1. **Phân biệt kiểu biến var, let, và const trong JavaScript**

Biến var

Với từ khóa var chúng ta có thể khai báo đa dạng các kiểu biến như number, string, boolean, etc. Trừ trường hợp được khai báo bên trong 1 function (khi đó biến var sẽ có scope là **function/locally scoped**), biến var sẽ có scope là **globally scoped**. Đặc biệt, biến var còn có thêm tính chất **hoisting**: nghĩa là dù khai báo ở đâu thì biến đều sẽ được đem lên đầu scope trước khi code được thực hiện.

## Biến let

Một trong những nguyên nhân khiến let có thể thay thế var để xử lý vấn đề nêu trên là vì biến let được khai báo sẽ có scope là **block scoped** chứ không phải globally hay locally scoped.

## Biến const

Tương tự với let cũng có scope là **block scoped**, và **hoisting** thì chúng ta có thêm 1 kiểu khai báo biến nữa là const. Trong biến const nếu trường hợp kiểu của biến là **primitive** (bao gồm string, number, boolean, null, và undefined) thì chúng ta sẽ không thể tái khai báo hay cập nhật giá trị mới để thay thế cho giá trị trước đó của biến.

1. **Js là ngôn ngữ bất đồng bộ**

 Trong Javascript, **hàm (function) sẽ không bao giờ được gọi xử lí trực tiếp, nó sẽ được giải quyết thông qua các message.**

Bộ máy xử lí bên trong của Javascript gồm có các phần chính như sau: một hàng đợi tin nhắn (message queue), một vòng lặp event (event-loop) và một stack xử lí (call stack**). Message queue tiếp nhận các xử lí được gọi tới**, mỗi hàm được gọi sẽ tương ứng là 1 message trong hàng đợi. **Event-loop làm nhiệm vụ điều phối việc lấy các message đang có trong queue**: mỗi lần sẽ lấy 1 message ra khỏi queue để đem đi xử lí, trong quá trình xử lí thì event-loop sẽ đứng chờ cho tới khi message cũ được xử lí xong, sau đó sẽ tiếp tục lấy message mới trong queue để xử lí, và cứ lặp lại như thế.

**Call-stack là 1 ngăn xếp quản lí việc thực thi hàm và gọi lồng hàm (nested function),** tức là: Với mỗi hàm vừa lấy ra khỏi queue để đem ra xử lí, thì nó sẽ tương ứng với 1 frame ở trong call-stack, nếu bên trong hàm này có gọi tới nhiều hàm con nữa, thì những hàm con này sẽ được thêm tiếp vào call-stack dưới dạng 1 frame mới của stack. Và Javascript sẽ lần lượt xử lí call-stack này theo dạng first-in last-out (xử lí dạng stack), cho đến khi stack rỗng thì coi như message đó đã được xử lí xong.

1. **Promise là gì**

Promise là một cơ chế trong JavaScript giúp bạn thực thi các tác vụ bất đồng bộ mà không rơi vào callback hell hay pyramid of doom, là tình trạng các hàm callback lồng vào nhau ở quá nhiều tầng. Các tác vụ bất đồng bộ có thể là gửi AJAX request, gọi hàm bên trong setTimeout, setInterval hoặc requestAnimationFrame, hay thao tác với WebSocket hoặc Worker

1. **destructurning trong JS**

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.

1. **Callback là gì?**

Callback là một hàm sẽ được thực hiện sau khi một hàm khác đã thực hiện xong

vì thế nó có tên là callback.

callback tức là ta truyền một đoạn code (Hàm A) này vào một đoạn code khác (Hàm B).

Tới một thời điểm nào đó, Hàm A sẽ được hàm B gọi lại (callback). Javascript là một

ngôn ngữ lập trình hướng sự kiện và bất đồng bộ nên callback function đóng vai trò rất quan trọng,

bạn sẽ truyền một callback function vào các sự kiện và xử lý bất đồng bộ đó..

1. **Async - Await là gì? Cú pháp của Asyns-Await ra sao?**

JavaScript luôn được cải tiến và trong bản cập nhật ES7 vào năm 2016, [JavaScript](https://wiki.tino.org/html-css-javascript-la-gi/) đã được cập nhật thêm rất nhiều function (*chức năng*) mới được như: Arrow Functions, Array methods, Loops for/in, Template literals,…

Và trong đó có một function mới để giải quyết bài toán bất đồng bộ một cách nhanh gọn hơn, đó chính là Async Await function.

Async Await được phát triển dựa trên nền tảng của Promise function và tương thích với tất cả Promise nào dựa trên API để xử lý các trường hợp bất đồng bộ.

Trong đó, ta có Async Await với tính năng như sau:

* **Async**được sử dụng để khai báo một hàm bất đồng bộ và hàm sẽ luôn trả về một giá trị. Nếu Promise không được thực thi, JavaScript sẽ tự động kết thúc quá trình.
* **Await**được sử dụng để chờ một Promises trong một khối Async, công dụng dễ hiểu của Await là “kêu” JavaScript phải chờ cho đến khi nào có một Promise trả kết quả. Trong đó, await chỉ có công dụng thực hiện chức năng tạo ra sự không đồng bộ bằng cách chờ 1 khối trả kết quả, await không chờ đợi toàn bộ kết quả trong chương trình được trả về.

Cú pháp Async :

async function myFunction() { return "Hello"; }

Cú pháp Await :

let value = await Promise;

1. **Ưu và nhược điểm của Asyns - Await với Promise.**

Async/await là cách mới để viết code bất đồng bộ. Các phương pháp làm việc với code bất đồng bộ trước đây là sử dụng callback và promise.

Async/await là khái niệm được xây dựng ở tầng trên promise. Do đó nó không thể sử dụng với callback thuần.

Async/await cũng giống như promise, là non-blocking.

Async/await làm cho code bất đồng bộ nhìn và chạy gần giống như code đồng bộ.