Tổng quan về ES6

Giúp viết code nhanh hơn, thông minh hơn so với ES5

1. Arrow Functions

**var sum = (a) => {**

**return a + 100;**

**}**

1. Template Literal: ``

### Class – Lớp:

### Rest parameter(…):

* Khai báo số lượng tham số không xác định

### Spread Attributes

**const *oldArray*** = [1, 2, 3];  
**const *newArray*** = [...***oldArray***, 4, 5];

**function** *sum*(x, y, z) { **return** x + y + z; }  
**const *numbers*** = [1, 2, 3]; ***console***.log(*sum*(...***numbers***));

1. For in, for of
2. Default Parameter: Đặt giá trị mặc định cho tham số

* Thay vì không truyền vào bị undifined thì nay có thể đặt 1 giá trị mặc định nếu không truyền vào
* Giá trị mặc định cho tham số có thể là số, chuỗi, mảng, lời gọi hàm, sử dụng các tham số khác, sử dụng các biểu thức
* Arguments: là đối tượng lưu các tham số ()

1. Destructuring:

**let *colors*** = [**"Xanh"**, **"Đỏ"**];  
**let** [***a***, ***b***] = ***colors***;

**let *school*** = { **name**: **"CodeGym"**, **age**: };  
**let** { ***name***, ***age*** } = ***school***;

Lớp và đối tượng:

* **Prototype** của mỗi đối tượng chính là cha của nó.
* Khi khai báo hàm tạo cũng là khi khai báo ra prototype của Lớp.
* Prototype cao nhất là đối tượng Object
* Prototype là đối tượng tham chiếu đến đối tượng cha.
* Mỗi đối tượng trong js được tạo ra thì luôn có 1 thuộc tính gọi là prototype.
* Prototype mặc định của các đối tượng là đối tượng thuộc lớp Object (lớp mặc định của js đã có nhiều phương thức mặc định)
* Nếu muốn định nghĩa lại prototype chúng ta cần hàm tạo (constructor) hay nói cách khác, khi tạo lớp có thì chúng ta có thể quy định prototype.
* *Đứa trẻ được sinh ra thì chúng mặc định là con người => Tương tự với việc 1 đối tượng được tạo ra thì là 1 object.*
* *Những đứa trẻ sinh ra là người họ Nguyễn khi chúng có cha họ Nguyễn => Tương tự với việc các đối tượng tạo ra có cùng thuộc tính chung ‘Họ’ = ‘Nguyễn’ thì chúng phải thuộc 1 lớp có prototype họ = ‘Nguyễn’.*
* Nguyễn mẫu vs Khung?
* Khung: Iphone(màn hình, camera, kiểu dáng) => Iphone 12, iphone 13, iphone 14.
* Khuôn mẫu: Iphone13 (Mẫu ip13): màu, giá.

