Modules

Khi ứng dụng phát triển lớn hơn, việc tổ chức tất cả mã JavaScript trong một file khiến mã trở nên khó bảo trì, thay vào đó một cách làm phổ biến là chia tách mã ra thành các file với chức năng riêng biệt, được gọi là các **module**

Module là một file JavaScript, mỗi file JavaScript là một module 🙂

Các module có thể được tải và sử dụng các chức năng của nó ở một module khác. Để cho phép một module khác sử dụng/hoặc sử dụng một chức năng từ một module khác, JavaScipt cung cấp 2 từ khóa đặc biệt **export** và **import**

- **export** cho phép các module khác quyền truy cập tới chức năng/biến trong module hiện tại
- **import** nhập các chức năng/biến từ module khác để sử dụng

Lưu ý: **export** và **import** chỉ hoạt động với giao thức **HTTP** (mở trang với **liveserver**)

Modules

```
// 📂 module.js
export function example() {
    console.log(`Heyyy, I Love YOU 😘`);
}

// 📂 index.js
import { example } from "./module.js";
example();

// 📂 index.html
<script src="index.js" type="module"></script>
```

Modules vs Regular JavaScript Files

Sự khác biệt giữa các file JavaScript "*thông thường*" và module:

- Module luôn *bật* mode **"use strict"**
- Module có scope của riêng nó, các biến/hàm/class/... không được **export** sẽ chỉ có thể truy cập được trong module hiện tại
- Các module sẽ được thực thi lần đầu tiên khi nhập, các câu lệnh **import** cùng một module sẽ chia sẽ chung dữ liệu đó
- Đối tượng **import.meta** chứa thông tin (phụ thuộc vào environment) về module hiện tại
- Trong module, *this* là *undefined*

Export

```
export let days = ["Mon", "Tue", " ... "];
export const MODULE_NAME = "module.js";
export function func() { }
export class User { }

function sayLove() { }
let name = "Ba";
export { sayLove, name };

export { sayLove as love, name as n };
```

Export Default

Lợi ích của module là chia nhỏ các tệp tin, mỗi chức năng đều nằm trong một module riêng. Ngoài các cách **export** thông thường, JavaScript hỗ trợ cú pháp **export default**, với cú pháp *gọn gàng* hơn 😃

Mỗi module chỉ có 1 **export default**, ngoài ra các giá trị **export default** không cần đặt tên

```
// 📂 module.js
export default () => console.log("Love You 😊");

// 📂 otherModule.js
function sayLove() { console.log("Love You 😊"); }
export { sayLove as default };

// 📂 index.js
// Đặt tên tùy ý khi import
import sayLove from "./module.js"
import sayLoveAgain from "./otherModule.js"
```

Import

```
import { func } from "./module.js";
import { func as f } from "./module.js";
// import default
import func from "./module.js";
import { default as func } from "./module.js";
// import everything as object
import * as m from "./module.js";
m.func(); // call function
// import and run
import "./module.js";
```

Exercises

Xây dựng form đăng nhập, với các yêu cầu:

- Module validate.js chứa các chức năng liên quan đến xác thực dữ liệu
- Module index.js thực hiện các thao tác xử lý form, và hiển thị kết quả trên giao diện
- Giao diện HTML sử dụng bootstrap (Form Component)

Class Basic

Lập trình hướng đối tượng (*Object-oriented Programing* - OOP) là một kỹ thuật lập trình cho phép mô tả (trừu tượng hóa) các **đối tượng trong thực tế** bằng ngôn ngữ lập trình.

Một đối tượng bao gồm:

- Thuộc tính (attribute): Là các thông tin, đặc điểm (trạng thái) của đối tượng
- Phương thức (method): Là các chức năng, hành vi của đối tượng

Trong lập trình hướng đối tượng, một class là một mẫu (blueprint) mà chương trình có thể sử dụng để tạo các đối tượng, cung cấp các giá trị ban đầu (thuộc tính - trạng thái) và triển khai hành vi (phương thức).

JavaScript ES5 sử dụng `function` và `new` để tạo ra các đối tượng. JavaScript ES6 sử dụng cú pháp `class` mới cung cấp nhiều tính năng mạnh mẽ hơn cho lập trình OOP.

Class Basic

Class vs Function

```
class User {
    constructor(name) {
        this.name = name;
    }

    love() {
        alert(`I Love ${this.name}`);
    }
}

let user = new User("Ba");
user.love();
```

```
function User(name) {
    this.name = name;
}

User.prototype.love = function () {
    alert(`I Love ${this.name}`);
};

let user = new User("Ba");
user.love();
```

Class Getter/Setter

```
constructor(name) {
    this.name = name;
}

get name() {
    return this._name;
}

set name(val) {
    if (val.length < 2) {
        alert("Tên quá ngắn");
        return;
    }
    this._name = val;
}
```

Exercises

Tạo `class Counter` bao gồm các chức năng hiển thị đồng hồ bấm giờ trên trình duyệt (HTML) và các phương thức `start()`, `stop()`, `reset()` cho phép chạy/dừng/reset đồng hồ bấm giờ.