TypeScript

2. Tipos de datos

0. Definir variables en TS

- Las variables en TypeScript se declaran usando la palabra clave let. Las variables se pueden declarar y asignar en la misma línea, o puede separar la declaración y la asignación en líneas diferentes.
- En el siguiente ejemplo, declaramos y asignamos un valor al mensaje variable en una sola línea.

```
let message: string = "Hello World";
console.log(message);
```

1. Number

- El tipo de datos **number** se usa para representar tanto enteros como valores de punto flotante.

```
let a: number = undefined; // Error
let b: number = null; // Error
let c: number = 3;
```

2. String

- El tipo de datos **string** se utiliza para almacenar información textual.

```
let a: string = undefined; // Error
let b: string = null; // Error
let c: string = "";
let d: string = "y";
let e: string = "building";
let f: string = 3; // Error
let g: string = "3";
```

3. Boolean

- El vaBoolean solo tiene dos valores válidos: **true** o **false**.

```
let a: boolean = true;
let b: boolean = false;
let c: boolean = 23; // Error
let d: boolean = "blue"; // Error
```

4. Null

- El tipo de datos **null** en TypeScript solo puede tener un valor válido: null

```
let a: null = null; // Ok
let b: undefined = null; // Error
let c: number = null; // Error
let d: void = null; // Error
```

5. Void

- El tipo de datos void se usa para indicar la falta de un type para una variable.
- Establecer variables con tipo void puede no ser muy útil,su mayor uso es establecer el tipo de retorno de las funciones que no devuelven nada a void.

```
let a: void = undefined; // Ok
let b: void = null; // Error
let c: void = 3; // Error
let d: void = "apple"; // Error
```

6. Undefined

- Cualquier variable cuyo valor no haya especificado se establece en **undefined**.
- Sin embargo, también podemos establecer explícitamente el tipo de una variable como indefinida

```
let a: undefined = undefined; // Ok
let b: undefined = null; // Error
let c: number = undefined; // Error
let d: void = undefined; // Ok
```

7. Array

- En un array especificaremos el tipo de elementos de array seguidos por [] lo cual que denota un array de ese tipo.

```
let a: number[] = [1, 12, 93, 5];
let b: string[] = ["a", "apricot", "mango"];
let c: number[] = [1, "apple", "potato"]; // Error
```

7. Array

- Otro método es utilizar el tipo de array genérico Array<elemTipo>

```
let d: Array<number> = [null, undefined, 10, 15];
let e: Array<string> = ["pie", null, ""];
```

8. Tupla

- El tipo de datos de tupla le permite crear un array donde el tipo de un número fijo de elementos se conoce de antemano.
- El tipo del resto de los elementos solo puede ser uno de los tipos que ya ha especificado para la tupla.

```
let a: [number, string] = [11, "monday"];
let b: [number, string] = ["monday", 11]; // Error
let c: [number, string] = ["a", "monkey"]; // Error
let d: [number, string] = [105, "owl", 129, 45, "cat"];
let e: [number, string] = [13, "bat", "spiderman", 2];
```

9. Enum

- El tipo de datos **enum** está presente en muchos lenguajes de programación como Java

```
enum Animals {cat, lion, dog, cow, monkey}
let c: Animals = Animals.cat;

console.log(Animals[3]); // cow
console.log(Animals.monkey); // 4
```

10. Any

- Hay ocasiones en las que no podrá establecer el tipo de una variable de antemano.
- La variable podría ser de cualquier tipo
- Este problema se puede resolver usando el tipo **any**. Esto también es útil cuando está creando arrays con elementos de tipos mixtos.

```
let a: any = "apple";
let b: any = 14;
let c: any = false;
let d: any[] = ["door", "kitchen", 13, false, null];
```

11. Never

- El tipo **never** se usa para representar valores que nunca se supone que ocurran.
- Por ejemplo, puede asignar never como el tipo de devolución de una función que no devuelve nada.

```
let a: never; // Ok
let b: never = false; // Error
let c: never = null; // Error
let d: never = "monday"; // Error
```