# Typescript

Es un superset fuertemente tipado de Javascript desarrollado y liberado como un lenguaje de código abierto por Microsoft.

Los navegadores actuales solo ejecutan Javascript; es por esto que el código que escribimos en Typescript debe ser traducido para que estos puedan ejecutarlo, a este proceso de traducción se le conoce como transpilar.

La ventaja más grande de usar Typescript para escribir tu aplicación es minimizar los errores en tiempo de ejecución lo que finalmente reducirá el tiempo de depuración y solución de incidentes; pero en sí mismo el uso de Typescript trae consigo lo que algunos desarrolladores consideran una desventaja, que es que finalmente vas a tener que escribir más código.

Existen transpiladores online como [TypeScript Playground](https://www.typescriptlang.org/play/index.html), que sirven para hacer pruebas e inspeccionar el resultado final de la transpilación.

Durante el curso usaremos el compilador de Typescript instalado como módulo de NodeJS, para instalarlo globalmente ejecutamos el siguiente comando:

**npm install -g typescript**

El anterior comando instalará la última versión estable del compilador al momento de escribir este documento 1.8.10; pero para el curso usaremos 2.x por tal motivo ejecutaremos el siguiente comando:

**npm install -g typescript@next**

El cual instalará la siguiente versión candidato, al momento de escribir este documento 2.1.0-dev.20161021.

Para compilar un archivo usamos el siguiente comando:

**tsc main.ts**

Para poder realizar depuración a los archivos Typescript es necesario crear sourcemaps. Los sourcemaps son archivos que mapean el código Typescript y el Javascript emitido. Para generar los source maps ejecutamos el siguiente comando:

**tsc --sourcemap main.ts**

Para indicar la versión de Javascript en la que deseamos sean emitidos los archivos Javascript usamos el parámetro --t del compilador:

**tsc --t ES5 main.ts**

Actualmente la versión por defecto es ES3.

Para configurar el compilador y evitar la necesidad de agregar parámetros al comando ejecutado para compilar podemos usar el archivo de configuración tsconfig.json.

Ejemplo:

**{**

**"compilerOptions": {**

**"target": "es5",**

**"module": "commonjs",**

**"emitDecoratorMetadata": true,**

**"experimentalDecorators": true,**

**"rootDir": "./src",**

**"outDir": "./js",**

**"sourceMap": true**

**},**

**"exclude": [**

**"node\_modules"**

**]**

**}**

## Características del lenguaje

### Tipos opcionales:

La declaración de tipos de en Typescript es opcional; pero si es provista en el código el transpilador puede detectar diferencias en los tipos especificados y asignados.

Ejemplo:

**var saludo:string = "Hola mundo"; //Declarando tipo**

**saludo = 123; //Falla porque el dato está definido como string y se está tratando asignar un entero**

**var saludo = "Hola mundo"; //Sin declarar tipo**

**saludo = 123; // No falla porque no se definió tipo**

### Funciones:

Las funciones en Typescript son similares a las funciones en Javascript; pero es posible especificar los tipos para los parámetros y el retorno de la función.

Ejemplo:

**function sumar(sumando1: number, sumando2: number): number {**

**return sumando1 + sumando2;**

**}**

Al igual que en ES6 las funciones en Typescript pueden declarar parámetros por defecto:

Ejemplo:

**function mostrarError(mensajeError: string = ‘Error no especificado’) {**

**console.log(mensajeError);**

**}**

También es posible de definir parámetros opcionales en las funciones de ES6.

Ejemplo:

**function optionalPow(base: number, exponent?: number): number {**

**if(exponent) {**

**return Math.pow(base, exponent);**

**}**

**return base;**

**}**

### Clases

Typescript soporta clases de manera similar a como ES6 las soporta, con las siguientes diferencias:

#### Modificadores de acceso

Typescript provee los siguientes modificadores de acceso para controlar el acceso a los miembros de los objetos en tiempo de desarrollo: public, protected y private.

#### Herencia

Typescript soporta herencia a través de la palabra reservada extends similar a como lo hace ES6.

