1. ¿Qué es Typescript?

TypeScript es un lenguaje fuertemente tipado creado por Microsoft, aunque está muy relacionado con el framework Angular, TypeScript es un lenguaje independiente que puedes usar para proyectos de back-end con Express o Front-end ya sea con Vue, React o Angular. Para que TypeScript pueda correr en el navegador debe ser transpilado a JavaScript con herramientas como Babel.

**Compilación**

Es el proceso por el cual el código fuente de un programa escrito en un lenguaje de programación “A” se traduce a código fuente de un lenguaje de programación “B”. Cuando el código origen está escrito en un lenguaje de programación de “alto nivel” (C, Java, PHP u otros lenguajes de tercera generación) y se traduce a código máquina que pueda ejecutar directamente el ordenador. Pero hay muchas otras posibilidades como pasar por una representación intermedia en bytecode como es el caso típico de Java.

**Transpilación**

La transpilación es un caso particular de la compilación. Es decir, todo transpilador es también un compilador pero al revés no es cierto, hay compiladores que no se pueden considerar transpiladores. ¿Y cuál es la característica que convierte un compilador en transpilador?: la relación entre los lenguajes origen y destino de la traducción.

Si el compilador traduce código entre dos lenguajes que están al mismo nivel de abstracción entonces, estamos ante un transpilador. Si traduce código entre lenguajes de diferente nivel de abstracción (típicamente de más alto a más bajo nivel) entonces no lo es.

**TypeScript** es un lenguaje de programación de código abierto desarrollado por Microsoft, el cual cuenta con herramientas de programación orientada a objetos, muy favorable si se tienen proyectos grandes. Anders Hejlsberg, arquitecto principal del desarrollo del lenguaje de programación C#, es el principal participante en el desarrollo de este lenguaje.

**TypeScript convierte su código en Javascript común**. Es llamado también Superset de Javascript, lo que significa que si el navegador está basado en Javascript, este nunca llegará a saber que el código original fue realizado con TypeScript y ejecutará el Javascript como lenguaje original.

**¿Qué es un superset?**

Se trata de un lenguaje escrito sobre otro lenguaje. En este caso Typescript es eso, un lenguaje basado en el original, ofreciéndonos grandes beneficios como el descrito anteriormente, aunque existen otros beneficios. Por ejemplo, mientras otros superset de JavaScript nos alejan del código original, Typescript, por el contrario, es muy similar a Javascript y a C# gracias a que su creador posee conocimientos de ambos lenguajes.

Actualmente Angular 2, uno de los frameworks más famosos de JavaScript, está siendo desarrollando en TypeScript, para lo cual conocer este lenguaje será fundamental para entender y darle un mejor uso a la nueva versión de Angular.

::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

**3-Declaración de Variables en TypeScript**  
’let’ y ‘const’ son tipos de declaradóres de variables similares a var, que permiten evitar algunos de los “problemas” de JavaScript.  
const, por ejemplo evita la reasignación a una variable.

**const** numLivesForCat = 9; // no se puede re asignar

let, impide el block-scoping  
Las variables ‘let’ no son visibles fuera de su bloque o bucle for más cercano.

**let** hello = "Hello!"; // declaración normal

// block scoping test

**var** a = 5;

**var** b = 10;

**if** (a === 5) {

**let** a = 4; // El alcance es dentro del bloque if

**var** b = 1; // El alcance es global

console.log(a); // 4

console.log(b); // 1

}

console.log(a); // 5

console.log(b); // 1

**Tipos de Datos en TypeScript**  
Boolean - El tipo de dato más básico, puede ser true o false

let isDone:**boolean** = **false**;

Number - Los valores numéricos reciben el tipo de dato ‘number’.  
También soporta decimal y hexadecimal. Binario y Octal a partir de ECMAScript 2015.

let decimal: number = 67;

let hex: number = 0xf00d;

let binary: number = 0b1010;

let octal: number = 0o744;

String - Usamos ‘string’ para referirnos a estos tipos de datos textuales.  
Podemos usar comillas dobles (") o simples (’) para rodear la cadena. Tambien los `` para mezclar las cadenas con otras variables.

**let**color: string = "blue";

color = 'red';

**let**fullName: string = `Bob Bobbington`;

**let**sentence: string = `Hola!, mi nombre es ${fullName}`;

Array - Los tipos de dato Array pueden ser escritos de dos maneras.  
1 - Colocamos el tipo de datos seguido de un [], señalamos que sera un array de ese tipo

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

2 - Usamos un generic, Array<elemType>

**let**list: Array<number> = [1, 2, 3];

Any - Cuando aun no sabemos que tipo de dato vamos a utilizar.

**let**notSure: **any** = 4;

// ...las reasignamos...

notSure = "quizas es un string";

notSure = **false**; // ahora es un boolean

// si la usamos en un Array podemos mezclar los tipos de datos

**let**list: **any**[] = [1, **true**, "free"];

Void - Lo opuesto a any  
Visto comunmente en las functions y quiere decir que no retorna ningun valor.

**functionwarnUser**(): **void** {

console.log("Este es mi mensaje");

}

**::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::**

**4-** todo sobre clases y programación de objetos está en el archivo

Dentro de TypeScript podemos realizar programación orientada a objetos y como su nombre lo dice vamos a programar objetos, pero para generar dichos objetos primero debemos crear su plantilla base que recibe el nombre de clase.

**Una clase** está integrada por propiedades y funciones, estas pueden ser privadas, públicas o protected.

A la variable que le asignamos el valor de un objeto debemos indicarle que su tipo de dato es igual a la clase.

Una característica de la programación orientada a objetos es que podemos crear clases padre que hereden sus propiedades y funciones a una clase hijo, esto es la herencia y dentro de TypeScript se representa con la palabra **extends.**

**::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::**

**5- Interfaces en type-script:archivo**

\*\*Una interfaz \*\*es un tipo abstracto que sirve como contrato para la estructura de un objeto y **al igual que las clases puede ser utilizada como un tipo de dato.** Para declarar una interfaz en TypeScript utilizamos la palabra clave interface.

Dentro de las interfaces en TypeScript podemos manejar propiedades opcionales añadiendo el signo de pregunta **‘?’** al final del nombre de la propiedad.

}

**::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::**

**-6 Shapes:archivo**

Cada uno de los tipos tiene una forma o estructura en que es representado, esto es mediante sus propiedades. Dentro de TypeScript si dos variables poseen propiedades idénticas aunque sean de distintas interfaces o clases, estas pueden ser asignadas entre sí.

**::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::**

9- **Function type: archivo**

Un function type representa la estructura que tendrá la función a la cual se aplica el tipo y únicamente funciona para tipos de funciones.