Campus

# Introducción a TypeScript

Teamcamp

# Typescript

## **Introducción**:

Es un lenguaje **tipado** construido sobre Javascript

Añade **sintaxis** a la ya existente en Javascript

Es un **superset** de Javascript. Javascript con tipos

Se **transpila/compila** a Javascript para poder usarse donde se usa Javascript

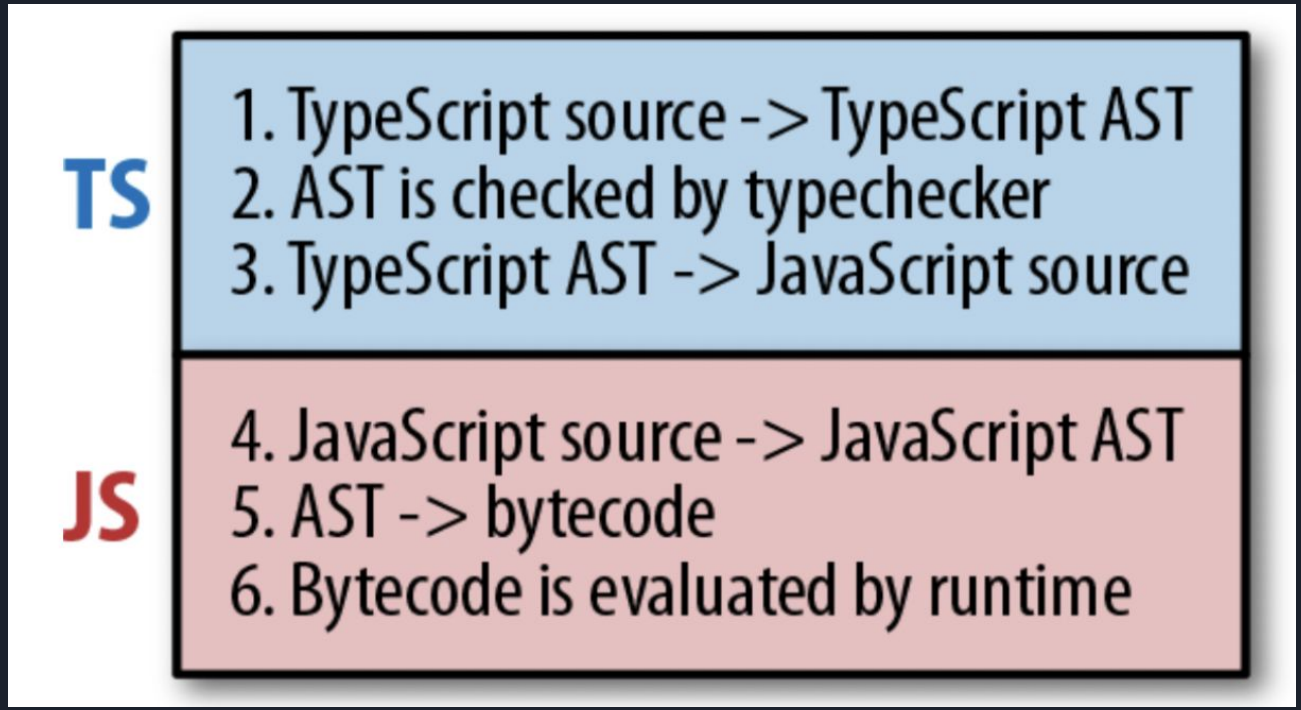
Implementa un motor de **inferencia de tipos** que nos permite añadir semántica a nuestro código sin necesidad de escribir código adicional para expresarla

Se puede añadir de forma **progresiva** a nuestra base de código existente

Ayuda a escribir código más **robusto** y **menos propenso a errores**

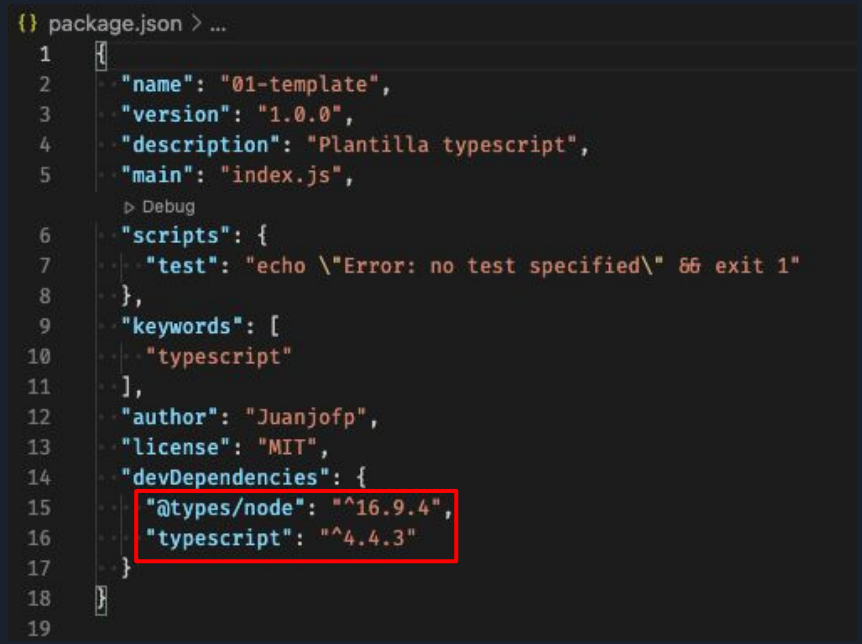
# Typescript

**Transpilación/Compilación:**



# Typescript

**Instalación**:
npm init
npm install --save-dev typescript @types/node
npx tsc --init



# Typescript

Instalación: **eslint**

package.json > { } scripts

```
{
  "name": "01-template",
  "version": "1.0.0",
  "description": "Plantilla typescript",
  "main": "index.js",
  ▷ Debug
  "scripts": {
    "lint": "eslint . --ext .ts",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [
    "typescript"
  ],
  "author": "Juanjofp",
  "license": "MIT",
  "devDependencies": {
    "@types/node": "^16.9.4",
    "@typescript-eslint/eslint-plugin": "^4.31.1",
    "@typescript-eslint/parser": "^4.31.1",
    "eslint": "^7.32.0",
    "eslint-config-prettier": "^8.3.0",
    "eslint-plugin-filenames": "^1.3.2",
    "typescript": "^4.4.3"
  }
}
```

.eslintrc > ...

```
{
  "parser": "@typescript-eslint/parser",

  "env": {
    "browser": false,
    "node": true,
    "jest": true,
    "es6": true
  },

  "plugins": ["@typescript-eslint", "filenames"],

  "parserOptions": {
    "sourceType": "module",
    "project": ["./tsconfig.json", "./test/tsconfig.json"]
  },

  "extends": [
    "plugin:@typescript-eslint/eslint-recommended",
    "plugin:@typescript-eslint/recommended",
    "eslint-config-prettier"
  ],

  "rules": {
    "filenames/match-exported": 0,
    "block-scoped-var": [0],
    "brace-style": [0, "stroustrup", { "allowSingleLine": true }]
  }
}
```

