**1. Browser xử lý JS như nào ?**

=> thông qua JS engine, js engine nổi tiếng nhất là V8 của chrome

**2. Cơ chế hoisting là gì ?**

=> Cơ chế hoisting là cơ chế đưa các phần khai báo biến và hàm lên các dòng đầu tiên của Scope chứa nó. Tuy nhiên, chỉ di chuyển các phần khai báo, phần gán giá trị vẫn giữ nguyên vị trí cũ.

**3. Callback function là gì?**

=> Callback function là một function A được truyền vào một function B dưới dạng một tham số. Tại một thời điểm nào đó tùy thuộc vào cách function B được xây dựng mà function A sẽ được function B gọi để thực thi.

*example:*

*const sayHello = (name, callback) => {*

*const msg = "Hello, " + name;*

*return callback(msg);*

*};*

*const useCallBack = sayHello("Khoa", (msg) => {*

*return msg;*

*});*

*console.log(useCallBack);*

**4. Em hiểu gì về closure?**

=> Closure là một hàm được viết lồng vào bên trong một hàm khác (hàm cha) mà nó có thể sử dụng và thay đổi các tham số của hàm cha lẫn của chính nó.

Example:

const increase = () => {

let x = 0;

const increaseInner = () => ++x;

return increaseInner;

};

console.log(increase()()); *// output: 1*

console.log(increase()()); *// output still: 1*

const useClosure = increase();

console.log(useClosure()); *// output: 1*

console.log(useClosure()); *// output: 2 (voila !?)*

**4.1. Có mấy cách khai báo function trong js. Khác nhau là gì?**

=> Có 2 cách để khai báo function:

* Declaration function (DF):

function logSomething() {

*// do something*

}

* Expression function (EF):

const logSomething = function () {

*// do something*

};

DF bị ảnh hưởng bởi cơ chế hoisting, còn EF thì không. Vì vậy, nếu gọi hàm viết dưới dạng EF chạy trước khi khai báo sẽ bị lỗi.

**5. Em hiểu gì về event loop**

=>

**6.Sử dụng gì để call api, ngoài axios?**

=> fetch và AJAX

**7.Cho câu sau, thứ tự in ra đúng là gì ?**

setTimeout(() => console.log(‘a’),200)

setTimeout(() => console.log(‘b’),0)

console.log(‘c’)

=> Explain: hàm setTimeout là hàm bất đồng bộ, sau khi đọc đến đây. JS Engine sẽ tạm thời đưa vào callback queue để chờ sau khi lệnh console.log(“c”) chạy xong, setTimeout nào có thời gian chờ ngắn hơn, sẽ được xếp trước vào callback queue. Vì vậy, kết quả lần lượt là: c

b

a

**8. Khác nhau giữa let,var và const**

=> var bị ảnh hưởng bởi cơ chế hoisting và hoạt động ở mọi nơi trong global scope

=> let không bị ảnh hưởng bởi hoisting, có thể tái gán giá trị và hoạt động trong block scope.

=> const không bị ảnh hưởng bởi hoisting, không thể tái gán giá trị và hoạt động trong block scope.

**9. Có cách nào thay đổi được giá trị của const ko? Tại sao?**

=> Có thể nhưng cũng không thể, nếu trường hợp giá trị của const là kiểu tham trị thì không thể thay đổi nhưng ngược lại, nếu giá trị của const là kiểu tham chiếu (array, object) thì chúng ta có thể thêm vào hoặc bớt đi phần tử bên trong. Nguyên nhân là khi chúng ta thay đổi giá trị bên trong array hay object của một biến const, việc đó chỉ góp phần thay đổi vùng nhớ của biến chứ không thay đổi giá trị của biến, bản chất biến const ta đang đề cập vẫn là một array hay một object.

**10. Arrow function khác gì với function thường ?**

=> Arrow function không làm thay đổi ngữ cảnh của con trỏ **this** và không thể sử dụng agruments

**11. ES6 offer các tính năng mới gì cho mình sử dụng, kể tên?**

=> Arrow function, classes, promise, let & const, template string.

**12. Kể tên các higher order function trong ES6**

=> reduce, map, filter, foreach

**13.Tại sao lại gọi là High order function ?**

=> function chứa tham số là một callback, hoặc function trả về một function khác (closure)

**14. Spread operator dùng khi nào ?**

=> Khi cần clone, merge array/object.

15. 4 tinh chất của đối tượng?

16. Nếu có 3 promise đang chạy, nếu muốn chờ xong cả 3 mới làm thì dùng cách nào ? => promise.all

17.Bubble sort, quick sort chạy như thế nào?

17.Dự án trước làm về gì ? Trong dự án đó có gì e cảm nhận là khó nhất

**------REACT-------**

**1. E hiểu gì về state , props. Giống và khác**

=> **Props** là nơi lưu trữ dữ liệu được lưu truyền giữa các Component, giúp chúng tương tác và trao đổi dữ liệu với nhau. Được truyền vào Component cha và có thể truy cập được ở các Component con.

=> **State** cũng lưu trữ dữ liệu như Props, nhưng cách hoạt động hoàn toàn khác. State là một thành phần của Component và chỉ có thể bị thay đổi bởi chính Component chứa nó.

=> **Giống:** Cả state và props đều được sử dụng để lưu trữ dữ liệu trong Components.

