Hito 2

CUESTIÓN 1. Declaración de variables en TypeScript

- Explica qué tipado (débil o fuerte) se aplica en TypeScript

  TypeScript se caracteriza por estructuras de tipado fuerte donde la declaración de variables exige la asociación con un tipo de datos de forma implícita, se puede cambiar el modo estricto en el archivo de configuración de un proyecto TS para que no exija la declaración de tipo de dato.

- Realiza un ejemplo con declaración de diferentes variables: texto, numéricas, booleanas y de listas. Identifica correctamente el tipo de dato de cada una.

  Declaración e impresión de variables

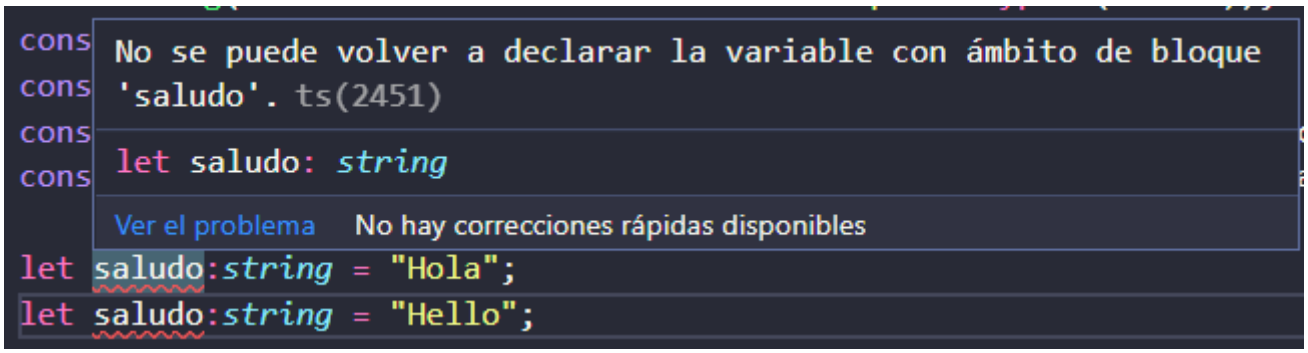
```
let nombre:string = "Raul";
let edad:number = 21;
let altura:number = 1.72;
let matriculado:boolean = true;
let asignaturas:string[] = ["DWEC","DAW","DIW","DWES"];

console.log("La variable: "+nombre+ " es de tipo: "+typeof(nombre));
console.log("La variable: "+edad+ " es de tipo: "+typeof(edad));
console.log("La variable: "+altura+ " es de tipo: "+typeof(altura));
console.log("La variable: "+matriculado+ " es de tipo: "+typeof(matriculado))
console.log("La variable: "+asignaturas+ " es de tipo: "+typeof(asignaturas))
```

  La consola muestra:

```
D:\2DAW\D.WEB.CLIENTE\1Trimestre Segundo Parcial\Hito2>node Cuestion1.js
La variable: Raul es de tipo: string
La variable: 21 es de tipo: number
La variable: 1.72 es de tipo: number
La variable: true es de tipo: boolean
La variable: DWEC,DAW,DIW,DWES es de tipo: object
```

- Explica la diferencia entre let, var y const con un ejemplo.

let: es una variable la cual puedes cambiar su valor, solo funciona dentro del bloque donde está declarada. Las variables let son block-scope (su ámbito es su bloque).



let no permite redefinir la misma variable en el mismo bloque.

```
let saludo:string = "Hola";
if(true){
    let saludo:string = "Hello";
    console.log("Dentro del if: " + saludo);
}
console.log("Fuera del if: " + saludo);
```

```
Dentro del if: Hello
Fuera del if: Hola
```

En diferentes bloques es posible definir múltiples variables con el mismo nombre siendo cada una de ellas una variable diferente.

var: es una variable la cual puedes cambiar su valor, el ámbito de una variable declarada con var, es su contexto de ejecución. El ámbito de una variable var declarada fuera de la función es global. En el caso de var las variables pueden ser de global-scope, local-scope y block-scope, es decir que su ámbito dependerá de donde las declaremos.

```
var saludo:string = "Hola";
var saludo:string = "Hello";
```

var permite redefinir la misma variable en el mismo bloque.

```
var saludo:string = "Hola";
if(true){
    var saludo:string = "Hello";
    console.log("Dentro del if: " + saludo);
}
console.log("Fuera del if: " + saludo);
```

```
Dentro del if: Hello
Fuera del if: Hello
```

