Hito 2

**CUESTIÓN 1. Declaración de variables en TypeScript**

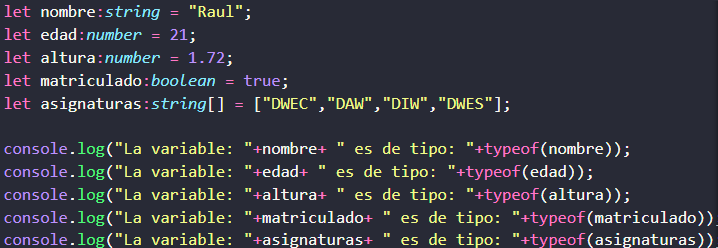
* Explica qué tipado (débil o fuerte) se aplica en TypeScript

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.

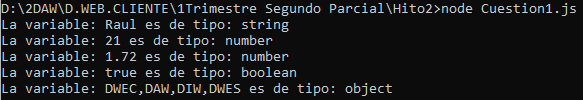
* Realiza un ejemplo con declaración de diferentes variables: texto, numéricas, booleanas

y de listas. Identifica correctamente el tipo de dato de cada una.

Declaración e impresión de variables

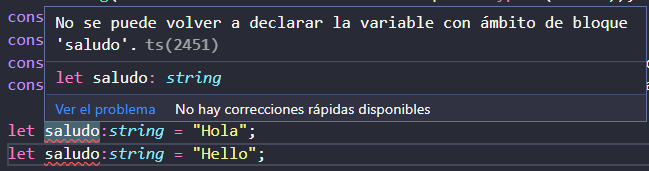


La consola muestra:

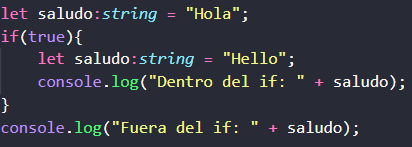


* Explica la diferencia entre let, var y const con un ejemplo.

**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.

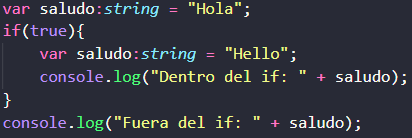
** 

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.

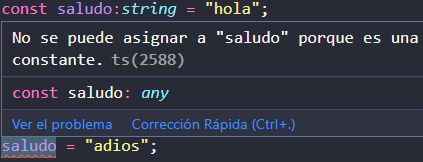


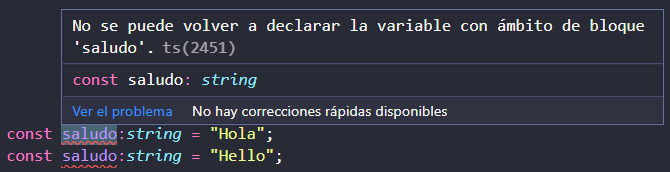
var permite redefinir la misma variable en el mismo bloque.

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).



