

Tema 6

TypeScript

Micael Gallego

micael.gallego@urjc.es

Twitter: @micael_gallego

TypeScript

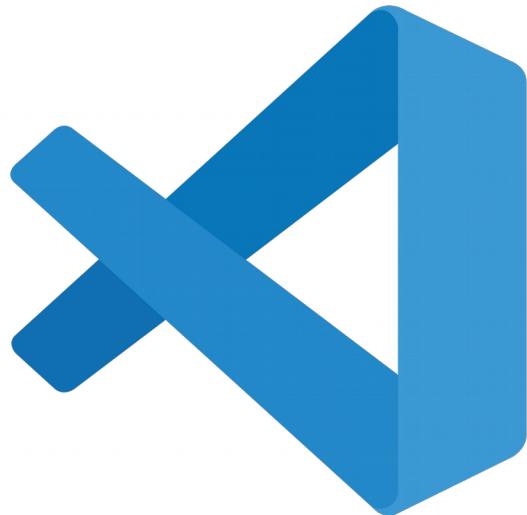


Qué es TypeScript ?

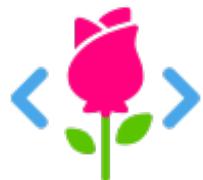
Una **extensión** de JavaScript **ES6**
para que puedas **ayudar** a las
herramientas
para que te **ayuden** a ti

<https://www.typescriptlang.org/>

Necesitamos un editor moderno



Visual Studio
Code



Preparando el entorno...

- **Node.js**

<https://nodejs.org/>

- **Visual Studio Code**

<https://code.visualstudio.com>

- **TypeScript**

npm install -g typescript

npm install -g ts-node

script2.ts - ejem2 - Visual Studio Code

EXPLORER

OPEN EDITORS

- script2.ts
- script.js
- tsconfig.json

EJEM2

- .vscode
- out
- script.js
- script2.ts
- tsconfig.json

script2.ts x script.js tsconfig.json

1 export function process(){
2
3 let num = 3;
4
5 num = "ss";
6
7 return num;
8
9 }
10
11
12
13
14

1 ERROR Filter by type or text

script2.ts 1

✖ [ts] Type 'string' is not assignable to type 'number'. (5, 5)

✖ 1 ▲ 0 Ln 9, Col 2 Spaces: 4 UTF-8 LF TypeScript 😊

Herramientas TypeScript

- Analizan el **código TS** y generan **JS ES6** (**quitando** el código no estándar)
- Si encuentra algún **problema** en tu código te **avisan**
- Aunque tu código tenga “**avisos**”, siempre se **genera** el JavaScript
- Un mismo proyecto puede tener **código JS y TS a la vez**



TypeScript

el lenguaje

Valores del mismo tipo en variables

- Si asignas un **valor de un tipo** al inicializar una variable y luego asignas un valor de **otro tipo**, TypeScript te **avisa**

```
function process(){  
  
    let num = 3;  
  
    num = "ss"; ←  
  
    return num;  
}
```

ERROR: Type 'string' is
not assignable to type
'number'

Verificación de uso correcto

- TS te avisa si usas una **propiedad** o **método** no **disponible** en el tipo de la variable

```
function process4(){  
  
    let num = 3;  
    let name = "Pepe";  
  
    console.log(num.length);  
    console.log(name.length);  
}
```

ERROR: Property 'length'
does not exist on type
'number'

Autocompletado de métodos

- Como sabe el tipo de la variable y los métodos que tiene, te **ayuda**

```
function process5(){  
  let name = "Pepe";  
  let char = name.ch  
}  
  ↴  
  ↴ charAt (method) String.charAt(pos: number): str...  
  Returns the character at the specified index.  
  ↴ charCodeAt  
  ↴ match  
  ↴ search
```

Variables no declaradas

- Si intentas usar una **variable no declarada**, te avisa

```
function process6(){  
  
    let name = "Pepe";  
  
    console.log(nam);  
}
```

ERROR: Cannot find
name 'nam'

Variables con cualquier valor

- Si no inicializas la variable, TS asume que esa variable puede contener **valores de cualquier tipo**

```
function process2(param){  
  
    let id;  
  
    let value = id.ddd() + 4;  
  
    id = 3;  
    id = "ss";  
  
    return id;  
}
```

Variable con valores
de diferente tipo

Te has dado cuenta?

Era código JavaScript
estándar

Empezamos con las
extensiones de TypeScript

Tipos de parámetros

- TS no puede **inferir el tipo** de los parámetros porque no se inicializan*

```
function show(name, age){  
    console.log("Name: "+name);  
    console.log("Age: "+age);  
}
```

De qué tipo son estos parámetros?

* En ES6 se puede dar valor por defecto a los parámetros

Tipos de parámetros

- Para poder ayudar, TS necesita saber **de qué tipo son** los parámetros

```
function show(name: string, age: number){
```

```
    console.log("Name: "+name);
    console.log("Age: "+age);
```

```
}
```

Etiquetas de tipo

- Tipos **básicos**

string number boolean

Tipo de retorno de funciones

- Si le decimos el **tipo de retorno** de la función, TS también nos ayuda

```
function sum2(name: number): number {  
    return name + "2";  
}
```

ERROR: Type 'string' is
not assignable to type
'number'

