plain concepts

TypeScript

https://www.typescriptlang.org

INTRODUCCIÓN

- Es un lenguaje de programación libre y de código abierto desarrollado y mantenido por **Microsoft** que compila a **JavaScript.**

- Es un superconjunto de **JavaScript**, que esencialmente añade tipado estático y objetos basados en clases.

- TypeScript extiende la sintaxis de **JavaScript**, por tanto, cualquier código JavaScript existente debería funcionar sin problemas.

TYPINGS

- Definiciones de tipos en TypeScript para trabajar con librerías externas en JS.
- Extensión .d.ts
- Se definen en el fichero package.json los typing utilizados.

```json
"devDependencies": {
  "typescript": "~2.7.2",
  "@types/jasmine": "~2.8.6",
  "@types/jasminewd2": "~2.0.3",
  "@types/node": "~8.9.4"
}
```

plain concepts

TSCONFIG.JSON

- Fichero que permite indicar las opciones de compilación
- Deber ubicarse en la raíz del proyecto

```json
"compileOnSave": false,
"compilerOptions": {
  "baseUrl": "./",
  "outDir": "./dist/out-tsc",
  "forceConsistentCasingInFileNames": true,
  "strict": true,
  "noImplicitOverride": true,
  "noPropertyAccessFromIndexSignature": true,
  "noImplicitReturns": true,
  "noFallthroughCasesInSwitch": true,
  "sourceMap": true,
  "declaration": false,
  "downlevelIteration": true,
  "experimentalDecorators": true,
  "moduleResolution": "node",
  "importHelpers": true,
  "target": "es2020",
  "module": "es2020",
  "lib": [
    "es2020",
    "dom"
  ]
```

plain concepts

COMPILACIÓN

Get TypeScript

Node.js

The command-line TypeScript compiler can be installed as a Node.js package.

INSTALL
```
npm install -g typescript
```

COMPILE
```
tsc helloworld.ts
```

Visual Studio

Visual Studio 2019

Visual Studio Code

Visual Studio 2017

And More...

Sublime Text

Atom

Eclipse

Emacs

WebStorm

Vim

https://www.typescriptlang.org/#download-links

plain concepts

TIPOS

```typescript
let isDone: boolean = false;

let decimal: number = 6;
let hex: number = 0xf00d;
let binary: number = 0b1010;
let octal: number = 0o744;

let color: string = "blue";
color = 'red';

let fullName: string = `Bob Bobbington`;
let age: number = 37;
let sentence: string = `
  Hello, my name is ${ fullName}.
  I'll be ${ age + 1} years old next month.`;

let list: number[] = [1, 2, 3];

enum Color { Red, Green, Blue }
let c: Color = Color.Green;

let notSure: any = 4;
notSure = "maybe a string instead";
notSure = false; // okay, definitely a boolean

let columnType: 'string' | 'number' | 'html' = "number";
columnType = "string";
```

plain concepts

DECLARACIÓN DE VARIABLES

- *let*
  - Su ámbito pertenece al bloque mas cercano en la que se encuentra definido.
  - Su valor puede ser reasignado.

- *const*
  - Su ámbito pertenece al bloque mas cercano en la que se encuentra definido.
  - Su valor NO puede ser reasignado.

plain concepts

INTERFACES

```typescript
interface SquareConfig {
  color?: string;
  width: number;
}


interface Square {
  color: string;
  area: number;
}


function createSquare(config: SquareConfig): Square {
  return {
    color: config.color || "white",
    area: config.width * config.width
  };
}


const mySquare = createSquare({ width: 50 });
```

plain concepts

CLASSES

```typescript
class Animal {
    name: string;
    constructor(theName: string) { this.name = theName; }
    move(distanceInMeters: number = 0) {
        console.log(`${this.name} moved ${distanceInMeters}m.`);
    }
}

class Snake extends Animal {
    constructor(name: string) { super(name); }
    move(distanceInMeters = 5) {
        console.log("Slithering...");
        super.move(distanceInMeters);
    }
}
```

```typescript
abstract class Animal {
  abstract makeSound(): void;
  move(): void {
    console.log("roaming the earth...");
  }
}
```

plain concepts

FUNCTIONS

```typescript
function add(x: number, y: number = 1): number {
    return x + y;
}


let result1: number = add(1, 2);
let result2: number = add(3);
```