Typescript

Xử lý bất đồng bộ

1. Phân biệt đồng bộ và bất đồng bộ
   1. Đồng bộ: các dòng lệnh chạy lần lượt từ trên xuống dưới
   2. Bất đồng bộ:

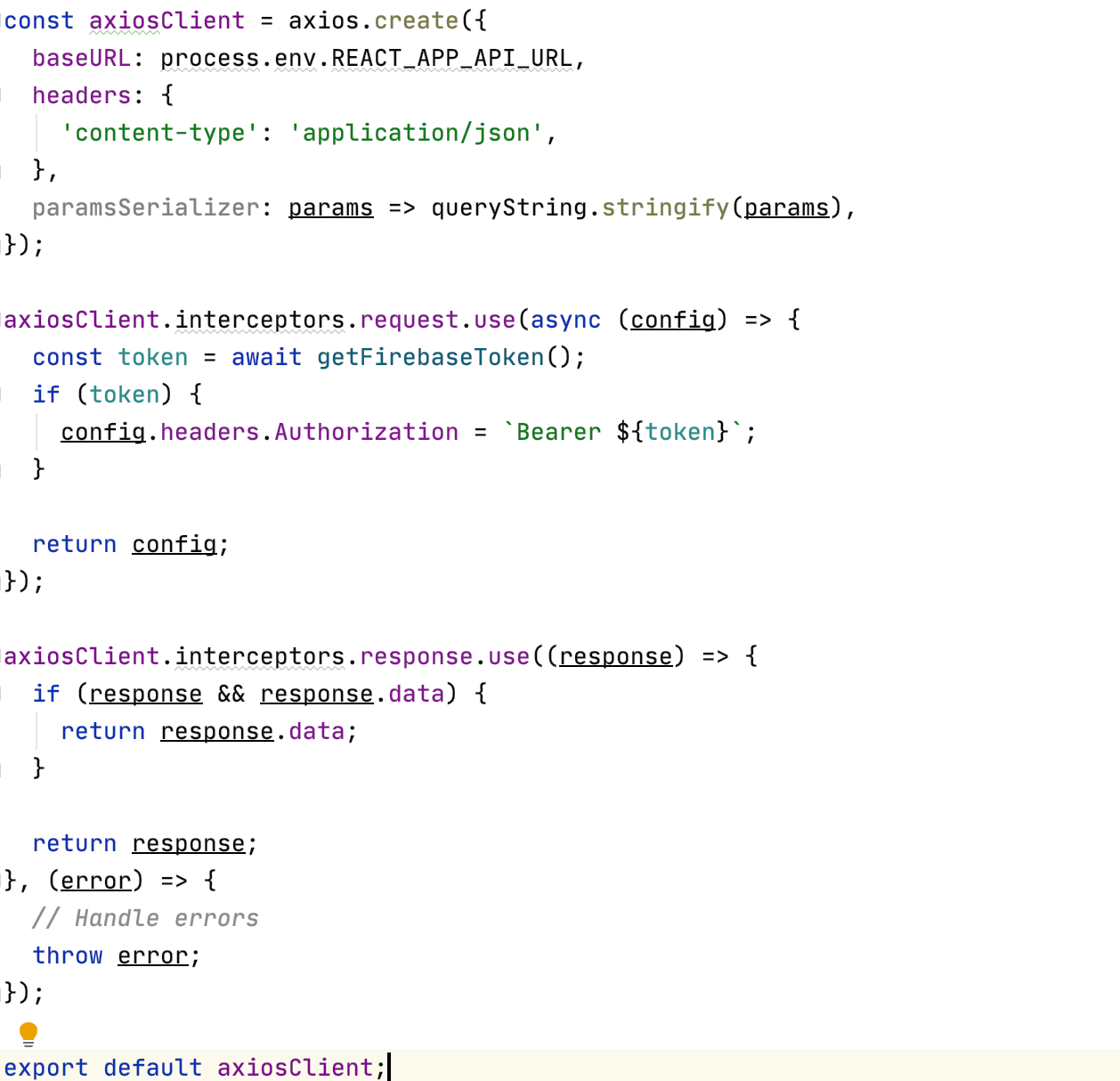
* Các dòng lệnh chạy không đúng thứ tự
* Xảy ra khi lấy dữ liệu một môi trường khác, nằm ngoài dự án

1. Xử lý bất đồng bộ
   1. Callback
   2. Async Await
   3. Promise

* Nói về DOM
* Document.createElement vs React.createElement.
* JSX: viết HTML(XML) trong js. Viết dạng html bình thường, thông qua babel sẽ được dịch sang react.
* Component rerender sau khi state thay đổi.
* Khi unmounted thì được coi là hết 1 vòng đời
* React Component:
  + Có 2 cách tạo 1 component: class, function
  + Tái sử dụng nhiều lần
  + Prop: khai báo tham số đầu vào
  + State:
    - khai báo biến trong component (chỉ dùng được trong class component)
    - Chỉ có thay state thì component mới re-render.
    - Khi prop được thay đổi(phải từ component cha thay đổi) thì component cũng re-render.
  + Khi unmounted thì được coi là hết 1 vòng đời
* useEffects:
  + Loại 1(chỉ callback): hỗ trợ tự động thực hiện 1 hành động ngay sau hành động render (re render thì cũng thực hiện)
  + Loại 2(callback, []): thực hiện hành động 1 lần sau render.
  + Loại 3(callback, [deps]):
    - callback gọi khi deps thay đổi
    - useEffect sẽ kiểm ra lại deps ra khi re-render
  + Có thể viết nhiều useEffect trong 1 component
  + Dùng với DOM event thì cần xóa event listener khi unmounted, nếu không sẽ gây ra lỗi window lắng nghe 2 sự kiện như nhau
  + Return(callback) trong userEffects () = clean up function: hàm chạy trước khi unmounted.
  + Từ lần 2 useEffect chạy thì callback luôn chạy sau clean up
* Formik:
  + Thư viên hỗ trợ làm form thuận lợi hơn
  + Cung cấp 1 số sự kiện: onSubmit, onChange, handleSubmit, handleChange,..
  + Cung cấp 1 số component:
    - Form: tự động kết nối các sự kiện với các sự kiện Formik cung cấp
    - Field: tự nối các giá trị trong values vào các input theo name, mặc định là input type text, có thể custom.
    - ErrorMessage: tương tự như field nhưng để báo lỗi.
  + Trong code thể hiện dưới dạng 1 component
  + Khi sử dụng thì có thể cung cấp nhiều props: onSubmit, onChange, validate, initailValue. Children của Formik thì là 1 đoạn jsx hoặc 1 function return ra jsx.
  + Các component bên trong luôn có đầy đủ các props tương ứng với props của Formik
* Hiện đầu tiên là trang xem danh sách sản phẩm(có nút chuyển sang giao diện thêm mới)
* Trong các giao diện thêm mới, sửa có nút về trang chủ.
* Dữ liệu lưu và cập nhật trong context.
* BrowserRouter, Routes, Route, navigate, *useParams*, *useLocation*