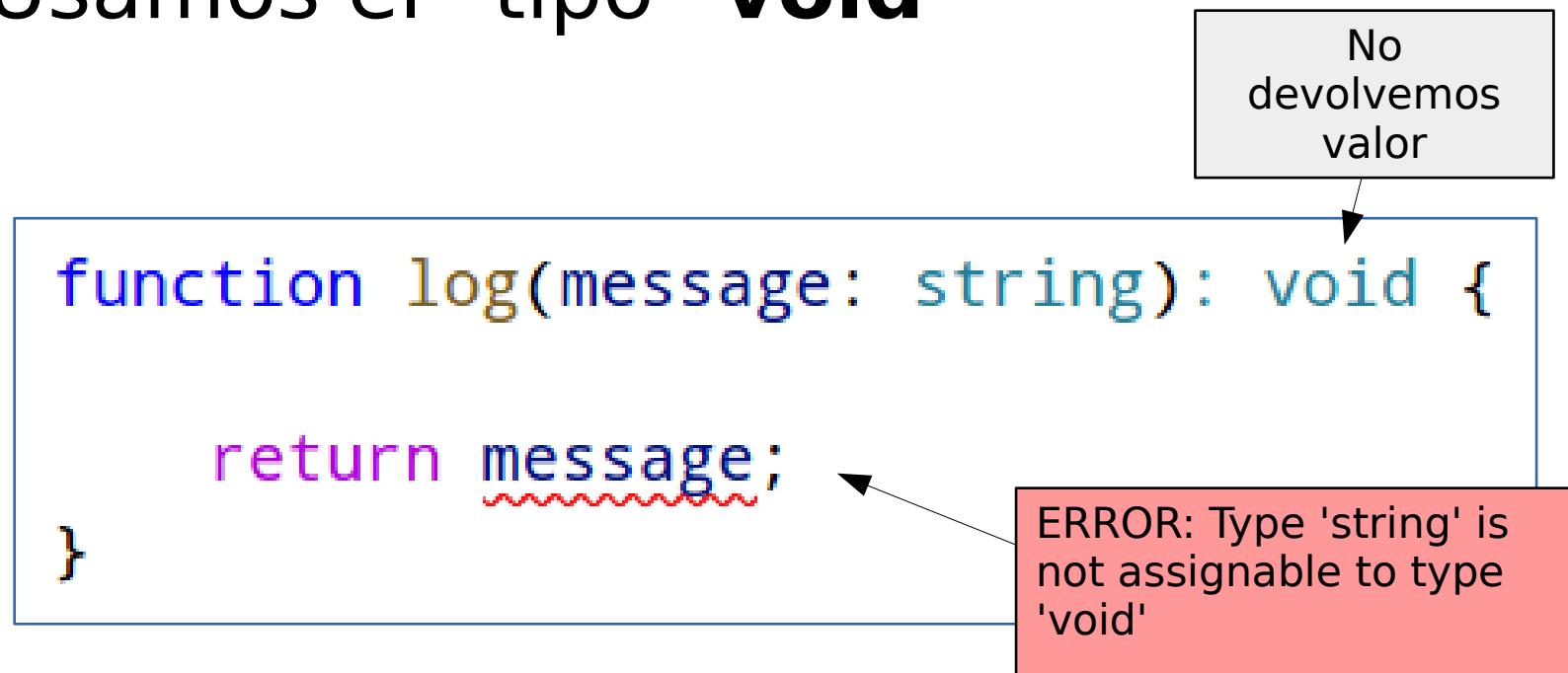
Tipo de retorno de funciones

- Si le decimos que **no vamos a devolver** valor, nos **avisa** si lo hacemos
- Usamos el “tipo” **void**

```
function log(message: string): void {  
    return message;  
}
```

No devolvemos valor

ERROR: Type 'string' is not assignable to type 'void'



Llamar a una función

- Si le decimos los tipos de la función, nos ayuda si no hacemos bien la **llamada**
- Y también puede **inferir** el tipo de la variable

```
+ function sum2(name: number): number { ...  
}
```

```
let num = sum2("kk");
```

TS conoce que
num es de tipo
number

ERROR: Argument of type
'string' is not assignable to
parameter of type 'number'

Parámetros opcionales

- En TS se asume que todos los **parámetros** son **obligatorios**, pero podemos hacer que sean **opcionales con ?**

```
+ function process(param1: string, param2?: number) { ...  
}  
  
process();  
process("ss");  
process("ss", 33);  
process("ss", 33, true);
```

parámetro
opcional

ERROR: Supplied parameters
do not match any signature
of call target.

Declaración del tipo en un variable

- Siempre podemos **indicar el tipo** de la variable al **declararla**
- También la podríamos **inicializar**

```
let message: string;  
  
message = "Hello from codemotion2016";
```

```
message = 34;
```

ERROR: Type 'number' is not
assignable to type 'string'

Arrays

- TS también nos ayuda con los **arrays**

```
let names = ["Pedro", "Juan", "Sofía"];  
  
let first = names[0]; ← Tipo string  
  
for (let name of names) {  
    console.log(name.length);  
}
```

Diagram illustrating type annotations for the code:

- A box labeled "Tipo string[]" points to the array declaration `names`.
- A box labeled "Tipo string" points to the index `[0]` in the assignment `first = names[0];`
- A box labeled "Tipo string" points to the `name` variable in the `for` loop declaration.

Arrays

- Incluso con sus **métodos**

```
let bools: boolean[] = [];
```

```
bools.push(true);
```

```
bools.push("ss");
```

```
let bool = bools.pop();
```

ERROR: Argument of type
'string' is not assignable to
parameter of type 'boolean'

¿Has visto todo lo que
te puede **ayudar** TypeScript
sabiendo el **tipo**?

Te gustaría que TS te
avisara cuando **no sabe el**
tipo de una variable?

El tipo any

- Cuando TS no puede adivinar un tipo mejor para una variable, considera que tiene el tipo **any**

Estos parámetros tienen tipo
any
Como puede venir cualquier
valor, TS no nos puede ayudar
mucho

```
function show(name, age){  
  
    console.log("Name: "+name);  
    console.log("Age: "+age);  
}
```

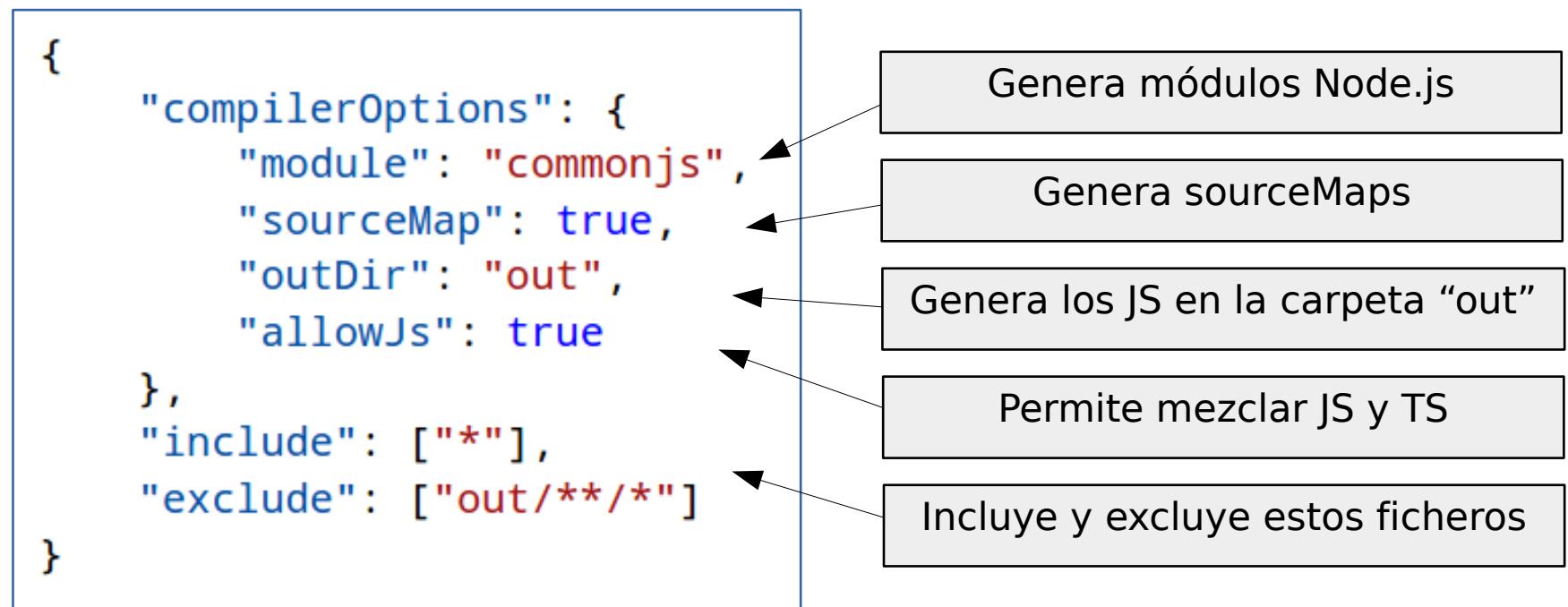
El tipo any

- Si tomamos la decisión de **poner tipo a todos los parámetros y variables**, estaría bien que TS te avisara si se te olvida
- TS te puede avisar si una variable tiene un tipo **any implícito**

```
--noImplicitAny
```

