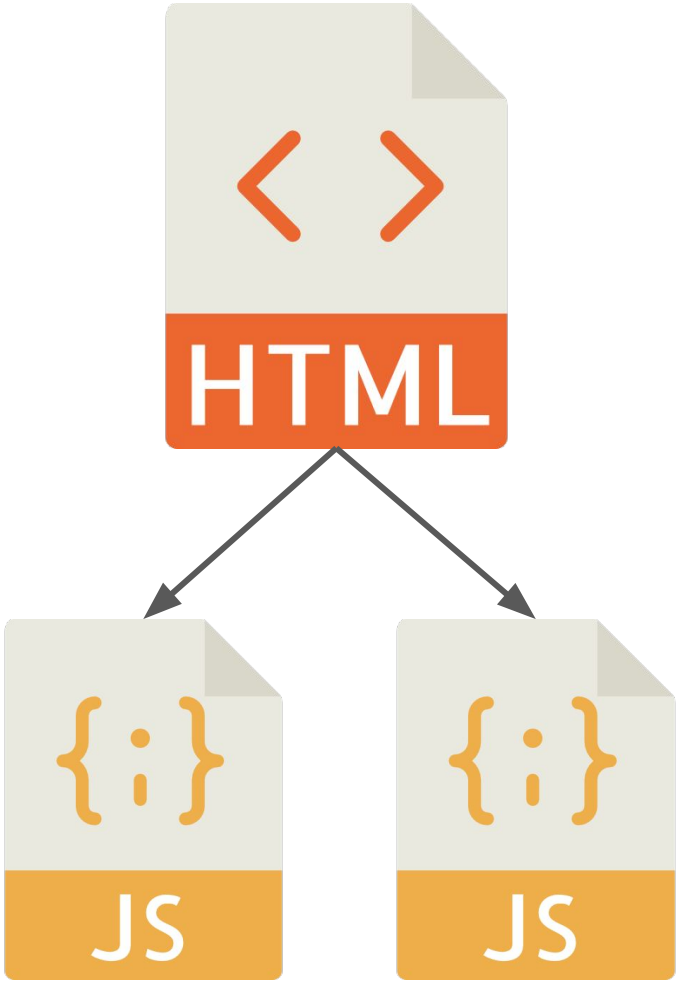
Gestión de Módulos con JavaScript

DEV.F:
DESARROLLAMOS(PERSONAS);

DEV.F:

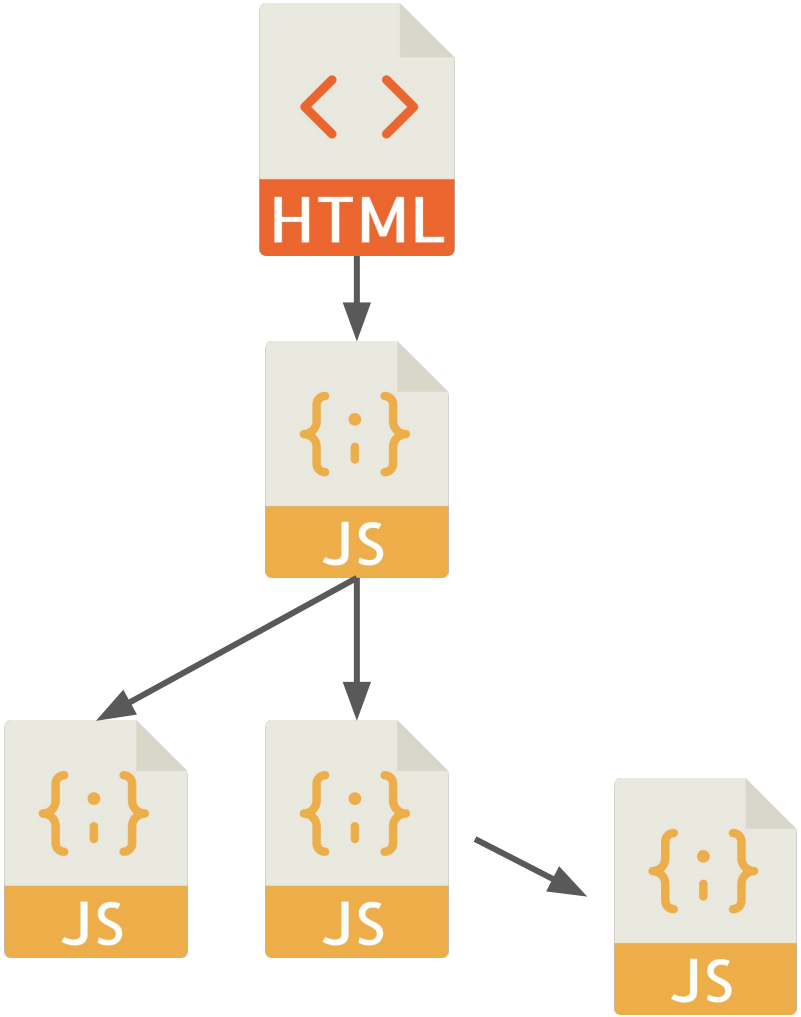
Objetivos

1. *Conocer la evolución y uso de módulos en JavaScript.*
2. *Comprender la diferencia entre require, import.*
3. *Comprender la diferencia entre CommonJS y EcmaScript Modules (ESM), y cuando usar cada uno.*
4. *Aprender a dividir nuestra lógica de JavaScript en archivos.*



Un Antecedente sobre Módulos (1)

Los programas JavaScript comenzaron siendo bastante pequeños, inicialmente eran fragmentos aislados que brindaban interactividad a las páginas web donde fuera necesario, por lo que generalmente no se necesitaban grandes scripts.



Un Antecedente sobre Módulos (2)

Avancemos unos años y ahora tenemos aplicaciones completas que se ejecutan en navegadores con mucho JavaScript.

Por lo tanto, en los últimos años se ha comenzado a pensar en proporcionar mecanismos para dividir programas JavaScript en módulos separados que se puedan importar cuando sea necesario.

DEV.F:

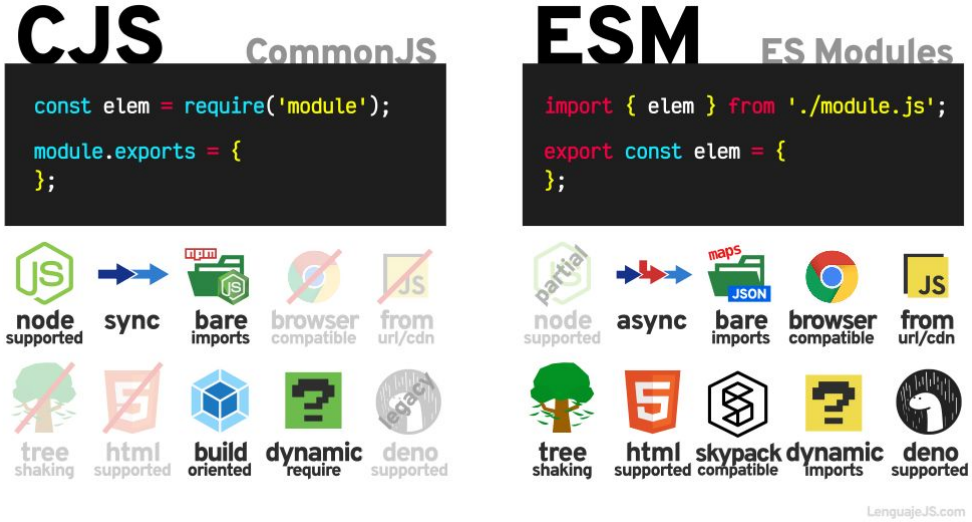
Actualmente existen varias alternativas para importar módulos de JavaScript, pero vamos a centrarnos en 2:

- *ES Modules (ES6)*
- *CommonJS*

CommonJs (CJS) vs ES Modules (ESM)

Hoy en día, de las opciones disponibles, lo más común suele ser utilizar CommonJS o ESM.

En ecosistemas donde predomina la utilización de NodeJS, es más frecuente encontrarse usando CommonJS, mientras que en sistemas más modernos, de navegador o, por ejemplo, Deno, es más habitual utilizar el enfoque de ESM.



CommonJS vs ESM: Algunas Diferencias

- **NodeJS soporta tradicionalmente la sintaxis require de CommonJS**, y aunque cada vez soporta mejor ESM, aún el soporte no es completo y tiene una amplia comunidad con paquetes utilizando CommonJS a través de NPM.

