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: 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.