MEDAC
Instituto Oficial de Formación Profesional

DESARROLLO WEB EN ENTORNO CLIENTE
TÉCNICO EN DESARROLLO DE APLICACIONES WEB

Lenguajes de alto nivel

13

ÍNDICE

/ 1. Introducción y contextualización práctica 3

/ 2. Nuevos lenguajes de script: TypeScript 4

2.1. TypeScript vs JavaScript 4

/ 3. Desarrollando en TypeScript: Tipos básicos y declaración de variables 5

/ 4. Caso práctico 1: "Tipo Any" 6

/ 5. Desarrollo en TypeScript 6

/ 6. Desarrollando en TypeScript: Interfaces y enumerados 7

/ 7. Manejo de errores 8

7.1. Captura y lanzamiento de excepciones 9

/ 8. Uso de clases y compilación de Typescript a JavaScript 9

/ 9. Caso práctico 2: "Permitirnos errores" 10

/ 10. Resumen y resolución del caso práctico de la unidad 11

/ 11. Bibliografía 11

OBJETIVOS

Comprender las limitaciones de JavaScript.

Entender los motivos de crear un nuevo lenguaje.

Comprender los fundamentos de los lenguajes de alto nivel.

Saber utilizar los lenguajes de alto nivel.

Comprender los mecanismos de compilación en entorno web.

Saber crear código en lenguajes web de alto nivel.

/ 1. Introducción y contextualización práctica

El número de tecnologías existentes para el desarrollo de soluciones en el *front-end* crece constantemente y los desarrolladores tienen que estar continuamente actualizando sus conocimientos. Un ejemplo de ello es la utilización de gran cantidad de librerías y *frameworks* para el desarrollo en entorno de cliente. Teniendo en cuenta que cada vez las aplicaciones son más exigentes y el time-to-market (TTM) más corto, es interesante poder utilizar tecnologías que faciliten el desarrollo en la medida de lo posible.

Cuando utilizamos JavaScript, nos encontramos ante un lenguaje relativamente antiguo que fue utilizado en sus orígenes para realizar pequeños scripts y que en la actualidad se utiliza como lenguaje base de la gran mayoría de las librerías existentes para el mundo web.

No es de extrañar que los esfuerzos de los desarrolladores de lenguaje de programación se centren en facilitar la escritura de soluciones web, creando nuevos lenguajes más seguros y que requieran menos cantidad de código.

En este tema, presentamos TypeScript para facilitar la escritura de código JavaScript y tener soluciones más robustas, en menor tiempo y con mejor calidad si es posible.

Fig. 1. Logo TypeScript.

Audio intro. "Migrado a TypeScript"

http://bit.ly/3afqQQp

/ 2. Nuevos lenguajes de script: TypeScript

TypeScript es un lenguaje de programación de código abierto desarrollado por Microsoft para el desarrollo basado en entornos JavaScript. Se trata de un lenguaje creado sobre JavaScript que modifica la forma de crear objetos y clases. TypeScript extiende la sintaxis de JavaScript, por lo que cualquier código desarrollado con JavaScript debería también poder ejecutarse sobre TypeScript.

Como norma general, TypeScript está pensado para desarrollar proyectos de gran tamaño, ofreciendo un compilador que traduce de TypeScript a JavaScript. En la actualidad, TypeScript es utilizado en muchos frameworks de desarrollo en el cliente e incluso en desarrollo en el servidor. Por ejemplo, Angular utiliza TypeScript para sus desarrollos como lenguaje de programación por defecto.

Una de las **ventajas** de utilizar TypeScript es la formalización de los tipos con respecto a JavaScript. Con TypeScript es posible definir los tipos de una variable para evitar posibles errores en tiempo de ejecución.