Redux:

* Giúp quản lý state dễ dàng hơn.
* Là 1 Pattern: Muốn nó hiệu quả thì phải tuân theo quy tắc của nó.
* Nguyên tắc hoạt động của React:
  + Khởi tạo, cập nhật state
    - => Render ra view
    - => Tác động view
    - => Cập nhật sate
    - …
* Các thành phần chính:
  + Store: lưu các state trong ứng dụng, store thì có nhiều reducer bên trong
  + Reducer:
    - Là 1 function để cập nhật lại 1 state
    - Đầu vào là state và action
    - Phần code bên trong hàm: cập nhật state tương ứng với action nhận được
    - Quy tắc:
      * Không được thay đổi state cũ mà dùng spread (…)
      * Không được có code bất đồng bộ => thunk
      * Dùng hành động chắc chắn kết quả !(Math.Random)
    - Có thể tách thành nhiều reducer rồi gộp lại vào root reducer và cung cấp cho store.
  + Action:
    - Là 1 object
    - Có 2 thuộc tính:
      * type (mô tả hành động là gì)
      * payload: chỉ sử dụng khi muốn state được tính toán dự trên giá trị lưu trong payload
  + Action creator:
    - Function để tạo ra action
    - Nhận vào giá trị cho payload
  + Dispatch (nằm ngoài store = “Tác động view”):
    - Là 1 function để cập nhật store
    - Nhận vào 1 action hoặc 1 function return ra action (Action creator)
    - Chỉ nhận 1 object đơn giản không bất đồng bộ, nếu có bất đồng bộ cần middleware
  + UseSelector: lấy ra state từ store

Axios: 

**Node JS**

NodeJS là một nền tảng được xây dựng trên V8 JavaScript Engine – trình thông dịch thực thi mã JavaScript, giúp xây dựng các ứng dụng web một cách đơn giản và dễ dàng mở rộng.

**Lý do nên học NodeJS là gì?**

* NodeJS được viết bằng JavaScript với cộng đồng người dùng lớn mạnh. Nếu bạn cần hỗ trợ gì về NodeJS, sẽ nhanh chóng có người hỗ trợ bạn.
* Tốc độ xử lý nhanh. Nhờ cơ chế xử lý bất đồng độ (non-blocking), NodeJS có thể xử lý hàng ngàn kết nối cùng lúc mà không gặp bất cứ khó khăn nào.
* Dễ dàng mở rộng. Nếu bạn có nhu cầu phát triển website thì tính năng dễ dàng mở rộng của NodeJS là một lợi thế cực kỳ quan trọng.
* Phần Core bên dưới của Nodejs được viết hầu hết bằng C++ nên cho tốc độ xử lý và hiệu năng khá cao.
* Nodejs tạo ra được các ứng dụng có tốc độ xử lý nhanh, realtime thời gian thực.
* Nodejs áp dụng cho các sản phẩm có lượng truy cập lớn, cần mở rộng nhanh, cần đổi mới công nghệ, hoặc tạo ra các dự án Startup nhanh nhất có thể.