En este caso siempre se conserva el valor ya que el rango de definición no es el bloque.

const: es una variable la cual no puedes cambiar su valor una vez definida, las variables const son block-scope (su ámbito es su bloque).

```
const saludo:string = "hola";
```
No se puede asignar a "saludo" porque es una constante. ts(2588)

```
const saludo: any
```
Ver el problema Corrección Rápida (Ctrl+.)

```
saludo = "adios";
```

No se puede volver a declarar la variable con ámbito de bloque 'saludo'. ts(2451)

```
const saludo: string
```
Ver el problema No hay correcciones rápidas disponibles

```
const saludo:string = "Hola";
const saludo:string = "Hello";
```

No te permite cambiar su valor, ni redefinir la variable

CUESTIÓN 2. POO en TypeScript

- Crea una clase llamada Animal. Esta clase tiene como propiedades nombreAnimal, númeroPatas, carnívoro y edad.

```typescript
class Animal{
    //Propiedades
    nombreAnimal:string;
    numeroPatas:number;
    carnivoro:boolean;
    edad:number;

    //constructor
    constructor(nombreAnimal:string, numeroPatas:number, carnivoro:boolean, edad:number){
        this.nombreAnimal = nombreAnimal;
        this.numeroPatas = numeroPatas;
        this.carnivoro = carnivoro;
        this.edad = edad;
    }
}
```

- Utiliza encapsulamiento, propiedades, constructores y getters y setters para gestionar estas propiedades

```typescript
class Animal{
    //Propiedades
    private _nombreAnimal: string;
    private _numeroPatas: number;
    private _carnivoro: boolean;
    private _edad: number;

    //constructor
    constructor(nombreAnimal:string, numeroPatas:number, carnivoro:boolean, edad:number){
        this._nombreAnimal = nombreAnimal;
        this._numeroPatas = numeroPatas;
        this._carnivoro = carnivoro;
        this._edad = edad;
    }

    //getters y setters
    public get nombreAnimal(): string {
        return this._nombreAnimal;
    }
    public set nombreAnimal(value: string) {
        this._nombreAnimal = value;
    }

    public get numeroPatas(): number {
        return this._numeroPatas;
    }
    public set numeroPatas(value: number) {
        this._numeroPatas = value;
    }

    public get carnivoro(): boolean {
        return this._carnivoro;
    }
    public set carnivoro(value: boolean) {
        this._carnivoro = value;
    }

    public get edad(): number {
        return this._edad;
    }
    public set edad(value: number) {
        this._edad = value;
    }

    public toString = () : string => {
        return `Animal (nombreAnimal: ${this.nombreAnimal}, numeroPatas: ${this.numeroPatas}, carnivoro: ${this.carnivoro}, edad: ${this.edad})`;
    }
}
```

He cambiado las propiedades a privadas para crear los getters y setters.

- Crea un array que guarde tres animales y los muestras en una caja de texto

  Para compilar el arvicho.ts he usado un comando más especifico ya que me daba un error por la versión de ECMAScript, he probado a cambiarla desde el archivo de configuración json pero no me funcionaba. El comando que he usado es este: tsc -t es5 Cuestion2.ts

```
const caballo = new Animal("Caballo", 4, false, 5);
const lobo = new Animal("Perro", 4, true, 7);
const araña = new Animal("Araña", 8, true, 1);

let listaAnimales:Animal[] = [caballo,lobo,araña];

let textarea = document.getElementById("txtArea");

var animales:string = "";
for (let iterator of listaAnimales) {
    animales += iterator.toString() + "\n";
}

textarea!.innerHTML = animales;
```

Cuestion 2

```
Animal (nombreAnimal: Caballo,
numeroPatas: 4, carnivoro:
false, edad: 5)
Animal (nombreAnimal: Perro,
numeroPatas: 4, carnivoro: true,
edad: 7)
Animal (nombreAnimal: Araña,
numeroPatas: 8, carnivoro: true,
edad: 1)
```

CUESTIÓN 3. Promises en TypeScript

- Explica con un ejemplo qué es comunicación síncrona y asíncrona

  La programación síncrona se ejecuta en secuencia, hasta que una no termine la otra no comenzará.
  La programación asíncrona hace que se puedan ejecutar varios procesos al mismo tiempo.