plain concepts

GENERICS

```
function getArray(items : any[] ) : any[] {
  return new Array().concat(items);
}

let myNumArr = getArray([100, 200, 300]);
let myStrArr = getArray(["Hello", "World"]);

myNumArr.push(400); // OK
myStrArr.push("Hello TypeScript"); // OK

myNumArr.push("Hi"); // OK
myStrArr.push(500); // OK

console.log(myNumArr); // [100, 200, 300, 400, "Hi"]
console.log(myStrArr); // ["Hello", "World", "Hello TypeScript", 500]
```

```
function getArray<T>(items : T[] ) : T[] {
  return new Array<T>().concat(items);
}

let myNumArr = getArray<number>([100, 200, 300]);
let myStrArr = getArray<string>(["Hello", "World"]);

myNumArr.push(400); // OK
myStrArr.push("Hello TypeScript"); // OK

myNumArr.push("Hi"); // Compiler Error
myStrArr.push(500); // Compiler Error
```

plain concepts

GENERICS

```typescript
function identity<T>(arg: T): T {
    return arg;
}

let output1: string = identity<string>("myString");
let output1b = identity("myString");

let output2: number = identity<number>(2);
let output2b: number = identity(2);
```

```typescript
interface Entity {
  key: number;
}

interface Config<T extends Entity> {
  name: string;
  callback: (data: T) => T;
}

interface Account extends Entity {
  amount: number;
}

const accountConfig: Config<Account> = {
  name: 'Account A',
  callback: (data: Account) => {
    data.amount + 10;
    return data;
  }
}
```

ARROW FUNCTIONS

- No hace falta escribir **function**
- Captura el significado de **this**

```typescript
const myFunc1 = function (h: number, w: number): number {
  return h * w * this.count;
};

const myFunc2 = (h: number, w: number): number => h * w * this.count;
```

MODULOS

"Modules are executed within their own scope, not in the global scope; this means that variables, functions, classes, etc. declared in a module are not visible outside the module unless they are explicitly exported using one of the [export forms](#). Conversely, to consume a variable, function, class, interface, etc. exported from a different module, it has to be imported using one of the [import forms](#)"

```typescript
export const numberRegexp = /^[0-9]+$/;

export class ZipCodeValidator {

    private isValid(s: string) {
        return s.length === 5 && numberRegexp.test(s);
    }

    isAcceptable1(s: string) {
        return this.isValid(s);
    }

    public isAcceptable2(s: string) {
        return this.isValid(s);
    }
}
```

```typescript
import * as Utils from './zip-code-validator-utils';
import { ZipCodeValidator } from './zip-code-validator-utils';

let myValidator = new ZipCodeValidator();

myValidator.
            isAcceptable1  (method) ZipCodeValidator.isAcceptabl…
            isAcceptable2
```

plain concepts

MIXINS

"A mixin class is a class that implements a distinct aspect of functionality. Other classes can then include the mixin and access its methods and properties. That way, mixins provide a form of code reuse that is based on composing behavior"
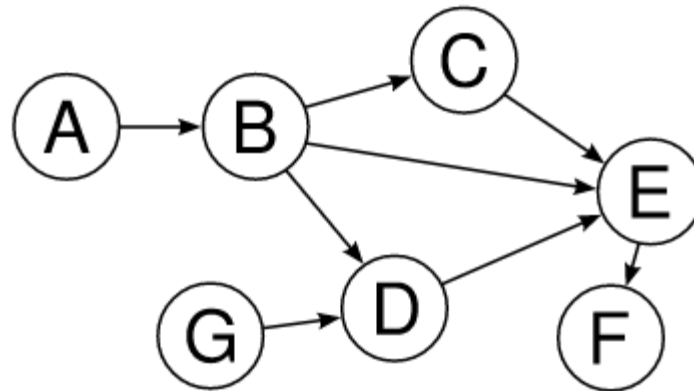
MIXINS

```
type Constructor<T = {}> = new (...args: any[]) => T;

function Timestamped<TBase extends Constructor>(Base: TBase) {
  return class extends Base {
    timestamp = Date.now();
  };
}

function Activatable<TBase extends Constructor>(Base: TBase) {
  return class extends Base {
    isActivated = false;

    activate() {
      this.isActivated = true;
    }

    deactivate() {
      this.isActivated = false;
    }
  };
}
```
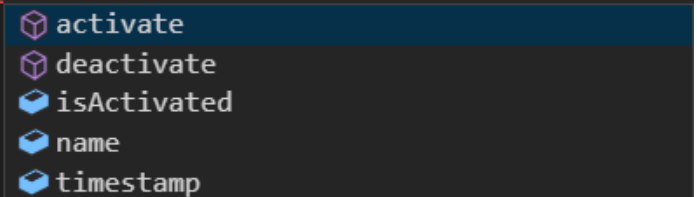
```
class User {

  constructor(
    public name: string) { }

}


const SpecialUser = Activatable(Timestamped(User));
const johnDoe = new SpecialUser("John Doe");
johnDoe.
        activate
        deactivate
        isActivated
        name
        timestamp
```
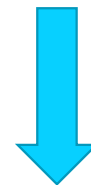
plain concepts