Algunas de las **principales características de TypeScript** son:

- **El código TypeScript convertido a código JavaScript.** Los navegadores no son capaces de entender código escrito en TypeScript, por ello, es necesario realizar una conversión de TypeScript a JavaScript utilizando un proceso de compilación
- **JavaScript es TypeScript.** Cualquier código escrito en JavaScript se considera código TypeScript, básicamente cambiando la extensión del archivo. En JavaScript, los archivos tienen extensión .js, mientras que en TypeScript, tienen extensión .ts.
- **TypeScript es un lenguaje de propósito general.** Este lenguaje no está vinculado a los navegadores, dispositivos, ni sistemas operativos. Gracias al compilador, se puede utilizar en cualquier contexto o, incluso, utilizarlo como lenguaje final.
- T**ypeScript es capaz de soportar las librerías en JavaScript.** TypeScript es un superconjunto de JavaScript, por ello, es posible utilizar las librerías de JavaScript desde TypeScript.

Vídeo 1. "Preparación del entorno para TypeScript"
https://bit.ly/3mk70Wx

2.1. TypeScript vs JavaScript

A la hora de utilizar TypeScript, es importante conocer las diferencias que introduce este lenguaje con respecto a JavaScript. A continuación, mostramos algunas de las diferencias más relevantes:

- **TypeScript se conoce como un lenguaje orientado a objetos** mientras que JavaScript se conoce como un lenguaje de *scripting*.
- TypeScript **ofrece un tipado estático**.
- TypeScript **permite utilizar módulos**, mientras que JavaScript no.
- TypeScript permite **definir interfaces**, mientras que en JavaScript no es posible.
- TypeScript **ofrece la posibilidad de crear funciones con parámetros opcionales.** En JavaScript, el número de parámetros directamente no es comprobado.

Algunas de las ventajas de utilizar TypeScript es que el proceso de compilación nos devuelve los posibles errores antes de pasar a ejecución. Esto es posible porque TypeScript es un lenguaje compilado, mientras que JavaScript es un lenguaje interpretado. Además, TypeScript es un lenguaje fuertemente tipado que soporta tipos estáticos, lo que significa que estos tipos se comprueban en tiempo de compilación. En JavaScript, la comprobación de tipos no existe, así, todos los errores suceden en tiempo de ejecución. Gracias al compilador de TypeScript, es posible generar código JavaScript para diferentes tipos de navegadores, obviando así las necesidades de cada uno de ellos.

La principal **desventaja de TypeScript** frente a JavaScript es el tiempo requerido para la compilación que, en proyectos de relativo tamaño, puede ser significativa, retrasando así las pruebas que se necesiten llevar a cabo.

Fig. 2. TypeScript vs JavaScript.

Audio 1. "TypeScript vs JavaScript
http://bit.ly/3ao8Ovg

/ 3. Desarrollando en TypeScript: Tipos básicos y declaración de variables

Una vez que ya estamos familiarizados con JavaScript y con algunas librerías existentes, añadir la capa de desarrollo de TypeScript es relativamente sencillo. Gracias a este lenguaje vamos a poder desarrollar nuestro código de una forma más rápida y segura. Como hemos dicho anteriormente, TypeScript es un lenguaje orientado a objetos y, como tal, permite la herencia. Todos los objetos del lenguaje heredan del tipo *Any*.

a. **Tipos básicos**. TypeScript ofrece un gran número de tipos básicos que pueden ser utilizados en cualquier contexto, pudiendo ser objetos o no. A continuación, se muestra una tabla con los principales tipos básicos:

Tipo	Descripción
Boolean	Permite definir valores booleanos
Number	Permite definir valores numéricos
String	Define una cadena de caracteres
Array	Permite definir listas
Enum	Define valores enumerados
Any	Permite definir un valor de cualquier tipo
Void	Permite definir la ausencia de tipo

Tabla 1. Tipos básicos TypeScript

b. **Declaración de variables**. La declaración de variables en JavaScript se ha realizado tradicionalmente utilizando la palabra reservada *var*. La declaración en TypeScript utiliza la palabra reservada *let*.

   Aunque *a priori* podemos pensar que la mayor diferencia es el cambio de palabra reservada, realmente supone un cambio más profundo, desde el punto de vista del éxito del uso del ámbito de visibilidad.

