**I. IIFE**

Immediately Invoked Function Expression === Self-Invoking Function

IIFE (Biểu thức hàm được gọi ngay lập tức) là một **hàm**[JavaScript](https://developer.mozilla.org/en-US/docs/Glossary/Function)[chạy](https://developer.mozilla.org/en-US/docs/Glossary/JavaScript) ngay khi được xác định.

(function () {

// …

})();

(() => {

// …

})();

(async () => {

// …

})();

Đây là một mẫu thiết kế còn được gọi là [Hàm ẩn danh tự thực thi](https://developer.mozilla.org/en-US/docs/Glossary/Self-Executing_Anonymous_Function) và chứa hai phần chính:

1. Đầu tiên là chức năng ẩn danh với phạm vi từ vựng được đặt trong tệp . Điều này ngăn việc truy cập các biến trong thành ngữ IIFE cũng như làm ô nhiễm phạm vi toàn cầu.[Grouping Operator](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Grouping) ()
2. Phần thứ hai tạo biểu thức hàm được gọi ngay lập tức ()qua đó công cụ JavaScript sẽ diễn giải trực tiếp hàm đó.

[**Trường hợp sử dụng**](https://developer.mozilla.org/en-US/docs/Glossary/IIFE#use_cases)

[**Tránh làm ô nhiễm không gian tên toàn cầu**](https://developer.mozilla.org/en-US/docs/Glossary/IIFE#avoid_polluting_the_global_namespace)

Vì ứng dụng của chúng ta có thể bao gồm nhiều hàm và biến toàn cục từ các tệp nguồn khác nhau nên điều quan trọng là phải giới hạn số lượng biến toàn cục. Nếu chúng tôi có một số mã khởi tạo mà chúng tôi không cần sử dụng lại, chúng tôi có thể sử dụng mẫu IIFE. Vì chúng tôi sẽ không sử dụng lại mã, sử dụng IIFE trong trường hợp này sẽ tốt hơn là sử dụng khai báo hàm hoặc biểu thức hàm.

(() => {

// some initiation code

let firstVariable;

let secondVariable;

})();

// firstVariable and secondVariable will be discarded after the function is executed.

### [Thực hiện một chức năng không đồng bộ](https://developer.mozilla.org/en-US/docs/Glossary/IIFE#execute_an_async_function)

IIFE [async](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/async_function)cho phép bạn sử dụng [await](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/await) và [for-await](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/for-await...of) ngay cả trong các trình duyệt cũ hơn và thời gian chạy JavaScript không có [mức cao nhất đang chờ](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/await#top_level_await) :

const getFileStream = async (url) => {

// implementation

};

(async () => {

const stream = await getFileStream("https://domain.name/path/file.ext");

for await (const chunk of stream) {

console.log({ chunk });

}

})();

const makeWithdraw = (balance) =>

((copyBalance) => {

let balance = copyBalance; // This variable is private

const doBadThings = () => {

console.log("I will do bad things with your money");

};

doBadThings();

return {

withdraw(amount) {

if (balance >= amount) {

balance -= amount;

return balance;

}

return "Insufficient money";

},

};

})(balance);

const firstAccount = makeWithdraw(100); // "I will do bad things with your money"

console.log(firstAccount.balance); // undefined

console.log(firstAccount.withdraw(20)); // 80

console.log(firstAccount.withdraw(30)); // 50

console.log(firstAccount.doBadThings); // undefined; this method is private

const secondAccount = makeWithdraw(20); // "I will do bad things with your money"

console.log(secondAccount.withdraw(30)); // "Insufficient money"

console.log(secondAccount.withdraw(20)); // 0

[**Vòng lặp for có var trước ES6**](https://developer.mozilla.org/en-US/docs/Glossary/IIFE#for_loop_with_var_before_es6)

Chúng ta có thể thấy cách sử dụng IIFE sau đây trong một số mã cũ, trước khi giới thiệu các câu lệnh **let** và **const** trong **ES6** và phạm vi khối. Với câu lệnh **var** , chúng ta chỉ có phạm vi chức năng và phạm vi toàn cầu. Giả sử chúng tôi muốn tạo 2 nút có nội dung Nút 0 và Nút 1 và khi chúng tôi nhấp vào chúng, chúng tôi muốn chúng cảnh báo 0 và 1. Đoạn mã sau không hoạt động:

for (var i = 0; i < 2; i++) {

const button = document.createElement("button");

button.innerText = `Button ${i}`;

button.onclick = function () {

console.log(i);

};

document.body.appendChild(button);

