

Módulos en Javascript

En ES6 se introduce el concepto de módulo que no es otro que permitir trocear el código en distintos ficheros, de forma que se permita la exportación de clases, funciones, variables o constantes e importarlos desde otro módulo.

Para poder usar módulos en tu aplicación web, debes añadir el texto `type="module"` en la etiqueta `<script></script>` de tu fichero .js principal. Esta propiedad ya incluye la ejecución en diferido “`defer`”, luego podrás ya quitarla.

Véase el siguiente ejemplo:

```
//---Fichero lib.js -----
export const sqrt = Math.sqrt;
export function square(x) {
    return x * x;
}
export function diag(x, y) {
    return sqrt(square(x) + square(y));
}
```

Ahora podremos llamar a estas funciones, variables, constantes, clases, ... desde otro fichero .js. Para ello usaremos la sintaxis siguiente:

```
/----- main.js -----
import { square, diag } from 'lib.js';
console.log(square(11)); // 121
console.log(diag(4, 3)); // 5
```

En caso de que nuestro fichero tuviera muchas funciones y quisieramos importarlas todas, pero no tener que declararlas una a una, podemos usar un *alias*:

```
import * as Lib from './circle.js';
console.log( 'El cuadrado de 4 es ' + Lib.square(4));
```

Antes de proseguir debes tener clara la diferencia entre **módulos** y **web workers**. El primero es simplemente una forma de dividir la funcionalidad JavaScript entre varios ficheros. El segundo, permite que varios ficheros trabajen de forma paralela y se puedan comunicar entre ellos.

Exportación de módulos

Por defecto, un fichero Javascript no tiene módulo de exportación si no se usa un `export` al menos una vez. Existen varias formas de exportar código mediante la palabra clave `export`:

| Forma | Descripción |
|--|--|
| <code>export { name };</code> | Añade el elemento <i>name</i> al módulo de exportación. |
| <code>export { n1, n2, n3... };</code> | Añade los elementos indicados (n1, n2, n3...) al módulo de exportación. |
| <code>export * from './file.js';</code> | Añade todos los elementos del módulo file.js al módulo de exportación. |
| <code>export declaration;</code> | Declara una variable, función o clase y la añade al módulo de exportación. |
| <code>export default declaration;</code> | Declara una función o clase y la añade al módulo de exportación. |

Es posible renombrar los elementos sobre la marcha utilizando *as* seguido del nuevo nombre.

```
export * as utils from './utils.js'
```

Además, si se indica *default* como nuevo nombre, ese elemento será la exportación por defecto. Sólo puede haber una exportación por defecto por fichero. Se suele usar para indicar cuál es la exportación que contiene el código más importante o principal.

Hay que tener en cuenta que, en el caso de utilizar una exportación *por defecto* en una declaración, no es posible utilizar *var*, *let* o *const*. Tampoco es posible usar *export* dentro de funciones, bucles o contextos específicos.

```
export default saludar;

const saludar = (nombre) => {
    return `¡Hola, ${nombre}! Bienvenido al código.`;
};
```

A la hora de importar...

```
// Nota que no usamos llaves y podemos llamarlo como queramos
import miFuncionDeBienvenida from './saludar.js';

console.log(miFuncionDeBienvenida('Alex'));
```

Importación de módulos

Para importar módulos en otro fichero .js, usaremos *import*. Existen varias formas de importar en un módulo:

| Forma | Descripción |
|---------------------------------------|---|
| import nombre from './file.js'; | Importa sólo el elemento por defecto (default) de file.js en nombre. |
| import { nombre } from './file.js'; | Importa sólo el elemento nombre de file.js. |
| import { n1, n2.. } from './file.js'; | Importa los elementos indicados desde file.js. |
| import * as obj from './file.js'; | Importa todos los elementos de file.js en el objeto obj. |
| import './file.js'; | No importa elementos, pero ejecuta el código de file.js. |

Recuerda que, al igual que con la exportación, también puedes renombrar elementos utilizando **as** seguido del nuevo nombre:

```
import * as utils from './utils.js'
```

En el primer caso, importamos el elemento por defecto desde el módulo *file.js* y lo guardamos en la variable *nombre*. En el segundo y tercer caso, importamos los elementos indicados en el interior de las llaves, desde el módulo *file.js*.

Si vamos a importar tanto exportaciones por defecto (default) como con nombre, la sintaxis cambia un poco. La exportación por defecto deberá ir fuera de las llaves:

```
// Exportaciones con nombre (Named exports)
export const PI = 3.14159;
export const sumar = (a, b) => a + b;

// Exportación por defecto (default)
const calculadoraPrincipal = (operacion, a, b) => {
    console.log(`Ejecutando la operación: ${operacion}`);
    return a + b;
};

export default calculadoraPrincipal;
```

Y para importarlos...

```
// 'Calculadora' es el default, { PI, sumar } son las importaciones con nombre
import Calculadora, { PI, sumar } from './mathUtils.js';

console.log(PI);
console.log(sumar(10, 5));
```

```
console.log(Calculadora( 'Suma' , 2 , 2)); // Ejecutando la operación: Suma -> 4
```

En la línea de import, el nombre que representa al default siempre debe ir primero, antes de las llaves {}.

Imports dinámicos **Nuevo!**

Las importaciones dinámicas en JavaScript dan la opción de importar archivos JS de forma dinámica como módulos en su aplicación de forma nativa. Esta función ayuda a enviar código de solicitud bajo demanda. Esta característica permite importar un módulo y no ensuciar el espacio de nombres global. También se puede cargar código condicionalmente en un bloque if-else si se desea. Véase el ejemplo siguiente:

```
if (myCondition) {  
    const module = await import('./dynamicmodule.js')  
    module.addNumbers(3, 4, 5) // Método de dynamicmodule.js  
}
```

Import.meta **Nuevo!**

Nos dará metainformación de un módulo. Considera un módulo, module.js:

```
<script type="module" src="module.js"></script>
```

Puedes acceder a la metainformación con el objeto import.meta:

```
console.log(import.meta) // {url: "file:///home/user/module.js"}
```

Devuelve un objeto con una propiedad de URL que indica la URL base del módulo. Esta será la URL de la que se obtuvo la secuencia de comandos (para secuencias de comandos externas) o la URL base del documento que lo contiene (para secuencias de comandos en línea).

Reglas para el uso de módulos

- Si queremos utilizar *import* y *export* desde el navegador directamente, deberemos añadir los archivos con módulos con la etiqueta <script> utilizando el atributo **type="module"**. Estas etiquetas de módulos se cargan en diferido, o lo que es lo mismo, como si fueran un <script defer>.
- Por norma general, a los archivos Javascript con módulos se les suele indicar la extensión **.mjs**. Aunque también se podría hacer especificando otra extensión como **.es6** o **.js**.
- Se aconseja utilizar las rutas UNIX en los *export* e *import*, ya que son las que tienen mejor soporte, tanto en navegadores como en *NodeJS*. También se pueden indicar rutas absolutas.