TypeScript

Introducción

JavaScript: evolución

- En los últimos años JavaScript ha crecido de forma exponencial.
- Pero cuando fue creado en los 90 no fue concebido para ser un lenguaje de programación completo.
- A lo largo de los años evolucionó.
- Empresas vieron una gran oportunidad en el lenguaje y su protagonismo en la web para desarrollar sus frameworks (Google – Angular, Facebook – React).
- También se pensó en JavaScript como lenguaje del servidor (NodeJS).

JavaScript: carencias

- Hoy en día el código de las aplicaciones es extenso.
- A menudo requiere un equipo de trabajo para desarrollarlas.
- El tipo dinámico de las variables en JavaScript supone un problema para este tipo de desarrollo.
- Pero también hay otras carencias:
  - Detección de errores limitado en tiempo de escritura de código.
  - Auto completado deficiente.
  - Tipado de respuestas http inexistente.
  - Clases y módulos todavía sin implementar como sí ocurre en otros lenguajes.
  - Errores porque una variable no está definida o un objeto no tiene una propiedad.
  - Errores porque se utilizan variables con el mismo nombre.
  - Errores porque escribimos mal el nombre de una variable, por ejemplo, cuando escribimos una mayúscula donde no procede.
  - Estos errores se producen en ejecución.

TypeScript: objetivo

- Busca tener la misma experiencia de desarrollo de lenguajes como Java o C# en JavaScript.
- TypeScript es un JavaScript mejorado, un *superset* que añade nuevas características.
- Todo lo que sabemos de JavaScript lo podremos aplicar en TypeScript.
- TypeScript no puede ejecutarse en el navegador. Se transpila a JavaScript para que sea ejecutable.
- Así que TypeScript nos permite escribir código moderno que será soportado por los navegadores web.
- Lo podremos utilizar en React y Node. Angular lo ha tomado como lenguaje obligado.

Tipos de datos

- Hasta donde sea posible, TypeScript va a tratar de adjudicar un tipo a cada variable o constante que creemos.
- A esto se le llama “inferir” el tipo.
- Los tipos de datos básicos son:
  - **string**: Para almacenar textos.
  - **number**: Bajo number se aglutinan todos los datos de tipo numérico.
  - **boolean**: true/false
  - **any**: Admite cualquier tipo de valor.
  - **arrays**: Como en JavaScript, los usaremos para implementar colecciones de información.

Otros tipos de datos: Tuplas

- Una tupla en TypeScript es un array de elementos que están tipados.
- De esta manera cada vez que haya que crear un elemento se validará que dicho elemento coincida con el tipo de dato establecido en la tupla.
- Puede ser útil para disponer de elementos compuestos de datos de diferente tipo y donde sea importante el orden en que se indexan estos datos.

Otros tipos de datos: Enumeraciones

- Ayudan a trabajar con datos que tienen un fuerte sentido semántico y cuya información esté circunscrita a unos determinados valores.
- Las enumeraciones permiten asociar unos valores semánticos a otros numéricos
- Pueden ser útiles para disponer de elementos compuestos de datos de diferente tipo y donde sea importante el orden en que se indexan estos datos.

Otros tipos de datos: void

- El tipo void es propio de las funciones. Cuando una función en TypeScript retorna algo, ese algo debe ser de un tipo.
- Las funciones void son aquellas que no retornan valores.

¿Qué es POO? (I)

- No es un lenguaje de programación sino una forma de programar, una metodología.
- Intenta llevar al código lo mismo que encontramos en el mundo real.
- Cuando miramos a nuestro alrededor, ¿qué vemos?
- Vemos OBJETOS.

¿Qué es POO? (II)

- OBJETOS.
  - Podemos distinguir un objeto por que pertenece a una **clase**.
  - Distinguimos un perro de un coche porque pertenecen a clases diferentes.
  - Distinguimos un televisor de otro porque aunque pertenezcan a la misma clase, cada objeto es diferente.
- Esta es el modelo que intenta seguir la POO para estructurar un sistema.

Características de la POO (I)

- **Abstracción**:
  - Es algo conceptual.
  - Define nuevos tipos de entidades en el mundo real.
  - Depende de la perspectiva del observador.
  - Se centra en la visión externa de un objeto, Ej.: Pastor, pekinés o bulldog son derivados de la idea abstracta “perro”.

Características de la POO (II)

- **Jerarquía:**
  - Las abstracciones forman una jerarquía.
  - Seccionamos el código en partes más pequeñas que al unirlas hacen el trabajo.
  - Ej.: Máquinas de locomoción →de tierra →sin motor→ con pedales → para niños → triciclo.