# Typescript

Instalación: **prettier**

{} .prettierrc > ...

```
{
    "arrowParens": "avoid",
    "bracketSpacing": true,
    "embeddedLanguageFormatting": "auto",
    "htmlWhitespaceSensitivity": "css",
    "insertPragma": false,
    "jsxBracketSameLine": false,
    "jsxSingleQuote": true,
    "printWidth": 80,
    "proseWrap": "preserve",
    "quoteProps": "as-needed",
    "requirePragma": false,
    "semi": true,
    "singleQuote": true,
    "tabWidth": 4,
    "trailingComma": "none",
    "useTabs": false,
    "vueIndentScriptAndStyle": false
}
```

{} package.json > ...

```
{
    "name": "01-template",
    "version": "1.0.0",
    "description": "Plantilla typescript",
    "main": "index.js",
    ▷ Debug
    "scripts": {
        "lint": "eslint . --ext .ts",
        "format": "prettier --write \"src/**/*.+(js|jsx|ts|tsx|json)\"",
        "test": "echo \"Error: no test specified\" && exit 1"
    },
    "keywords": [
        "typescript"
    ],
    "author": "Juanjofp",
    "license": "MIT",
    "devDependencies": {
        "@types/node": "^16.9.4",
        "@typescript-eslint/eslint-plugin": "^4.31.1",
        "@typescript-eslint/parser": "^4.31.1",
        "eslint": "^7.32.0",
        "eslint-config-prettier": "^8.3.0",
        "eslint-plugin-filenames": "^1.3.2",
        "prettier": "^2.4.1",
        "pretty-quick": "^3.1.1",
        "typescript": "^4.4.3"
    }
}
```

# Typescript

Instalación: **tsconfig**

```
→  01-template npx tsc --init
message TS6071: Successfully created a tsconfig.json file.
```

tsconfig.json > ...

```json
{
    "include": ["./src/**/*.ts"],
    "exclude": ["node_modules"],
    "compilerOptions": {
        "module": "commonjs",
        "target": "ES2015",
        "noImplicitAny": true,
        "declaration": true,
        "resolveJsonModule": true,
        "esModuleInterop": true,
        "sourceMap": true,
        "outDir": "build",
        "watch": false,
        "lib": ["es2018"],
        "moduleResolution": "node",
        "noUnusedLocals": true,
        "strict": true
    }
}
```

{} package.json > ...

```json
{
    "name": "01-template",
    "version": "1.0.0",
    "description": "Plantilla typescript",
    "main": "build/main.js",
    ▷ Debug
    "scripts": {
        "build": "tsc",
        "start": "node build/main.js",
        "start:dev": "ts-node src/main.ts --inspect=0.0.0.0:9229",
        "start:dev:watch": "nodemon -L src/main.ts --inspect=0.0.0.0:9229 --exec ts-node",
        "lint": "eslint . --ext .ts",
        "format": "prettier --write \"src/**/*.+(js|jsx|ts|tsx|json)\"",
        "test": "echo \"Error: no test specified\" && exit 1"
    },
    "keywords": [
        "typescript"
    ],
    "author": "Juanjofp",
    "license": "MIT",
    "devDependencies": {
        "@types/node": "^16.9.4",
        "@typescript-eslint/eslint-plugin": "^4.31.1",
        "@typescript-eslint/parser": "^4.31.1",
        "eslint": "^7.32.0",
        "eslint-config-prettier": "^8.3.0",
        "eslint-plugin-filenames": "^1.3.2",
        "prettier": "^2.4.1",
        "pretty-quick": "^3.1.1",
        "ts-node": "^10.2.1",
        "typescript": "^4.4.3",
        "nodemon": "^2.0.12"
    }
}
```

# Typescript

Configuración: **tsconfig**

## files

Lista de ficheros a incluir en la compilacion

```
{
    "compilerOptions": {},
    "files": ["core.ts", "sys.ts", "types.ts", "scanner.ts"]
}
```

## extends

Heredar una configuracion base sobre la que aplicar cambios

```
{
    "extends": "./configs/base",
    "compilerOptions": {
        "strictNullChecks": false
    },
    "files": ["main.ts", "supplemental.ts"]
}
```

## include

Array con la lista de ficheros o patron que identifique a los ficheros que se van a incluir en la compilacion

```
{
    "include": ["src/**/*", "tests/**/*"]
}
```

## exclude

Array con la lista de ficheros o patron que identifique a los ficheros que se van a excluir de la lista de include

```
{
    "exclude": ["node_modules"]
}
```

# Typescript

Configuración: **tsconfig**

compilerOptions

outDir

Especifica el dirtectorio donde se guardarán los resultados de la compilación

module

Especifica el tipo de módulos que estamos usando en el programa. Si queremos generar código ES5 o ES3 usaremos CommonJS

moduleResolution

Especifica como debe hacer la resolución de módulos, node para Node.js y classic para el resto de tipos

target

Versión de JAvascript a la que se va a compilar

lib

Especifica las bibliotecas y apis que queremos tener disponibles, por ejemplo ES2018 nos permite usar finally en las promesas

declaration

Genera un fichero de tipos para cada fichero encontrado

noImplicitAny

Avisa cunado un tipo no ha sido declarado y no es capaz de inferirlo, desactivado asignará any a esta clase de tipos

