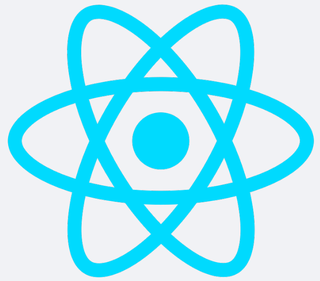
Introducción a TypeScript

{dev/talles}

JavaScript



JavaScript

Hoy en día es fácil superar miles de líneas de código en una aplicación de JavaScript

Y trabajar en equipo en un proyecto de JavaScript puede ser retador

Supongamos...

```
const calculateAge = (birthday) => {

    var ageDifMs = Date.now() - birthday.getTime();
    var ageDate = new Date(ageDifMs);

    return Math.abs(ageDate.getUTCFullYear() - 1970);

}
```

Le falta...

- Tipado de variables
- Errores en tiempo de escritura (linter)
- Auto completado dependiendo de las variables
- Clases y módulos (ES6)
- Validación de objetos dentro de objetos
- Tipado de respuestas http

Problemas comunes

- Errores porque una variable no estaba definida.
- Errores porque el objeto no tiene la propiedad.
- Errores porque no se tiene idea cómo trabaja una función.
- Errores porque se sobre escriben variables, clases o funciones.

Problemas comunes

- Errores porque el código colisiona con el de otro.
- Errores porque colocamos una mayúscula o minúscula.
- Errores porque el IDE o Editor no me dijo.

Lo peor de todo...

JS

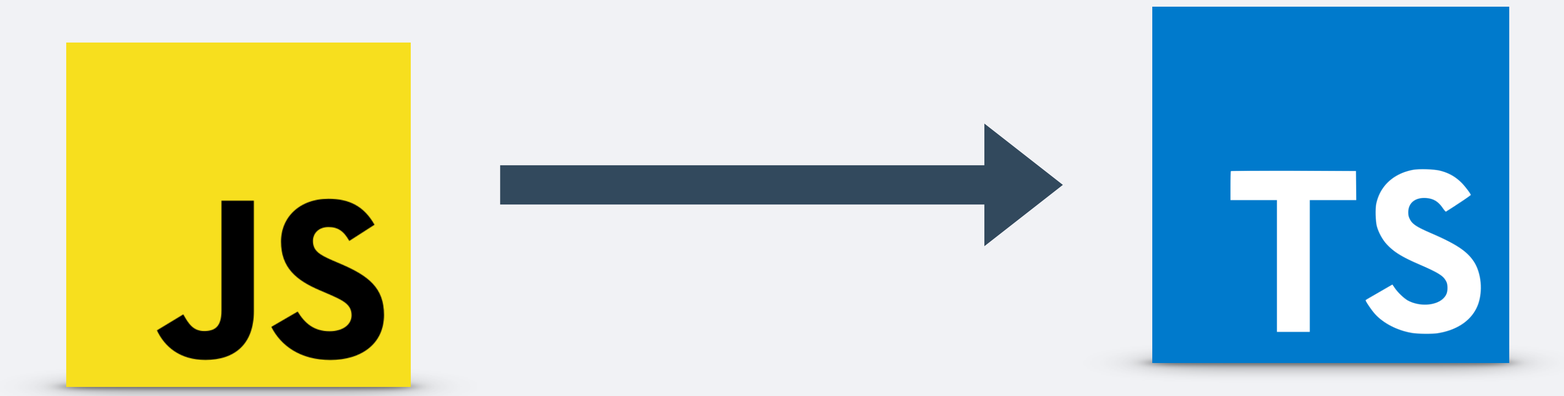
Nos damos cuenta hasta que nuestro programa esta corriendo

