JavaScript TUTORIAL

# JavaScript Export

**Summary**: In this tutorial, you will learn how to use the JavaScript export keyword to export values from a module.

## Introduction to JavaScript export keyword

ES6 modules allow you to structure JavaScript code in a modular fashion. Modules provide a standardized way for defining and importing/exporting reusable pieces of code within a JavaScript application.

By default, ES6 modules encapsulate their code. This means that values (variables, functions, classes, etc.) defined in a module are not accessible from outside of the module by default. This prevents naming conflicts and promotes better code structure.

Modules can export values (variables, functions, classes, etc.,) using the `export` keyword.

The `export` keyword exports values from a module so that you can use them in other modules. There are two types of exports:

- Named exports
- Default exports

A module can have **multiple named exports** but **only one default export**.

## Named exports

A module can have multiple named exports. In practice, you use named exports when you need to export multiple values from a module.

### Exporting variables

The following example shows how to export a variable `count` from a module:

```
let count = 1;
export { count };
```

In this example:

- First, declare a variable `count` and initialize its value to `100`.
- Then, export the `count` variable by placing them in curly braces that follow the `export` keyword.

It's possible to combine the variable declaration and export in a single statement like this:

```
export let count = 1;
```

Similarly, you can export variables declared using the `const` keyword:

```
export const MIN = 0;
```

To export multiple variables, you separate them using a comma (`,`):

```
let count = 1;
const MIN = 0, MAX = 10;

export { MIN, MAX, count };
```

## Exporting functions

Export functions share the same syntax as exporting variables:

```
function increase() {
  // ..
}
export { increase };
```

And:

```
export function increase() {
  // ...
}
```

## Exporting classes

Likes variables and functions, you can export classes using the `export` keyword for example:

```
class Counter {
  constructor() {
    this.count = 1;
  }
  increase() {
    this.count++;
  }
  get current() {
    return this.count;
  }
}

export { Counter };
```

In this example, we define a `Counter` class and export it. You can define a class and export it using a single statement as follows:

```
export class Counter {
  constructor() {
    this.count = 1;
  }
  increase() {
    this.count++;
  }
  get current() {
    return this.count;
  }
}
```

When importing named exports, you need to use the exact name and specify them in curly braces. For example, the following import the `Counter` class:

```
import { Counter } from 'module.js';
```

## Default exports

A module can have one default export. To export a value using a default export, you use the `default export` keyword. For example:

```
let message = 'Hi';
export { default as message };
```

It's equivalent to the following:

```
export default let message = 'Hi';
```

When importing a default export, you don't need to place the variable inside curly braces:

```
import message from 'module.js';
```

Note that if the message was exported using a named export, you would place it inside the curly braces:

```
import { message} from 'module.js';
```

This is the main difference between importing a named export and a default export.

Similarly, you can export a function or a class from a module using default exports:

```
export default function increase() {
   // ..
}
```

And:

```
export default class Counter {
   // ...
}
```

## Renaming named exports

When exporting a value using a named export, you can rename it like this:

```
class Counter {
  // ..
}

export { Counter as MyCounter };
```

Renaming a value when exporting helps avoid naming conflict. Note that the modules that import the class may also rename it when importing:

```
import { Counter as MyCounter } from 'module.js';
```

## Re-exporting

A module can import values from another module and export them immediately as follows:

```
import { Counter } from 'module.js';
export { Counter };
```

JavaScript allows you to shorten this using the `export from` syntax:

```
export { Counter } from 'module.js';
```

## Summary

- Use JavaScript `export` keyword to export variables, functions, and classes from a module.
- Exports can be named exports and default exports. A module can have multiple named exports but only one default export.