Node JS là 1 nền tảng, hay nói cách khác nó là công nghệ hỗ trợ:

* Chạy JS độc lập với HTML, ứng dụng sau này có thể viết các ứng dụng không chỉ web.
* Nhiều thư viện thuận tiện trong quá trình xây dựng sản phẩm:
  + Realtime
  + Socket
  + File
  + Http
  + …
* Thư viện ở trên trong Node được gọi là “**Module**”, các thư viện sẽ được lưu trọng node\_modules, đó là lý do mỗi khi dùng “**npm i**” để cài thêm thư viện thì node\_modules sẽ có xuất hiện thư viện đó.
* Từ Module 2 chúng ta đã làm việc với nodeJS.
* Debug là công nghệ, tool hỗ trợ JS debug rất hay ho(Demo).
* NodeJS vs Browser:
  + Giống: Đều có JS engine – công nghệ hỗ trợ chạy file Javascript
  + Khác:
    - Browser có: Window, Document, local Storage,…
    - Node chạy JS trên Server, Browser chạy JS trên Client.
    - Node có “modules” – các thư viện hỗ trợ các chức năng chuyên biệt.
    - Browser có giao diện là HTML.
    - Node có thể có giao diện nếu có sử dụng modules hỗ trợ.

**Bất đồng bộ**

1. Đồng bộ, bất động bộ?
   1. Đồng bộ: là việc code chạy từ trên xuống dưới, tuần tự (*nêu ví dụ*).
   2. Bất đồng bộ: là việc các đoạn code có thể chạy song song, đồng thời. (*nêu ví dụ*)
   3. Ưu nhược điểm:

* Bất đồng bộ nhanh hơn nhưng sẽ khó kiểm soát các giá trị hơn.
* Một số thao tác bất đồng bộ hay được dùng:
  + SetInterval
  + Lấy dữ liệu từ 1 nơi khác (file, database, api)
* Điều gì xảy ra khi đoạn code B cần kết quả từ đoạn code A khi chúng đang chạy song song? (*nêu ví dụ*)

=> **LỖI!**

=> Lại để đồng bộ??? Thế thì bất đồng bộ sinh ra vô nghĩa!

Và chưa kể một vài thao tác từ thư viện ngoài bắt buộc là bất động bộ

(*ví dụ:* sử dụng dữ liệu từ bên thứ 3, cần chờ bên kia gửi dữ liệu xong mới làm tiếp được hoặc có trường hợp còn bị lỗi)

* + - * “XỬ LÝ BẤT ĐỘNG BỘ” (~~đồng bộ~~)

Bằng cách nào đó làm cho các thao tác chạy tuần tự nhưng chỉ khi cần thiết (đồng bộ tạm thời => *nêu ví dụ*) và bằng cách nào đó có các thao tác khác nếu lỗi.

* + - * 1. Promise: là một đối tượng để thực hiện bất đồng bộ, rất nhiều thư viện có những hàm có kết quả trả về là promise.

Đối tượng promise(state, result, then, catch, finally)

Tạo mới 1 promise: hàm tạo(resolve, reject):

Tạo ra đối tượng promise, trong hàm tạo chúng ta viết code đoạn code bất đồng bộ và chúng ta đặt ra các trường hợp cho đoạn code bất đồng bộ đó.