Características de la POO (III)

- **Encapsulación y ocultamiento**:
  - La encapsulación se encarga de mantener ocultos los procesos internos que se necesitan para hacer lo que sea que haga. → El programador accede sólo a lo que necesita.

Características de la POO (IV)

- **Encapsulación y ocultamiento**:
  - Un objeto se comunica con el resto del mundo mediante su **interfaz** → la lista de métodos y propiedades que hace público.
  - Ventaja: Lo que hace el usuario puede ser controlado internamente, incluso los errores, evitando que todo colapse.

Características de la POO (V)

- **Herencia**:
  - Capacidad que tiene una clase de derivar las propiedades y métodos de otra.
  - Decimos que una gallina es un ave → las gallinas tienen características comunes con otras aves (pico, plumas…) → la gallina hereda las características comunes de aves. → Además un ave es un animal → comparte características comunes con el caballo, el perro, el hombre.

Características de la POO (VI)

- **Herencia**:
  - Ventajas: Evita tener que escribir el mismo código una y otra vez ya que al definir una categoría (que llamaremos clase) que pertenece a otra, automáticamente atribuimos las características de la primera sin tener que definirlas de nuevo.
  - Ejemplo: Controles Formularios Windows →Cuadro de texto → Cuadro de texto con máscara

Clase (I)

- Sabemos que “ave” se refiere a algo que cumple unas determinadas características, se comporta de una forma concreta → tiene plumas, pico, dos patas… → “ave” es una palabra que nos permite clasificar las cosas.
- “ave” identifica a un grupo, no a un ave en concreto.
- Es un concepto, una abstracción.

Clase (II)

- La clasificación (crear clases) es la tarea de agrupar cosas según sus características y sus capacidades.
- **Cada objeto debe pertenecer a una clase específica**, de acuerdo a sus características y si esa clase no existe, se crea.

Clase (III)

- Por lo tanto, una clase es una serie de código que define a todos los elementos relacionados → es una plantilla, un molde a partir de la que se crean los objetos.
- La clase posee todas las variables y funciones que deben tener un objeto → el objeto tendrá valores propios en esas variables cuando sea creado.
- Un objeto no es una clase, sino el **resultado de crear un ejemplar de esa clase**.

Clase (IV)

- JavaScript tiene muchas clases preconstruidas que utilizamos para desarrollar aplicaciones.
- Formato de una clase en JavaScript:

```
class NombreClase {
    constructor
    propiedades
    métodos
}
```

- La palabra clave **class** siempre encabeza la definición de una clase.

Objeto

- Un objeto es cualquier cosa del mundo real: una bici, un perro, un televisor, una excursión…
- Todos los objetos tienen 2 características:
  - Propiedades/características
  - Comportamiento
- Ej.: perro → nombre, color, raza → ladrar, buscar, menearCola.
- En nuestras clases almacenaremos las propiedades de un objeto como propiedades y los comportamientos como métodos.

Ejemplares/Objetos de una clase (I)

- Una clase es una acción declarativa → no ocurre nada hasta que no creamos su objeto.
- A la creación de un ejemplar de una clase se le denomina instanciación. Al objeto, instancia.
- Ej.: creamos un objeto de la clase TV

```
TV miTV = new TV()
```

- El operador new seguido del nombre de la clase, crea un objeto de dicha clase en una posición de memoria a la que accederemos con el nombre miTV.

Ejemplares/Objetos de una clase (II)

- Ahora puedo usar libremente lo que me permita la interfaz de miTV.
- Ejemplos:

```
miTV.Encender()
miTV.CambiarCadena(2)
miTV.Apagar()
```

Atributos / variables

- Forman la estructura interna de los objetos de una clase. Son los que hacen a un objeto diferente de otro.
- Se declaran exactamente igual que cualquier otra variable pudiendo incluso inicializarse con un valor.

Constructores (I)

- Son métodos que se llaman automáticamente al crear un objeto de una clase.
- Su misión principal es iniciar nuevos objetos de una clase.
- El método constructor no se puede llamar, se autoejecuta al crear un objeto con la palabra reservada **new**.

Métodos (I)

- Un método es un conjunto de declaraciones asociadas a un nombre.
- Se ejecutarán todas cuando se llame al método.
- Forman lo que se denomina interfaz o medio de acceso a la estructura interna de las clases.
- Definen las operaciones que pueden realizarse con las propiedades.