noUnusedLocals

Avisa de variables locales que no se están usando

resolveJSONModule

Permite importar un fichero json

# Typescript

Configuración: **tsconfig**

**strict**

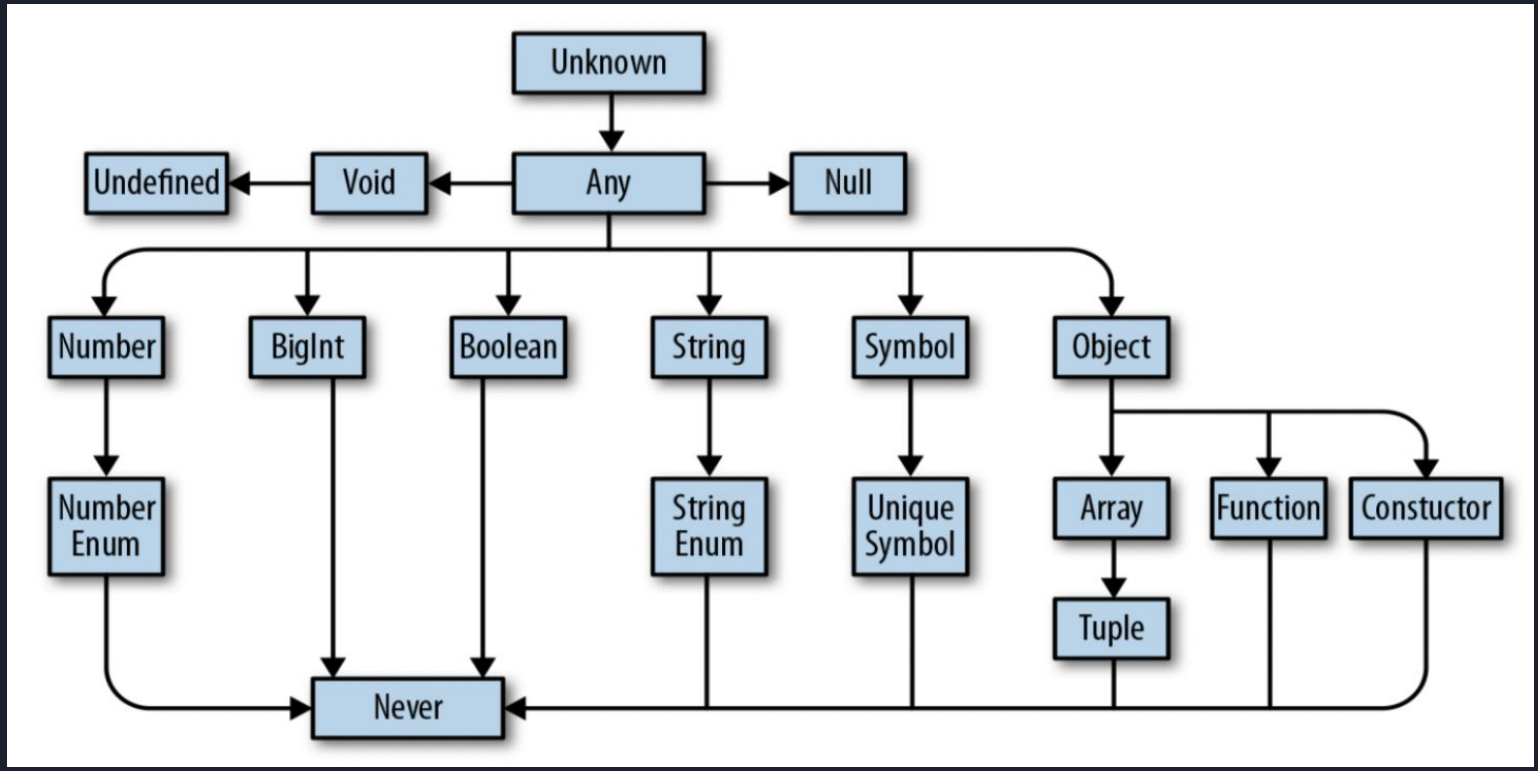
Habilita una amplia gama de comportamientos de verificación de tipos que dan como resultado mayores garantías de corrección del programa. Activar esto equivale a habilitar todas las opciones de la familia de modo estricto. Las futuras versiones de TypeScript pueden introducir un control más estricto bajo esta propiedad, por lo que las actualizaciones de TypeScript pueden resultar en nuevos errores de tipo en su programa

## Ejercicios

- Crear un proyecto de Node.js preparado para soportar TypeScript
- Añadir ESLint a la configuración
- Añadir Prettier a la configuración
- Añadir Nodemon a la configuración
- Genera el código JavaScript en el directorio build

# Typescript

## Tipos



# Typescript

## Tipos primitivos

**any**: No aplica ninguna restricción. Es el tipo por defecto, es intercambiable por cualquier tipo
**unknown**: Es el más restrictivo de todos los tipos, es intercambiable por cualquier tipo
**boolean**: Es el conjunto formado por true y false
**number**: Es el conjunto de todos los números
**bigint**: Permite trabajar con enteros grandes evitando los efectos del redondeo
**string**: Es el conjunto de todas las cadenas de caracteres
**symbol**: Es el tipo para los valores Symbol de JS

Cualquier tipo primitivo con **const** se convierte en un **Tipo Literal**

# Typescript

## Tipos primitivos

```ts
// [let|const|var] variable: TIPO = valor;
let name: String = 'Juanjo'; //string
let city = 'Murcia'; // string
let x = true; // boolean
const z = true; // true
let a = 1234; // number
var b = 1234; // number
const c = 1234; // 1234
let d = a > b; // boolean
let e: number = 100; // number
let f: 2.5 = 2.5; // 2.5
let g: 2.5 = 3.3; // Error el valor 3.3 no es asignable al tipo 2.5
```

Por lo general dejaremos que sea TS quien infiera los tipos de valores primitivos

# Typescript

## Tipados

**Tipado Estructural**: Definido por la forma del objeto. **Typescript**
**Tipado Nominal**: Definido por el nombre del objeto. **Java**

## Objetos

**object**: Restringe a ser un objeto no nulo
**object literal**: Restringe a la forma con la que es declarado
**class**: Restringe a la forma del objeto declarado

# Typescript

## Objetos

```ts
let p1: object = { x: 0, y: 0 }; // {}
console.log(p1.x, p1.y);

let p2 = { x: 0, y: 0 }; // {x: number; y: number;}
console.log(p2.x, p2.y);

class Point3 {
    x = 0;
    y = 0;
}
let p3 = new Point3(); // {x: number; y: number;}
console.log(p3.x, p3.y);

let p4: Point3 = {x: 0, y: 0};
let p5: {x: number; y: number;} = new Point3();
let p6: Point3 = {x: 0};
let p7: Point3 = {x: 0, y: 0, z: 0};
```

```ts
// Evitar tipar con {}
let danger = {}; // {}
danger = {a: 34};
danger = 2;
danger = () => undefined;
danger = [];
danger = 'DANGER!!!';
danger = null;
danger = undefined;
```

# Typescript

## Objetos

```ts
let p1: object = { x: 0, y: 0 }; // {}
console.log(p1.x, p1.y);

let p2 = { x: 0, y: 0 }; // {x: number; y: number;}
console.log(p2.x, p2.y);

class Point3 {
    x = 0;
    y = 0;
}
let p3 = new Point3(); // {x: number; y: number;}
console.log(p3.x, p3.y);
```

```ts
function printPoints(point: Point3) {
    console.log(point.x, point.y);
}
printPoints(p1);
printPoints(p2);
printPoints(p3);
```

# Typescript

## Objetos

```
// Propiedades opcionales ?
let film: {
    title: string;
    description?: string;
    year?: number;
}
film = {title: 'Dune', description: 'No Spoiler', year: 2021};
film = {title: 'Black Widow', description: 'BW deserves this film'};
film = {title: 'Spiderman'};
film = {title: 'Dr. Strange 2', format: 'DVD'}
```

```
// Propiedades de solo lectura
let user: {readonly id: string; name: string};
user = {id: 'jjfp', name: 'Juanjo'};
user.id = 'jrfp';
```

```
// Propiedades desconocidas (index signature)
let shopCart: {[key: string]: number};
shopCart = {potatoes: 3, milk: 5, bananas: 6};
shopCart = {};
shopCart = {beans: '3kg'};
```

```
// Propiedades desconocidas (built in Record)
let shopCart: Record<string, number>;
shopCart = {potatoes: 3, milk: 5, bananas: 6};
shopCart = {};
shopCart = {beans: '3kg'};
```

# Typescript

## Alias (type)

Permite declarar un nuevo tipo

```
type COOD = number;
type Point3D = {
    x: COOD;
    y: COOD;
    z: COOD;
}

const center: Point3D = {x: 0, y: 0, z: 0};
const topLeft: Point3D = {x: -10, y: -10, z: 0};
```