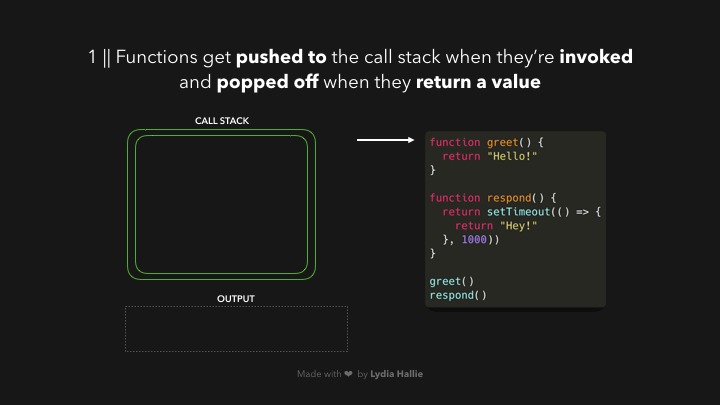
* + - * 1. Async Await

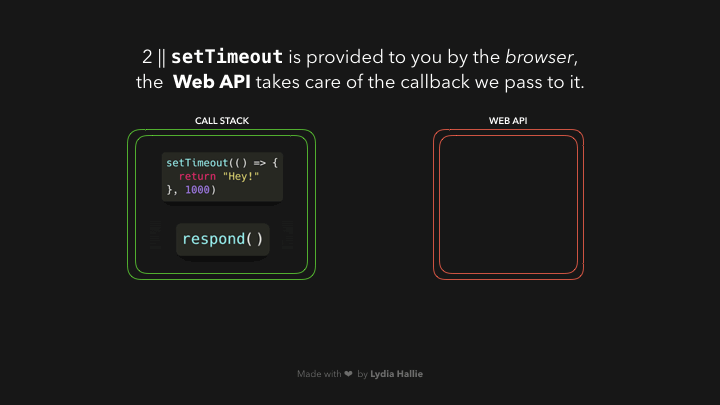
Event Loop

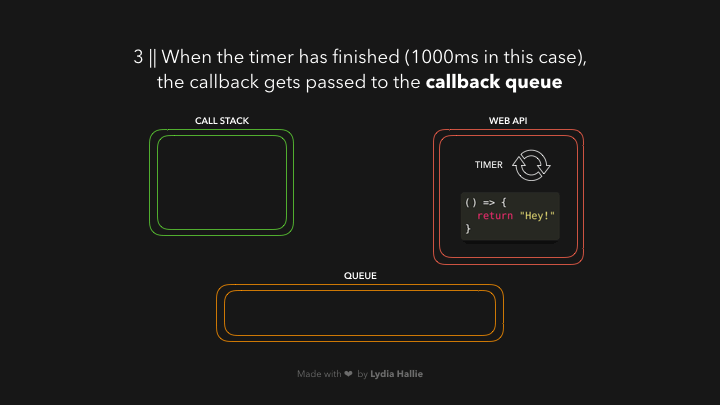
# Hiểu về cơ chế xử lý sự kiện Event Loop trong Javascript

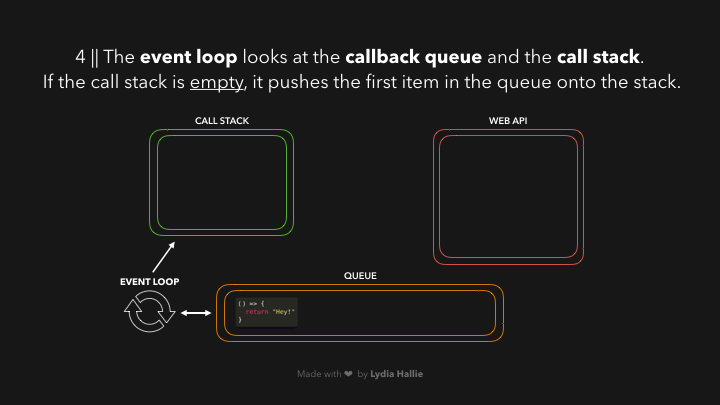
Bài đăng này đã không được cập nhật trong 2 năm

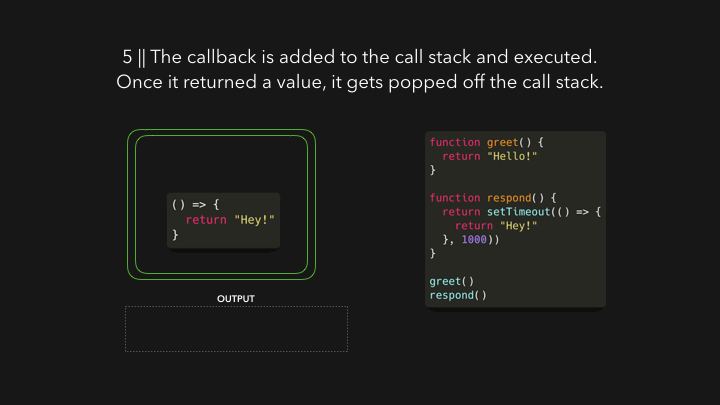
Xin chào mọi người, hôm nay mình quay lại với một bài viết về Event Loop một trong những khai niệm vẫn còn khá mơ hồ và khó hiểu đối với những người mới làm quen với Javascript. Trước tiên hay nói một chút về Javascript ra đời năm 1996 sau đó 1 năm thì tổ chức ECMA dựa trên Javascript đưa qua quy chuẩn cho các ngôn ngữ Script language đó là sự ra đời của ECMAScript. Để hiểu sâu hơn về lịch sử mình sẽ có 1 bài viết về nó hôm nay mình sẽ đi sâu vào cơ thế hoạt động và cách Javascript thực hiện các sự kiện lặp như thế nào nha.

