**Objetivo:**

- *un primer vistazo a*
  - *los **tipos básicos** de TypeScript*
  - *la **inferencia de tipos** de TypeScript*
  - *la **composición de tipos** en TypeScript*

- *conocer las ventajas que ofrece la **anotación de tipos** a la hora de prevenir errores en la escritura de código y mejorar la mantenibilidad del mismo*

Los tipos de TypeScript son la característica que más define este lenguaje. Los tipos de variables y constantes pueden ser definidos de forma explícita ***(anotación de tipos)*** o implícita ***(inferencia de tipos).***

Ejemplo de definición de tipos mediante anotación de tipos ***(definición explícita del tipo)***:

```
// Basic types
let myTypeString: string = 'Hello world!' ;
let myTypeNumber: number = 33;
let myTypeBoolean: boolean = true;


// Array

let arrNumber : number[] = [1, 2, 3];
// let arrNumber2 : Array<number> = [1, 2, 3];
let arrStrings : string[] = ["Uno", "Dos"];

let arrAny : any[] = ['Hola', 33, false];
```

Ejemplo de **definición implícita** de tipos: las tres primeras variables del ejemplo anterior podríamos haberlas definido tal y como se muestra a continuación. Si posicionas en tu editor el cursor sobre las variables, puedes comprobar que TypeScript ha inferido el tipo a las variables en base al valor asignado a las mismas. Es la característica de TypeScript conocida como ***inferencia de tipos.***

```
// Basic types
let myTypeString = 'Hello world!' ;
let myTypeNumber = 33;
let myTypeBoolean= true;
```

**TS-Tarea2-tiposBasicos.ts**

Define **explicitando el tipo** al menos una variable de cada uno de los siguientes tipos de datos y muestra en pantalla su valor. Comprueba cómo al asignar valores erróneos, el editor te advierte de ello.

- string,
- number,
- boolean,
- any *(el tipo any es un tipo que deshabilita la verificación de tipos y permite el uso de todos los tipos; podemos decir que este tipo "resta autoridad" a typeScript ya que lo correcto es tener siempre el control de los tipos de datos con los que trabajamos, pero en ocasiones cuando por ejemplo estamos trabajando con una API y no sabemos claramente lo que nos va a devolver, el uso de este tipo nos puede ayudar en el desarrollo)*
- array de números
- array de strings
- array de valores booleanos
- array cuyos items puedan ser de cualquier tipo (any)
- tupla: *cuando conocemos de antemano qué tipo de datos vamos a recibir en cada una de las posiciones del array podemos utilizar una tupla. Una tupla en TypeScript es un array de elementos que están tipados. De esta manera cada vez que haya que insertar un elemento se validará que dicho elemento coincida con el tipo de dato establecido en la tupla.*

```
let variable:[tipo1,tipo2,...,tipoN];
```

*Ejemplo:*

```
let tupla: [string, number,string];
tupla = ['España',10,'Madrid'];
tupla = ['España',10]; //Error
```

*Se trata de un array que puede tener datos de varios tipos, pero en los que las claves son índices numéricos. Consulta la documentación oficial sobre este tipo.*

Define una tupla con datos acerca de un país, en concreto cuatro datos:

- ➢ el primero será el nombre del país, de tipo `string`
- ➢ el segundo será el nombre de su capital, también de tipo `string`
- ➢ el tercer dato será el número de habitantes, un dato de tipo `number`
- ➢ por último, el idioma oficial del país, que también es `string`
- ➢ Después, asigna los datos y muestra un listado de los mismos.

- Ejemplo de **unions o composición de tipos**: typeScript permite declarar que una variable pueda ser de dos o más tipos *(útil cuando por ejemplo trabajo con una api y no tengo claro el tipo de dato que recojo)*

define una variable de este tipo y verifica posibles valores prohibidos y permitidos para la misma.

Extra *(no son tipos básicos)*:

- ❖ define un enum
- ❖ define un object

**TS-Tarea2-2.ts (tipado en funciones)**

- ○ define e invoca una función que reciba como parámetro un number y retorne un boolean
- ○ define e invoca una función que reciba dos strings y retorne su concatenación
- ○ define e invoca la función getSum (recibe como parámetro dos números y devuelve su suma)

**TS-Tarea2-3.ts (repasando var vs let)**

Transcribe a typescript el siguiente código ¿por qué no se puede ver el último document.write?

```
<!DOCTYPE html>
<html><head><meta charset="utf-8"></head>
<script language="javascript">
{
  var contador;
  contador=1;

  while (contador<5)
  {
     let dato=prompt('Introduce número del 1 al 10:','');
     let num=parseInt(dato);

     document.write(` El numero introducido es  ${num} </br> `);
     document.write(` El contador es  ${contador} </br> `);

     contador = contador +1 ;
  }

    document.write( ` </br><h2>
                      Fin del programa
                      contador ya NO es menor que 5.
                      </h2>
                  `);

    document.write (`</br> Ultimo numero introducido es ${num}`)

}
</script>
</html>
```