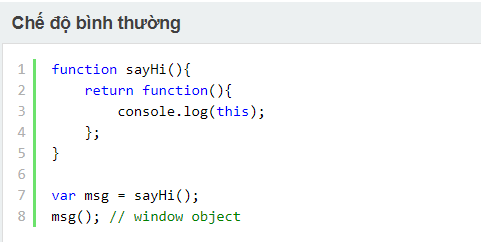
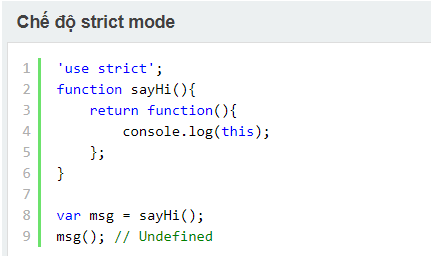
**Tìm hiều về JavaScript nâng cao và ES6**

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.

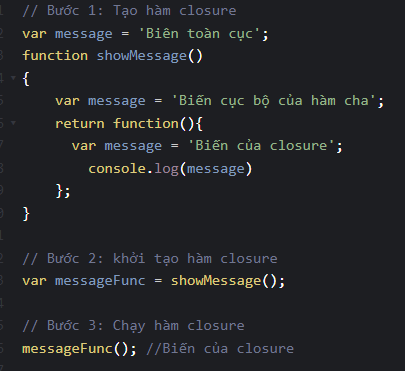




* Độ ưu tiên các biến trong closure function

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**

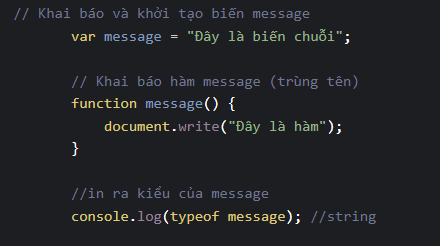
Hàm **bind**() dùng để gán dữ liệu vào đối tượng **this** của hàm đang sử dụng.

1. **Hoisting**

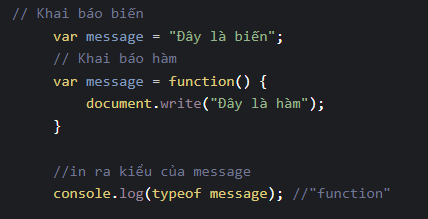
Hoisting là cơ chế của JavaScript cho phép các khai báo biến hoặc hàmđược dời lên trên đầu phạm vi của chúng trước khi thực thi đoạn code.

Thứ tự ưu tiên trong khai báo hàm:

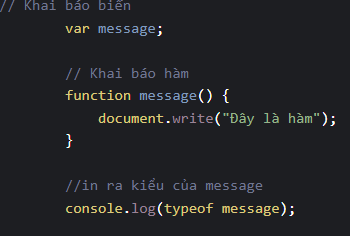
* Gánbiếnưutiênhơn khai báo hàm



* Biểuthức hàm ưu tiên hơn gán biến



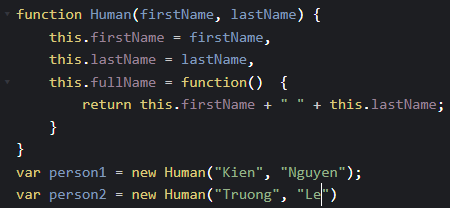
* Khaibáo hàm ưu tiên hơn khai báo biến không có giá trị



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]].



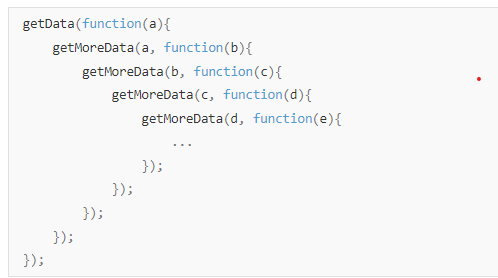
1. **Bất đồng bộ**

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

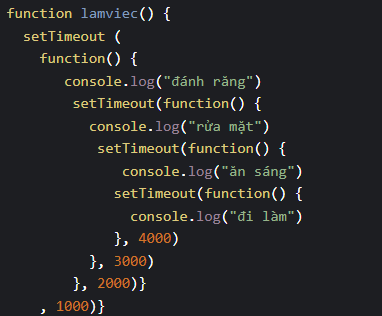
Nếu câu lệnh thứ 2 xong trước thì có thể hiển thị kết ra trước câu lệnh thứ nhất.