Fichero tsconfig.json

- Un proyecto TS suele tener un fichero **tsconfig.json** en la raíz para configurar las herramientas



nolmplicityAny

- Avisa cuando se **infiere el tipo any** para una variable o parámetro

```
{  
  "compilerOptions": {  
    "module": "commonjs",  
    "sourceMap": true,  
    "outDir": "out",  
    "allowJs": true,  
    "noImplicitAny": true  
  },  
  "include": ["*"],  
  "exclude": ["out/**/*"]  
}
```

nolmplicitAny activada

nolImplicitAny

- Si está activada **nolImplicitAny** se obtiene el siguiente aviso

ERROR: Parameter 'name'
implicitly has an 'any' type

```
function show(name, age){  
  
    console.log("Name: "+name);  
    console.log("Age: "+age);  
}
```

nolmplicityAny

- Si no sabemos el tipo, ponemos el tipo del parámetro como **any**

```
function show(name: any, age: any){  
    console.log("Name: "+name);  
    console.log("Age: "+age);  
}
```

Elegir el tipo más preciso

- **Tipos básicos**
 - **any**:
 - Cualquier valor
 - Demasiado genérico
 - **number, string o boolean**:
 - Algunas veces demasiado restringido

y si en mi código JS un
parámetro **id** puede ser
number o **string**?

Tipos Avanzados

A large word cloud composed of various advanced data types in Spanish, including: REAL, ARRAYS, OBJECTS, NUMBERS, DATA, INTEGERS, STRUCTURES, BINARY, VARIABLES, DATE, LISTS, QUERIES, and TIME.

REAL
ARRAYS
OBJECTS
BOOLEAN NUMBERS
DATA INTEGERS
STRUCTURES BINARY
VARIABLES DATE
QUERIES TIME LISTS

TypeScript
**no te obliga a cambiar
tu código JavaScript**

**Tiene un sistema de tipos
adaptado a los idiomas
JavaScript**

Tipos unión

- Una variable puede tener valores de **varios tipos**

Tipo unión

```
function findById(id: string | number) {  
  
    console.log("Searching: "+id);  
}  
  
findById("r4");  
findById(45);  
findById(true);
```

Tipos unión

- Si la definición del tipo se **complica**, se puede definir como un tipo, darle **nombre** y **usar** ese nombre

```
type Id = string | number;

function findById(id: Id){

    console.log("Searching :" + id);
}
```

Enumerados

- La unión también se puede usar para definir **enumerados de string**

```
+ function process(response: "yes" | "no" | "maybe"): number { ...  
}  
  
console.log(process("yes"));  
console.log(process("no"));  
console.log(process("meybe"));
```

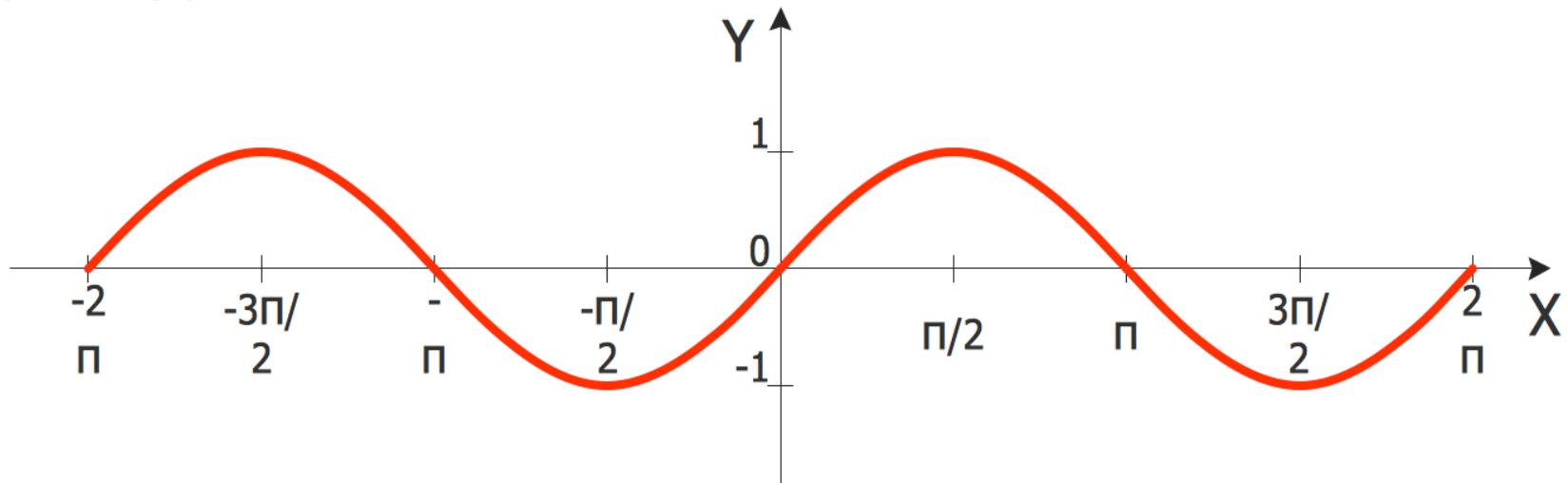
Enumerado
basado en
strings

ERROR: Argument of type ""meybe"" is not assignable to parameter of type "yes" | "no" | "maybe"

* Existen mecanismos **más avanzados**
para declarar enumerados en TypeScript

Tipos de Funciones

$$y = \sin(x)$$



Tipos de funciones

- Cuando asignamos una función a una variable se **infiere** el **tipo** de la **función**

```
+ function random():number { ...  
}  
  
let generator = random;  
  
generator();  
generator(5);
```

ERROR: Supplied parameters do not
match any signature of call target

Tipos de funciones

- Podemos declarar el **tipo** de la función de forma **explícita**

```
+ function random():number { ...  
}  
  
let generator: () => number;  
  
generator = random;  
  
generator();  
generator(5);
```

Tipo de la función
Parámetros =>
Tipo de retorno

Tipos de funciones

- Ahora con **parámetros**

```
+ function sumOne(num: number): number { ...  
}  
  
let sumator: (num: number) => number;  
  
sumator = sumOne;  
  
sumator(3);
```

Tipos de funciones

- Y si nos equivocamos, **TS nos ayuda**

```
+ function twoParam(p:number,p2:any): number { ...
}

+ function randomStr(): string { ...

let sumator: (num: number) => number;

sumator = twoParam;
sumator = randomStr;
```



