Módulos en TypeScript (importar y exportar)

1. ¿Qué es un módulo en TypeScript /JavaScript?

Un módulo es un archivo de JavaScript que exporta o importa código (funciones, variables, clases) para una mejor organización del código (legibilidad, escalabilidad, mantenibilidad).

¿Por qué es útil?

- ✓ Evita archivos enormes y difíciles de entender.
- ✓ Permite reutilizar código fácilmente.
- ✓ Facilita el mantenimiento de proyectos grandes.

¿Desde cuándo existen los módulos en JavaScript?

Se introdujeron oficialmente en el estándar ECMAScript 2015 (ES6).

Antes de ES6 se utlizaban trucos con funciones anónimas o bibliotecas como RequireJS.
Ahora, los navegadores modernos soportan módulos de forma nativa usando ***type="module"*** en HTML.

2. ¿Cómo funcionan los módulos?

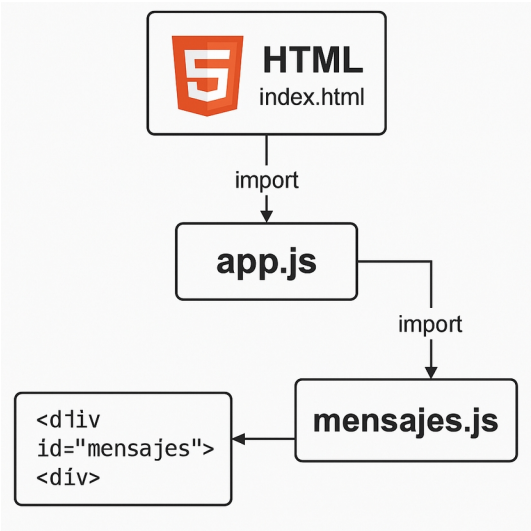
Exportar: Compartir funciones, variables o clases desde un archivo.

Importar: Traer esas funciones, variables o clases a otro archivo para usarlas.

En navegadores modernos, para que un archivo .js funcione como módulo, debe declararse como módulo en el HTML, así:

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

3. Flujo de la aplicación




```
index.html
  └── app.js (importa de mensajes.js)
        └── mensajes.js (define saludo y despedida)
```

Archivo	Contenido
`index.html`	Estructura semántica y enlace a CSS y JS.
`style.css`	Estilos separados y responsivos.
`app.js`	Lógica principal + eventos + documentación JSDoc.
`mensajes.js`	Funciones de mensaje exportadas + documentación JSDoc.

4. Configuración tsconfig.json

```
{
  "compilerOptions": {
    /* Language and Environment */
    "target": "es2016",

    …

    /* Modules */
    "module": "es6",                     /* Specify what module code is generated. */

    …

  }
}
```

index.html

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Módulos en TypeScript (importar y exportar)</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>

  <header>
    <h1>Módulos en TypeScript</h1>
    <h2>(importar y exportar)</h2>
  </header>

  <main>
    <section>
      <h2>Mensajes</h2>
      <div class="botones">
        <button id="boton-saludo">Mostrar Saludo</button>
        <button id="boton-despedida">Mostrar Despedida</button>
      </div>
      <div id="mensajes"></div>
    </section>
  </main>

  <footer>
    <p>Juan Carlos Varela Iglesias @ 2025</p>
  </footer>

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

style.css

```
/* Estilos generales */
body {
  font-family: Arial, sans-serif;
  margin: 0;
  padding: 20px;
  display: flex;
  flex-direction: column;
  align-items: center;
  min-height: 100vh;
  box-sizing: border-box;
}

/* Contenedor principal */
header, main, footer {
  width: 100%;
  max-width: 600px;
}

/* Centrar contenido del footer */
footer {
  text-align: center;
}

/* Estilos de los botones */
.botones {
  display: flex;
  flex-wrap: wrap;
  gap: 10px;
  margin-bottom: 20px;
}

.botones button {
  flex: 1 1 45%;
  padding: 10px;
  font-size: 1rem;
  cursor: pointer;
  border: none;
  border-radius: 5px;
  background-color: #4CAF50;
  color: white;
  transition: background-color 0.3s;
}

.botones button:hover {
  background-color: #45a049;
}

/* Estilos del contenedor de mensajes */
#mensajes {
  padding: 15px;
  border: 1px solid #ccc;
  border-radius: 5px;
  min-height: 50px;
  background-color: #f9f9f9;
}

/* Responsivo para móviles */
@media (max-width: 400px) {
  .botones button {
    flex: 1 1 100%;
  }
}
```

