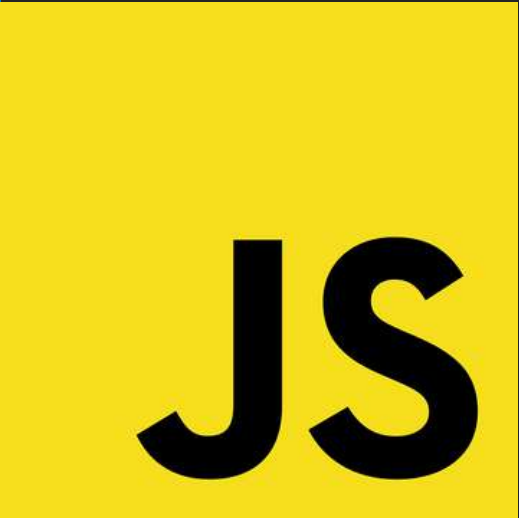


TypeScript - introducción al lenguaje y tipos de dato básicos

The TypeScript logo, consisting of a blue square with the white letters 'TS' inside.

TS

The JavaScript logo, consisting of a yellow square with the black letters 'JS' inside.

JS

Introducción a TypeScript

¿Qué es TypeScript? Es un súper set de JavaScript, es decir que expande todo lo que nos brinda JavaScript, pero con la diferencia que nos permite agregar tipado cuando declaramos variables, entre otras cosas.

TypeScript no corre en el navegador, tiene que transpilar el código a código de JavaScript puro para que pueda ser soportado por los navegadores.

Es un archivo de configuración de TypeScript, se puede configurar de diferentes maneras para que TypeScript trabaje tan estricto como nosotros querremos. Por ejemplo que JavaScript tenga el modo estricto, la versión de EcmaScript, así como también dejar el modo "noImplicitAny" en valor true o false. entre otras opciones más.

Este archivo también nos ayudará a transpilar todos los archivos .ts a archivos .js.

Para crear este archivo de configuración basta con escribir "tsc --init" en nuestra terminal, dentro de la carpeta de nuestro proyecto.

Este modo detectará cualquier archivo de TypeScript, por lo cual convertirá y actualizará a un archivo de JavaScript de manera inmediata.

Para activar el "modo observador" basta con ejecutar el comando `"tsc --watch"` dentro de nuestra terminal en la carpeta de nuestro proyecto.

Los tipos de datos que maneja TypeScript son todos los que maneja JavaScript. Por Ejemplo los datos primitivos y compuestos.




TypeScript a diferencia de JavaScript, contiene también otro tipo de datos como pueden ser:

- Interfaces

- Genéricos

- Tuplas

Como ya se sabe, los booleans son un tipo de dato que su valor será verdadero o falso ("true" y "false"). En TypeScript funcionan igual, sin embargo mediante la configuración que nosotros le demos, estos datos pueden llegar a ser del tipo null o undefined. Como buena práctica siempre declarar el tipo de dato que almacenará una variable:

A code editor window with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. It contains two lines of TypeScript code, each preceded by a line number.

```
1 const isTrue : boolean = true;  
2 const isFalse : boolean = false;
```

Numbers

TS


Son valores de tipo numérico, es decir que podemos almacenar tanto números enteros como decimales:



```
1  const number : number = 21;  
2  const decimalNumber : number = 21.55;
```



Son cadenas de caracteres, comúnmente encerradas entre comillas, comillas simples o comillas invertidas o backticks: "Esto es un string", 'Esto es un string', `Esto es un string`.

La manera de declarar variables de tipo string en TypeScript es la siguiente:



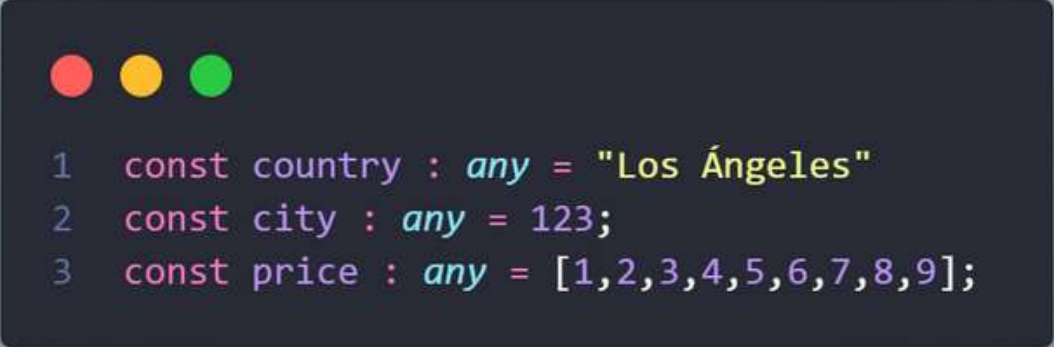
```
1  const NAME : string = "Ricardo";  
2  const LAST_NAME : string = 'Guevara';  
3  const NICK_NAME : string = `@ricardo_guevarag`;
```

