



THỰC HỌC – THỰC NGHIỆP

LẬP TRÌNH ECMASCRIPT

MODULE

- ⦿ Javascript Module là gì
- ⦿ Khái niệm Export
- ⦿ Khái niệm Import

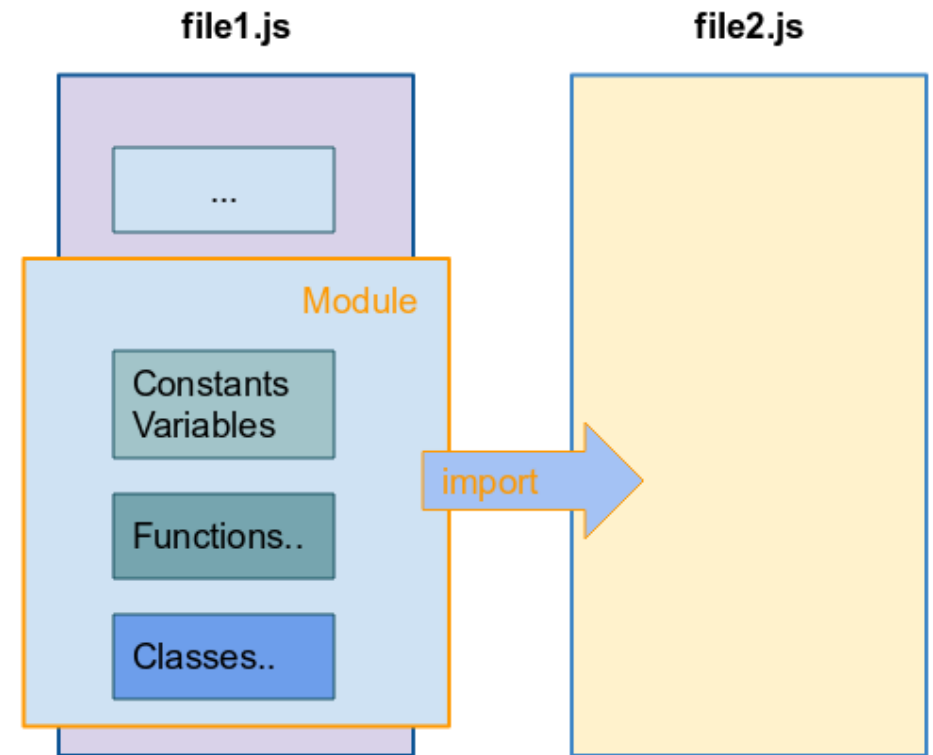






PHẦN 1: TỔNG QUAN

- ❑ JS **modules** hay còn gọi là ES modules, ECMAScript modules là một tính năng quan trọng mới của javascript.
- ❑ Trước đây chúng ta sử dụng CommonJS trong Node.js hay AMD để có thể sử dụng tính năng này.
- ❑ Tính năng Module đã được tích hợp sẵn trong trình duyệt, thông qua sử dụng các khai báo xuất(export) và nhập(import).



- ❖ Để khai báo một script là ES Modules, bạn phải thêm **attribute** cho nó là: type="module".

```
<!-- tải một file module Javascript-->
<script type="module" src="module.js"></script>
<!-- Nhúng một module dạng inline -->
<script type="module">
  import { sum } from "../example.js";
  let result = sum(1, 2);
</script>
```

- ❖ Trình duyệt không hỗ trợ module sẽ tự động bỏ qua dòng type="module"
- ❖ Các module được thực thi theo thứ tự mà chúng xuất hiện trong tệp HTML.

❖ Ví dụ:

- nếu chúng ta có tệp **sayHi.js** export một hàm:
- ... Sau đó, một tệp khác có thể nhập và sử dụng nó:

❖ Lệnh **import** sẽ tải module theo đường dẫn **./sayHi.js** liên quan đến tệp hiện tại và gán biến **sayHi** cho đường dẫn tương ứng

❖ Truy cập liên kết để xem ví dụ: [Link](#)

```
// sayHi.js
export function sayHi(user) {
  alert(`Hello, ${user}!`);
}
```

```
// main.js
import {sayHi} from './sayHi.js';

alert(sayHi); // function...
sayHi('John'); // Hello, John!
```

1. Trong **Module** tính năng "use strict" luôn được bật mặc định, ví dụ:

```
<script type="module">
  |   a = 5; // error
</script>
```

2. Tính năng **undefined**

```
<script>
  |   alert(this); // window
</script>
```

```
<script type="module">
alert(this); // undefined
</script>
```


