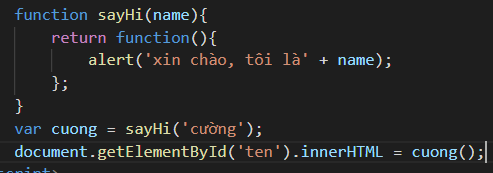
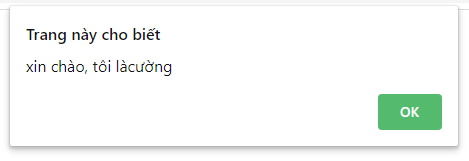
**Tìm hiểu kiến thức js nâng cao**

1. **Closure:**

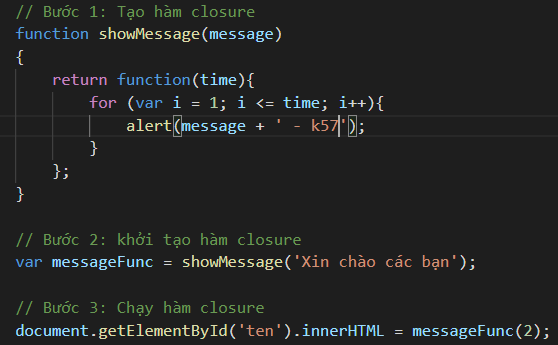
* Là một hàm được tạo ra từ bên trong một hàm khác
* Có thể dùng biến toàn cục, biến cục bộ của hàm cha và biến cục bộ của chính nó
* Giúp code nhìn sáng và dễ quản lý hơn, linh hoạt hơn
* Ví dụ:

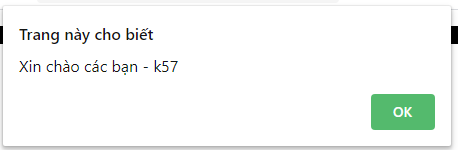




* Với biến của hàm cha, mặc dù sau khi hàm cha chạy xong thì biến của hàm này sẽ bị hủy nhưng nó vẫn tồn tại trong closure function
* Nếu chạy strict mode thì con trỏ this là undefined còn không là đối tượng window
* Có thể tạo closure trong các phương thức của class nhưng không thể truy cập đến các thuộc tính của class vì this trong closure là undefined. Việc này có thể xử lý bằng việc gán con trỏ this vào một biến khác.
* Ví dụ về closure

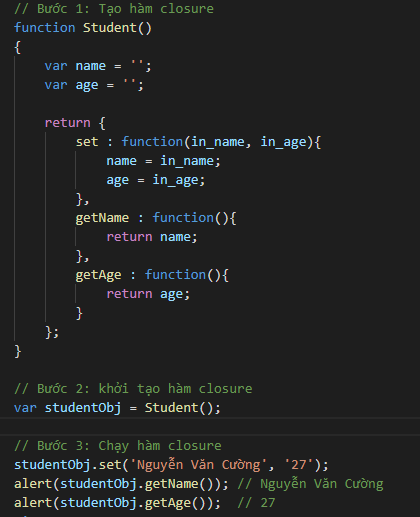
+ Closure có tham số

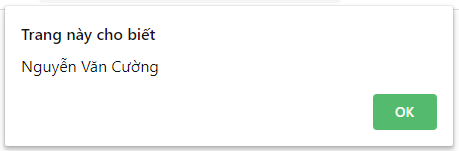


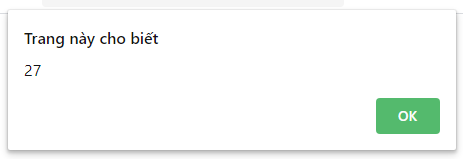


+ Return nhiều closure function

+ Closure thay đổi giá trị biến toàn cục lẫn cục bộ







* Độ ưu tiên các biến:

+ Bước 1: Xem biến có nằm trong closure function không, nếu không thì qua bước 2, nếu có thì dùng

+ Bước 2: Xem biến có nằm trong hàm cha không, nếu không thì qua bước 3, nếu có thì dùng