* **Callback hell**

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



Ví dụ về Callback hell:



* **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.

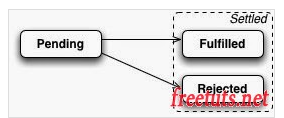
Với trường hợp thực hiện Callback Function thì sẽ dễ xảy ra lỗi **Callback Hell**, Promise sinh ra có thể giúp chúng ta giải quyết những vấn đề này.

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 xong 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 để xử lý callback hell như ví dụ trên:

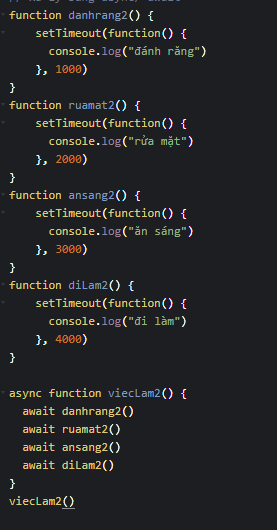


* **Async await**

**Async / Await** là một tính năng của JavaScript giúp chúng ta làm việc với các hàm bất đồng bộ theo cách thú vị hơn và dễ hiểu hơn. Nó được xây dựng trên Promises và tương thích với tất cả các Promise dựa trên API. Trong đó:

* **Async** - khai báo một hàm bất đồng bộ (async function someName(){...}).
* Tự động biến đổi một hàm thông thường thành một Promise.
* Khi gọi tới hàm async nó sẽ xử lý mọi thứ và được trả về kết quả trong hàm của nó.
* Async cho phép sử dụng Await.
* **Await** - tạm dừng việc thực hiện các hàm async. (Var result = await someAsyncCall ().
* Khi được đặt trước một Promise, nó sẽ đợi cho đến khi Promise kết thúc và trả về kết quả.
* Await chỉ làm việc với Promises, nó không hoạt động với callbacks.
* Await chỉ có thể được sử dụng bên trong các function async.

Ví dụ xử lý hàm callback hell theo asyn/await

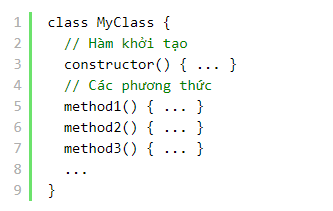


1. **Class**

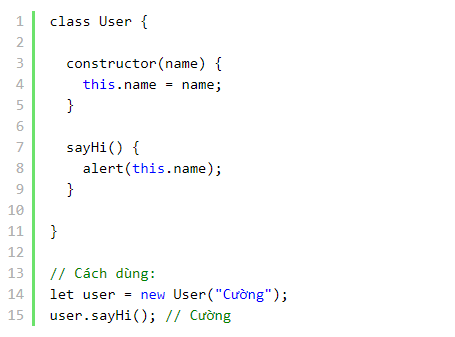
Class trong javascript là một kỹ thuật giúp ta tạo ra các lớp mẫu chương trình để thể hiện cho các đối tượng. Qua đó, ta có thể thêm các hành động và thuộc tính của đối tượng vào trong lớp thông qua hai khái niệm thuộc tính và phương thức.

Ví dụ, đối tượng Sinh Viên sẽ có các đặc điểm như: Mã sinh viên, tên, tuổi, quê quán ... Và các hành động như: Thêm / xóa / sửa và thi cử.. Lúc này ta sẽ cài đặt đối tượng sinh viên này trong javascript thông qua class.

Cú pháp:



Ví dụ tạo 1 method trong trong class:



**ES6**

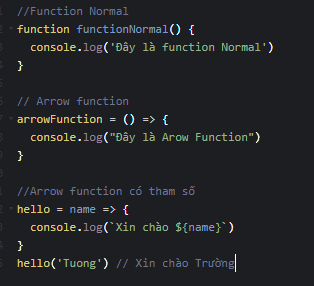
1. **Block-scoped: let, const**

Khi khai báo biến bằng let và const trong 1 khối mã (block) thì chúng chỉ có thể dùng trong khối mã đó.