3. Đối tượng **import.meta** chứa thông tin về mô-đun hiện tại.

```
<script type="module">  
  alert(import.meta.url); // hiển thị thông tin của module  
</script>
```

4. Module sẽ chỉ được thực thi một lần sau khi import

```
// Import the same module from different files  
// Import chung một module, có cùng tên nhưng từ 2 file khác nhau
```

```
// 📁 1.js  
import `./alert.js`; // Module được chạy!
```

```
// 📁 2.js  
import `./alert.js`; // Không chạy được
```



PHẦN 2: KHÁI NIỆM EXPORT

❖ Lệnh **export** được sử dụng khi tạo các module JavaScript để xuất các hàm, đối tượng hoặc giá trị nguyên thủy trong module để chúng có thể được sử dụng bởi các chương trình khác bằng lệnh import

❖ Cú pháp:

```
export { name1, name2, ..., nameN };  
export { variable1 as name1, variable2 as name2, ..., nameN };  
export let name1, name2, ..., nameN; // còn có thể là var, function  
export let name1 = ..., name2 = ..., ..., nameN; // còn có thể là var, const  
  
export default expression;  
export default function (...) { ... } // còn có thể là class, function*  
export default function name1(...) { ... } // còn có thể là class, function*  
export { name1 as default, ... };  
  
export * from ...;  
export { name1, name2, ..., nameN } from ...;  
export { import1 as name1, import2 as name2, ..., nameN } from ...;
```

- ❖ Có hai kiểu export khác nhau. **Named** và **default** Mỗi kiểu tương ứng với một trong các cú pháp ở phía trên:

```
// export chức năng được định nghĩa trước  
export { myFunction, myVariable };
```

```
// export các tính năng riêng lẻ ( có thể biến, function hoặc class)  
export let myVariable = Math.sqrt(2);  
export function myFunction() {  
  | // ...  
};
```

- ❖ Export giá trị mặc định hàm và lớp:

```
export { myFunction as default };  
export default function () { /*...*/ }  
export default class { /*...*/ }
```

- ❖ Export tên (**named export**) được sử dụng để xuất nhiều thứ từ một module bằng cách thêm **export** vào khai báo của chúng.


```
// export data
export let color = "red";
// export function
export function sum(num1, num2) {
  return num1 + num1;
}
// export class
export class Rectangle {
  constructor(length, width) {
    ...
  }
}
```

- ❖ Bạn có thể sử dụng export sau khi định nghĩa một biến, hàm hoặc một lớp

```
// Định nghĩa một hàm và export sau
function multiply(num1, num2) {
    return num1 * num2;
}
export { multiply };
```

- ❖ Khi export nếu cần thay đổi tên, bạn cũng có thể sử dụng từ khóa **as** để thay đổi tên

```
class ZipCodeValidator extends StringValidator {
    isAcceptable(value) {
        return value.length === 5 && numberRegexp.test(value);
    }
}
export { ZipCodeValidator as mainValidator };
```



- ❖ Trong một vài trường hợp, khi bạn muốn xuất lại **(re-export)** gì đó từ module mà bạn import:
- Bạn đã tạo một module tổng hợp từ những module nhỏ hơn
 - Module được kế thừa từ module khác, và sử dụng một phần tính năng của module nhỏ đó

```
// re-export sum từ module "./example.js"
export { sum } from "./example.js";
// sum được import từ "./example.js" và được gán bằng tên mới là add
export { sum as add } from "./example.js";
// re-export mọi thứ từ "./example.js"
export * from "./example.js";
```

*Lưu ý: nếu bạn muốn sử dụng toàn bộ tính năng trong module cần import thì có thể sử dụng từ khóa **

- ❖ Trong Javascript ES6 Giá trị mặc định của một module có thể là một biến, hàm hoặc lớp được chỉ định bằng từ khóa **default**
- ❖ Vì vậy bạn chỉ có thể sử dụng một **export default** cho từng module.

```
export default function(num1, num2) {  
  |   return num1 + num2;  
}  
// Hoặc  
function sum(num1, num2) {return num1 + num2;}  
export default sum;
```




PHẦN 3: KHÁI NIỆM IMPORT

- ❖ Khi bạn có một module cần export, bạn có thể truy cập tính năng trong một module khác bằng cách sử dụng từ khóa **import**

```
// chỉ nhập một hoặc nhiều tính năng trong module export
import { sum } from "./example.js";
import { multiply, magicNumber } from "./example.js";
console.log(sum(1, magicNumber));
console.log(multiply(1, 2));
```