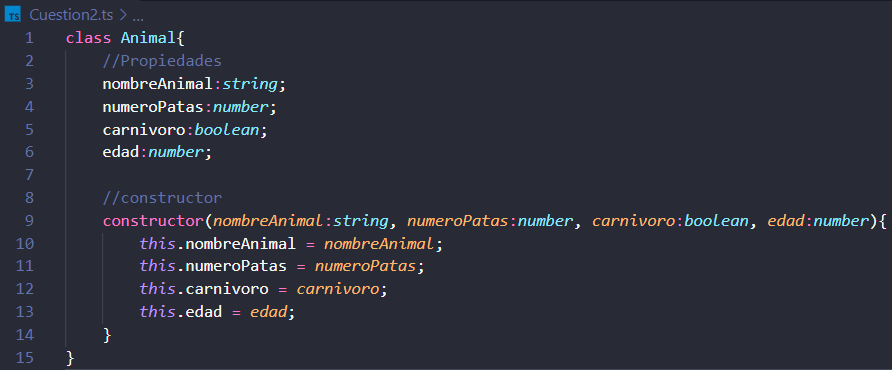


No te permite cambiar su valor, ni redefinir la variable

**CUESTIÓN 2. POO en TypeScript**

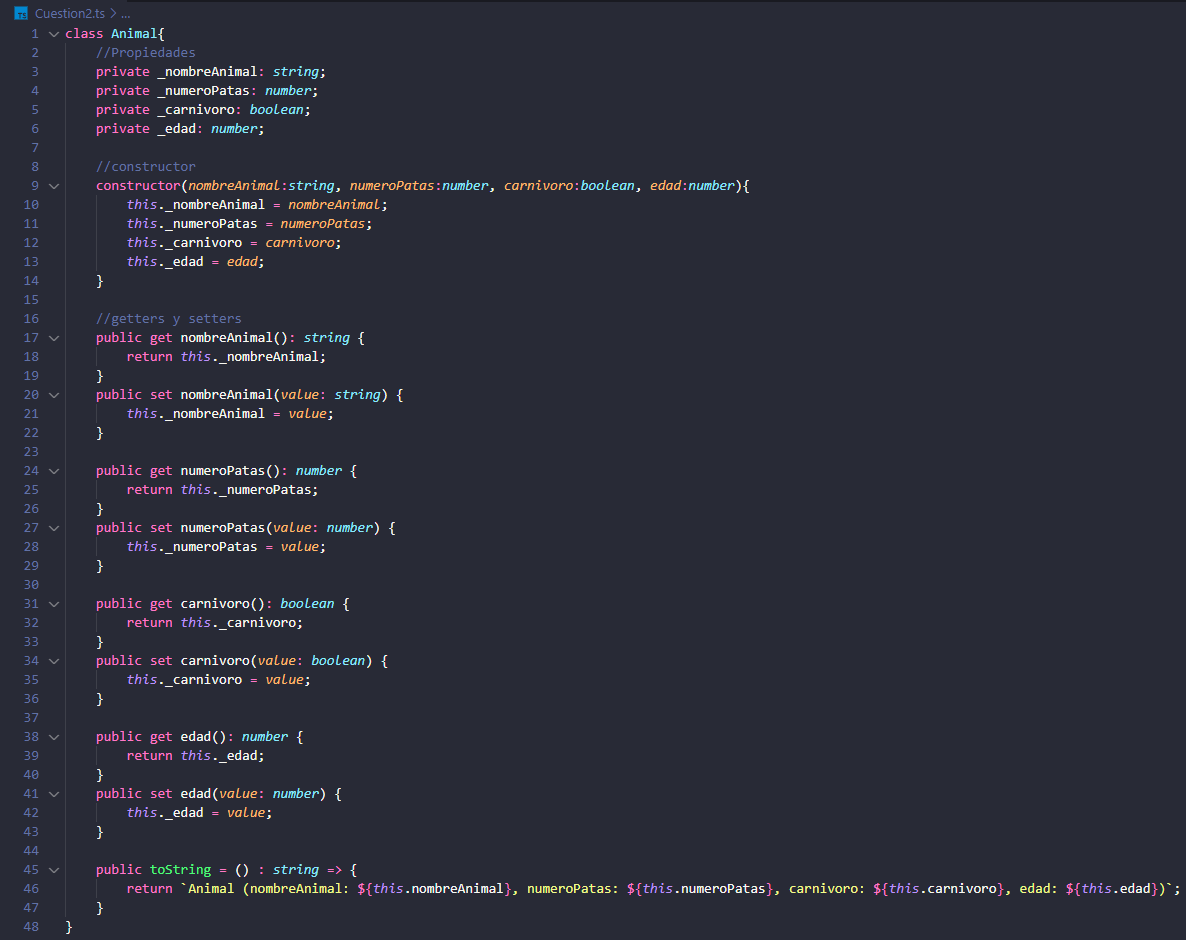
* Crea una clase llamada Animal. Esta clase tiene como propiedades nombreAnimal,

númeroPatas, carnívoro y edad.



* Utiliza encapsulamiento, propiedades, constructores y getters y setters para gestionar

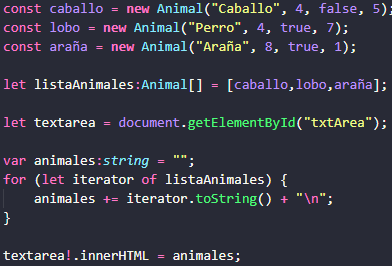
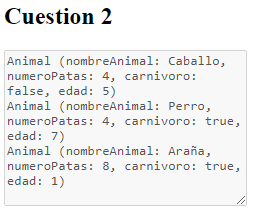
estas propiedades



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

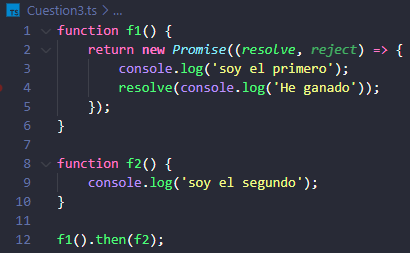
 

**CUESTIÓN 3. Promises en TypeScript**

* 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.

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

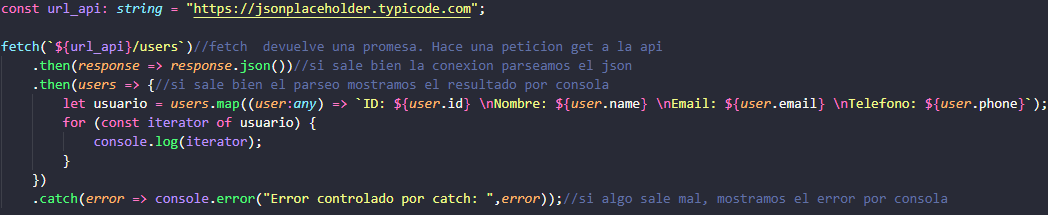
resultado en consola.

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.

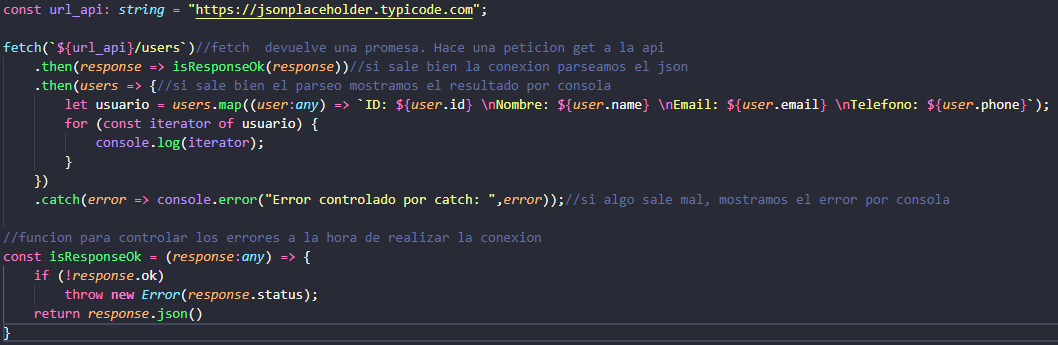
Me he conectado a ella mediante fetch que devuelve una promesa, la cual he manejado para mostrar por consola algunos datos relevantes obtenidos y en el caso de que hubiera errores los he manejado con un catch.





* 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.



Raúl Medina García-Navas