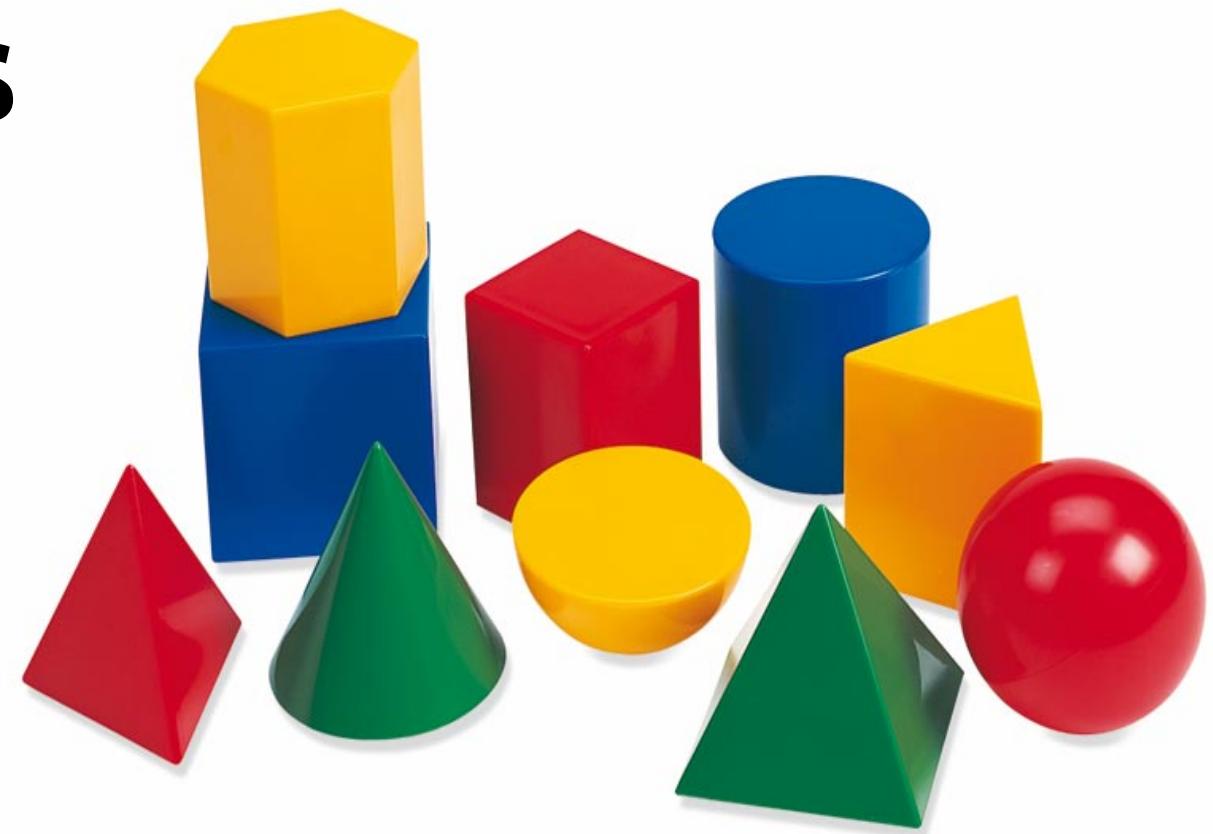
ERROR: Type '() => string' is not assignable to type '(num: number) => number'. Type 'string' is not assignable to type 'number'

Tipos de funciones

- Para soportar **idiomas** típicos TS es **flexible**
- Permite asignar **funciones con menos parámetros** que los declarados en el tipo, siempre que sus tipos sean compatibles

```
+ function random(): number { ...  
}  
  
let sumator: (num: number) => number;  
  
sumator = random;
```

Tipos de Objetos



Tipos de objetos

- En los **objetos literales** se infiere el tipo de la inicialización de la variable

```
let empleado = {  
    nombre: "Pepe",  
    salario: 300  
}  
  
empleado.nombre = "Juan";
```

```
console.log(empleado.|  
    nombre  
    salario
```

(property) nombre: string

Tipos de objetos

- Por defecto no podemos **añadir propiedades**

```
let empleado = {  
    nombre: "Pepe",  
    salario: 300  
}  
  
empleado.nombre = "Juan";  
  
console.log(empleado.salario);  
  
empleado.telefono = "34324234";
```

ERROR: Property 'telefono'
does not exist on type
'{ nombre: string; salario:
number; }'

Tipos de objetos

- Podemos declarar la variable como **any**, pero no se realizará **ninguna validación**

```
let empleado: any = {  
    nombre: "Pepe",  
    salario: 300  
}  
  
empleado.nombre = "Juan";  
  
console.log(empleado.salario);  
  
empleado.telefono = "34324234";
```

Variable de tipo
any

Tipos de objetos

- En los parámetros de las funciones, el tipo **no se puede inferir**
- Si queremos ayuda, hay que especificar el **tipo de forma explícita**

```
function printLabel(obj: any) {  
    console.log(obj.label);  
}  
  
printLabel({ size: 10, label: "Size 10" });  
printLabel({ size: 3, labol: "Size 3" });
```

Error no
detectado

Tipos de objetos

- Se le puede dar un nombre al **tipo de objeto**

Tipo de objeto

```
type Labelled = {
    label: string;
}

function printLabel(obj: Labelled) {
    console.log(obj.label);
}

printLabel({ label: "Size 10 Object" });
```

Tipos de objetos

- Se le puede definir **propiedades optionales**

Opcional

```
type SquareConfig = {  
  color: string;  
  width?: number;  
}  
  
function createSquare(config: SquareConfig) { ... }  
  
createSquare({ color: "black" });  
createSquare({ color: "black", width: 20 });
```

Tipos de objetos

- Para **prevenir errores**, TS nos avisa cuando pasamos una propiedad **no declarada**

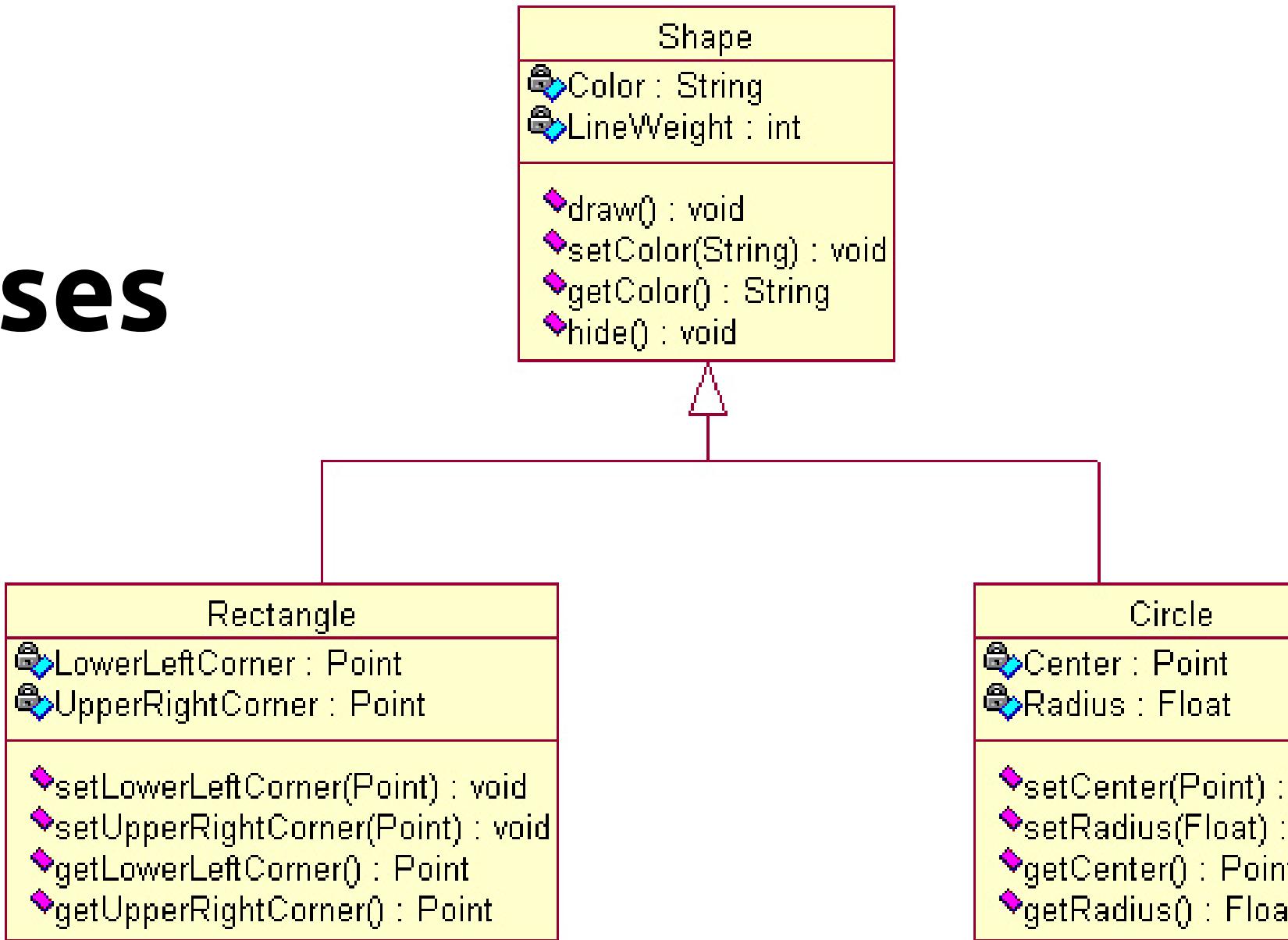
```
type SquareConfig = {  
  color: string;  
  width?: number;  
}
```

```
function createSquare(config: SquareConfig) { ...  
}
```

```
createSquare({ color: "black", widht: 20 });  
createSquare({ color: "black", width: 20, stroke: 3 });
```