#### Interfaces

Typescript soporta interfaces, para definir una interfaz usamos la palabra reservada interface.

Ejemplo:

**interface Persona {**

**nombre: string;**

**apellidos: string;**

**}**

Para implementar una interfaz usamos la palabra reservada implements.

Ejemplo:

**interface Renderable {**

**render();**

**}**

**class TextRenderer implements Renderable {**

**render(){**

**console.log("Render implementation");**

**}**

**}**

#### Genéricos

Typescript soporta tipos parametrizados, que pueden ser usados para declarar una función que soporta cualquier tipo; pero al momento de ser usada especificar el tipo de dato sobre el que operara.

Ejemplo:

**function logObject<T>(object: T){**

**console.log(object);**

**}**

También podemos usar los tipos parametrizados para restringir los parámetros de una función a una jerarquía de clases.

Ejemplo:

**interface Persona {**

**nombre: string;**

**apellidos: string;**

**}**

**function getNombreCompleto<c extends Persona>(persona: c): string**

**{**

**return persona.nombre + " " + persona.apellidos;**

**}**

#### Módulos

Typescript comparte el mismo concepto de módulos ofrecido por ES6/2015, cualquier archivo que contenga un import o export de nivel superior es considerado un módulo.

Los módulos son ejecutados al interior de su propio alcance, esto significa que las funciones, variables, clases y demás, declaradas al interior de un módulo solo tienen visibilidad al interior de dicho modulo; a no ser, que sean exportados usando la palabra reservada ***export*** y posteriormente sean importados en otros módulos usando la palabra reservada ***import***.

**Exportando declaraciones:**

Cualquier tipo de declaración (tales como una variable, función, clase, alias o interfaz) puede ser exportada usando la palabra reservada ***export***.

Ejemplo:

renderable.ts

**export interface Renderable {**

**render();**

**}**

paginator.ts

**export const REGISTROS\_PAGINA: number = 20;**

user.ts

**export class User implements Persona {**

**}**

**Importando declaraciones:**

Para importar declaraciones exportadas por otro modulo, usamos la palabra reservada import de la siguiente manera:

**import {<nombre de las declaraciones a importar separadas por coma>} from “<ruta relativa al archivo que exporta la declaración sin la extensión del archivo>”;**

Ejemplo:

user.ts

**import {Persona} from “./persona”;**

**export class User implements Persona {**

**}**

También podemos usar la palabra reservada ***import*** para importar y asignar un alias a declaraciones expuestas por otros modulos.

Ejemplo:

externalUser.ts

**import {User as Usuario} from “./user”;**

**export class ExternalUser extends Usuario {**

**}**

Typescript también ofrece la posibilidad de importar un módulo completo y asignarlo a una única variable, para usarlo posteriormente a través de dicha variable; esto es especialmente útil cuando existe colisión de nombres entre las declaraciones de diferentes módulos.

Ejemplo:

userRepository.ts

**import \* as user from “./user”;**

**let currentUser = new user.User();**

**Exportar declaraciones por defecto:**

Cada módulo tiene la opción de exportar una y solo una declaración por defecto usando la palabra reservada ***default***, las declaraciones exportadas por defecto son importadas usando una forma diferente de ***import***.

Ejemplo:

emailValidator.ts

**const emailRegexp = /^[-a-z0-9~!$%^&\*\_=+}{\'?]+(\.[-a-z0-9~!$%^&\*\_=+}{\'?]+)\*@([a-z0-9\_][-a-z0-9\_]\*(\.[-a-z0-9\_]+)\*\.(aero|arpa|biz|com|coop|edu|gov|info|int|mil|museum|name|net|org|pro|travel|mobi|[a-z][a-z])|([0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}))(:[0-9]{1,5})?$/i;**

**export default function (s: string) {**

**return emailRegexp.test(s);**

**}**

registration.ts

**import isValidEmail from “./emailValidator”;**

**console.log(isValidEmail(“luissanchez@elpoli.edu.co”));**