Vậy Event Loop là gì và tại sao chúng ta phải hiểu về cách hoạt động của nó? Như mọi người đã biết JS hoạt động đơn luồng, chỉ một nhiệm vụ được thực thi trong 1 lúc. Nó sẽ không là vấn đề gì nhưng thử tưởng tượng 1 task chạy 30 giây, và bạn có đến 30 task cứ phải chờ như vậy là một vấn đề cực lớn trong chương trình. Đặc biệt trong thời đại công nghệ lên ngôi không một ai muốn phí phạm thời gian chờ đợi một ứng dụng không đáp ứng hiệu năng và không phản hồi kết quả. May mắn trình duyệt đưa cho chúng ta chức năng đó là Web API bao gồm DOM API và Http Request,… giúp chúng ta tạo nên các tác vụ chạy bất đồng bộ, non-blocking. Ok bây giờ bắt đầu đi sâu hơn vào cơ chế hoạt động nhé, khi một hàm được gọi trong JS, hàm đó được thêm vào 1 vùng nhớ được gọi là call stack. Call stack là một phần của JS Engine không phải của Browser nhé mọi người. Nói về Stack chắc mọi người cũng hiểu cơ chế FILO (first in last out). Sau khi hàm đã được đẩy vào call stack, nó trả về giá trị thực thi sau đó được lấy ra khỏi stack để đẩy hàm tiếp theo vào.

Việc phản hồi hay giá trị trả về như các hàm setTimeout được cung cấp bởi Web API. Các hàm này cho phép chúng ta thực hiện các hàm callback, các hàm callback này được đẩy vào vùng Web API là nơi chứa các hàm mà Web API cung cấp ở đây nó sẽ chờ thời gian được set ví dụ 1 giây sau đồng thời phẩn hồi lại với stack pop hàm setTimeout ra khỏi Call Stack.

Trong Web API, có một đồng hồ tính giờ được chạy ngay khi các hàm setTimeout, setInterval được đẩy vào, các hàm callback được truyền vào lúc này sẽ không nhảy qua Call Stack ngay mà nó được chuyển vào một hàng đợi Queue chờ được gọi lại. Trước giờ mọi người vẫn hình dung cứ xong thời gian setTimeout thì hàm callback được gọi ngay nhưng nó vẫn phải nằm trong hàm đợi đấy nhé đế lượt mới được thực thi nếu chúng ta xếp chồng setTimeout, setInterval đó là lý do các hàm này chúng ta chỉ sử dụng khi cần thiết.

Đây là phần khó hiểu nhất nè: hàm callback đã được đẩy vào quêu vậy khi nào nó được gọi nếu như đầu bài chúng ta đã nói hàm chỉ thực hiện trong vùng Call Stack? Đây là lúc Event Loop hoạt động, Event Loop có một nhiệm vụ duy nhất là đồng bộ Queue với Call Stack. Nếu Call Stack trống thì chúng ta gọi hàm trong Queue, Queue hoạt động theo cơ chế FIFO nên hàm nào nằm trong hàng đợi trước thì được gọi thực thi trong Call Stack trước. Mọi người lưu ý lúc này Call Stack trống thì các hàm trong Queue mới được Event Loop lấy lên và đẩy vào Call Stack để thực thi nhé.

Hàm callback sau khi đẩy vào Call Stack thì nó thực thì chờ trả về giá trị và lấy ra khỏi Call Stack cho việc thực hiện các hàm tiếp theo.

Ok chúng ta đã hiểu quy trình vây bây giờ hãy thử áp dụng một ví dụ nhé:

**Core Modules:** Node.js có nhiều mô-đun tích hợp là một phần của nền tảng và đi kèm với cài đặt Node.js. Các mô-đun này có thể được tải vào chương trình bằng cách sử dụng hàm require. <http>

**Local Modules:** Mô-đun cục bộ được tạo cục bộ trong ứng dụng Node.js của bạn. Hãy tạo một mô-đun tính toán đơn giản để tính toán các phép toán khác nhau. <Code mình viết ở M2>

**Third-party Modules:** là các mô-đun trực tuyến có sẵn bằng cách sử dụng Node Package Manager(NPM). Các mô-đun này có thể được cài đặt trong thư mục dự án. Một số mô-đun third-party phổ biến là mongoose, express, angular, và react <readlinesync>

/: trang login

Login nhuanh, 123456 => /admin: trang admin

Login lichcau, 123456 => /user: trang user

Tren menu:

* Ấn home thì hiện: list user ở phần dưới menu
* Ấn add thì hiện: form add ở phần dưới menu

Trang User: hiện list

Trang Admin:

* Hiện list
* Thêm
* Sửa
* Xóa