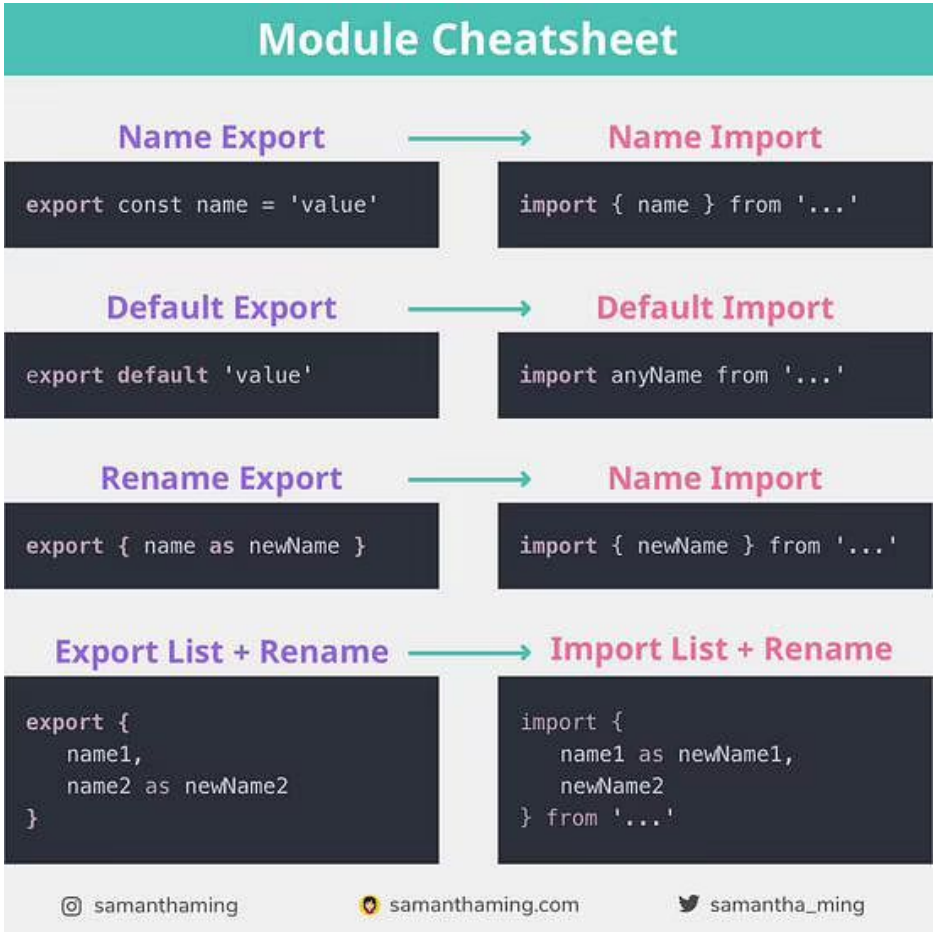
- **CommonJS sólo permite cargar módulos de forma síncrona**, mientras que **ESM permite carga síncrona y asíncrona**.

- **Los *require* de CommonJS no son compatibles en el navegador de forma directa**, mientras que los import de ESM si lo son si se indica el atributo ***<script type="module">*** en los scripts que los utilicen.

- **CommonJS no permite cargar directamente desde una URL o CDN un módulo**, mientras que con **ESM puedes hacerlo sin problemas** y funciona directamente desde un navegador.

Gestión de Módulos con ES Modules (ES6)

DEV.F:
DESARROLLAMOS(PERSONAS);



