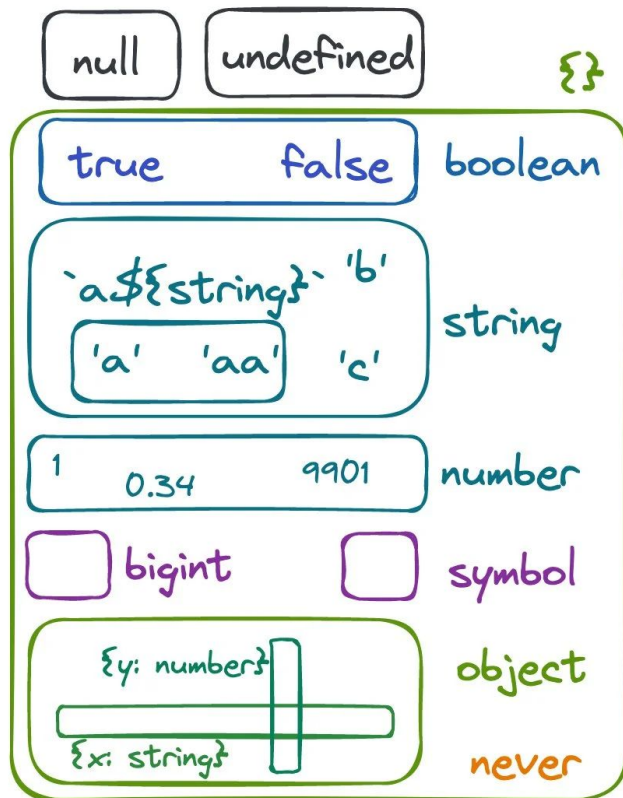


TypeScript vs Javascript

JavaScript	TypeScript
Lenguaje de <i>scripting</i> orientado a objetos	Lenguaje de <i>scripting</i> orientado a objetos
Puede ser interpretado y ejecutado	Compila a JavaScript
Lenguaje interpretado, ejecutado directamente en un navegador	Lenguaje compilado, no se puede ejecutar directamente en un navegador
Dinamicamente tipado	Puede ser estáticamente tipado
Más flexible	Mejor estructurado
Bueno para proyectos pequeños y simples	Ideal para proyectos complejos


Tipos de datos en Typescript

Typescript usa los tipos básicos de Javascript (number, string, boolean, object, etc) y nos permite tener nuevos tipos u operaciones sobre estos



Void

Se utiliza principalmente para declarar el tipo de funciones que no devuelven nada



```
1 function warnUser(): void {  
2     alert("This is my warning message");  
3 }
```

Funciones



```
1 function sumar(a: number, b: number): number {  
2     return a + b;  
3 }
```

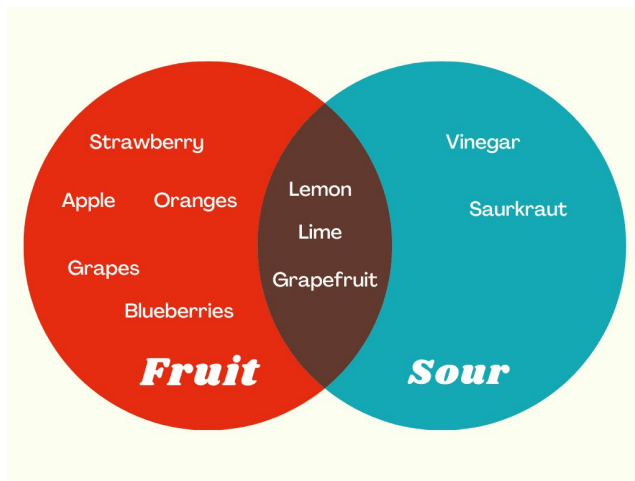
Parámetros opcionales

Puedes hacer que un parámetro sea opcional en una función agregando ? al final de su nombre o especificando un valor por defecto

```
1 function saludar(nombre: string, apellido?: string): string {  
2     if (apellido) {  
3         return 'Hola, ' + nombre + ' ' + apellido;  
4     } else {  
5         return 'Hola, ' + nombre;  
6     }  
7 }
```

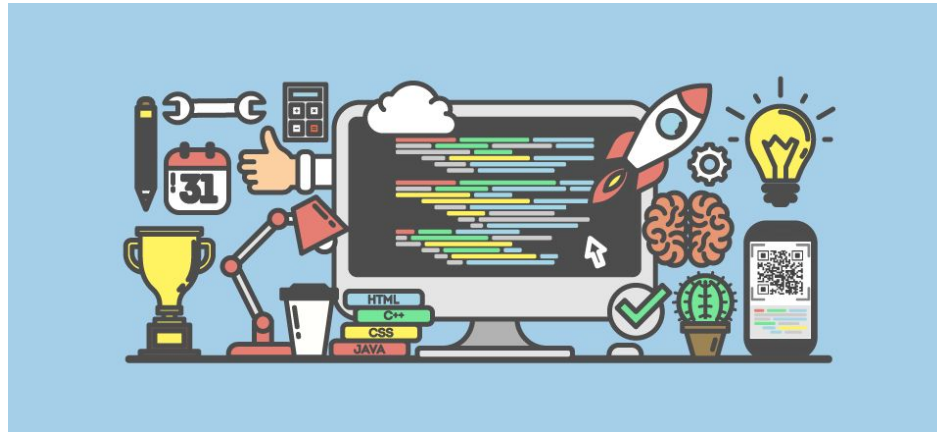
Union Type

Permiten definir una variable, parámetro o propiedad que puede tener más de un tipo. Es decir, puedes especificar que una variable pueda contener valores de diferentes tipos en lugar de solo un tipo específico



Objects

En TypeScript, puedes agregar tipos a los objetos para especificar la estructura y los tipos de las propiedades que contienen. Esto te permite definir la forma esperada de un objeto y obtener comprobaciones de tipo durante la compilación.



Inline type



```
1 const persona: { nombre: string; edad: number } = {  
2     nombre: 'Juan',  
3     edad: 30,  
4 };
```

Type vs Interface

```
1 type Persona = {  
2     nombre: string;  
3     edad: number;  
4 };  
5  
6 const persona: Persona = {  
7     nombre: 'Juan',  
8     edad: 30,  
9 };
```

```
1 interface Persona {  
2     nombre: string;  
3     edad: number;  
4 }  
5  
6 const persona: Persona = {  
7     nombre: 'Juan',  
8     edad: 30,  
9 };
```

Express - Typescript

Podemos usar **Request** y **Response** para agregar los tipos a los controladores, rutas y middlewares



```
1 import jwt from "jsonwebtoken";
2 import { Request, Response } from "express";
3
4 exports.login = (req: Request, res: Response) => {
5
6     const { email } = req.body;
```


Implementemos Express usando Typescript

Let's Play

Ver el repositorio de github de la clase y leer el readme file

TAREA PARA LA CLASE 11

- Convertir a Typescript un Controller, Una ruta y un Middleware
- La clase 11 vamos a realizar el deploy de la aplicación, por lo que el repositorio debe estar funcionando y corriendo localmente