ERROR: ...Object literal may
only specify known properties,
and 'widht' does not exist...

Clases



Clases en ES6

```
class Empleado {  
  
    constructor(nombre, salario) {  
        this.nombre = nombre;  
        this.salario = salario;  
    }  
  
    incSalario(incremento) {  
        this.salario += incremento;  
    }  
  
    toString() {  
        return "N:" + this.nombre + " S:" + this.salario;  
    }  
}
```

Herencia de clases en ES6

```
class Jefe extends Empleado {  
  
    constructor(nombre, salario, despacho) {  
        super(nombre, salario);  
        this.despacho = despacho;  
    }  
  
    getNombre() {  
        return this.nombre;  
    }  
  
    toString() {  
        return super.toString() + " D:" + this.despacho;  
    }  
}
```

Creación de objetos en ES6

```
let empleado = new Empleado("Pepe", 800);
empleado.incSalario(200);
console.log(empleado.salario);

let jefe = new Jefe("Juan", 1200, "D22");
console.log(jefe.salario);
console.log(jefe.despacho);
```

Clases en ES6

- **No se declaran los atributos**
- Los atributos se **crean** al usarse (con **this**) en el **constructor** o los **métodos**
- No existe el concepto de **visibilidad** de los atributos, todos son **públicos**
- Hay bastante **controversia** sobre su uso

<https://medium.com/javascript-scene/how-to-fix-the-es6-class-keyword-2d42bb3f4caf#.vvmrzg9fo>

<https://medium.com/@housecor/in-defense-of-javascript-classes-e50bf2270a95#.bqp1arnuy>

**TypeScript nos puede
ayudar a implementar clases
y crear objetos
(si queremos usarlas)**

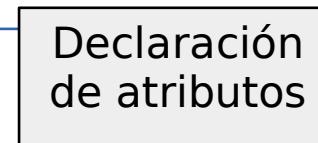
Clases en TypeScript

- Es obligatorio **declarar los atributos**
- Podemos usar visibilidad **public** (por defecto), **private** y **protected**
- **TypeScript** nos avisa de:
 - Uso de atributos que **no existen**
 - Llamadas a métodos que **no existen** o **parámetros** no adecuados
 - Violaciones de la **visibilidad**

Clases en TypeScript

```
class Empleado {  
    nombre: string;  
    salario: number;  
  
    constructor(nombre: string, salario: number) {  
        this.nombre = nombre;  
        this.salario = salario;  
    }  
  
    incSalario(incremento: number) {  
        this.salario += incremento;  
    }  
  
    toString() {  
        return "N:" + this.nombre + " S:" + this.salario;  
    }  
}
```

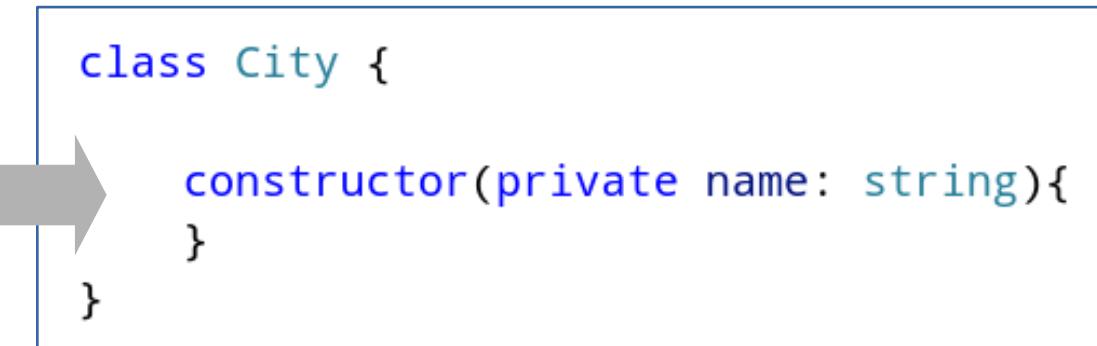
Declaración de atributos



Clases en TypeScript

- Para **evitar** tanta **repetición**, un parámetro del constructor con una visibilidad se considera también un atributo (y se inicializa)

```
class City {  
  
    private name: string;  
  
    constructor(name: string){  
        this.name = name;  
    }  
}
```



The diagram illustrates the transformation of a class definition. On the left, a code block contains a class named 'City' with a private attribute 'name' and a constructor that initializes it. A large grey arrow points from this block to the right, indicating the transformation process. On the right, the resulting code is shown: the class 'City' now has a constructor that takes a parameter 'name' of type 'string'. This parameter is annotated with the same 'private' visibility modifier as the original attribute.

```
class City {  
  
    constructor(private name: string){  
    }  
}
```

Clases y tipos de objetos

- Los **objetos** de una **clase** se pueden **asignar** a **variables** definidas con un **tipo de objeto** si cumplen con su **estructura**

```
type Named = {  
    name: string;  
}  
  
class Person {  
    name: string;  
    age: number;  
}  
  
+ function log(obj: Named){ ...  
}  
  
log({ name: "Obj12"});  
log(new Person());
```

Person cumple la estructura de **Named**

Person es assignable a **Named**

Interfaces en TypeScript

- Un **tipo de objeto** se parece mucho a un **interfaz** de lenguajes como Java o C#
- Podemos declarar un **tipo de objeto** como un **interfaz**

```
type Named = {  
    name: string;  
}
```

```
interface Named {  
    name: string;  
}
```

Interfaces en TypeScript

- Las **clases** pueden **implementar interfaces**
- TS nos **avisa** si la clase **no cumple** la estructura del interfaz

```
interface Named {  
    name: string;  
}  
  
class Person implements Named {  
    nam: string;  
    age: number;  
}
```

ERROR: Class 'Person'
incorrectly implements
interface 'Named'.