Fig. 3. TypeScript se basa en JavaScript.

/ 4. Caso práctico 1: "Tipo Any"

Planteamiento: Al realizar el análisis del código de nuestra aplicación TypeScript, observamos que todas las variables están declaradas con el tipo *Any*.

Nudo: ¿Crees que es una buena solución?

Desenlace: Como hemos visto en el apartado sobre el desarrollo en TypeScript, el lenguaje incluye grandes medidas de control de tipo, sin embargo, cuando utilizamos el tipo *Any*, perdemos información de tipos.

Con esta aproximación, es posible que se produzcan errores de tipo en tiempo de ejecución, justo uno de los puntos que intenta solucionar TypeScript.

Para solucionar esto, habría que redefinir todos los elementos al tipo que le corresponde. Aunque es un trabajo bastante tedioso, y en muchas ocasiones "peligroso", es la única forma de poder recuperar toda la potencia que introduce TypeScript para el control de tipos.

Fig. 4. Cada elemento con su tipo.

/ 5. Desarrollo en TypeScript

En JavaScript, las funciones se pueden utilizar exactamente igual que lo haríamos con otro objeto del lenguaje, es decir, se pueden asignar a variables, se pueden pasar a otras funciones por parámetro, pueden ser devueltas en otras funciones, etc.

Cuando utilizamos TypeScript, todo esto se realiza de una forma más elegante y segura, pues se permite introducir mayor cantidad de información a las funciones que permite ayudar al programador y crear una sintaxis más amigable.

Fig. 5. TypeScript permite declarar funciones.

TypeScript puede inferir el tipo de las variables que forman parte del cuerpo de una función, pero, generalmente, no puede inferir el tipo de los parámetros de una función, siendo necesario indicar el tipo de los parámetros cuando declaremos la función. Gracias a ello, podremos evitar errores de tipo y podremos saber el tipo de datos devuelto, simplemente leyendo su definición.

Si lo comparamos con JavaScript, por un lado, las funciones no tendrían que declarar el tipo de los parámetros en su definición. Por otro lado, las funciones no tienen que indicar el tipo devuelto en su definición, es más, no sabemos si devuelve algo hasta que no analicemos el cuerpo de la función.

```
function optionalParam(name: string,
surname?: string) {
   if (surname)
      return name + " " + surname;
   else
      return name;
}
}
;
```

Código 1. Función.

Una función en TypeScript puede tener parámetros opcionales a través del operador "?".

Puedes ver un ejemplo de utilización de funciones en el anexo del tema.

/ 6. Desarrollando en TypeScript: Interfaces y enumerados

a. **Interfaces**.

Al utilizar clases, es muy común que se vinculen ellas con interfaces. Una interfaz no es más que una forma de nombrar a un tipo que no puede ser definido en una línea. Las interfaces permiten establecer una jerarquía y poder definir diferentes tipos de elementos que contendrán campo de datos. Con las interfaces podemos ahorrar líneas de código.

```
interface Comida {
  calorias: number
  sabroso: boolean
}
interface Asiatica extends Comida {
  salada: boolean
}
interface Occidental extends Comida {
  dulce: boolean
}
```

Código 2. Jerarquía de interfaces.

Como se puede ver en el ejemplo anterior, se define una jerarquía de interfaces en la que podemos trabajar con las dos interfaces que heredan de **comida** de la misma forma, pues ambas son interfaces de **comida**.

b. **Enumerados**.

Los enumerados permiten definir un conjunto de nombres constantes que facilitan el manejo de un documento, la creación de conjuntos para diferentes casos, etc.

Con TypeScript, se pueden crear enumerados de tipo numérico o con cadena de caracteres.

```
enum EnumNumerico {
   Arriba = 1,
   Abajo,
   Izquierda,
   Derecha,
}

enum EnumCadena {
   Arriba="Arriba",
   Abajo="Abajo",
   Izquierda="Izquierda",
   Derecha="Derecha",
}
```

Código 3. Enumerados.