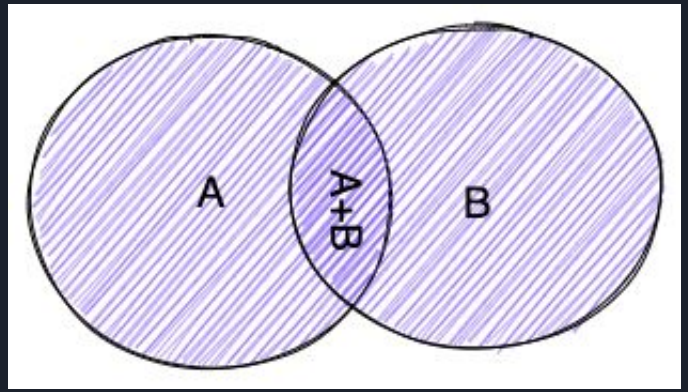
Ámbito y shadowing

```
type COORD = number;
type COORD = string;
```

```
type COORD = number;
if(scene === '3D') {
    type Point = {
        x: COORD;
        y: COORD;
        z: COORD;
    }
    let center: Point = {x: 0, y: 0, z: 0};
    console.log(center)
}
else {
    type Point = {
        x: COORD;
        y: COORD;
    }
    let center: Point = {x: 0, y: 0};
    console.log(center)
}
```

# Typescript

## **Uniones** (A | B)



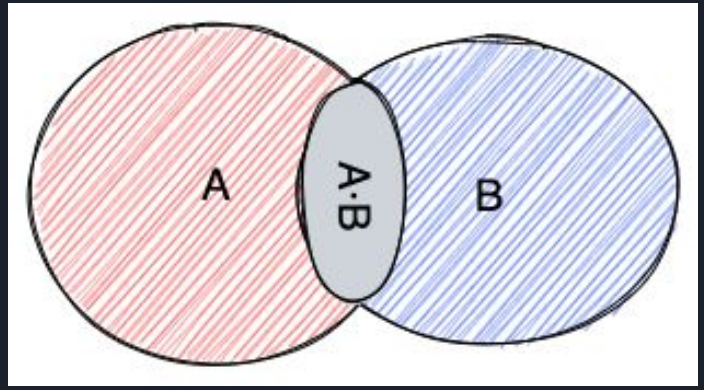

Es la unión dos o más tipos
Acepta conjuntos de A o de B o de A+B

```
type Humanoid = { arms: number; legs: number; heads: number };
type Squidoid = { tentacles: number; heads: number };
type Alien = Humanoid | Squidoid;

const tanos: Alien = { heads: 1, arms: 2, legs: 2 };
const kang: Alien = { heads: 1, tentacles: 8 };
const octavius: Alien = { heads: 1, arms: 2, legs: 2, tentacles: 4 };
const invalidAlien: Alien = {heads: 2, arms: 2};
const me: Alien = {heads: 1, arms: 2, legs: 2, hands: 2};
```

# Typescript

## **Intersecciones** (A **&** B)




Es la intersección de dos o más tipos
Acepta el conjuntos de A·B

```
type Humanoid = { arms: number; legs: number; heads: number };
type Squidoid = { tentacles: number; heads: number };
type Alien = Humanoid & Squidoid;

const tanos: Alien = { heads: 1, arms: 2, legs: 2 };
const kang: Alien = { heads: 1, tentacles: 8 };
const octavius: Alien = { heads: 1, arms: 2, legs: 2, tentacles: 4 };
const invalidAlien: Alien = {heads: 2, arms: 2};
const me: Alien = {heads: 1, arms: 2, legs: 2, hands: 2};
```

# Typescript

## Ejemplo

```ts
type Humanoid = { arms: number; legs: number; heads: number };
type Squidoid = { tentacles: number; heads: number };
type Alien = Humanoid | Squidoid;
type AlienCombined = Humanoid & Squidoid;

const tanos: Alien = { heads: 1, arms: 2, legs: 2 };
const kang: Alien = { heads: 1, tentacles: 8 };
const octavius: AlienCombined = { heads: 1, arms: 2, legs: 2, tentacles: 4 };
const invalidAlien: Alien = { heads: 2, arms: 2 };
const me: Alien = { heads: 1, arms: 2, legs: 2, hands: 2 };
```

```ts
function printAlien(alien: Alien) {
    let description = `Heads: ${alien.heads}`;
    if (isHumanoid(alien))
        description += `, arms: ${alien.arms}, legs: ${alien.legs}`;
    if (isSquidoid(alien)) description += `, tentacles: ${alien.tentacles}`;
    console.log(description);
}

printAlien(tanos);
printAlien(kang);
printAlien(octavius);

function printAlienCombined(alien: AlienCombined) {
    return `Heads: ${alien.heads}, arms: ${alien.arms}, legs: ${alien.legs}, tentacles: ${alien.tentacles}`;
}

printAlienCombined(tanos);
printAlienCombined(kang);
printAlienCombined(octavius);
```

# Typescript

## Arrays

TS puede inferir el tipo de array a partir de su inicialización
Especificamos su tipo usando **T[]** o **Array<T>**
Es aconsejable mantener los arrays heterogéneos

```ts
let myArray = [1, 2, 3]; // number[]
myArray.push(42);
myArray.push('Juanjo');

let mixArray = ['hi', 42, [5], true]; // (string | number | boolean | number[])[]
mixArray.push(42);
mixArray.push('Juanjo');

let anyArray: any[] = []; // any[]
anyArray.push(42);
anyArray.push('Juanjo');

let names: string[] = []; // string[]
names.push('Juanjo');
names.push(42)

let implicitAnyArray = []; // any[]
implicitAnyArray.push(42);
implicitAnyArray.push('Juanjo');
```

```ts
function buildArray() {
    let inferMyType = []; // any[]
    inferMyType.push(true);
    inferMyType.push(42);
    inferMyType.push('Juanjo');
    return inferMyType;
}

let constrainedArray = buildArray(); //(string | number | boolean)[]
```

# Typescript

## Tuplas

Son un subtipo de Array que nos permite aumentar las restricciones sobre los Arrays
Tienen que ser tipados de forma explícita al declararse

```ts
// Array de tamaño fijo
type Coords = [number, number, number];
const houseCoords: Coords = [-1.6343, 37.9422, 675];
```

```ts
// Array de tamaño fijo con valores opcionales
type Coords = [number, number, number?];
const houseCoords: Coords = [-1.6343, 37.9422];
const teideCoords: Coords = [1.6343, 7.9422, 3005];
```

```ts
// Array con valores iniciales fijos
type Node = string;
type Route = [Node, Node, ...Node[]];
const goHome: Route = ['office', 'home'];
const goHomeAndBuyEgss: Route = ['office', 'market', 'home'];
const goNoWhere: Route = [];
const goOfficeFomOffice: Route = ['office'];
```

```ts
// Inmutabilidad
type Inmutable = readonly string[];
const team: Inmutable = ['Jose', 'Pablo', 'Juan'];
team[2] = 'Juanjo';
team.push('Juanjo');

type InmutableCoords = readonly [number, number];
const someWhere: InmutableCoords = [0, 0];
someWhere[0] = 1;
```

# Typescript

## null, undefined, void, never

**undefined**: Variables que aún no han sido definidas
**null**: Ausencia de valor
**void**: Función que no devuelve nada
**never**: Función que nunca devuelve.