Property 'name' is missing in
type 'Person'

Tipos de las Librerías



Bibliotecas?

Tipos de las librerías

- Lo **ideal** sería que todas las bibliotecas que usamos tuvieran los **tipos ya definidos**
- TypeScript viene de **serie** con librerías del **navegador**

dom webworker es5 es6

- Podemos **usar** las que queramos dependiendo del **entorno de ejecución**

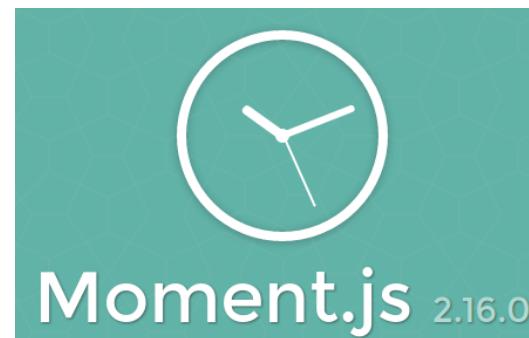
Librerías de terceros?

- La comunidad ha definido el tipo de **2344 librerías**



socket.io

UNDERScore.js



Lo

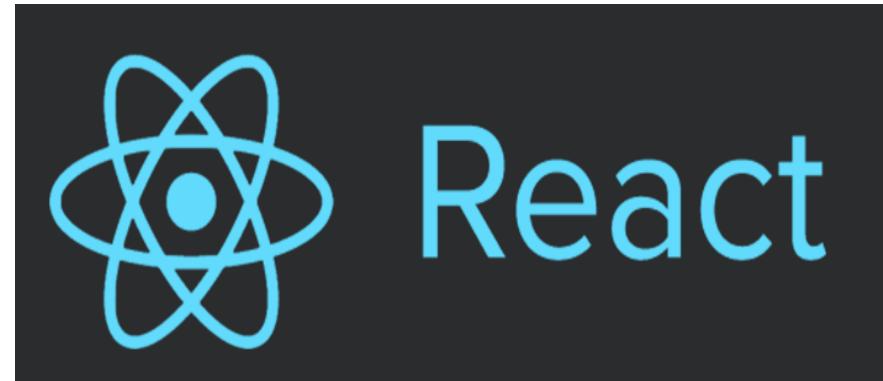
<https://www.npmjs.com/~types>

Librerías de terceros?

- La comunidad ha definido el tipo de **2344 librerías**



express



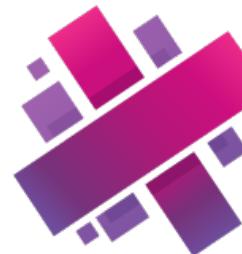
<https://www.npmjs.com/~types>

Librerías de terceros?

- Algunas incluso están **implementadas en TS**



ReactiveX

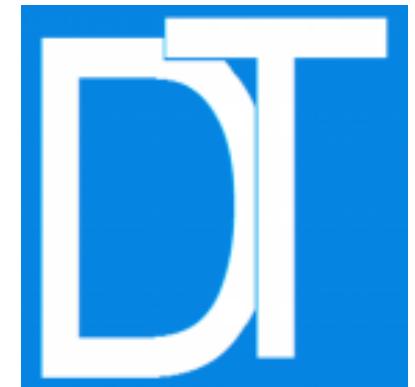


aurelia

<https://www.npmjs.com/~types>

Y si la librería no tiene tipos?

- Puedes usarla como en **JavaScript** (**sin ninguna ayuda** de TS)
- O puedes ir **definiendo** los tipos que te **vayan haciendo falta** partiendo de la documentación
- Y si la definición es de calidad, puedes **compartirla con la comunidad** en **DefinitelyTyped**



<http://definitelytyped.org/>

**Y mucho
más...**



Otras características de TypeScript

- **Lenguaje**
 - Generics en clases y funciones
 - Async / Await (ES7)
 - Anotaciones (ES7)
- **Herramientas**
 - Generar ES5 (evitando el uso de Babel)
 - Transpilador en el browser (prototipado)



Interesante...
Y quién ha hecho
esto?

File Edit Search Run Compile Debug Tools Options Window Help

[■] HELLOWOR.PAS 1=[▲]

```
program HelloWorld;
var
  i:String;
begin
  WriteLn('TechApple.Net Turbo Pascal 7_World');
  ReadIn(i);
end.
```



5:39

F1 Help | Welcome to Turbo Pascal. Press Enter to close this dialog box

Quién ha hecho TypeScript?

- Un proyecto desarrollado en **abierto en GitHub**
- Liderando **Anders Hejlsberg**
 - (Turbo Pascal, Delphi, C#)
- Financiado por **Microsoft**
- Licencia **Apache 2**

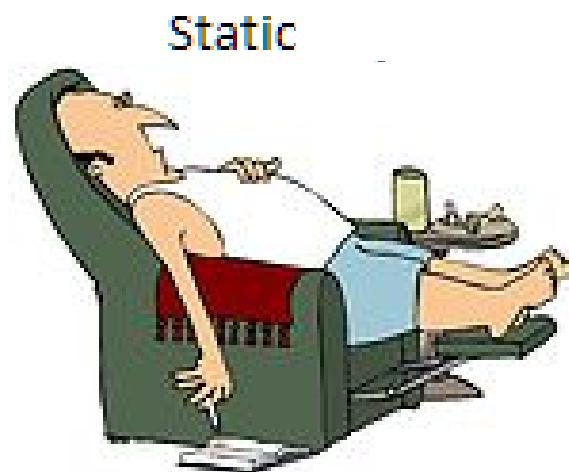


<http://www.typescriptlang.org/>

<https://www.gitbook.com/book/basarat/typescript/details>

Volvemos a la eterna
discusión de qué lenguajes son mejores?

Tipado estático vs dinámico?



El eterno debate

- **Static vs Dynamic**
 - <https://pchiusano.github.io/2016-09-15/static-vs-dynamic.html>
- **Static typing will not save us from bugs**
 - <http://www.drmaciver.com/2016/10/static-typing-will-not-save-us-from-broken-software/>
- **Types**
 - <https://gist.github.com/garyberhardt/122909856b570c5c457a6cd674795a9c>
- **What to know before debating type systems**
 - <http://blogs.perl.org/users/ovid/2010/08/what-to-know-before-debating-type-systems.html>
- **Unit testing isn't enough. You need static typings too**
 - <http://evanfarrer.blogspot.com.es/2012/06/unit-testing-isnt-enough-you-need.html>



Uncle Bob

Robert C. Martin

You **don't need static** type checking if
you have **100% unit test** coverage

As TDD becomes more accepted, **dynamic languages** will be preferred

Encuesta totalmente sesgada hacia **Javeros torpes** como yo ;)



Micael Gallego

@micael_gallego

Prefieres un lenguaje con tipado estático o uno con tipado dinámico?