+ Bước 3: Xem có phải biến cục bộ không, nếu có thì dùng, nếu không thì nó sẽ khởi tạo biến mới

1. **Context, this, bind**
   1. **Context**

+ Có thể hiểu đơn giản là ngữ cảnh, cung cấp thông tin về môi trường thực thi đoạn mã

+ Thường được xác định bằng cách gọi hàm

+ Khi hàm được gọi như một phương thức của object, this sẽ là đối tượng đó

+ Mặc định this trỏ đến ngữ cảnh toàn cục hoặc window object, nếu function ở strict mode thì context mặc định là undefined

+ Khi trình thông dịch js thực thi code, nó mặc định nằm trong ngữ cảnh thực thi toàn cục (global execution context). Mỗi lần gọi hàm sau đó sẽ tạo ngữ cảnh thực thi mới

+ Một bộ ngữ cảnh thực thi sẽ tạo thành một stack, đáy stack là ngữ cảnh thực thi toàn cục, trên cùng là ngữ cảnh thực thi hiện tại

+ Một ngữ cảnh thực thi được chia thành 2 pha: tạo và thực thi. Ở pha tạo sẽ tạo đối tượng biến chứa tất cả biến, khai báo hàm, tham số. Ở pha thực thi, code được thông dịch và chạy

* 1. **This**

+ Khi dùng this ở cấp ngoài cùng chương trình thì nó chính là đối tượng window, vì vậy có thể thay thế window object

+ Ví dụ 2 đoạn code dưới đây là tương đương

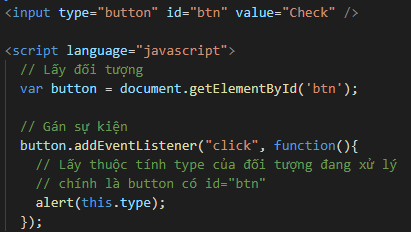


+ Mặc định this trỏ đến ngữ cảnh toàn cục hoặc window object, nếu function ở strict mode thì context mặc định là undefined

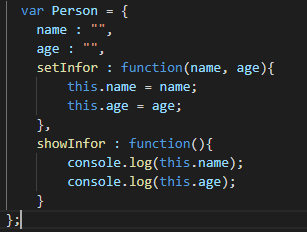
+ Khi dùng hàm như một constructor thì this chính là các instance của hàm

+ Khi gán hành động cho sự kiện thì this chính là đối tượng HTML đang được gán sự kiện đó

+ Ví dụ:

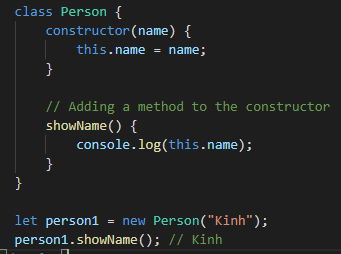


+ This trong js object



Trong phương thức setInfor và showInfor đã dùng this để lấy thuộc tính

+ This trong js class

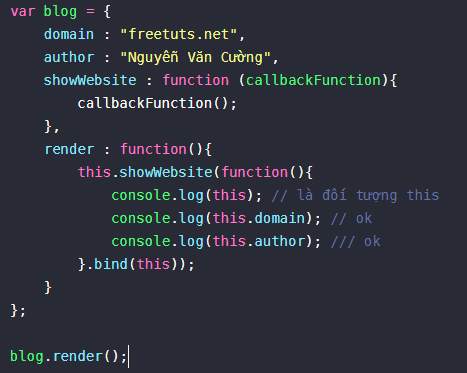


+ This trong arrow function: Arrow function không tồn tại đối tượng this, khi sử dụng thì nó lấy đối tượng global window

* 1. **Bind**

+ Dùng để gán dữ liệu vào đối tượng this của hàm đang dùng

+ Ví dụ:



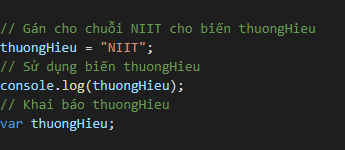
Kết quả:



+ Như vậy khi truyền bất kỳ một loại dữ liệu nào trong tham số của hàm bind thì đối tượng this sẽ nhận chính dữ liệu đó

1. **Hoisting**

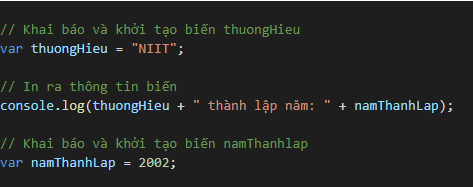
* Là cơ chế của js cho phép các khai báo biến hoặc hàm được dời lên trên đầu phạm vi của chúng trước khi thực thi đoạn code (chỉ di chuyển phần khai báo)
* Ví dụ:



Kết quả:



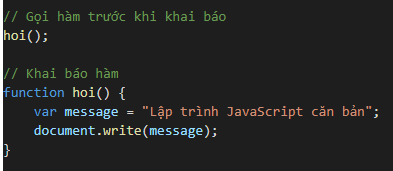
* Js chỉ lưu trữ các khai báo, không lưu trữ khởi tạo



Kết quả:



* Khi khai báo biến mà không gán dữ liệu thì giá trị là undefined, và hoisting chi chuyển phần khai báo lên đầu chứ không chuyển phần gán giá trị
* Đối với khai báo hàm thì hoisting cũng giống như biến js, việc khai báo được chuyển lên đầu



* Đối với biểu thức hàm thì cơ chế hoisting không áp dụng vì biểu thức hàm bản chất là hàm được gán cho một biến
* Trong hoisting gán biến có độ ưu tiên cao hơn gán hàm, biểu thức hàm ưu tiên hơn gán biến, khai báo hàm ưu tiên hơn khai báo biến
* Trong một dự án js nên hạn chế hoisting (bật chế độ use strict) hoặc dùng let hoặc const để khai báo biến, mặc dù biến khai báo vẫn được đưa lên đầu nhưng nó không cho phép dùng cho đến khi được khai báo

1. **Prototype**

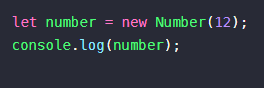
* Là cơ chế mà các object trong js kế thừa tính năng từ một object khác
* Tất cả object trong js đều có một prototype
* Bản thân prototype là một object
* Cơ chế hoạt động

+ Bước 1: Kiểm tra thuộc tính có sẵn trong đối tượng không, nếu có thì sử dụng và kết thúc, nếu không thì qua bước 2

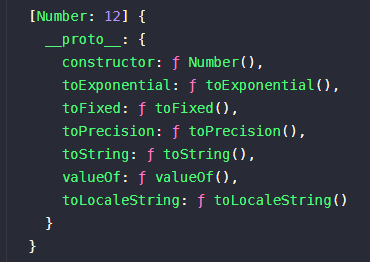
+ Bước 2: js sẽ kiểm tra trong prototype của đối tượng đó có hay không, nếu có thì dùng và kết thúc, không thì qua bước 3

+ Bước 3: Trả về undefined nếu đang truy cập thuộc tính hoặc thông báo lỗi nếu truy cập phương thức

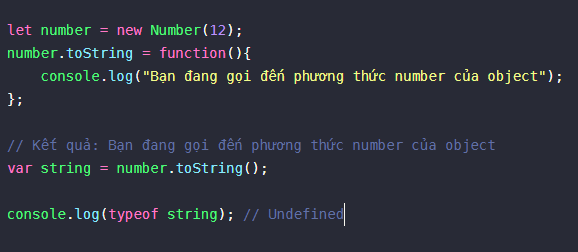
* Ví dụ:



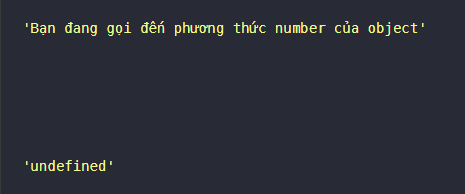
Kết quả:



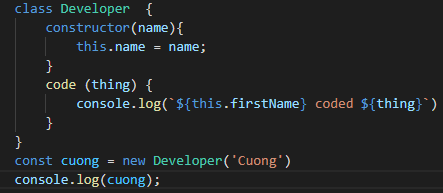
* Nếu đối tượng và prototype của đối tượng đều có cùng phương thức thì khi dùng sẽ ưu tiên phương thức của đối tượng



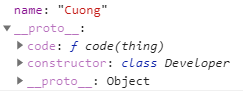
Kết quả:



* Prototype của lớp không có kế thừa

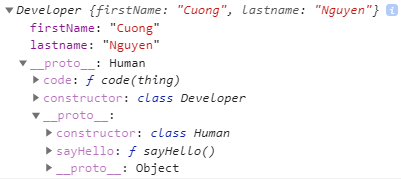


Kết quả:



* Prototype của lớp có kế thừa





* Mỗi đối tượng js được chia làm 2 phần, thứ nhất là thuộc tính đối tượng, thứ 2 là prototype của đối tượng. Khi bổ sung một thuộc tính cho đối tượng thì có thể chọn một trong 2 cách trên, cụ thể:

+ Thêm trực tiếp vào đối tượng

+ Thêm vào prototype

1. **Bất đồng bộ**
   1. **Callback hell**

+ Hiểu đơn giản thì callback hell là thực hiện nhiều callback lồng nhau

+ Ví dụ



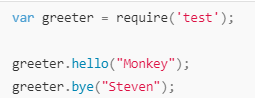
+ 6 cách xử lý callback hell trong js:

* Thiết kế ứng dụng theo dạng module

Ví dụ: Tạo một module tên Test



Sau đó gọi ở ứng dụng:



* Đặt tên cho callback trong js



* Định nghĩa hàm trước khi gọi



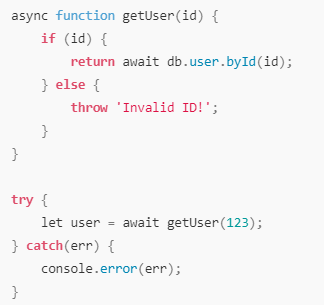
* Sử dụng module Asynce.js



* Dùng Promises



* Async/Await nhằm giảm khả năng xảy ra callback hell



* 1. **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 và ngược lại?”

+ Khi một Promise được khởi tạo thì nó có 3 trạng thái sau:

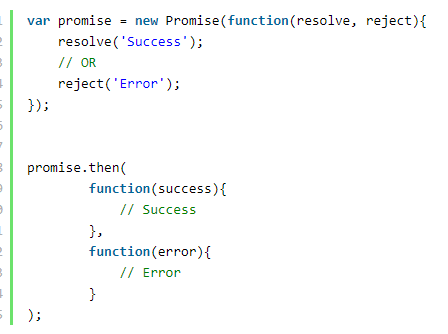
* Fulfilled: Hành động xử lý xong và thành công
* Rejected: Hành động xử lý xong và thất bại
* Pending: Hành động đang chờ xử lý hoặc bị từ chối

+ Tạo một Promise:



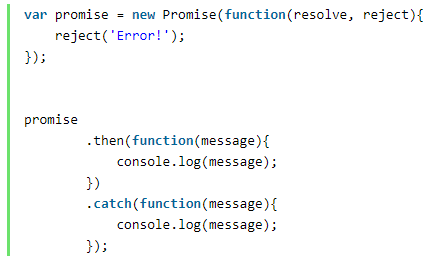
Trong đó callback có 2 tham số truyền vào là resolve dùng để xử lý hành động thành công và reject dùng để xử lý hành động thất bại

+ Thenable trong Promise là một phương thức ghi nhận kết quả của trạng thái được khai báo ở Resolve và Reject. Nó có 2 tham số truyền vào là 2 callback function xử lý cho Resolve và Reject



Hai hàm callback trong then chỉ xảy ra một trong hai, nếu khai báo cả hai thì chỉ tác dụng với khai báo đầu tiên