/ 7. Manejo de errores

TypeScript realiza todo lo posible para detectar el mayor número de errores en tiempo de compilación, sin embargo, existen errores que no se pueden detectar en este momento y tendrán que pasar a tiempo de ejecución.

Cuando un error se produce en tiempo de ejecución, el comportamiento del código es indeterminado. Debido a esto, los lenguajes de programación más modernos incluyen un manejador de excepciones.

Utilizando el manejador en tiempo de ejecución, se permite que el código se comporte de una manera controlada y pueda permitir que el resto de la aplicación continúe funcionando con normalidad.

En JavaScript, esto no es posible, ya que, por un lado, no existe la fase de compilación, por lo que no se pueden evitar errores, y por otro lado, los errores en tiempo de ejecución no se pueden capturar, provocando que la aplicación deje de funcionar sin motivo aparente.

Una de las fuentes más propensas a introducir errores es la interfaz de usuario. Desde el punto de vista de la programación en entorno cliente, la tarea principal es la interacción con el usuario, por lo tanto, es el punto por el cual pueden surgir el mayor número de errores en nuestro ámbito de desarrollo.

Un error muy típico es que el usuario olvide introducir un campo y en nuestro procesamiento ese campo sea nulo.

Veamos un ejemplo sencillo en el siguiente apartado.

Fig. 6. TypeScript puede manejar y recuperarse de errores.

7.1. Captura y lanzamiento de excepciones

a. **Captura de excepciones**

Basándonos en el caso expuesto anteriormente, en el que un usuario pueda olvidar introducir un campo, será necesario comprobar constantemente que ese campo sea diferente de nulo. Con esta aproximación, el código se puede complicar, simplemente por preguntas sobre el estado de un campo.

Un manejador de excepciones permite simplificar los errores cuando tienen lugar.

Veamos un ejemplo:

```
// ...
try {
  let date = procesa(interaccion())
  console.info('Date is', date.toISOString())
} catch (e) {
  console.error(e.message)
}
console.error(e.message)
}
```

Código 4. Manejador de excepciones.

En el ejemplo anterior, utilizando la sintaxis try-catch, podemos implementar un manejador de excepciones. La idea es dejar que se produzcan excepciones y gestionarlas correctamente.

b. **Lanzamiento de excepciones**

Al igual que podemos capturar las excepciones, en ocasiones, podemos decidir delegar esa tarea en el usuario de nuestro código. TypeScript proporciona métodos para poder definir funciones que lancen excepciones.

Cuando usemos una función que lanza una excepción, es obligatorio capturarla o propagarla.

```
function parse(birthday: string): Date {
  let date = new Date(birthday)
  if (!isValid(date)) {
    throw new InvalidDateFormatError('Fecha no
válida')
  }
  ...
  return ...
}
```

Código 5. Lanzamiento de excepciones.

/ 8. Uso de clases y compilación de Typescript a JavaScript

a. **Clases**

JavaScript utiliza funciones y el prototipado basado en herencia para poder construir componentes que puedan ser reutilizados. Sin embargo, esto puede ser bastante tedioso para aquellos programadores acostumbrados a la orientación a objetos, donde se usan las clases y objetos.

En TypeScript, es posible que los programadores utilicen la orientación a objetos tal y como la conocen, pudiendo compilar dicho código a JavaScript.

```
class Saludador {
   saludo: string;
   constructor(texto: string) {
      this.saludo = texto;
   }
   greet() {
      return "Hola, " + this.saludo;
   }
}
//uso de la clase
let greeter = new Saludador("Mundo");
```

Código 6. Uso de clases.

b. **Compilación de TypeScript a JavaScript**

Para poder compilar código TypeScript, es necesario instalar el compilador conocido como **tsc.**

El compilador debe ser instalado de manera global en nuestra máquina para que pueda ser accesible desde cualquier línea de comando. La forma más sencilla de instalar el compilador de TypeScript es a través del gestor de paquetes Node.js