60% Estático>Compiler=amigo

25% Dinámico>+Libertad +Facil

6% Me da igual

9% Depende. Tuit respuesta

162 votos • Resultados finales

https://twitter.com/micael_gallego/status/787916015648768000

Según Anders, TS no es ni tipado estático ni dinámico



Anders Hejlsberg
@ahejlsberg



Seguir

@tomasz_ducin TypeScript is "gradually typed", so really in a new category. Aim is to catch mistakes while still allowing common idioms.

<https://twitter.com/ahejlsberg/status/792762247239995392?s=09>

¿TypeScript o JavaScript?

**Depende del programador
y del proyecto**



Hay quien considera que TypeScript lo
complica todo innecesariamente
(y es comprensible)



Micael Gallego @micael_gallego · 14 oct. 2015

@pinchito El compilador y el IDE son tus amigos... y con tipos te pueden ayudar mucho más ;) aunque cada vez soy más abierto de mente



Pinchita

@pinchito



Siguiendo

@micael_gallego El compilador y el IDE son los típicos amigos gorriones que te comen la vida. Tus auténticos amigos son el intérprete y vi XD



<https://twitter.com/pinchito/status/654293829013643265>

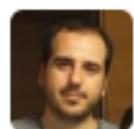


Carlos Hernández
@CodingCarlos



Seguir

@micael_gallego @carlosazaustre
@codemotion_es me valdría con entender por
qué complicar el lenguaje me facilita la vida
jajajajaja



Micael Gallego @micael_gallego · 26 sept.

@CodingCarlos @carlosazaustre @codemotion_es justo de eso vamos a hablar.
Te gustan los tipos?



...



Carlos Hernández @CodingCarlos · 26 sept.

@micael_gallego pues tengo un conflicto moral, porque aunque se que son el bien, me parecen un atraso en flexibilidad y tiempo



...



Micael Gallego @micael_gallego · 27 sept.

@CodingCarlos No son el bien ni el mal. Después de años y años parece que es una cuestión de gustos y depende mucho de uno mismo

<https://twitter.com/CodingCarlos/status/780536033712738305>

**TypeScript es un nuevo enfoque,
tu JavaScript de siempre pero con
una red de seguridad**

Cada vez hay más **JavaScripters**
convencidos que ven
bondades en **TypeScript**



javier.velez.reyes

@javiervelezreye



Siguiendo

@micael_gallego @codemotion_es ajajja pues
he de decirte que le estoy cogiendo el gusto a
typescript ;)





Max Lynch @maxlynch · 20 may.

I don't work on Ionic 2 anymore (we have better devs for that), but been dogfooding and love TypeScript and ng2 (and Ionic 2 of course 🔎)



15

...



Max Lynch
@maxlynch



Siguiendo

Honestly, I was resistant to TypeScript for a long time. It felt *more* complex than ES6. But now, I think it reduces the complexity of it



<https://twitter.com/maxlynch/status/733669142985908225>

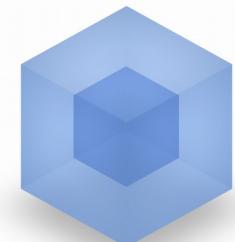


Sean T. Larkin
Webpack core developer

Sustaining webpack for the future: Part 1

What will webpack do with my sponsorship?

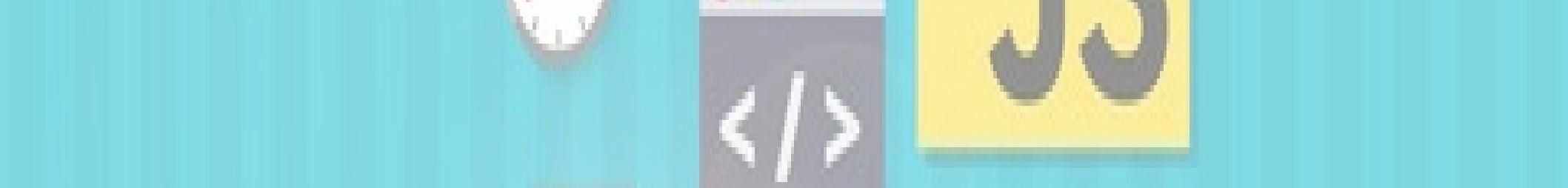
- Support for taking the time to convert the entire project to TypeScript.
We believe that TypeScript allows users to better write scaled and complex code with rich APIs (like webpack). If we can make code easier for you to write, then its worth our time to invest in for your contributions.



<https://medium.com/webpack/sustaining-webpack-for-the-future-part-1-32bea7f9e8a2#.7mbnncr89>



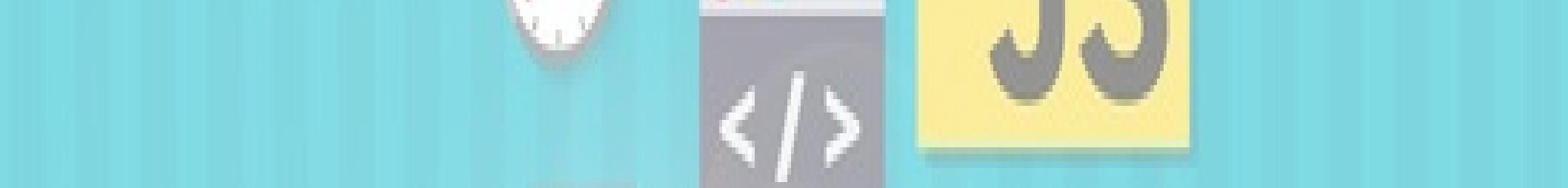
La experiencia de un **JavaScripter** cualquiera...



Llevo trabajando desde hace mucho tiempo con lenguajes dinámicos (**JavaScript**, **Clojure**, **Ruby**) sin verificación de tipos ni autocompletado en mi editor.



Mi cerebro ha aprendido a **trabajar sin esas cosas**. Sabía que estas funcionalidades son las ventajas que aporta TypeScript, pero no **eran para mí**. No creo que **valga la pena** añadir nuevas herramientas y la gestión de los tipos de las librerías.



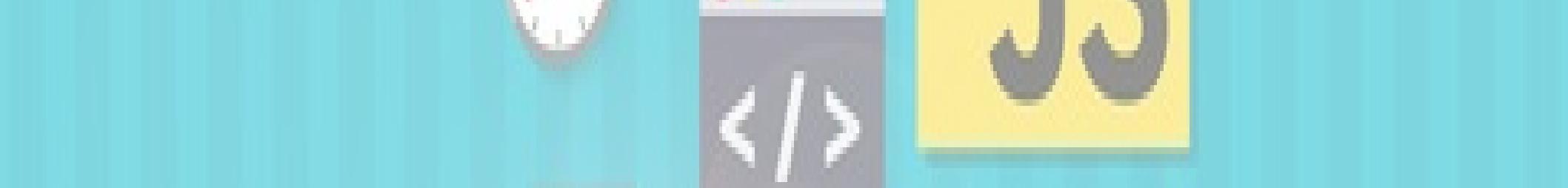
Pero **empecé** a usar **TypeScript** en algunos proyectos y algo ocurrió. Me **olvidaba** de que **TypeScript** estaba ahí.

Seguía programando **como había hecho** con **JavaScript** durante años.