app.ts

```
/**
 * Importación de funciones
 */
import { saludo, despedida } from './mensajes.js';

/**
 * Inicializa los eventos de los botones de saludo y despedida.
 */
function inicializarEventos(): void {
  const divMensajes = document.getElementById('mensajes') as HTMLDivElement;
  const botonSaludo = document.getElementById('boton-saludo') as HTMLButtonElement;
  const botonDespedida = document.getElementById('boton-despedida') as HTMLButtonElement;

  botonSaludo.addEventListener('click', () => {
    mostrarMensaje(divMensajes, saludo());
  });

  botonDespedida.addEventListener('click', () => {
    mostrarMensaje(divMensajes, despedida());
  });
}

/**
 * Muestra un mensaje dentro del div de mensajes.
 * @param {HTMLElement} contenedor - Elemento HTML donde mostrar el mensaje.
 * @param {string} mensaje - Texto del mensaje a mostrar.
 */
function mostrarMensaje(contenedor: HTMLElement, mensaje: string): void {
  contenedor.innerHTML = `<p>${mensaje}</p>`;
}

/**
 * Ejecutamos la función de inicialización
 */
inicializarEventos();
```

¿Por qué se usa as HTMLDivElement

```
const divMensajes = document.getElementById('mensajes') as HTMLDivElement;
```

Cuando usamos **document.getElementById('mensajes'), TypeScript por defecto no sabe qué tipo exacto de elemento HTML es.**

Sólo sabe que devuelve un **HTMLElement | null** (o sea, podría ser cualquier elemento o incluso null si no existe).

Pero nosotros sí sabemos que #mensajes es un ** ** (* *).

Entonces, le decimos manualmente a TypeScript:

> "Confía, te garantizo que esto es un HTMLDivElement."

Esto es **hacer un "cast" en TypeScript, usando as.**

Por eso:

- HTMLDivElement es el tipo específico para un .
- Si fuera un <button>, usaríamos HTMLButtonElement.
- Si fuera un <input>, usaríamos HTMLInputElement.

Así TypeScript nos dará autocompletado, detección de errores y protección de tipos.

mensajes.js

a) Exportando cada una de las funciones en su definición.

Usamos export en la definición de cada una de las funciones.

```
/**
 * Devuelve un mensaje de saludo.
 * @returns {string} Mensaje de saludo.
 */
export function saludo(): string {
  return "¡Hola! Bienvenid@ a esta aplicación web.";
}

/**
 * Devuelve un mensaje de despedida.
 * @returns {string} Mensaje de despedida.
 */
export function despedida(): string {
  return "¡Hasta luego! Gracias por tu visita.";
}
```

Importación en app.js

```
/**
 * Importación de funciones
 */
import { saludo, despedida } from './mensajes.js';
```

b) Exportando todo al final del archivo JavaScript.

Ya no usamos export en la definición de cada una de las funciones.

```
/**
 * Devuelve un mensaje de saludo.
 * @returns {string} Mensaje de saludo.
 */
function saludo(): string {
  return "¡Hola! Bienvenid@ a esta aplicación web.";
}

/**
 * Devuelve un mensaje de despedida.
 * @returns {string} Mensaje de despedida.
 */
function despedida(): string {
  return "¡Hasta luego! Gracias por tu visita.";
}

/**
 * Exportación agrupada al final
 */
export { saludo, despedida };
```

Importación en app.js

La Importación en *app.js* es igual que en caso anterior

```
/**
 * Importación de funciones
 */
import { saludo, despedida } from './mensajes.js';
```

c) Exportando todo al final del archivo JavaScript e importando todo en app.js con * y un alias.