}

console.log(i); // 2

Khi được nhấp vào, cả Nút 0 và Nút 1 đều cảnh báo 2 vì ilà chung, với giá trị cuối cùng là 2. Để khắc phục sự cố này trước ES6, chúng tôi có thể sử dụng mẫu IIFE:

for (var i = 0; i < 2; i++) {

const button = document.createElement("button");

button.innerText = `Button ${i}`;

button.onclick = (function (copyOfI) {

return function () {

console.log(copyOfI);

};

})(i);

document.body.appendChild(button);

}

console.log(i); // 2

Khi được nhấp, Nút 0 và 1 cảnh báo 0 và 1. Biến iđược xác định trên toàn cầu. Sử dụng câu lệnh **let** , chúng ta có thể thực hiện một cách đơn giản:

for (let i = 0; i < 2; i++) {

const button = document.createElement("button");

button.innerText = `Button ${i}`;

button.onclick = function () {

console.log(i);

};

document.body.appendChild(button);

}

console.log(i); // Uncaught ReferenceError: i is not defined.

Sao chép vào clipboard

Khi được nhấp vào, các nút này sẽ cảnh báo 0 và 1.

VD:

(function js(x) {

const y = (j) => j \* x;

console.log(y(s()));

function s() {

return j();

}

function j() {

return x \*\* x;

}

})(3)

// Output: ?

## Giải thích

Hàm js() được thực thi ngay lập tức sau khi được khai báo và được gọi là IIFE. Lưu ý rằng tham số x của hàm js được truyền thực tế với giá trị 3.

Giá trị trả về của hàm là y(s()), nghĩa là gọi ba hàm khác là y(), s() và j() vì hàm s() trả về j().

j() trả về 3 ^ 3 = 27 để s() trả về27.

y(s()) có nghĩa là y(27) trả về 27 \* 3 = 81.

**II. Scope**

- Các loại phạm vi:

* Global: Toàn cầu
* Code block: Khối mã let, const
* Local scope: Hàm var, function

- Khi gọi mỗi hàm luôn có 1 phạm vi mới được tạo

- Các hàm có thể truy cập các biến được khai báo trong phạm vi của nó và bên ngoài nó

- Cách thức một biến được truy cập

- Khi nào một biến bị xóa khỏi bộ nhớ?

* Biến toàn cầu?
* Biến trong code block & trong hàm?
* Biến trong hàm được tham chiếu bởi 1 hàm?

Phạm **vi(Scope)** là ngữ cảnh thực thi hiện tại trong đó [các giá trị](https://developer.mozilla.org/en-US/docs/Glossary/Value) và biểu thức "hiển thị" hoặc có thể được tham chiếu. Nếu một [biến](https://developer.mozilla.org/en-US/docs/Glossary/Variable) hoặc biểu thức không nằm trong phạm vi hiện tại, nó sẽ không có sẵn để sử dụng. Các phạm vi cũng có thể được xếp lớp trong một hệ thống phân cấp, để các phạm vi con có quyền truy cập vào các phạm vi cha, nhưng không phải ngược lại.

JavaScript có các loại phạm vi sau:

* Phạm vi toàn cầu: Phạm vi mặc định cho tất cả các mã đang chạy trong chế độ tập lệnh.
* Phạm vi mô-đun: Phạm vi mã chạy trong chế độ mô-đun.
* Phạm vi chức năng: Phạm vi được tạo bằng một [chức năng](https://developer.mozilla.org/en-US/docs/Glossary/Function) .

Ngoài ra, các biến được khai báo với [let](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/let)hoặc [const](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/const)có thể thuộc về một phạm vi bổ sung:

* Phạm vi khối: Phạm vi được tạo bằng một cặp dấu ngoặc nhọn (một [khối](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/block) ).

Một [hàm](https://developer.mozilla.org/en-US/docs/Glossary/Function) tạo ra một phạm vi, do đó (ví dụ) một biến được xác định riêng trong hàm không thể được truy cập từ bên ngoài hàm hoặc bên trong các hàm khác. Chẳng hạn, những điều sau đây là không hợp lệ:

function exampleFunction() {

const x = "declared inside function"; // x can only be used in exampleFunction

console.log("Inside function");

console.log(x);

}

console.log(x); // Causes error

Tuy nhiên, đoạn mã sau hợp lệ do biến được khai báo bên ngoài hàm, làm cho nó trở nên toàn cục:

const x = "declared outside function";

exampleFunction();

function exampleFunction() {

console.log("Inside function");

console.log(x);

}

console.log("Outside function");

console.log(x);

Chỉ chặn phạm vi letvà constkhai báo, chứ không phải varkhai báo.

{

var x = 1;

}

console.log(x); // 1

{

const x = 1;

}

console.log(x); // ReferenceError: x is not defined