Modules

CommonJS (old) and ES Modules (new) Styles

Node.js CommonJS Modules

In early Node.js, the **CommonJS** system was used before ES Modules (`import/export`).

We use:

- `require()` to **import**
- `module.exports` to **export**

Also:

- Adding ".js" is an option.

Class Module Example

lib/Calculator.js

```
class Calculator {
  add(a, b) {
    return Number(a) + Number(b);
  }
  multiply(a, b) {
    return Number(a) * Number(b);
  }
}

module.exports = Calculator;
```

Exporting a class using module.exports = Calculator.

require to use the module

```
// Demo consumer using require(...)
const Calculator = require('./lib/Calculator');

const calc = new Calculator();

const a = 12;
const b = 3;

console.log('Using Calculator class:');
console.log(`add(${a}, ${b}) =`, calc.add(a, b));
console.log(`multiply(${a}, ${b}) =`, calc.multiply(a, b));
```

Function Module Example

lib/mathFns.js

```
function sub(a, b) {
  return Number(a) - Number(b);
}

function div(a, b) {
  if (b === 0) throw new Error("Division by zero");
  return Number(a) / Number(b);
}

module.exports = { sub, div };
```

Exporting multiple functions as an object.

require to use the module

```
//  object destructuring assignment
const { sub, div } = require('./lib/mathFns');

const a = 12;
const b = 3;

console.log(`sub(${a}, ${b}) =`, sub(a, b));
console.log(`div(${a}, ${b}) =`, div(a, b));
```

We also can use:

```
const arith = require('./lib/mathFns');
console.log(`sub(${a}, ${b}) =`, arith.sub(a, b));
console.log(`div(${a}, ${b}) =`, arith.div(a, b));
```

Node.js ES Modules (Modern Syntax)

In modern Node.js (v14+), we use **ES Modules (ESM)** instead of CommonJS.

We use:

- `import` to **import**
- `export` to **export**

We can specify the default export.

Also:

- We must add the ".js" when to specify the module name.

Class Module Example

```
// Named exports
export class Calculator {
  add(a, b) {
    return Number(a) + Number(b);
  }
  multiply(a, b) {
    return Number(a) * Number(b);
  }
}
```

Exporting a class using export class.

import to use the module

```
// Demo consumer using import
import { Calculator } from './math.js';

const calc = new Calculator();

const a = 12;
const b = 3;

console.log('Using Calculator class:');
console.log(`add(${a}, ${b}) =`, calc.add(a, b));
console.log(`multiply(${a}, ${b}) =`, calc.multiply(a, b));
```

Function Module Example

```
// Default export
export default function sub(a, b) {
  return Number(a) - Number(b);
}

// Named export
export function div(a, b) {
  if (Number(b) === 0) throw new Error("Division by zero");
  return Number(a) / Number(b);
}
```

Here we export:

- sub as the default export
- div as a named export

import to use the module

```
// import default and named exports
import sub, { div } from './math.js';

const a = 12; const b = 3;

console.log(`sub(${a}, ${b}) =`, sub(a, b));
console.log(`div(${a}, ${b}) =`, div(a, b));
```

In this example:

- sub ← comes from export default ...
- { div } ← comes from export function div(...)

They come from two different “export namespaces”:

- The default export has no name.
- The named exports are grouped inside { ... }.

import all exports

We can use `* as name` to access all the exports using the `name` (we can use any name for it).

```
import sub, * as math from './math.js';

const a = 12;
const b = 3;

console.log('Using ES Module Functions:');
console.log(`add(${a}, ${b}) =`, math.add(a, b));
```

import CommonJS module

When we need to use th CommonJS module in the ES6 module, we need to assign a variable to access all the exports from the CommonJS module.

```
import math from './math.js';
console.log(`add(${a}, ${b}) =`, math.add(a, b));
```