El tipo de dato Any es un tipo de dato que puede recibir cualquier tipo de dato, ya sea un number, un booleano, un string, etc. Al trabajar con TypeScript se recomienda usar lo menos posible este tipo de dato, incluso en nuestro archivo tsconfig.json podemos inhabilitar el uso de este tipo de dato.



```
1  "noImplicitAny": true,
```

Este tipo de dato de igual manera se tiene que especificar a la hora de declarar una variable:



```
1  const country : any = "Los Ángeles"
2  const city : any = 123;
3  const price : any = [1,2,3,4,5,6,7,8,9];
```

Puedo escribir cualquier tipo de dato y no arrojará ningún error, por eso no es recomendado utilizar `any`, ya que TypeScript no nos ayudaría con un tipado estricto en nuestro código.

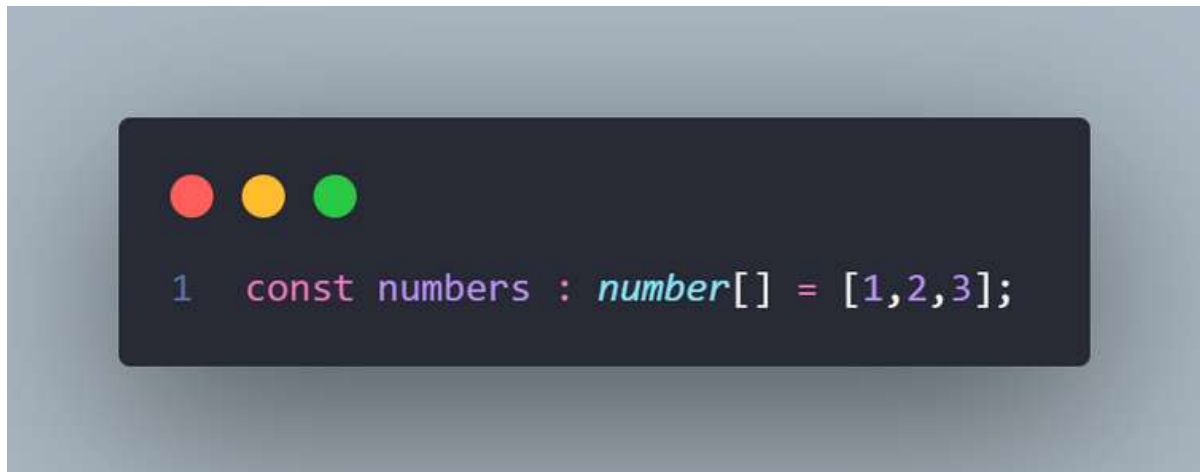
Los arrays son un listado de cualquier tipo de datos, pueden ser cualquier tipo de colección, por ejemplo una colección de strings, de booleans, objetos, e incluso puede contener diversos tipos de datos.

En JavaScript basta con declarar nuestro arreglo en una variable y podíamos insertar cualquier tipo de dato, incluso combinarlos:



```
1 var exampleArray = [1, 2, 3, "4", "5", true, false, null, undefined];
```

En TypeScript podemos indicarle el tipo de dato que queremos que tenga o que permita tener nuestro array:

A screenshot of a code editor window with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. The code displayed is a single line: `1 const numbers : number[] = [1,2,3];`. The text is color-coded: `1` is light blue, `const` is pink, `numbers` is light blue, `:` is light blue, `number` is teal, `[]` is light blue, `=` is light blue, and `[1,2,3];` is light blue.

```
1 const numbers : number[] = [1,2,3];
```

En este ejemplo le indicamos a TypeScript que nuestro arreglo solo contendrá valores numéricos.

Si nosotros deseamos tener más de un tipo de dato también debemos indicárselo.