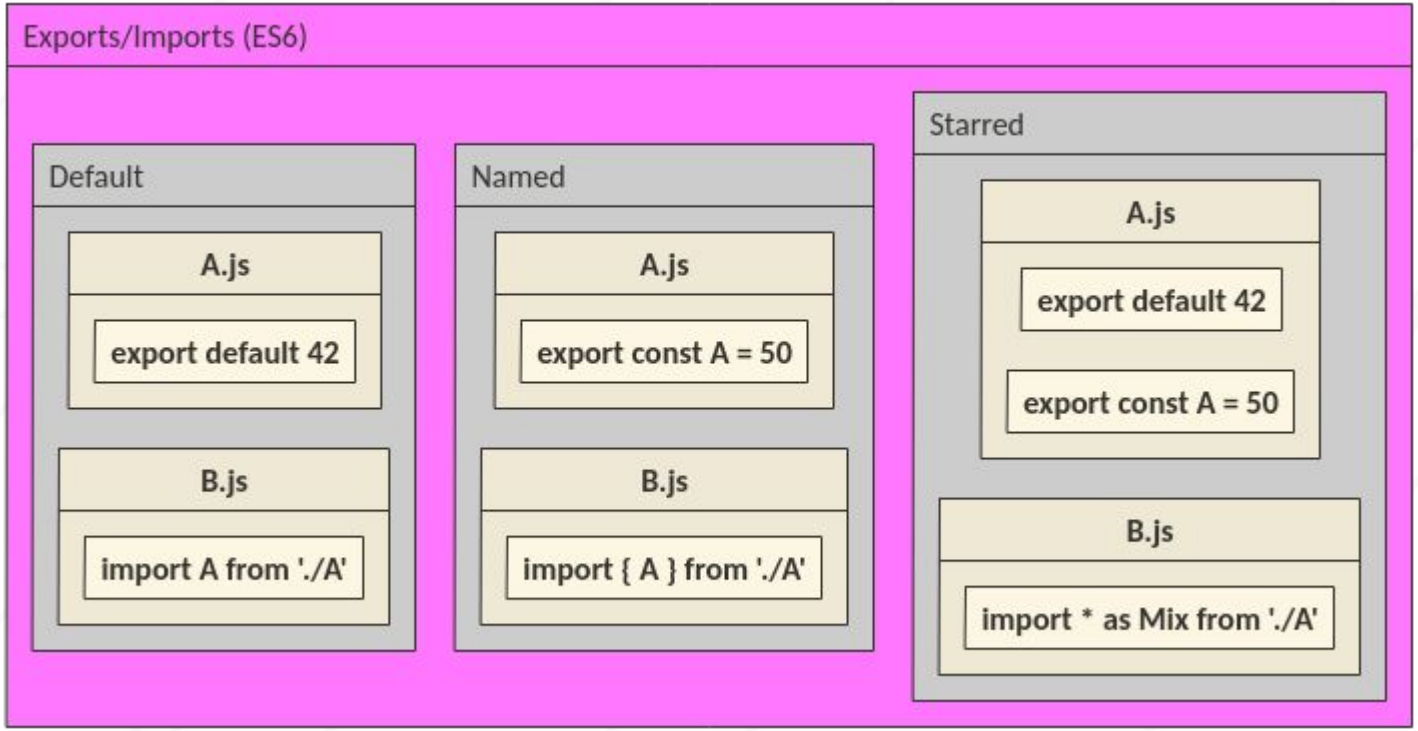
Referencia: [Samantha Ming (s.f.). Module CheatSheet]

Formas de Exportar e Importar en ESM

1. Podemos exportar e importar un elemento por su nombre (name export/import).
2. Podemos usar un export default con alguna función o valor para que sea anónima y poder llamarla como queramos al momento de importar.
3. Podemos renombrar alguna variable al momento de exportar para importarla posteriormente con dicho nombre.
4. Podemos exportar una lista de elementos e importarlas de igual forma, esto como un objeto.

DEV.F::

Resumen de uso Export/Import con ESM



Ejemplo en código

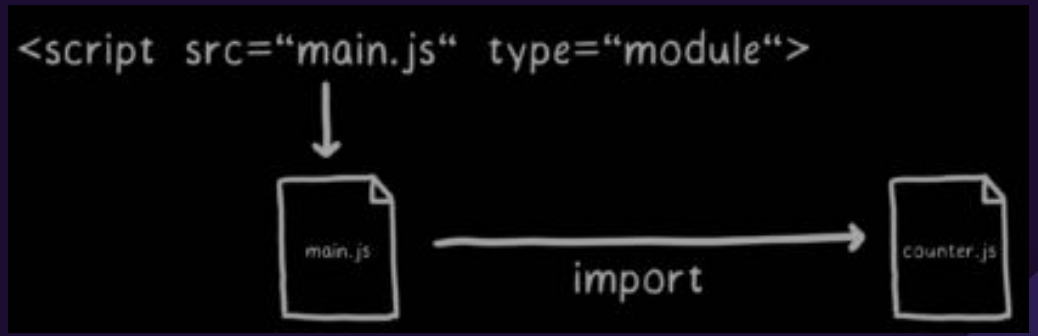
```JS
// module.js
export const data = 42;
export const method = () => console.log("Hello");

// index.js
import { data, method } from "./module.js";
```