OPTIONAL CHAINING

```
let x = foo === null || foo === undefined || foo.bar === null || foo.bar === undefined
  ? undefined
  : foo.bar.baz();
```

```
let x = foo?.bar?.baz();
```

```
if (foo && foo.bar && foo.bar.baz) {
  // ...
}
```

```
if (foo?.bar?.baz) {
  // ...
}
```

plain concepts

EJERCICIO

```typescript
let borja = new Person('borja', 31);
let lassie = new Dog('lassie', 10, true);
let milu = new Dog('milu', 5, false);

borja.printAge();    // borja is 31 years old
lassie.printAge();   // lassie is 10 years old
milu.printAge();     // milu is 5 years old

borja.turnYear();
lassie.turnYear();
milu.turnYear();

borja.printAge();    // borja is 32 years old
lassie.printAge();   // lassie is 18 years old
milu.printAge();     // milu is 11 years old
```

main.ts

plain concepts

LINTING - TSLINT

TSLint is an extensible static analysis tool that checks TypeScript code for readability, maintainability, and functionality errors. It is widely supported across modern editors & build systems and can be customized with your own lint rules, configurations, and formatters.

Reglas:

https://palantir.github.io/tslint/rules/

DEPRECATED

```
"rules": {
  "arrow-return-shorthand": true,
  "callable-types": true,
  "class-name": true,
  "comment-format": [
    true,
    "check-space"
  ],
  "curly": true,
  "deprecation": {
    "severity": "warn"
  },
  "eofline": true,
  "forin": true,
  "import-blacklist": [
    true,
    "rxjs/Rx"
  ],
  "import-spacing": true,
  "indent": [
    true,
    "spaces"
  ],
  "interface-over-type-literal": true,
  "label-position": true,
  "max-line-length": [
    true,
    140
  ],
  "member-access": false,
  "member-ordering": [
    true,
    {
      "order": [
        "static-field",
        "instance-field",
        "static-method",
        "instance-method"
      ]
    }
  ],
  "no-arg": true,
  "no-bitwise": true,
  "no-console": [
```

plain concepts

LINTING - ESLINT

ESLint is a tool for identifying and reporting on patterns found in ECMAScript/JavaScript code, with the goal of making code more consistent and avoiding bugs.

Reglas:

https://eslint.org/docs/rules/

```json
{
    "extends": "airbnb",
    "parser": "babel-eslint",
    "globals": {
        "window": true,
        "document": true,
        "navigator": true,
        "location": true
    },
    "plugins": [
        "jsx-a11y",
        "import"
    ],
    "rules": {
        "no-undef": ["error"],
        "comma-dangle": ["error", "never"],
        "no-underscore-dangle": ["off"],
        "func-names": ["off"],
        "no-console": ["error", { "allow": ["warn",
        "jsx-a11y/no-static-element-interactions":
        "no-param-reassign": ["off"],
        "class-methods-use-this": ["off"],
        "prefer-template": ["off"],
        "import/prefer-default-export": ["off"],
        "object-shorthand": ["off"],
        "no-plusplus": ["off"],
        "no-useless-constructor": ["off"],
        "max-len": ["error", { "code": 200 }]
    }
}
```

plain concepts

¡GRACIAS!

plain concepts

plain concepts Bilbao

afole@plainconcepts.com

Ledesma 10 bis, 2ª planta

48001 Bilbao, España

+34 94 6008 168