Para hacerlo debemos declara una sintaxis así:



```
1 const exampleArray: (number | string)[] = [1, "2", 3, "4"];
```

De esta manera le indicamos que queremos un arreglo de diversos tipos de datos, en este caso de strings y numbers.

Los tuples o las tuplas son similares a los arreglos, igual son una colección de elementos. La diferencia es que se puede indicar que tipo de dato quiero en las posiciones del arreglo y también controlar la longitud del mismo.

Ejemplo de una tupla:



```
1 const exampleTuple : [string, number, boolean] = ["ricardo", 21, false];
```

Las enumeraciones son una característica que no contiene JavaScript, son un conjunto de constantes con nombre. Existen enumeraciones numéricas (numbers), de cadenas de texto ("strings"), de booleanos e incluso de diferentes tipos de datos.

Si no le damos valores a las variables dentro de la enumeración por defecto serán valores numéricos empezando en 0 y aumentando de 1 en 1, es decir "0, 1, 2, etc...".

Para definir una enumeración es necesario utilizar la palabra reservada `enum` seguida del nombre que queremos asignarle y las llaves que tendrán dentro nuestra enumeración.

Un ejemplo de enumeración en TypeScript:

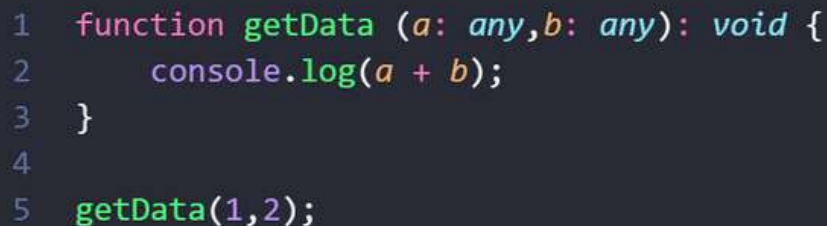


```
1  enum Levels {  
2      easy,  
3      medium,  
4      hard  
5  }
```

Como se mencionó antes, al no declarar valores a las variables de la enumeración estas tendrán valores numéricos empezando desde el 0.

Void ("vacío") es un tipo de retorno de las funciones que no retornan nada, es decir, que no tienen la palabra reservada `return` dentro del cuerpo.

La palabra reservada `void`, la podemos utilizar en este tipo de funciones, declarándola de la siguiente manera:



```
1 function getData (a: any, b: any): void {  
2     console.log(a + b);  
3 }  
4  
5 getData(1,2);
```

Never es un tipo de retorno de las funciones, el cual nos va indicar que no tiene que tener un punto alcanzable al finalizar la ejecución, es decir que nunca debe ocurrir o que nunca ocurre.

Nos puede servir a la hora de crear funciones que manden algún tipo de error y que el resto del código no se siga ejecutando.

Un ejemplo de una función con retorno `never` puede ser así:



```
1 function newError (messageError: string) : never {  
2     throw new Error(messageError);  
3 }  
4  
5 newError("Algo sucedió mal...")
```

Le indicamos a la función el tipo de retorno con `never` y dentro de la función especificamos lo que queremos que se ejecute.

Undefined

TS

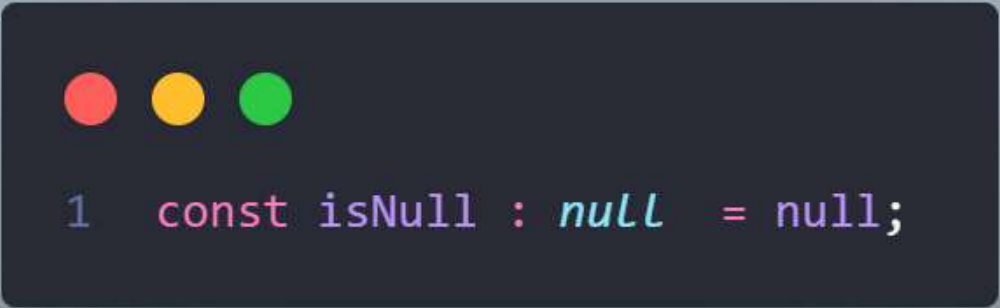
El tipo de dato `undefined` funciona para indicar que algo no está definido. Este tipo de dato también se debe especificar a la hora de declarar una variable:



```
1  const isUndefined : undefined = undefined;
```

Null

Null es un tipo de dato que nos indica que algo es nulo, de igual manera debe definirse a la hora de declarar una variable:



```
1  const isNull : null = null;
```