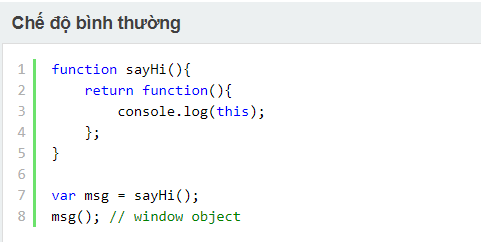
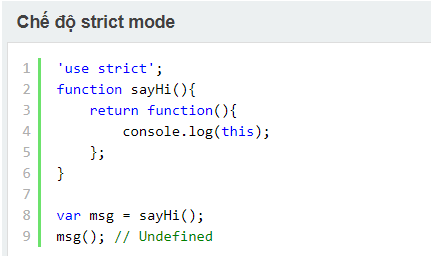
**Tìm hiều về JavaScript nâng cao**

1. **Closure**

* Closure là một **hàm được tạo ra từ bên trong một hàm khác**, nó có thể sử dụng các biến toàn cục, biến cục bộ của hàm cha và biến cục bộ của chính nó. Việc viết hàm theo kiểu closure trong một số trường hợp sẽ giúp code nhìn sáng và dễ quản lý hơn, linh hoạt hơn trong một số trường hợp.
* Các biến bên trong hàm sẽ kết thúc khi hàm đó được chạy xong.
* Cũng như những function khác, nếu chúng ta đang chạy chế độ **strict mode** thì con trỏ this sẽ là **undefined**, còn không thì nó là đối tượng window.

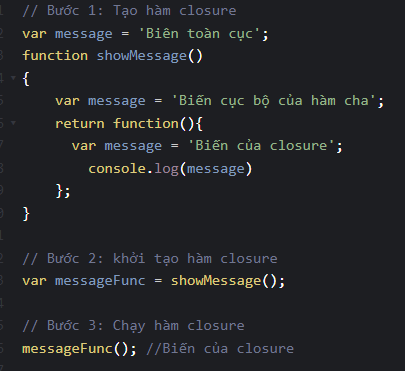




* Độ ưu tiên các biến trong closure function

Trường hợp này nó sẽ ưu tiên từ trong ra ngoài như sau:

* **Bước 1**: Xem biến có nằm trong closure function không? Nếu không thì nhảy qua bước 2, nếu có thì sử dụng.
* **Bước 2**: Xem biến có nằm trong hàm cha không? Nếu không thì qua bước 3, nếu có thì sử dụng.
* **Bước 3**: Xem có phải là biến cục bộ không? Nếu có thì sử dụng, nếu không thì nó sẽ khởi tạo biến mới mới.



1. **Context, this, bind**

* **Context**

Context (hay ngữ cảnh) thường được xác định bằng cách gọi hàm. Khi một hàm được gọi như một phương thức của object, this sẽ là đối tượng đó.

* **This**

Trong javascript, chúng ta dùng từ khóa **this** để đại diện cho một đối tượng (Object). Đối tượng đó là chủ thế của ngữ cảnh, hoặc là chủ thế của code đang được chạy.

* **Bind**

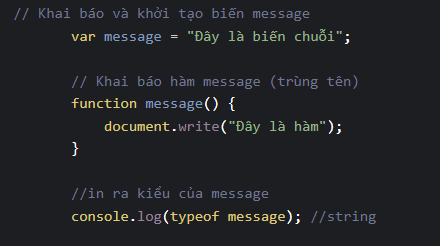
Hàm **bind**() dùng để gán dữ liệu vào đối tượng **this** của hàm đang sử dụng.

1. **Hoisting**

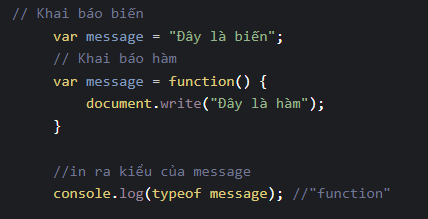
Hoisting là cơ chế của JavaScript cho phép các khai báo biến hoặc hàmđược dời lên trên đầu phạm vi của chúng trước khi thực thi đoạn code.