## Enum

Especifican los posibles valores que puede adoptar un tipo
Pueden mapearse a números o a cadenas

```
enum Colors {
    Red = '#CC0010',
    Green = '#00BE54',
    Blue = '#1060AF'
}
const myTextColor = Colors.Red;
console.log(myTextColor); // #CC0010
```

```
// Enums
enum Language {
    English,
    Spanish,
    Russian
}
const myLanguage = Language.Spanish;
const learningLanguage = Language['English'];
const otherLanguage: Language = 2;
otherLanguage === Language.Russian; // true
```

# Typescript

## Funciones

**Podemos tipar los parámetros y el tipo devuelto**
**Podemos hacer uso de valores por defecto, opcionales y parámetros variables**

```typescript
function functionName(
    param1: number,
    param2 = 'param2', // Default Params
    param3?: boolean, // Optional Params
    ...param4: bigint[] // Variable Params
): string {
    return `Param1: ${param1} Param2: ${param2} Param3: ${
        param3 ? param3 : 'Undefined'
    } Param4: ${param4.length}`;
}
```

```typescript
functionName(45);
// Param1: 45 Param2: param2 Param3: Undefined Param4: 0

functionName(45, 'second param');
// Param1: 45 Param2: second param Param3: Undefined Param4: 0

functionName(45, 'second param', true);
// Param1: 45 Param2: second param Param3: true Param4: 0

functionName(45, 'second param', true, 3n, 5n, 7n);
// Param1: 45 Param2: second param Param3: true Param4: 3
```

# Typescript

## Funciones

**Podemos indicar el uso de ‘this’ en la función**

```
function whoAmI(this: { me: string }) {
    console.log('My name is ', this.me);
}
// The 'this' context of type 'void' is not assignable
// to method's 'this' of type '{ me: string; }'
whoAmI();
const itsMe = { me: 'Juanjo', whoAmI };
itsMe.whoAmI();
whoAmI.call({me: 'Juanjo'});
```

# Typescript

## Funciones

### Tipado de la firma de funciones

```
// Tipar funciones
type AddFunction = (a: number, b: number) => number;
const add: AddFunction = (a, b) => a + b;
add(3, 5);

type IncrementTotal = (total: number, increment: number) => number;
type SumArrayNumbers = (values: number[]) => number;

const incrementTotal: IncrementTotal =  (total, increment) => total += increment;
const sumArrayNumber: SumArrayNumbers = function (values) {
    return values.reduce(incrementTotal);
}
sumArrayNumber([1, 2, 3, 4, 5]); // 15
```

# Typescript

## Funciones

### Tipado de funciones

```ts
type TransFormNumber = (a: number, b: number) => number;
function transformArrayNumber(values: number[], predicate: TransFormNumber) {
    return values.reduce(predicate);
}

const addPredicate: TransFormNumber = (a, b) => a + b;
const substractPredicate: TransFormNumber = (a, b) => a - b;
const productPredicate: TransFormNumber = (a, b) => a * b;

const values = [1, 2, 3, 4, 5];
transformArrayNumber(values, addPredicate); // 15
transformArrayNumber(values, substractPredicate); // -13
transformArrayNumber(values, productPredicate); // 120
```

# Typescript

## Funciones

### Tipado de funciones

```
transformArrayNumber(values, (a, b) ⇒ a + b);

const predicate = (a: number, b: number) ⇒ a % b;
transformArrayNumber(values, predicate);

const invalidPredicate = (a: number, b: string) ⇒ a + b;
// Argument of type '(a: number, b: string) ⇒ string'
// is not assignable to parameter of type 'TransFormNumber'.
transformArrayNumber(values, invalidPredicate);
```

# Typescript

## Funciones

### Tipado de funciones

```ts
// Distintas formas de tipar una funcion
type FN1 = (a: number, b?: string) => string;
type FN2 = {
    (a: number, b?: string): string;
    count: number;
};
type FNCtor = new () => User;
type FNCtor2 = {
    new (): unknown;
    count: number;
};
```

```ts
const fnCtor: FNCtor2 = function constructMe(this: User) {
    this.name = 'Juanjo';
    fnCtor.count += 1;
} as unknown as FNCtor2;
fnCtor.count = 0;

new fnCtor();
new fnCtor();
fnCtor.count; // 2
fnCtor(); // Value of type 'FNCtor2' is not callable. Did you mean to include 'new'?
```

# Typescript

## Funciones

**Tipado de funciones dentro de objetos**

```ts
type Node = {
    weight: number;
    right?: Node;
    left?: Node;
    traverse(): void;
    getWeight: () => number;
};
```

```ts
function traverse(this: Node) {
    if (this.left) this.left.traverse();
    if (this.right) this.right.traverse();
}
const root: Node = {
    weight: 0,
    traverse,
    getWeight() {
        return this.weight;
    }
};
```

```ts
const root2: Node = {
    weight: 0,
    traverse() {
        if (this.left) this.left.traverse();
        if (this.right) this.right.traverse();
    },
    getWeight() {
        return this.weight;
    }
};
```

# Typescript

## Funciones

### Sobrecarga de funciones

```ts
// Sobrecarga con declaraciones de funciones
// Definimos los metodos sobrecargados
function asyncSum(a: number, b: number): Promise<number>;
function asyncSum(a: number, b: number, cb: asyncSumCb): void;
function asyncSum(): Promise<number>;

// Definimos la implementacion
function asyncSum(a = 0, b = 0, cb?: asyncSumCb) {
    const result = a + b;
    if (cb) return void cb(result);
    else return Promise.resolve(result);
}

asyncSum(3, 4, result => console.log('Suma:', result));
asyncSum(1, 2).then(result => console.log('Suma:', result));
asyncSum().then(result => console.log('Suma:', result));
```

# Typescript

## Funciones

### Sobrecarga de funciones

```ts
function createElement(tag: 'a'): HTMLAnchorElement;
function createElement(tag: 'canvas'): HTMLAnchorElement;
function createElement(tag: 'table'): HTMLAnchorElement;
function createElement(tag: string): HTMLElement;
function createElement(tag: string): HTMLElement {…
}

const a = createElement('a');
const table = createElement('table');
const canvas = createElement('canvas');
const htlmElement = createElement('div');
```

```ts
class LayerFactory {
    createFeatureLayer(a1: string, a2: number): string;
    createFeatureLayer(a1: number, a2: boolean, a3: string): number;
    createFeatureLayer(
        a1: string | number,
        a2: number | boolean,
        a3?: string
    ): number | string {
        if (typeof a3 === 'string') return a1;
        return a2 as number;
    }
}
```

# Typescript

## Genéricos

```
type Filter ={
    (array: number[], f: (item: number) => boolean): number[];
    (array: string[], f: (item: string) => boolean): string[];
    (array: object[], f: (item: object) => boolean): object[];
}
const filter: Filter = (array: any[], f: (item: any) => boolean): any[] => {
    const filtered = [];
    for (let i = 0; i < array.length; i++) {
        if (f(array[i])) filtered.push(array[i]);
    }
    return filtered;
}

const names = ['Juanjo', 'Juan', 'Jose', 'Pablo'];
filter(names, name => name.startsWith('J'));

const marks = [8, 5, 3, 9, 5];
filter(marks, mark => mark.);
```

```
★ toFixed          (method) Number.toFixed(fractionDigits?: nu…
★ toPrecision
toExponential
toLocaleString
toString
valueOf
```

# Typescript

## Genéricos

```
function filter(array: number[], f: (item: number) ⇒ boolean): number[];
function filter(array: string[], f: (item: string) ⇒ boolean): string[];
function filter(array: object[], f: (item: object) ⇒ boolean): object[];
function filter(array: any[], f: (item: any) ⇒ boolean): any[] {
    const filtered = [];
    for (let i = 0; i < array.length; i++) {
        if (f(array[i])) filtered.push(array[i]);
    }
    return filtered;
}

const names = ['Juanjo', 'Juan', 'Jose', 'Pablo'];
filter(names, name ⇒ name.startsWith('J'));

const people = [{name: 'a', age: 12}, {name: 'b', age: 18}, {name: 'c', age: 21}];
filter(people, (person) ⇒ person.age ⩾ 18);
```