DEV.F

IMPORTANTE

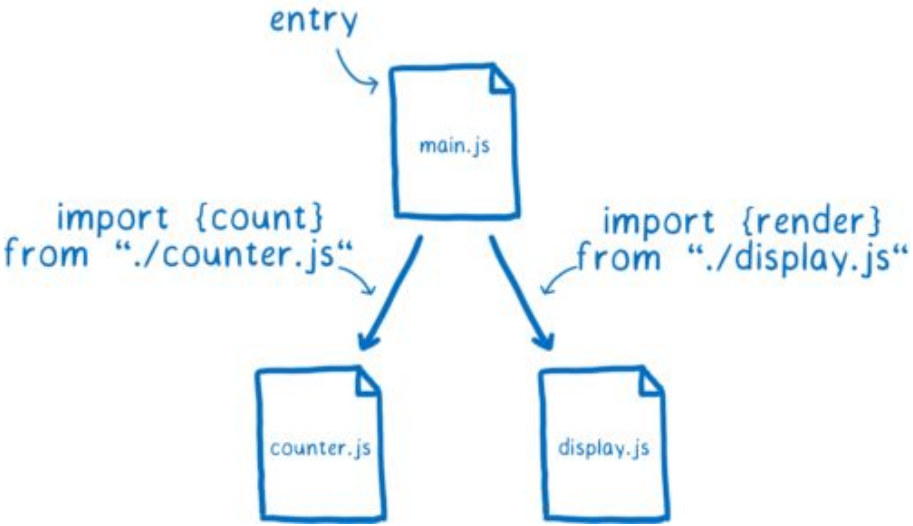
Si queremos usar ESM en conjunto con HTML, deberemos usar en nuestro archivo HTML el atributo:
<script type="module">



Este se usa para indicar cuándo se está apuntando a un módulo. En caso contrario obtendremos un error en la consola.

Gestión de Módulos con CommonJS

DEV.F:
DESARROLLAMOS(PERSONAS);



Acerca de Common JS

El lenguaje JavaScript no tenía una forma nativa de organizar el código antes del estándar ES2015.

Node.js llenó este vacío con el formato del módulo CommonJS, por lo que ya trae esta funcionalidad incluida de fábrica y por eso comúnmente usamos *require* en NodeJS para leer y ejecutar módulos de CommonJS.

Usando módulos con CommonJS

logger.js

```
const error = 'ERROR';
const warning = 'WARNING';
const info = 'INFO';

function log(message, level = info) {
    console.log(`${level}: ${message}`);
}

module.exports.log = log;
module.exports.error = error;
module.exports.info = info;
module.exports.warning = warning;
```

app.js

```
const {
    log,
    error,
    info,
    warning
} = require('./logger');


log('Node.js module demo 1');
log('Node.js module demo 2', warning);
```

Exportar:
module.exports

Importar:
require

DEV.F

En Resumen...

PRINCIPAL DIFERENCIA

En Node.js se crearon un sistema de módulos llamado CommonJS, este se usa con: const modulo = require('modulo');

En ES2015/6 se agregó al lenguaje un sistema nativo de módulos usando la sintaxis de: import modulo from 'modulo'; Aunque básicamente parece que hacen lo mismo, en realidad funcionan de forma distinta a más bajo nivel.

CommonJS cuando cargas un módulo te trae una copia del módulo para que lo puedas usar, mientras que ES2015 trae una referencia a ese módulo. Esto quiere decir que si tu módulo tiene una variable y exporta una función que modifica esa variable, cualquier otra parte de tu aplicación que haga uso de dicha variable va a ver reflejado el cambio, mientras que con CommonJS cada uno tendría su copia de la variable.

Recapitulando...