TypeScript

{dev/talles}

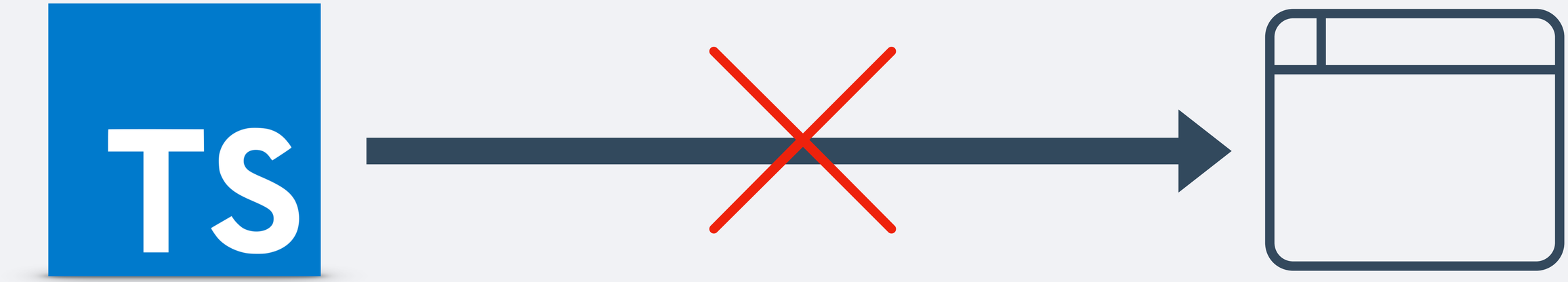
TypeScript

Busca tener la misma experiencia de lenguajes como:
Java, C#, Objective-C…

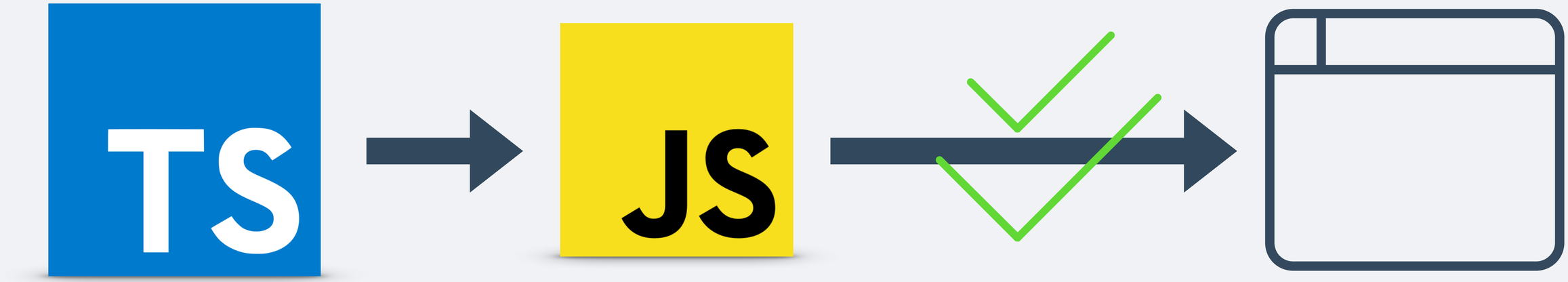


TypeScript es...

Pero...



Pero...



Pero no sólo lo transpila

ESNext

Clases, Arrow functions, tuples…

ES5/6

Te permite escribir código moderno que será soportado por la mayoría de navegadores web o plataforma objetivo

Tipos de datos en TypeScript

{dev/talles}

SuperSet - Los mismo de JS ++

Tipos de datos

Primitivos

- String
- Number
- Boolean
- Symbol

Compuestos

- Objetos literales
- Funciones
- Clases
- Arreglos

{dev/talles}

Strings

“María Perez”

‘Tesla’

`<h1> hola mundo </h1>`

{dev/talles}

Number

PI = 3.14159265359

salary = 1500.01

age = 30

Booleans

isActive = true

keepAlive = false

{dev/talles}

Null y Undefined

age = null

hero = undefined

{dev/talles}

Null y Undefined

age = null

hero = undefined

{dev/talles}

Symbol

```
sym = Symbol()
sym2 = Symbol('myProperty')
```

{dev/talles}

Object literal

```
person = {
    name: ‘Fernando’,
    Age: 35
}
```

Class

```
class Person {
  name;
  age;
}
```

Functions

```
function sayHello () {}
```

```
const sayHello = () => { }
```

Pero TypeScript nos permite

- Crear nuevos tipos
- Interfaces
- Genericos
- Tuplas