=> **Khác:**

|  |  |
| --- | --- |
| **State** | **Props** |
| State là dữ liệu có thể bị thay đổi, nên sẽ được dùng để theo dõi sự thay đổi bên trong component và re-render Component. | Được thiết lập từ Component cha, truyền vào các Component con và cố định trong suốt vòng đời. Vì vậy, props được dùng trong các trường hợp cần trao đổi dữ liệu giữa các Component. |

**2. React là thư viện hay framework? Thư viện và framework khác nhau ở điểm nào ?**

tham khảo :<https://viblo.asia/p/su-khac-nhau-giua-framework-va-library-maGK7D6DZj2>

**3.React là oneway binding hay two way binding?**

=> Oneway vì khi đổi model thì view ko đổi, phải thay đổi state

**4. Em hiểu gì về lifecycle của react?**

=>

**5. Khi thay đổi state,component render lại, có cách nào để khi thay đổi state mà component ko cần render lai ko?**

=> dùng shouldComponentUpdate => return false

**6. Sử dụng react hook thì làm sao để quản lý render?**

=> Sử dụng useEffect., useCallback, useMemo

**7. Em đã từng sử dụng các thư viện quản lý state nào ?**

=> Redux, mobx.

**8. Cơ chế so sánh của redux?**

=> Shallow equality checking (hay so sánh tham chiếu) là phép so sánh chỉ cần kiểm tra xem 2 biến được so sánh có cùng tham chiếu đến 1 object hay không, ngược lại với nó là thuật ngữ deep equality checking (hay so sánh giá trị) phải kiểm tra mỗi giá trị của từng thuộc tính ở cả 2 objects. Như vậy, chỉ cần 2 biến cùng tham chiếu đến 1 object thì Shallow equality checking sẽ trả về giá trị là true, ngược lại, nếu chúng không cùng tham chiếu đến 1 object thì Shallow equality checking sẽ trả về giá trị là false (dù cho 2 biến có giống hệt nhau đi nữa mà không cùng tham chiếu đến 1 object thì Shallow equality checking vẫn sẽ trả về giá trị là false)

=> Do đó shallow equality check đơn giản hơn và nhanh hơn deep equality checking => Redux sử dụng shallow equality check để cải thiện performance Ngoài ra, điểm đặc biệt của shallow equality checking đó là: nó không thể phát hiện được rằng 1 object có bị thay đổi hay không sau khi được xử lý bởi 1 function nếu function đó thay đổi chính object được truyền vào cho nó

**9. Công dụng của middleware là gì?**

=> Là cầu nối giữa DB và App, xử lí các request/response từ actions và handle nếu có lỗi xảy ra.

**10.componentWillUnmount chạy trước hay sau khi component bị huỷ?**

=> Chạy trước khi Component bị hủy

**11.PureComponent và memo dùng để làm gì?**

=> Để ngăn việc re-render không cần thiết.

**12. Hai cái trên sử dụng kiểu so sánh nào ? Shallow hay deep ?**

=> Shallow

**13. Làm sao để so sánh deep comparation?**

=> sử dụng useCallBack

**14. Em đã sử dụng context API bao giờ chưa? context API truyền dữ liệu như nào? có thể truyền giữa các component anh em được không?**

=>

**15. Một số cách để tối ưu ứng dụng reactjs**

=> Sử dụng useCallBack, memo, useMemo để ngăn việc re-render không đáng có.

=> Tối ưu dung lượng ảnh, hạn chế sử dụng svg.

=> sử dụng lazyload.

**16. Cho một code react, tìm lỗi sai?**

**=>**

**17. HOcs là gì? thường trong dự án e sử dụng khi nào?**

=> Sử dụng khi cần wrap các Screen trong một Layout chung như có chung Header, Foot, Side drawer.

**18. Khi sử dụng các hook của react như useCallback, useEffect, ta hay truyền 2 tham số, callback function và 1 mảng dependencies, sự khác biệt của truyền mảng rỗng và không truyền là gì?**

=> Truyền mảng rỗng tương ứng với việc hook chỉ chạy một lần và không bao giờ chạy lại trong suốt vòng đời của Component.

19.Ưu và nhược điểm của single page application so với multiple page application?

=> SPA không cần phải load lại trang nhiều lần, tăng trải nghiệm người dùng. Nhược điểm: Giảm SEO.

20. Cho lần lượt các api 1 2 3 4 5 => muốn api 1 2 3 4 cùng thực thi và thực thi xong thì api 5 mới được gọi thì dùng Promise.all

var p1 = new Promise((resolve, reject) => {

setTimeout(resolve, 1000, "one");

});

var p2 = new Promise((resolve, reject) => {

setTimeout(resolve, 2000, "two");

});

var p3 = new Promise((resolve, reject) => {

setTimeout(resolve, 3000, "three");

});

var p4 = new Promise((resolve, reject) => {

setTimeout(resolve, 4000, "four");

});

Promise.all([p1, p2, p3, p4]).then(values => { //<= Đợi 4 promise kia cùng chạy xong thì mới thực hiện gọi api 5 thực thi hoặc trong trường hợp có bấ kỳ api nào bị lỗi cũng sẽ đừng

console.log(values);

console.log('request api 5');

}, reason => {

console.log(reason)

});

21. Pure component là gì

22. So sánh sự khác biệt giữa state và props