Una vez que tengamos el gestor de paquetes instalado, podremos instalar TypeScript con el siguiente comando:

```
npm install -g typescript.
```

Para compilar un archivo typescript a javascript, utilizamos el siguiente comando:

```
tsc inputFile.ts
```

Vídeo 2. "TypeScript y declaración de clases"
https://bit.ly/2Wi3DFd

/ 9. Caso práctico 2: "Permitirnos errores"

Planteamiento: Cuando analizamos el código de nuestra aplicación TypeScript, observamos que hay muchas sentencias condicionales para comprobar cada uno de los campos de entrada. Esto está generando que algún campo no se compruebe y se produzca un error irrecuperable.

Nudo: ¿Se te ocurre alguna forma de solucionar este problema?

Desenlace: Como hemos visto en el apartado sobre el manejo de errores, en TypeScript no es necesario realizar la comprobación de todos los casos de entrada.

Un enfoque que suele utilizarse es el tratamiento de errores a posteriori, es decir, al tener manejador de excepciones, podemos "permitirnos" que se produzcan los errores e informar cuando se produzcan.

Con esta aproximación, es muy importante que el manejo del error se realice correctamente para obtener el resultado deseado. Al realizarlo de esta forma, nos ahorramos tener muchos "ifs" anidados, dejando un código más ligero y legible.

Fig. 7. TypeScript puede manejar errores a posteriori.

/ 10. Resumen y resolución del caso práctico de la unidad

A lo largo de esta unidad, hemos presentado el lenguaje **TypeScript** como alternativa a JavaScript para desarrollar en el lado del cliente.

Como hemos visto, TypeScript ofrece una organización del código más eficiente y permite realizar mayor cantidad de tareas con menos cantidad de líneas. Se trata de un lenguaje relativamente nuevo que sigue la filosofía de escribir menos código, por ejemplo, los puntos y comas son opcionales.

TypeScript ofrece la posibilidad definir **clases,** crear **objetos**, definir **funciones**, etc.

Una característica de TypeScript es que no puede ser interpretado directamente por un navegador web, sino que tiene que ser **compilado** a un lenguaje destino que, en este caso, es JavaScript.

TypeScript introduce algunas mejoras como el **manejo de errores** y un sistema de tipado seguro, garantizando que los errores de tipos se producirán, en su gran mayoría, en tiempo de compilación.

Resolución del caso práctico inicial

Como hemos visto en este tema, TypeScript es un lenguaje muy potente que facilita el desarrollo de aplicaciones para generar código JavaScript. Lo que nos pide nuestro cliente, simplemente, no es posible con la tecnología existente en la actualidad. Los navegadores existentes no son capaces de interpretar ni ejecutar código TypeScript. Por ello, debemos introducir una fase previa de compilación para generar código JavaScript a partir del código TypeScript. El código generado sí podrá ser interpretado por el navegador.

Además, la mayoría de las librerías existentes para el desarrollo en el *front-end* están basadas en JavaScript. Si no nos deja utilizar dichas librerías, no podríamos implementar gran parte de la funcionalidad que se requiere en la web.

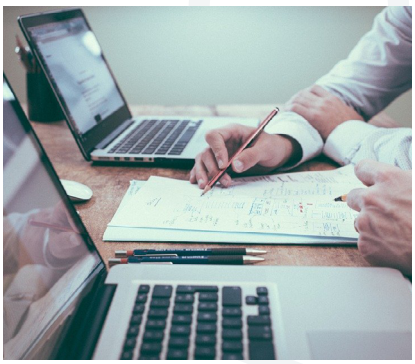
Fig. 8. Exposición de motivos al cliente.

/ 11. Bibliografía

Cherny, B. (2019). Programming *TypeScript : making your JavaScript applications scale*. Sebastopol, CA: O'Reilly Media, Inc.

Freeman, A. (2019). Essential TypeScript : *from beginner to pro*. Berkeley, CA: Apress.