- ❖ Một vài trường hợp đặc biệt, bạn cho phép import toàn bộ module như một đối tượng. Bạn có thể sử dụng * và chỉ định (**as**) vào một biến để truy xuất toàn bộ tính năng của module đó.

```
// import tất cả tính năng từ 'example.js'
import * as example from "./example.js";
console.log(example.sum(1, example.magicNumber));
console.log(example.multiply(1, 2));
```

- ❖ Nếu module đang import là một biến, hàm hoặc lớp mà bạn muốn sử dụng một tên khác, bạn có thể sử dụng **as**

```
// import hàm add() và thay đổi tên thành sum()
import { add as sum } from "./example.js";
console.log(typeof add);    // "undefined"
console.log(sum(1, 2));    // 3
```

❖ Một số module thiết lập một số trạng thái toàn cục hoặc mở rộng các mô-đun hiện có. Các mô-đun này có thể không có bất kỳ export nào.

❖ Ví dụ: nếu bạn muốn thêm một phương thức pushAll () vào tất cả các mảng, *Module này không có export trong example.js* sau:

```
Array.prototype.pushAll = function(items) {
    //...
};
```

❖ Phương thức pushAll () *Module này không có export trong example.js* bên trong module này.

```
import "./example.js";
let colors = ["red", "green", "blue"];
let items = [];
items.pushAll(colors);
```

- ❖ Trong một vài trường hợp đặc biệt, module import liên kết không thể thay đổi giá trị của nó. Nhưng module export có thể thay đổi nó!

- ❖ Ví dụ

```
// module example.js
export var name = "Nicholas";
export function setName(newName) {
  name = newName;
}

// module index.js import 'example.js'
import { name, setName } from './example.js';
console.log(name); // "Nicholas"
setName("Greg");
console.log(name); // "Greg"
name = "Nicholas"; // error
```

- ❖ Lệnh gọi **setName("Greg")** truy cập trở lại module **example.js** mà **setName ()** đã được khai báo trước đó, sau đó hàm thực thi và đặt tên thành "Greg".

- ❖ Một module có thể import những tính năng từ các mô-đun khác. Nó kết nối đến các mô-đun đó thông qua các chỉ định mô-đun, các chuỗi là:
 - Đường dẫn tương đối ('../model/user') được giải thích tương đối với vị trí của mô-đun nhập. Phần mở rộng tệp thường có thể bị bỏ qua.
 - Đường dẫn tuyệt đối ('/ lib / js / helpers') trỏ trực tiếp đến tệp của mô-đun được nhập.
 - Tên ('dùng'). Tên mô-đun nào tham chiếu đến phải được cấu hình. Một cấu hình thường trong các dự án JavaScript đề cập đến thư mục node_modules dưới dạng thư mục gốc.

```
// import một module không có bất kì ràng buộc nào
import 'jquery';
// import mặc định của một mô-đun
import $ from 'jquery';
// import tên của module
import { $ } from 'jquery';
// import module nhưng sử dụng một tên khác
import { $ as jQuery } from 'jquery';
// import tất cả tính năng module
// và được chỉ định bằng một đối tượng
import * as crypto from 'crypto';
```

```
// export tên một hàm
export function foo() {};
// export mặc định một hàm
export default function foo() {};
// export một biến
export { encrypt };
// export một biến và chỉ định là một biến mới
export { decrypt as dec };
// export một tính năng từ module khác
export { encrypt as en } from 'crypto';
// export tất cả tính năng từ module khác
export * from 'crypto';
```

▶ DEMO

- ❖ Sử dụng export riêng lẻ ở cuối file module. Cách sử dụng này được gọi là "named export" do ta định nghĩa tên của object đang được export trong khi đang export.

```
// foobar.js
function foo() { console.log('foo')}
function bar() { console.log('bar')}
export { foo, bar };
```

- ❖ Ta có thể sử dụng object được export từ các file khác như sau:

```
// main.js
import {foo, bar} from 'foobar';
foo();
bar();

import * as lib from 'foobar';
lib.foo();
lib.bar();
```

- ❖ Một format khác khi sử dụng "named export" là ta có thể export objects/function khi ta tạo ra chúng. Cấu trúc sau có phần tiện hơn so với ở ví dụ 1.

```
// foobar.js
export function foo() { console.log('foo')}
export function bar() { console.log('bar')}
```

- ❖ Kết quả hiển thị giống ví dụ 1

```
// main.js
import {foo, bar} from 'foobar';
foo();
bar();

import * as lib from 'foobar';
lib.foo();
lib.bar();
```

- ☑ Hiểu và sử dụng hàm Arrow Function
- ☑ Giải thích con trỏ 'this' trong Arrow Function
- ☑ Nắm được khái niệm Default Parameters
- ☑ Nắm được khái niệm Generator Function





thank
you!