La sintaxis de un lenguaje de programación se define como el conjunto de reglas que deben seguirse al escribir el código fuente de los programas para considerarse como correctos para ese lenguaje de programación.

La sintaxis de JavaScript es muy similar a la de otros lenguajes de programación como Java y C. Las normas básicas que definen la sintaxis de JavaScript son las siguientes:

No se tienen en cuenta los espacios en blanco y las nuevas líneas: como sucede con XHTML, el intérprete de JavaScript ignora cualquier espacio en blanco sobrante, por lo que el código se puede ordenar de forma adecuada para entenderlo mejor (tabulando las líneas, añadiendo espacios, creando nuevas líneas, etc.)

Se distinguen las mayúsculas y minúsculas: al igual que sucede con la sintaxis de las etiquetas y elementos XHTML. Sin embargo, si en una página XHTML se utilizan indistintamente mayúsculas y minúsculas, la página se visualiza correctamente, siendo el único problema la no validación de la página. En cambio, si en JavaScript se intercambian mayúsculas y minúsculas el script no funciona.

No se define el tipo de las variables: al crear una variable, no es necesario indicar el tipo de dato que almacenará. De esta forma, una misma variable puede almacenar diferentes tipos de datos durante la ejecución del script.

No es necesario terminar cada sentencia con el carácter de punto y coma (;): en la mayoría de lenguajes de programación, es obligatorio terminar cada sentencia con el carácter ;. Aunque JavaScript no obliga a hacerlo, es conveniente seguir la tradición de terminar cada sentencia con el carácter del punto y coma (;).

Se pueden incluir comentarios: los comentarios se utilizan para añadir información en el código fuente del programa. Aunque el contenido de los comentarios no se visualiza por pantalla, si que se envía al navegador del usuario junto con el resto del script, por lo que es necesario extremar las precauciones sobre la información incluida en los comentarios.

JavaScript define dos tipos de comentarios: los de una sola línea y los que ocupan varias líneas.

Ejemplo de comentario de una sola línea:

```
// a continuación se muestra un mensaje

alert("mensaje de prueba");
```

Los comentarios de una sola línea se definen añadiendo dos barras oblicuas (//) al principio de la línea.

Ejemplo de comentario de varias líneas:

```
/* Los comentarios de varias líneas son muy útiles
```

```
cuando se necesita incluir bastante información
en los comentarios */
alert("mensaje de prueba");
```

Los comentarios multilínea se definen encerrando el texto del comentario entre los símbolos `/*` y `*/`.

TypeScript

Typescript es un lenguaje que creó **Microsoft**. Su sintaxis es una especie de mezcla entre **Javascript y Java**. Todo el código que escribas en Typescript será compilado en código Javascript.

Typescript está pensado para **añadir funcionalidad a Javascript**, extenderlo, y por tanto puedes usar código Javascript dentro de métodos y clases de Typescript.

Veamos todo lo que añade Typescript respecto a Javascript:

Tipado estático

Typescript añade tipado estático a Javascript, puedes crear variables con un tipo fijado (**string, números, etc**). El tipado estático es opcional por lo que deja al desarrollador si quiere utilizarlos.

Los tipos que vienen con typescript son:

- **boolean**: (true/false).
- **number**: integers, floats, Infinito y not a number.
- **string**: caractéres o lista de caracteres.
- **[]**: Arrays of other types, like number[] or boolean[].
- **{}**: Objeto literal.
- **undefined**.
- **any**: Tipo especial que permite que una variable pueda tener cualquier tipo.

Para crear funciones se hace igual que en ES6:

```
calcArea() {
  return this.height * this.width;
}
```

Se definen como **un grupo o conjunto de sentencias que solucionan un problema particular**.

Tanto en Javascript como en TypeScript, las funciones tienen una cabecera (donde se define el nombre de la función) y un cuerpo (las instrucciones). Pueden ser definidas de diferentes formas, aunque en todos :

- Funciones con nombre
- Funciones anónimas o métodos anónimos
- Funciones lambda
- Funciones definidas en las clases

Todo un ramillete de opciones, para empezar. En todos los casos, las funciones aceptan argumentos, que además se puede forzar a que correspondan a cierto tipo. Pero ¿que pasa cuando queremos un número no determinado de argumentos? En estos casos, se utiliza ? tras el último parámetro.

Otra circunstancia interesante es la posibilidad de asignar valores predeterminados a los parámetros (cómo en Python), simplemente asignando el valor en la definición. También es interesante el uso de **…variable: string[]**, que es la forma de permitir un número indeterminado de parámetros, que acaban recogidos en la variable *array* **variable**.

```
1	function buildName(firstName: string, ...restOfName: string[]) {
2	return firstName + " " + restOfName.join(" ");
3	}
4
5	var foo = buildName("Joseph", "Samuel", "Lucas"); var foo = buildName("Maria", "Lu");
```

Cómo veis, se cubren todo tipo de opciones respecto a los argumento de las funciones.

Y ahora, empezamos con las funciones

Funciones con nombre

La primera de las opciones se diferencia de las funciones con nombre de Javascript en que los argumentos pueden asignarse tipos, y lo que devuelven las funciones también

```
1	function foo(msg: string) { console.log(msg); }
```

Pocas diferencias respecto a esta misma función si se escribiera directamente en Javascript:

```
1	function foo(msg) { console.log(msg); }
```

Funciones anónimas

Quizás, esta forma de crear funciones es la que yo más utilizo en Javascript, y al igual que antes, la diferencia es que pueden asignarse tipos en los argumentos y en el resultado:

```
1	var foo = function(x: number, y: number): number { return x+y; }
```

En Javascript, es muy similar:

```
1	var foo = function(x, y) { return x+y; }
```

Si la función no devuelve nada, puede marcarse como void, esto es, sin devolución (cómo en Java o en C#).

Funciones lambda, o funciones de flechas gordas

Las funciones lambda son funciones de una sola instrucción (normalmente) que se almacenan en una variable, o que se ejecutan directamente. Se caracterizan de que NO hace falta escribir return,

porque se sobre entiende que la función devuelve algo. Estas funciones lambda no son exclusivas de TypeScript, pues lenguajes como C# o Python también las tienen.

En TypeScript, es identifican por **=>**, que hace de separador entre los argumentos, y lo que devuelve:

```
1	var foo = (x: number, y: number) => x+y;
```

En realidad, la función sin lambda sería tal que así:

```
1	var foo = function (x, y) { return x + y; };
```

En este caso, se trata de ofrecer diferentes formas de escribir el mismo código a los desarrolladores, algo que a mi, personalmente, me parece lioso. Pero, es mi opinión personal.

Funciones en las clases

Cuando se definen funciones en las clases, una de sus peculiaridades es que no aparece function, sino que se escriben directamente:

```
1	class NewClass {
2
3	foo(x: number, y:number): number { return x+y; }
4
5	}
```

Para llamarlas basta con llamar el método y sus respectivos parámetros si son necesarios.