# Typescript

## Genéricos. Funciones

```
type Filter = {
    <T>(array: T[], predicate: (item: T) => boolean): T[];
}
const filter: Filter = (array, predicate) => {
    const filtered= [];
    for(let i = 0; i < array.length; i++) {
        if(predicate(array[i])) filtered.push(array[i]);
    }
    return filtered;
};

const people = [{name: 'a', age: 12}, {name: 'b', age: 18}, {name: 'c', age: 21}];
filter(people, (person) => person.| >= 18);
```

```
age                    (property) age: number
name
```

# Typescript

## Genéricos. Funciones

```
function filter<T>(array: T[], predicate: (item: T) => boolean): T[] {
    const filtered= [];
    for(let i = 0; i < array.length; i++) {
        if(predicate(array[i])) filtered.push(array[i]);
    }
    return filtered;
};

const people = [{name: 'a', age: 12}, {name: 'b', age: 18}, {name: 'c', age: 21}];
filter(people, (person) => person.age >= 18);
```

```
const filter = <T>(array: T[], predicate: (item: T) => boolean): T[] => {
    const filtered= [];
    for(let i = 0; i < array.length; i++) {
        if(predicate(array[i])) filtered.push(array[i]);
    }
    return filtered;
};

const people = [{name: 'a', age: 12}, {name: 'b', age: 18}, {name: 'c', age: 21}];
filter(people, (person) => person.age >= 18);
```

# Typescript

## Genéricos. Múltiples marcas

```
function map<T, U>(array: T[], transform: (item: T) => U): U[] {
    const result: U[] = [];
    for(let i = 0; i < array.length; i++){
        result.push(transform(array[i]));
    }
    return result;
}
```

```
function map<number, string>(array: number[], transform: (item: number) => string): string[]
map([1, 2, 3, 4], (item) => `Number ${item}`);
```

```
function map<string, number>(array: string[], transform: (item: string) => number): number[]
map(['1', '2', '3'], (item) => parseInt(item))
```

# Typescript

## Genéricos. Anotaciones

```
const p = new Promise((resolve) => {
    resolve(45);
});

p.then((value) => value);
```

```
(parameter) value: unknown
```

```
const p2: Promise<number> = new Promise((resolve) => {
    resolve(45);
});

p2.then((value) => value);
```

```
(parameter) value: number
```

# Typescript

## Genéricos. Alias

```
type Filter<T> = {
    (array: T[], predicate: (item: T) => boolean): T[];
}

type Filter<T> = (array: T[], predicate: (item: T) => boolean) => T[]

Generic type 'Filter' requires 1 type argument(s). ts(2314)

View Problem    No quick fixes available

type Filter<T> = {
    (array: T[], pre
}
const filterString: Filter = (array, predicate) => {
    const filtered= [];
    for(let i = 0; i < array.length; i++) {
        if(predicate(array[i])) filtered.push(array[i]);
    }
    return filtered;
}
```

```
type Filter<T> = {
    (array: T[], predicate: (item: T) => boolean): T[];
}
const filterNumber: Filter<number> = (array, predicate) => {
    const filtered= [];
    for(let i = 0; i < array.length; i++) {
        if(predicate(array[i])) filtered.push(array[i]);
    }
    return filtered;
}
filterNumber([1, 2, 3], (item) => item > 5);
filterNumber(['a', 'b'], (item) => item > 5);
```

```
type Filter<T> = {
    (array: T[], predicate: (item: T) => boolean): T[];
}
const filterString: Filter<string> = (array, predicate) => {
    const filtered= [];
    for(let i = 0; i < array.length; i++) {
        if(predicate(array[i])) filtered.push(array[i]);
    }
    return filtered;
}

filterString(['a', 'b'], (item) => item === item.toUpperCase());
filterString([1, 2, 3], (item) => item > 5);
```

# Typescript

## Genéricos. Composición

```ts
type MyEvent<T> = {
    target: T;
    type: string;
};

type ButtonEvent = MyEvent<HTMLButtonElement | null>;
const myButton: ButtonEvent = {
    target: document.querySelector('#buttonId'),
    type: 'onclick'
};

type DivEvent = MyEvent<HTMLDivElement | null>;
const myDiv: DivEvent = {
    target: document.querySelector('#'),
    type: 'onfocus'
};
```

```ts
type TimeEvent<T> = {
    event: MyEvent<T>;
    from: Date;
    to: Date;
};

const now = new Date();
const twoHoursLater = new Date(now);
twoHoursLater.setHours(now.getHours() + 2);
const myTimeEvent: TimeEvent<HTMLButtonElement | null> = {
    event: myButton,
    from: now,
    to: twoHoursLater
};
```

# Typescript

## Genéricos. Composición

```
function triggerTimeEventEach(
    timeEv: TimeEvent<HTMLButtonElement | null>,
    delay: number
) {
    // Es seguro confiar en TimeEvent
    const element = timeEv.event.target;
    if (element !== null) {
        setTimeout(() => {
            const now = Date.now();
            if (now > timeEv.from.getTime()) {
                element.click();
            }
            if (now < timeEv.to.getTime()) {
                setTimeout(() => triggerTimeEventEach(timeEv, delay));
            }
        }, delay);
    }
}
triggerTimeEventEach(myTimeEvent, 500);
```

# Typescript

## Genéricos. Extender

```
type TimeEvent<T extends HTMLElement | null = HTMLElement | null> = {
    event: MyEvent<T>;
    from: Date;
    to: Date;
};

const myTimeButtonEvent: TimeEvent<HTMLButtonElement | null> = {
    event: myButtonEvent,
    from: now,
    to: twoHoursLater
};

const myTimeHTMLElementEvent: TimeEvent = {
    event: myDivEvent,
    from: now,
    to: twoHoursLater
};
```

# Typescript

## Genéricos. Extender

```ts
function triggerTimeEventEach<T extends HTMLElement | null = HTMLElement>(
    timeEv: TimeEvent<T>,
    delay: number
) {
    // Es seguro confiar en TimeEvent
    const element = timeEv.event.target;
    if (element !== null) {
        setTimeout(() => {
            const now = Date.now();
            if (now > timeEv.from.getTime()) {
                element.click();
            }
            if (now < timeEv.to.getTime()) {
                setTimeout(() => triggerTimeEventEach(timeEv, delay));
            }
        }, delay);
    }
}

triggerTimeEventEach(myTimeButtonEvent, 500);
triggerTimeEventEach(myTimeHTMLElementEvent, 500);
```

# Typescript

## Genéricos. Demo

Anotar una función que reciba:

1. Como primer parámetro una función que puede recibir cualquier número de parámetros de cualquier tipo y devolver un valor de cualquier tipo
2. Tantos parámetros restantes como parámetros reciba la función que recibe como primer parámetro

```
complexFn(
    (one: number, two: string, three: boolean) => console.log(one, two, three),
    1,
    '2',
    true
);

complexFn(
    (array: number[], user: {name: string}) => ({ ...user, array}),
    [1, 2, 3],
    {name: 'Juanjo'}
);
```

```
complexFn(
    (one: number, two: string, three: boolean) => console.log(one, two, three),
    1,
    '2',
    4
);
```

