Intro to JavaScript Modules

JavaScript modules are reusable pieces of code that can be exported from one program and brought into another program. Modules are particularly useful for a number of reasons:

- Maintainability: Modules can be updated individually without affecting other parts of the codebase.
- Namespace: Variables defined inside a module won't conflict with those in another module or global scope.
- Reusability: Functions defined in a module can be reused across different parts of your application.

Exporting Modules: Named

In JavaScript, you can export functions, objects, or primitive values from a module so that they can be used by other scripts.

```
// math.js
export const add = (a, b) => a + b;
export const subtract = (a, b) => a - b;
```

In the above example, `add` and `subtract` are named exports from the `math.js` module.

Then you can import these functions in another file like this:

```
import { add, subtract } from './mathUtils.js';
```

More on Named exports:

- You use named exports when a module exports multiple items, like functions or objects.
- Named exports are useful when you are exporting many values.
- Each named export can then be imported individually into other files using the same name surrounded by curly braces `{ }`.
- Named exports can be renamed upon import using the `as` keyword.
- Named exports are good for utility modules that export several functions.

Exporting Modules: Default

If a module is designed to export a single value or object, we can use the default export.

```
// person.js
export default class Person {
  constructor(name) {
    this.name = name;
  }
}
```

In the above example, `Person` class is the default export from `person.js`. When importing a default export, you don't need to use curly brackets and you can name it whatever you want.

You can import it like this:

```
import Person from './person.js';
```

More on Default exports:

- You use default exports when a module only exports one item, like a function or a class.
- Each JavaScript file can have zero or one default export.
- Default exports are useful when a module only exports one thing. It doesn't require using curly braces `{ }` while importing.
- You can name a default export whatever you want when importing it.

In general, if a module defines a primary functionality (like a `Person` class, or a `createServer` function in an HTTP server module), you'd use `export default`. For additional, secondary functionalities (like utility functions), you'd use named exports.

Exporting Modules: Multiple

You can also export multiple items from a module. These items can be a mix of named and default exports.

```
// utils.js
export const helperFunc1 = () => {};
```

```
export const helperFunc2 = () => {};
export default () => {
  // default function...
};
```

In the above example, `helperFunc1` and `helperFunc2` are named exports and the unnamed function is the default export from the `utils.js` module.

Renaming Imported Modules

When importing named exports, you can rename them using the `as` keyword. This can be useful to avoid naming conflicts with local variables.

```
import { helperFunc1 as utilityFunc1, helperFunc2 as utilityFunc2 } from
'./utils.js';
```

In the above example, `helperFunc1` and `helperFunc2` are renamed to `utilityFunc1` and `utilityFunc2` respectively in the local scope.

Renaming isn't possible with default imports as you can give any name you want during default import. Also, you can combine default and named imports in a single import statement.

```
import defaultExport, { namedExport as Alias } from './module.js';
```

In the above example, `defaultExport` is the default export from the `module.js` and `namedExport` is the named export which has been renamed to `Alias` in the local scope.

Resources

MDN Web Docs

- JavaScript Modules
- export
- import

JavaScript Info

- Modules

W3 Schools

- Modules

Video Tutorials

- JavaScript Modules ES6 Import and Export by The Net Ninja
- ES6 Modules in JavaScript by Web Dev Simplified

Code Example

- GitHub Repo
- Repl

Practice - Try it Yourself

Building a Math Module:

- Prompt: Create a JavaScript file named `math.js`. This module should export four functions: `add()`, `subtract()`, `multiply()`, and `divide()`, each performing the respective mathematical operation. In a separate JavaScript file, import these functions and use them to perform some calculations.
- Hint: Use named exports to export functions from the `math.js` module and remember to import these functions using their correct names in the other file.

Creating a Personal Information Module:

- Prompt: Develop a module named `person.js`. This module should have a default export of a `Person` class with properties `name`, `age`, and `location`. Also, add a named export `greet()` function that logs a greeting to the console. Import this module in another JavaScript file and use the `Person` class and `greet()` function.
- Hint: When importing the `Person` class (a default export), you can name it anything you like. But, you need to import `greet()` with its specific name.

Renaming on Import:

- Prompt: Create a module that exports a few functions or variables. In another JavaScript file, import these functions or variables but rename them upon import. Use these renamed functions or variables in your code.
- Hint: Use the `as` keyword to rename imports. For instance, `import { originalName as newName } from './module.js';`