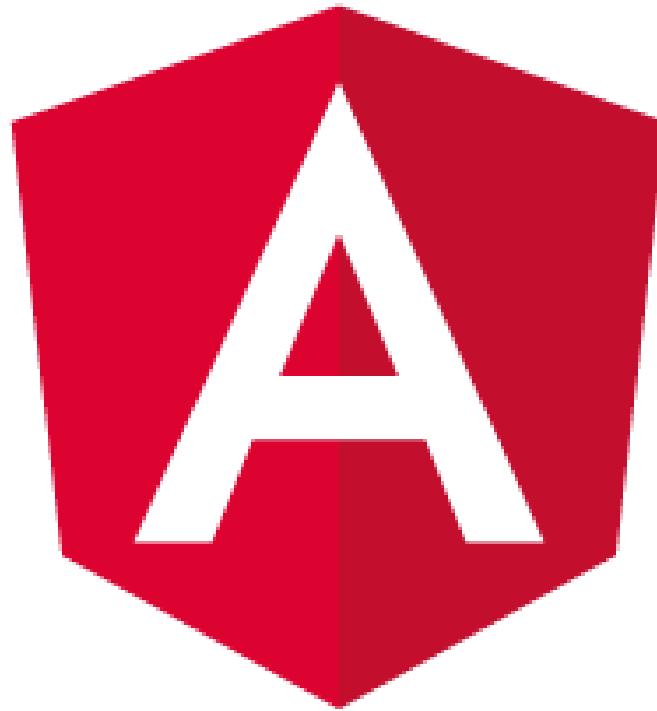
La gente dice que **TypeScript** es como una
pequeña **capa encima de JavaScript**, pero
hasta que no lo pruebas no te das cuenta.

Es sólo **JavaScript** pero con cinturón de
seguridad.

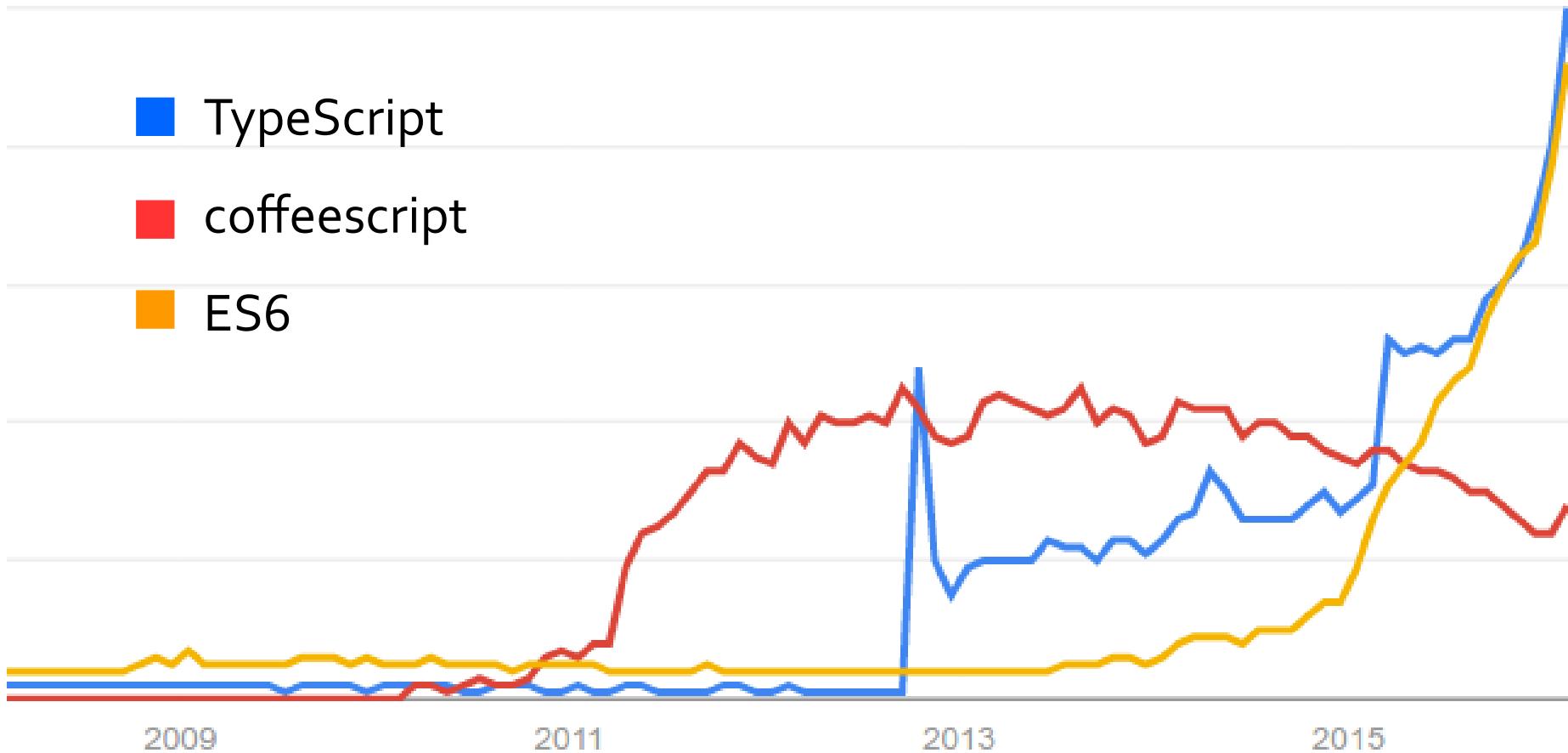


He empezado a **detectar problemas en mis proyectos JavaScript** donde he pensado “podríamos haber detectado este error antes si hubiéramos tenido una **interfaz TypeScript**”

**Cuál es la adopción de
TypeScript?**



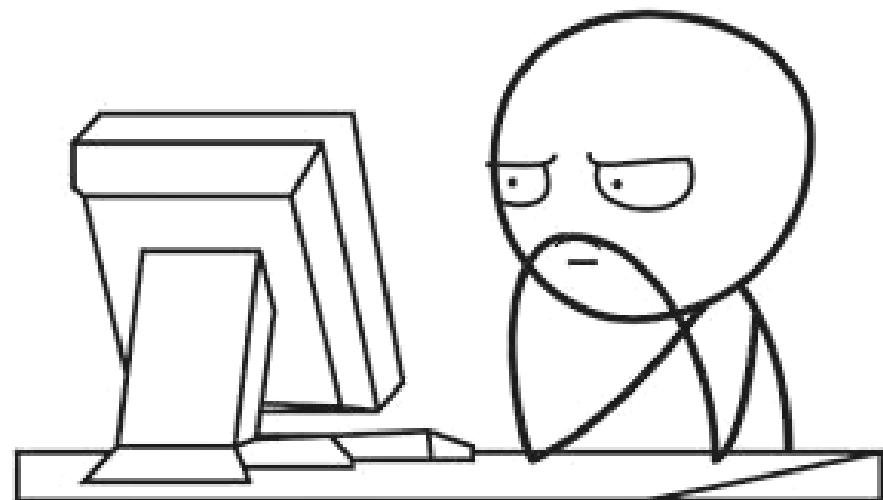
Es el lenguaje recomendado por **Google**
para aplicaciones **Angular 2**



Google Trends

estarás pensando...

TypeScript es para mí?



Depende de
lo torpe que
seas