Argument of type 'number' is not assignable to parameter of type 'boolean'. ts(2345)

View Problem  No quick fixes available

# Typescript

## Clases

```
class Animal {
    constructor(name) {
        this.name = name;
    }

    speak() {
        console.log(`${this.name} makes a noise.`);
    }
}

class Dog extends Animal {
    constructor(name) {
        super(name);
    }

    speak() {
        console.log(`${this.name} barks.`);
    }
}

const toby = new Dog("Mitzie");
toby.speak();
```

```
class Animal {
    constructor(protected name: string) {}

    speak() {
        console.log(`${this.name} makes a noise.`);
    }
}

class Dog extends Animal {
    constructor(name: string) {
        super(name);
    }

    speak() {
        console.log(`${this.name} barks.`);
    }
}

const toby = new Dog('Mitzie');
toby.speak();
```

# Typescript

## Clases

```ts
type Color = 'Black' | 'White';
type Row = 'A' | 'B' | 'C' | 'D' | 'E' | 'F' | 'G' | 'H';
type Column = 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8;
```

```ts
// Modificadores en los parametros del constructor
class Position {
    constructor(private row: Row, private column: Column) {}
}

abstract class Piece {
    protected position: Position;
    constructor(private readonly color: Color, row: Row, column: Column) {
        // this.position = new Position(row, column);
    }

    changeColor(color: Color) {
        this.color = color;
    }
}

class King extends Piece {
    getPosition() {
        return this.position;
    }
}
```

```ts
class Position {
    constructor(private row: Row, private column: Column) {}
}

abstract class Piece {
    protected position: Position;
    constructor(private readonly color: Color, row: Row, column: Column) {
        this.position = new Position(row, column);
    }

    abstract canMoveTo(newPosition: Position): boolean;
}

class King extends Piece {
}
```

Non-abstract class 'King' does not implement inherited abstract member 'canMoveTo' from class 'Piece'. ts(2515)

class King

View Problem  Quick Fix... (⌘.)

```ts
const piece = new Piece('White', 'A', 3);
const king = new King('White', 'A', 3);
console.log(king.getPosition());
king.position;
```

```ts
// Realmente privados (JS)
class Rectangle {
    #height = 0;
    #width;

    constructor(height: number, width: number) {
        this.#height = height;
        this.#width = width;
    }
}
```

```ts
class Game {
    private pieces = Game.makePieces();
    private static makePieces() {…
    }
    movePiece(piece: Piece, position: Position) {…
    }
}
```

# Typescript

## Clases

```ts
// Las clases son otra forma de contruir objetos!!!
class Rectangle {
    height = 0;
    width;

    constructor(height: number, width: number) {
        this.height = height;
        this.width = width;
    }

}

function calculateArea(rect: Rectangle) {
    return rect.width * rect.height;
}

calculateArea(new Rectangle(2, 2));
calculateArea({height: 2, width: 2});
```

```
King { color: 'White', position: Position { row: 'A', column: 3 } }


Game {
 pieces: [
   King { color: 'White', position: [Position] },
   King { color: 'Black', position: [Position] }
 ]
}


Rectangle2 { height: 5, width: 5 }
```

# Typescript

## Genéricos. Clases

```ts
class Diccionario<K, V> {
    private _diccionario: Map<K, V>;

    constructor() {
        this._diccionario = new Map<K, V>();
    }

    get(clave: K): V | null {
        return this._diccionario.get(clave) ?? null;
    }

    set(clave: K, valor: V): void {
        this._diccionario.set(clave, valor);
    }

    merge(diccionario: Diccionario<K, V>): void {
        diccionario._diccionario.forEach((valor, clave) => {
            this.set(clave, valor);
        });
    }

    static from<K, V>(diccionario: Diccionario<K, V>): Diccionario<K, V> {
        const resultado = new Diccionario<K, V>();
        resultado.merge(diccionario);
        return resultado;
    }
}
```

```ts
const translate = new Diccionario<string, string>();
translate.set('play', 'jugar');
translate.set('learn', 'aprender');

const periodicTable = new Diccionario<string, number>();
periodicTable.set('H', 1);
periodicTable.set('He', 2);
periodicTable.set('Li', 3);

const clone = Diccionario.from(periodicTable);
clone.get('H');
```

# Typescript

## Interfaces

```ts
type Employed = {
    id: string;
};

type Stuff = Employed & { storeId: number };

type Manager = Stuff & { team: Stuff[] };

const me: Stuff = { id: 'juanjofp', storeId: 544 };

const pablo: Manager = { id: 'pablogp', storeId: 544, team: [me] };
```

```ts
interface Employed {
    id: string;
}

interface Stuff extends Employed {
    storeId: number;
}

interface Manager extends Stuff {
    team: Stuff[];
}

const me: Stuff = { id: 'juanjofp', storeId: 544 };

const pablo: Manager = { id: 'pablogp', storeId: 544, team: [me] };
```

```ts
interface Employed {
    id: string;
}

type Stuff = Employed & {
    storeId: number;
}

interface Manager extends Stuff {
    team: Stuff[];
}

const me: Stuff = { id: 'juanjofp', storeId: 544 };

const pablo: Manager = { id: 'pablogp', storeId: 544, team: [me] };
```

# Typescript

## Interfaces

```typescript
interface Employed {
    readonly id: string;
}

type Stuff = Employed & {
    readonly storeId: number;
};

interface Manager extends Stuff {
    team: Stuff[];
}

// implementar Alias
class Cashier implements Stuff {
    constructor(readonly id: string, readonly storeId: number){}
}

const me = new Cashier('juanjofp', 445);
me.
```

```
id                    (property) Cashier.id: string
storeId
```

```typescript
interface Manager extends Stuff {
    team: Stuff[];
}

// implementar interface
class Boss implements Manager {
    team: Stuff[] = [];
    constructor(readonly id: string, readonly storeId: number){}

    addTeamMember(member: Stuff) {
        this.team.push(member);
    }
}

const juan = new Boss('myboss', 445);
juan.addTeamMember(me);
juan.
```

```
addTeamMemb…    (method) Boss.addTeamMember(member: Stuff):…
id
storeId
team
```

# Typescript

## Interfaces

```ts
// Un alias puede recibir expresiones de tipos
interface Resource {
    id: string;
}
interface Book extends Resource {
    pages: number;
}
interface Video extends Resource {
    duration: number;
}

// interface Content ??? Video | Book

type Content = Book | Video;
```

```ts
class Basket {
    private items: Content[] = [];

    addItemToBasket(item: Content) {
        this.items.push(item);
    }
}
const myOrder = new Basket();
const lotr: Video = {id: 'bestfilmever', duration: 179};
const dune: Book = {id: 'mustberead', pages: 412};
myOrder.addItemToBasket(lotr);
myOrder.addItemToBasket(dune);
```