Thứ tự ưu tiên trong khai báo hàm:

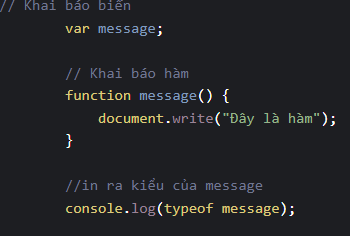
* Gánbiếnưutiênhơn khai báo hàm



* Biểuthức hàm ưu tiên hơn gán biến



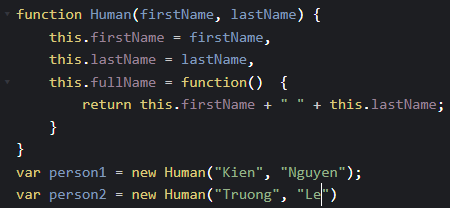
* Khaibáo hàm ưu tiên hơn khai báo biến không có giá trị



1. **Prototype**

**Prototype** là cơ chế mà các object trong javascript kế thừa các tính năng từ một object khác.

Mỗi một object trong javascript đều có một thuộc tính nội bộ (**internal property**) gọi là [[Prototype]].



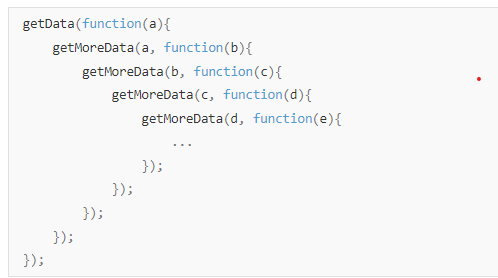
1. **Bất đồng bộ**

Đây là quá trình mà các câu lệnh có thể chạy cùng một lúc chứ không cần chờ câu lệnh trước.

Nếu câu lệnh thứ 2 xong trước thì có thể hiển thị kết ra trước câu lệnh thứ nhất.

* **Callback hell**

**Callback hell** trong javascript là việc chúng ta thực hiện quá nhiều callback lồng nhau.



* **Promise**

Promise được đưa vào Javascript từ ES6, đây có thể coi là một kỹ thuật nâng cao giúp xử lý vấn đề bất đồng bộ hiệu quả hơn.

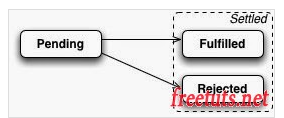
Với trường hợp thực hiện Callback Function thì sẽ dễ xảy ra lỗi **Callback Hell**, Promise sinh ra có thể giúp chúng ta giải quyết những vấn đề này.

Vậy promise sinh ra để xử lý kết quả của một hành động cụ thể, kết quả của mỗi hành động sẽ là thành công hoặc thất bại và Promise sẽ giúp chúng ta giải quyết câu hỏi "Nếu thành công thì làm gì? Nếu thất bại thì làm gì?". Cả hai câu hỏi này ta gọi là một hành động gọi lại (callback action).

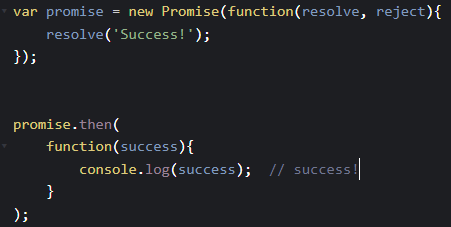
Khi một Promise được khởi tạo thì nó có một trong ba trạng thái sau:

* **Fulfilled** Hành động xử lý xông và thành công
* **Rejected** Hành động xử lý xong và thất bại
* **Pending** Hành động đang chờ xử lý hoặc bị từ chối

Trong đó hai trạng thái **Reject** và **Fulfilled** ta gọi là **Settled**, tức là đã xử lý xong.



Ví dụ:



* **Async await**