+ Catch trong Promise dùng để bắt lỗi



Kết quả:



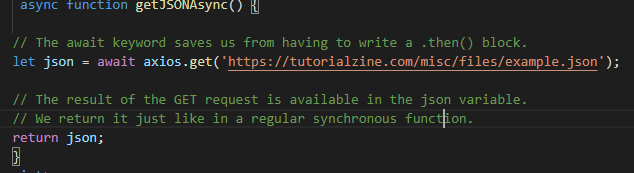
Nếu vừa truyền callback error và vừa dùng catch thì nó sẽ chạy hàm callback error còn catch sẽ không chạy.

* 1. **Async await**

+ Là một tính năng của js giúp làm việc với hàm bất đồng bộ theo cách dễ hiểu hơn. Nó được xây dựng trên Promises và tương thích với tất cả Promise dựa trên API. Trong đó:

* Async: Khai báo một hàm bất đồng bộ
  + 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
  + 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:



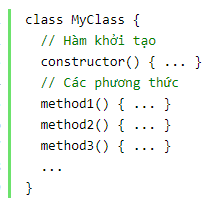
+ Async / Await cho phép bắt lỗi bằng cacgs dùng try / catch hoặc .catch

+ Async / Await có thể dùng trong hầu hết trình duyệt trừ IE11

1. **Class**

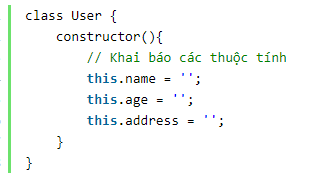
+ Là một kỹ thuật giúp tạo ra các lớp mẫu chương trình để thể hiện cho các đối tượng, qua đó có thể thâm hành động và thuộc tính của đối tượng vào lớp thông qua thuộc tính và phương thức

+ Cú pháp:

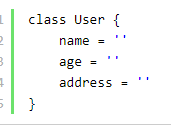


+ Để khai báo thuộc tính cho class thì có 2 cách:

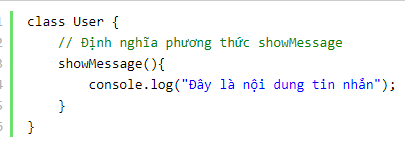
Cách 1: Khai báo trong hàm constructor



Cách 2: Khai báo trực tiếp



+ Khai báo phương thức:



+ Lưu ý: Trong class không cần dùng dấu phẩy để ngăn cách giữa cái thuộc tính hoặc phương thức

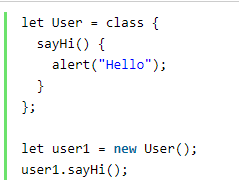
+ Để tạo class trong js thì qua 2 bước sau:

Bước 1: Phân tích đối tượng có thuộc tính, phương thức nào

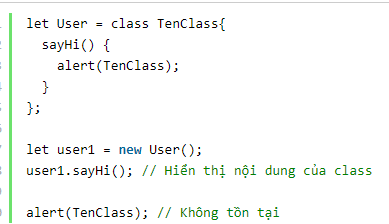
Bước 2: Cài đặt class trong js

+ Trong js, class là một function nên nó sẽ có prototype và ta có thể dùng function để khai báo thay cho class

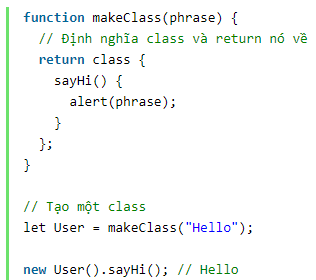
+ Class có thể được khai báo và gắn trực tiếp nó vào một biến



+ Có thể đặt tên ảo cho class và tên này chỉ được phép dùng trong class



+ Có thể khở tạo class trong một hàm và cho hàm đó return về chính class đó



+ Getters trong class là những hàm có nhiệm vụ lấy giá trị của thuộc tính

+ Setters trong class là những hàm có nhiệm vụ thiết lập giá trị cho các thuộc tính

