* **Hooks vs Class: Hooks là 1 function reusable (hàm tái sử dụng)**

\* nguyên tắc khi sử dụng hook:

+sử dụng bên trong cá function component. Không nên sử dụng trong các class component

+hook không thể được gọi bên trong các điều kiện, vòng lặp, hàm lòng nhau. Vd: useState có thể bị thay đổi khi gọi bên trong hàm if. Do state ban đầu nhận vào là 1 giá trị cố định.

+hook không thể có điều kiện

* **useSate():**

useState() sẽ trả về 1 mảng có 2 giá trị là state và setState. State: là giá trị ban đầu, và setState là giá trị để cập nhật lại biến state ( \* useState có thể nhận vào 1 object {}, 1 mảng [], 1 string bất kỳ)

* **Strict Mode (chế độ nghiêm ngặc)**

Giúp code javascript an toàn hơn, tránh được khỏi những lỗi vô ý xảy ra

\*cách sử dụng: bắt buộc phải sử dụng đầu tiên.

+”use strict” vào đầu file.js

+”use strict” vào ngay sau thẻ mở <script> “” </script> phạm vi nhỏ hơn dùng ở đầu file

+”use strict” vào đầu phạm vi hàm

* **States and Shallow Merge:**

State: lưu trữ giá trị của các biến trong 1 component

* **map() + key attribute**

.map() giống như 1 vòng lặp. Tạo ra 1 mảng mới và thường được sử dụng trong JSX (tạo ra danh sách các phần tử). Lời khuyên từ React là nên dùng thuốc tính key={} cho thẻ cha (key attribure).

Key attribute giúp cho map() nhận biết được các phần và hoạt động được chính xác hơn. Thông thường thì ‘key’ phải là giá trị duy nhất và thường là ID

* **Promises:**

Sinh ra để xử lý các nguyên tắc bất đồng bộ. (vd: thông thường 1 đoạn code sẽ được thực thi từ trên xuống, tuy nhiên có thể gặp nguyên tắc bất đồng bộ khi hàm đó được gọi trước và thực thi sau; hàm được gọi sau và thực thi trước)   
Trước khi sinh ra Promises thì chúng ta sử dụng callback (tuy nhiên callback có thể xảy ra ‘callback hell’) [có thể tìm hiểu thêm về callback hell]

\*Để tạo ra được ‘promise’ chúng ta sẽ sư dụng toán tử new + Promise(). Trong ‘constructor chúng ta sẽ truyền vào 1 **‘excutor’** function. => trong excutor function sẽ nhận vào 2 tham số: function(**resole, reject**). **Resole:** được gọi khi thao tác sử lý thành công, ngược lại nếu thất bại thị sẽ gọi **reject.**

**Promise**

**.then(‘** nhận vào 1 call back và sẽ được thực thi khi promise **resol’)**

**.catch(‘** nhận vào 1 callback và sẽ được thực thi khi promise **reject’)**

**.finally(‘** nhận vào 1 callback và sẽ được thực thi ngay cả khi **resol** **hoặc** **reject’)**

* **Render & Rerender**

Render: mỗi lần có sự thay đổi trong state thì render sẽ được thực thi và cập nhật lại giao diện người dùng (JSX)

Rerender: là trong 1 function component sẽ có return(). Thì rerender là thằng return sẽ được render lại. (state, props)

* **Funtional & Class Component**
* **Component life cycle**
* **Pure & Impure Function**
* **useEffect: (tuân thủ theo dependence của useEffect để giảm thiểu việc thực hiện side Effect không mong muốn)**

useEffect(function callback() { side-effect }, [dependence]) | **usually:** useEffect(() => { side-effect }, [dependence])

nói chung khi sử dụng useEffect phải truyền vào dependence. Khi khởi chạy trang web thì useEffect được gọi và chỉ khi dependence thay đổi thì useEffect mới tiếp tục được gọi.

* **DOM vs Virtual DOM (document object model)**

DOM: các element được trình duyệt đọc và tạo ra 1 mô hình, thì mô hình đó được gọi là DOM. Trong DOM có 3 thành phần: element(các thẻ html, h,p,…), attribute (thuộc tính của các thẻ vd: className, value,…), text. Nodes là những phần giao nhau trong Cây DOM, củng được xem như là element, attribute, text.

Vitual DOM : cây Dom ảo và chỉ react có thể thấy được. Nó sẽ tạo ra 1 cây Dom ảo để so sánh với cây DOM thật để nhận biết element nào đang thay đổi và nó chỉ cập nhật element đó thôi (có thể tìm hiểu thêm: react fiber architecture). Từ đó giúp tăng tốc độ của trang web.

* **ReactDOM**

**Nâng cao**

Advanced   
Authentication  
  
State management   
Context  
Redux  
Jotai  
Reducer  
Design Pattern   
Observer pattern  
Styling   
Styled-component  
css-in-js