1. **Arrow Function**

**Arrow Function** (hay còn gọi là hàm mũi tên) là một trong những tính năng mới và rất hay của ES6. Nó thừa hưởng cách viết ngắn gọn bằng cú pháp ES6. Tuy Arrow Function và Function đều là function, nhưng cách viết và cách tham chiếu tới bối cảnh (context) khác nhau.

Ví dụ về **Arrow function** với normal function:



Sự khác biệt của arrow function với normal function:

* **Arrow function không định nghĩa giá trị this của riêng nó giống như function normal.**

Ví dụ về this trong Normal function và arrow function

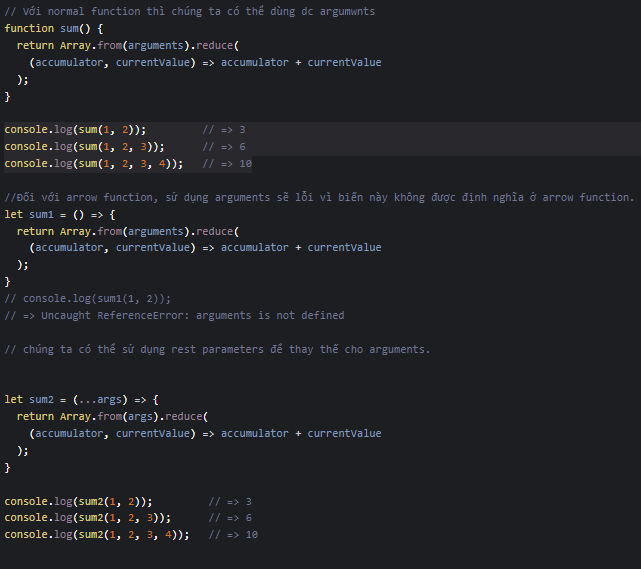


Vì arrow function không có địng nghĩa lại **this.** Do đó, giá trị của this sẽ ứng với ngữ cảnh gần nhất của nó. Vì vậy, trong đoạn code trên this.x sẽ tương ứng với giá trị x được truyền vào và bằng 1.

* **Arrow function không bind arguments**

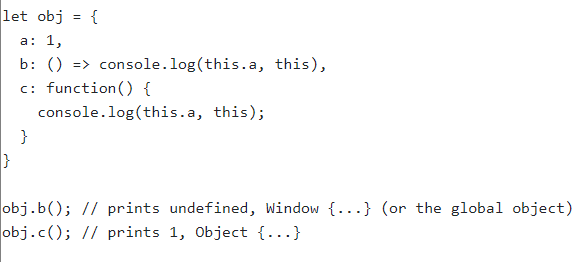
Tương tự như đối với this, arrow function cũng không định nghĩa giá trị arguments của riêng nó.

Đối tượng arguments đặc biệt hữu ích đối với những function không xác định trước số lượng tham số.



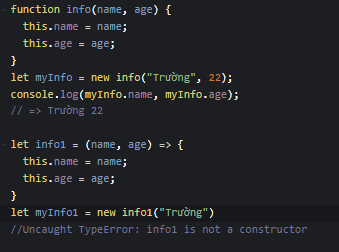
* **Arrow function không phù hợp làm method cho object**

Chính vì arrow function không định nghĩa giá trị this của riêng nó, nên cũng không phù hợp là method cho object. Vì vậy, nếu muốn định nghĩa method cho object thì bạn nên sử dụng function

****

* **Arrow function không thể sử dụng làm hàm constructor**

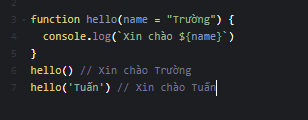
Nếu thay đoạn code trên bằng arrow function thì bạn sẽ bị lỗi: **TypeError: info is not a constructor**.



* Arrow function không có thuộc tính prototype
* Arrow function không được hoisted

1. **Default Parameter**

Là giá trị mặc định được gán ngay vị tri khai báo tham số



1. **Rest Parameter**

**Rest Parameter** (hay còn gọi là đại diện của các tham số còn lại) cho phép bạn biểu diễn một số lượng vô hạn đối số dưới dạng một **mảng**.