Ya no usamos export en cada una de las definiciones de las funciones.

```
/**
 * Devuelve un mensaje de saludo.
 * @returns {string} Mensaje de saludo.
 */
function saludo(): string {
  return "¡Hola! Bienvenid@ a esta aplicación web.";
}

/**
 * Devuelve un mensaje de despedida.
 * @returns {string} Mensaje de despedida.
 */
function despedida(): string {
  return "¡Hasta luego! Gracias por tu visita.";
}

/**
 * Exportación agrupada al final
 */
export { saludo, despedida };
```

Importación en app.js

```
/**
 * Importación de TODO lo que exporta mensajes.js
 */
import * as mensajes from './mensajes.js';

/**
 * Inicializa los eventos de los botones de saludo y despedida.
 */
function inicializarEventos(): void {
  const divMensajes = document.getElementById('mensajes') as HTMLDivElement;
  const botonSaludo = document.getElementById('boton-saludo') as HTMLButtonElement;
  const botonDespedida = document.getElementById('boton-despedida') as HTMLButtonElement;

  botonSaludo.addEventListener('click', () => {
    mostrarMensaje(divMensajes, mensajes.saludo());
  });

  botonDespedida.addEventListener('click', () => {
    mostrarMensaje(divMensajes, mensajes.despedida());
  });
}

/**
 * Muestra un mensaje dentro del div de mensajes.
 * @param {HTMLElement} contenedor - Elemento HTML donde mostrar el mensaje.
 * @param {string} mensaje - Texto del mensaje a mostrar.
 */
function mostrarMensaje(contenedor: HTMLElement, mensaje: string): void {
  contenedor.innerHTML = `<p>${mensaje}</p>`;
}

/**
 * Ejecutamos la función de inicialización
 */
inicializarEventos();
```

d) Exportando con `export default`

`export default` sirve para **exportar una única cosa principal** desde un archivo.

Permite que al importar, podamos poner **el nombre que queramos** (no tiene que coincidir con el nombre exportado).

```
/**
 * Devuelve un mensaje de saludo.
 * @returns {string} Mensaje de saludo.
 */
function saludo(): string {
  return "¡Hola! Bienvenid@ a esta aplicación web.";
}

/**
 * Devuelve un mensaje de despedida.
 * @returns {string} Mensaje de despedida.
 */
function despedida(): string {
  return "¡Hasta luego! Gracias por tu visita.";
}

const mensajes = {
  saludo,
  despedida
};

/**
 * Exportamos un objeto como valor por defecto
 */
export default mensajes;
```

Importación en app.js

```
/**
 * Importación del objeto completo
 */
import mensajes from './mensajes.js';

function inicializarEventos(): void {
  const divMensajes = document.getElementById('mensajes') as HTMLDivElement;
  const botonSaludo = document.getElementById('boton-saludo') as HTMLButtonElement;
  const botonDespedida = document.getElementById('boton-despedida') as HTMLButtonElement;

  botonSaludo.addEventListener('click', () => {
    mostrarMensaje(divMensajes, mensajes.saludo());
  });

  botonDespedida.addEventListener('click', () => {
    mostrarMensaje(divMensajes, mensajes.despedida());
  });
}

/**
 * Muestra un mensaje dentro del div de mensajes.
 * @param {HTMLElement} contenedor - Elemento HTML donde mostrar el mensaje.
 * @param {string} mensaje - Texto del mensaje a mostrar.
 */
function mostrarMensaje(contenedor: HTMLElement, mensaje: string): void {
  contenedor.innerHTML = `<p>${mensaje}</p>`;
}

// Ejecutamos la función de inicialización
inicializarEventos();
```

Consideraciones técnicas

Ruta relativa:

Siempre que importes, usa la ruta relativa.

CORS y módulos:

Si abres el index.html directamente haciendo doble clic, puede haber restricciones (especialmente en Chrome) por temas de seguridad.

Recomendación: usar un pequeño servidor local para pruebas, como:

- Extensión "Live Server" en VS Code.
- npx serve (Node.js).