Cuestion3.ts > ...

```
function f1() {
    return new Promise((resolve, reject) => {
        console.log('soy el primero');
        resolve(console.log('He ganado'));
    });
}

function f2() {
    console.log('soy el segundo');
}

f1().then(f2);
```

```
soy el primero
He ganado
soy el segundo
```

En este ejemplo la función f2 no se ejecuta hasta que la función f1 se haya completado.

- Realiza un ejemplo usando promises que se conecte a una fuente de datos y pinte el resultado en consola.

  Tanto para este ejercicio como para el anterior he tenido que instalar @types de node con este comando: npm install --save @types/node.
  He usado una fake API para realizar pruebas, se llama JSONPlaceholder, esta dispone de varias fuentes de datos, he usado una que te proporciona usuarios.
  Me he conectado a ella mediante fetch que devuelve una promesa, la cual he manejado para mostrar por consola algunos datos relevantes obtenidos y en el caso de que hubiera errores los he manejado con un catch.

```
const url_api: string = "https://jsonplaceholder.typicode.com";

fetch(`${url_api}/users`)//fetch  devuelve una promesa. Hace una peticion get a la api
    .then(response => response.json())//si sale bien la conexion parseamos el json
    .then(users => {//si sale bien el parseo mostramos el resultado por consola
        let usuario = users.map((user:any) => `ID: ${user.id} \nNombre: ${user.name} \nEmail: ${user.email} \nTelefono: ${user.phone}`);
        for (const iterator of usuario) {
            console.log(iterator);
        }
    })
    .catch(error => console.error("Error controlado por catch: ",error));//si algo sale mal, mostramos el error por consola
```

```
ID: 1                                   Cuestion3.js:19
Nombre: Leanne Graham
Email: Sincere@april.biz
Telefono: 1-770-736-8031 x56442

ID: 2                                   Cuestion3.js:19
Nombre: Ervin Howell
Email: Shanna@melissa.tv
Telefono: 010-692-6593 x09125

ID: 3                                   Cuestion3.js:19
Nombre: Clementine Bauch
Email: Nathan@yesenia.net
Telefono: 1-463-123-4447

ID: 4                                   Cuestion3.js:19
Nombre: Patricia Lebsack
Email: Julianne.OConner@kory.org
Telefono: 493-170-9623 x156

ID: 5                                   Cuestion3.js:19
Nombre: Chelsey Dietrich
Email: Lucio_Hettinger@annie.ca
Telefono: (254)954-1289

ID: 6                                   Cuestion3.js:19
Nombre: Mrs. Dennis Schulist
Email: Karley_Dach@jasper.info
Telefono: 1-477-935-8478 x6430

ID: 7                                   Cuestion3.js:19
Nombre: Kurtis Weissnat
Email: Telly.Hoeger@billy.biz
Telefono: 210.067.6132

ID: 8                                   Cuestion3.js:19
Nombre: Nicholas Runolfsdottir V
Email: Sherwood@rosamond.me
Telefono: 586.493.6943 x140

ID: 9                                   Cuestion3.js:19
Nombre: Glenna Reichert
Email: Chaim_McDermott@dana.io
Telefono: (775)976-6794 x41206

ID: 10                                  Cuestion3.js:19
Nombre: Clementina DuBuque
Email: Rey.Padberg@karina.biz
Telefono: 024-648-3804
```

- Controla si la conexión genera un error.

  He modificado un poco el código anterior añadiendo una función que comprueba el estado de la petición, aun así, el código anterior ya disponía de un catch en caso de errores.

```
const url_api: string = "https://jsonplaceholder.typicode.com";

fetch(`${url_api}/users`)//fetch  devuelve una promesa. Hace una peticion get a la api
    .then(response => isResponseOk(response))//si sale bien la conexion parseamos el json
    .then(users => {//si sale bien el parseo mostramos el resultado por consola
        let usuario = users.map((user:any) => `ID: ${user.id} \nNombre: ${user.name} \nEmail: ${user.email} \nTelefono: ${user.phone}`);
        for (const iterator of usuario) {
            console.log(iterator);
        }
    })
    .catch(error => console.error("Error controlado por catch: ",error));//si algo sale mal, mostramos el error por consola

//funcion para controlar los errores a la hora de realizar la conexion
const isResponseOk = (response:any) => {
    if (!response.ok)
        throw new Error(response.status);
    return response.json()
}
```

https://github.com/Raulmedinagn/DWEC_Hito2_Raul_Medina