1. **Spread**

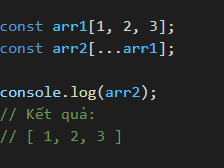
Toán tử **Spread** là một tính năng bổ sung mới cho các tính năng có sẵn trong phiên bản JavaScript ES6 với cú pháp có 3 dấu chấm phía trước

(… spread). Toán tử Spread được sử dụng để mở rộng một đối tượng hoặc một mảng có thể lặp.

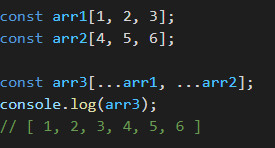
Những cách sử dụng toán tử **Spread:**

* Coppymột mảng bằng toán tử Spread

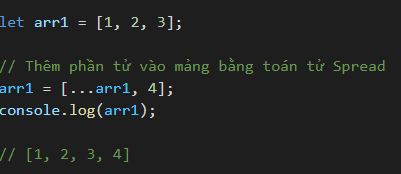
Chúng ta có thể sử dụng toán tử Spread để sao chép 1 mảng.



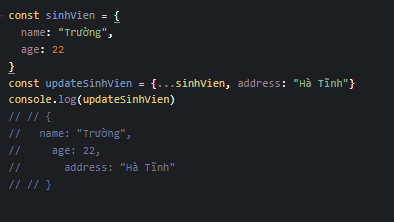
* Nối các mảng lại với nhau



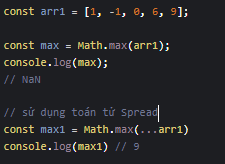
* Thêm phần tử vào mảng



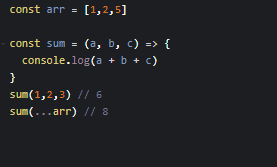
* Thêm thuộc tính vào object



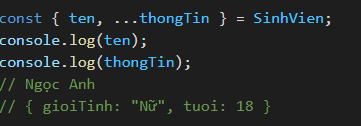
* Spread kết hợp với Math()



* Truyềnmảng như đối số với toán tử spread

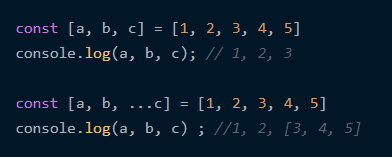


* **Phép gán hủy cấu trúc đối tượng**

****

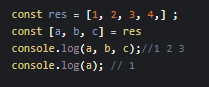
1. **Destructuring Assignment**

Destructuring là một cú pháp cho phép bạn gán các thuộc tính của một Object hoặc một Array. Điều này có thể làm giảm đáng kể các dòng mã cần thiết để thao tác dữ liệu trong các cấu trúc này. Có hai loại Destructuring: Destructuring Objects và Destructuring Arrays.

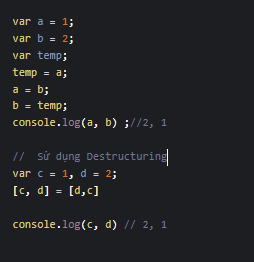


Các ứng dụng của **Destructuring:**

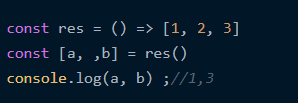
* Gán biến

****

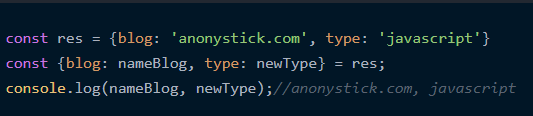
* **Swapping**

****

* Bỏ bớt các giá trị



* **Assignment to new variables**

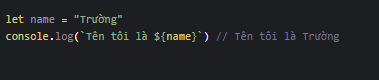


* **Nested object and array destructuring**

****

1. **Template Literals**

**Template Literals hay còn gọi là Template Strings là một cú pháp mới để khai báo String trong JavaScript được giới thiệu trong ES2015/ES6. Nó cho phép chúng ta sử dụng multi-line string, biến, biểu thức, hàm bên trong String mà không phải thông qua phép cộng String.**

****