# Typescript

## Interfaces

```
interface ServiceResponse {
    validate(input: string): string;
    error(input: number): string;
}
```

```
Interface 'MqttResponse' incorrectly extends interface 'ServiceResponse'.
  Types of property 'error' are incompatible.
    Type '(input: string) ⇒ string' is not assignable to type '(input: number) ⇒ string'.
      Types of parameters 'input' and 'input' are incompatible.
        Type 'number' is not assignable to type 'string'. ts(2430)

interface MqttResponse

View Problem   No quick fixes available

interface MqttResponse extends ServiceResponse {
    validate(input: string | number): string;
    error(input: string): string;
}
```

```
type MqttResponse = ServiceResponse & {
    validate(input: string | number): string;
    error(input: string): string;
};

const rabb
    valida
    },
    error(input) {
    }
};

error(input) {
    return (input = '');
}

(parameter) input: string | number
```

# Typescript

## Interfaces

```ts
/** This Fetch API interface represents the response to a request. */
interface Response extends Body {
    readonly headers: Headers;
    readonly ok: boolean;
    readonly redirected: boolean;
    readonly status: number;
    readonly statusText: string;
    readonly type: ResponseType;
    readonly url: string;
    clone(): Response;
}
```

```ts
interface Response {
    trackUuid: string;
}

const serviceResponse: Response = {

};
```

```
body
bodyUsed
clone
formData
headers
json
ok
redirected
status
statusText
text
trackUuid          (property) Response.trackUuid: string
```

```ts
TS service.d.ts > ...
interface User {
    id: string;
}

interface UserService {
    getUser(id: string): User;
}
```

```ts
const service: UserService = {

};
```

```
getUser            (method) UserService.getUser(id: string): U…
getUsersByName
getUsersByRegistrationDate
#endregion                                        Region End
#region                                           Region Start
```

```ts
interface UserService {
    getUsersByName(): User[];
}

interface UserService {
    getUsersByRegistrationDate(date: Date): User[];
}
```

```ts
type UserService = {
    getUsersByName(): User[];
}

type UserService = {
    getUsersByRegistrationDate(date: Date): User[];
}
```

```ts
enum Language {
    Esperanto
}

const local = Language.
```

```
English
Esperanto
Greek
Russian
Spanish
```

# Typescript

## Mixins

```
// Crear Mixins de clases que sean debugables y que tenga un metodo debug
type ClassConstructor<T extends object = object> = new (...args: any[]) => T;
type Debugable = { getDebugValue(): object };
function withDebugMixin<C extends ClassConstructor<Debugable>>(Class: C) {
    return class extends Class {
        debug() {
            const name = Class.name;
            const value = this.getDebugValue();
            return `${name} (${JSON.stringify(value)})`;
        }
    };
}
```

```
class UserService implements Debugable {
    constructor(public user: string) {}

    getDebugValue() {
        return {
            user: this.user
        };
    }
}

const UserServiceWithDebug = withDebugMixin(UserService);
const userService = new UserServiceWithDebug('John');
console.log(userService.debug());
```

# Typescript

## Decoradores

```
@withDebugMixin
class MailSender implements Debugable {
    constructor(public email: string) {}

    getDebugValue() {
        return {
            email: this.email
        };
    }
}
```

```
const mailSender = new MailSender('juanjo@juanjofp.com');

console.log(mailSender.debug());
```

Property 'debug' does not exist on type 'MailSender'. ts(2339)

any

View Problem  Quick Fix... (⌘.)

```
type MaisSenderDecored = MailSender & { debug(): string };
console.log((mailSender as MaisSenderDecored).debug());
```

# Typescript

## Decoradores

- Stage 0 represents an initial idea for addition or change to the specification
- Stage 1 is a formal proposal describing a problem and suggesting a proper solution
- Stage 2 is an initial draft of the proposal specification
- Stage 3 represents the draft when it's almost final but ready for last feedback
- Stage 4 is when the proposal specification completely ready and included within the next edition

```
{
  "compilerOptions": {
    "target": "ES5",
    "experimentalDecorators": true
  }
}
```

# Typescript

## Patrones. Factory

```typescript
export type ShapeNames = 'circle' | 'square' | 'rectangle' | 'triangle';
export abstract class Shape {
    constructor(public name: ShapeNames) {}
    abstract draw(): void;

    static create(shapeName: ShapeNames): Shape {
        switch (shapeName) {
            case 'circle':
                return new Circle();
            case 'square':
                return new Square();
            case 'rectangle':
                return new Rectangle();
            case 'triangle':
                return new Triangle();
        }
    }
}
```

```typescript
> class Circle extends Shape {…
  }
> class Square extends Shape {…
  }
> class Rectangle extends Shape {…
  }
> class Triangle extends Shape {…
  }

  const shapes: Shape[] = [
      Shape.create('circle'),
      Shape.create('square'),
      Shape.create('rectangle'),
      Shape.create('triangle')
  ];

  shapes.forEach(shape ⇒ shape.draw());
```

# Typescript

## Patrones. Builder

```ts
type METHODS = 'GET' | 'POST' | 'PUT' | 'DELETE';
class Request {
    public url: string;
    public method: METHODS;
    public data: unknown;
    public headers: Record<string, string>;

    constructor() {
        this.url = '';
        this.method = 'GET';
        this.data = {};
        this.headers = {};
    }

    send() {
        console.log(this.url, this.method, this.data, this.headers);
    }
}
```

```ts
export class RequestBuilder {
    private request: Request;

    constructor() {
        this.request = new Request();
    }

    setUrl(url: string): RequestBuilder {
        this.request.url = url;
        return this;
    }

    setMethod(method: METHODS): RequestBuilder {
        this.request.method = method;
        return this;
    }

    setData(data: unknown): RequestBuilder {
        this.request.data = data;
        return this;
    }

    build(): Request {
        return this.request;
    }
}
```

```ts
const request = new RequestBuilder()
    .setMethod('GET')
    .setUrl('https://google.com')
    .build();
request.send();
```

# Typescript

## Consejo general

1. Dejamos que TS infiera los tipos por nosotros (!any)
2. Ayudamos a TS a inferir los tipos
3. Anotamos los tipos explícitamente

# Typescript

## Ejercicios

Crear un proyecto con soporte para TS, ESLint y Prettier

Implementar una función de assertions que permita:

- Comparar valores del mismo tipo.
- Aceptar cualquier número de parámetros

Anotar una función que reciba:

1. Como primer parámetro una función que puede recibir cualquier número de parámetros de cualquier tipo y devolver un valor de cualquier tipo
2. Tantos parámetros restantes como parámetros reciba la función que recibe como primer parámetro

Modificar la función anterior para que solo acepte como primer parámetro funciones cuyo segundo parámetro sea un string

# Typescript

## Ejercicios

Crear un módulo que nos permita gestionar los usuarios de nuestra aplicación mediante una API sencilla. Los usuarios se componen de la propiedad id, nombre y edad.

1. Con un método para crear un usuario
2. Con un método para modificar un usuario, el id no se puede modificar
3. Con un método para listar los usuarios creados
4. Con un método para eliminar un usuario por id
5. Con un método para buscar un usuario por id
6. Con un método para buscar un usuario por nombre

Añadir al API la función **forEach**, esta función recibirá un callback de usuario, la función iterate recorrerá todos los usuarios almacenados en nuestro módulo y llamará a la función callback tantas veces como usuarios haya almacenados usando cada vez a un usuario como argumento.

Añadir al módulo la función **toString**, esta función usará forEach para mostrar un listado bien formateado de los usuarios almacenados