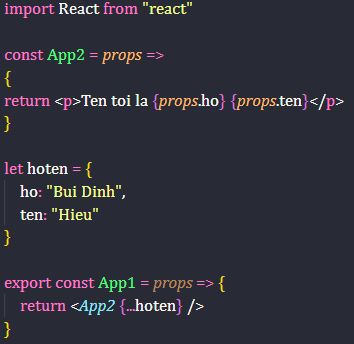
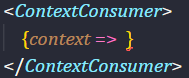
Trong bootstrap đa số (có thể là tất cả) các class có chiều rộng màn hình (sm, md, lg, xl) thì tính chất của nó là hoạt động khi màn hình lớn hơn hoặc bằng giá trị tương ứng. Ví dụ với class **px-md-3** thì nó hoạt động khi chiều rộng màn hình lớn hơn hoặc bằng 768px.

Ta có thể truyền props vào 1 component theo dạng đối tượng nhưng không truyền theo cách bình thường truyền đối tượng vào mà truyền theo cách như sau:

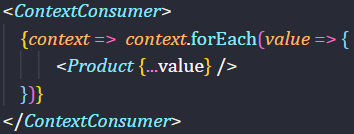


Đầu tiên ta phải khởi tạo đối tượng ở bên ngoài trước, sau đó sử dụng code **{…hoten}** để truyền các props vào component con đó. Khi đó chương trình sẽ hiểu các props **ho=“Bui Dinh” ten=“Hieu”**. Tuy nhiên nếu ta truyền vào bằng cách đưa cả đối tượng vào hoặc viết đối tượng ngay bên trong component thì gây ra lỗi.

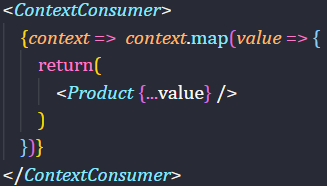
Khi tương tác dữ liệu sử dụng Consumer, giả sử có ví dụ sau:



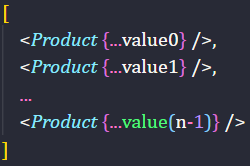
Phần đằng sau **context** phải là dữ liệu dạng mảng, dạng JSX, dạng chuỗi hoặc dạng số (nói chung nó là 1 dạng có giá trị) và trong context này ta không được dùng hàm forEach để xử lý dữ liệu vì thực chất hàm forEach là hàm xử lý dữ liệu chứ không phải hàm trả về giá trị dữ liệu. Ta có thể làm rõ hơn qua ví dụ sau:



Khi mảng context được xử lý qua forEach thì hàm thực thi các đoạn code bên dưới với lần lượt các phần tử trong mảng. Do là hàm thực thi code nên **<Product {…value} />** chẳng có ý nghĩa gì cho việc hiển thị nên chương trình báo lỗi và thay vào đó ta phải sử dụng hàm map do hàm này trả về giá trị mới. Thay forEach bằng map ta được:

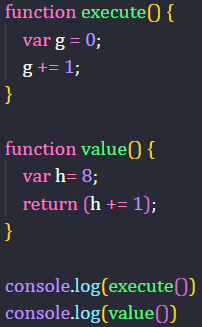


Chương trình đó tương đương với 1 mảng mới giả sử có tên là test bằng:



Đó chính là lí do vì sao hàm map lại thường được sử dụng trong JSX hơn forEach.

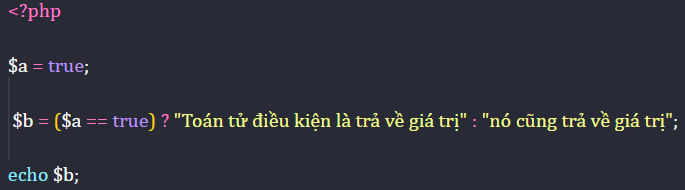
Ta có thể thấy rõ hơn sự khác nhau đối với hàm trả về giá trị và hàm thực thi như sau:



Ta thấy function execute() là hàm thực thi còn function value() là hàm trả về giá trị. Khi đó nếu ta console.log(execute()) thì chương trình gây ra lỗi còn console.log(value()) thì chương trình hiển thị ra kết quả bằng 9. Hàm trả về giá trị thì nó vừa thực thi vừa trả về giá trị còn hàm thực thi thì chỉ thực thi không thôi.

Đối với hàm pop và 1 số hàm xử lý mảng thì đều thuộc hàm trả về giá trị vì khi cho var a = arr.pop() thì a đều có giá trị.

Toán tử điều kiện trong cả JavaScript và PHP đều thuộc loại hàm trả về giá trị. Cụ thể như sau:



$b sẽ có giá trị là:



Để không bị nhầm lẫn khi truy cập các đường link trong reactjs thì route trang chủ ta nên để dấu “/”. Ví dụ:



Tránh để không có dấu “/” như sau:



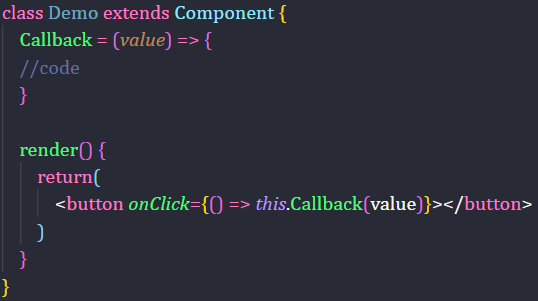
Bởi vì không có dấu “/” thì khi ta truy cập vào đường link bất kì ví dụ ta truy cập vào đường link **localhost:3000/detail** thì chương trình vẫn chỉ hiện ra component **Index** mặc dù ta đã **exact** route đó.

Trong 1 thẻ div được định dạng bởi class container khi ta chia grid trong container đó thì mỗi cột có giá trị bằng widthcontainer/12 và mỗi phần tử chiếm số col đó không bao gồm margin. Ví dụ 1 phần tử chiếm 2 col thì chiều rộng của nó là widthcontainer/6 và phần chiều rộng này không bao gồm margin của phần tử đó.

Khi sử dụng các hàm trong component được tạo bởi class, đối với những hàm arrow function (biểu thức hàm) thì ta không cần phải bind nó vào trong hàm constructor còn đối với hàm không phải arrow (khai báo hàm) thì cần phải bind.

Hàm callback là hàm được thực hiện khi có 1 sự kiện diễn ra và sự kiện đó mang 1 đặc trưng nhất định.

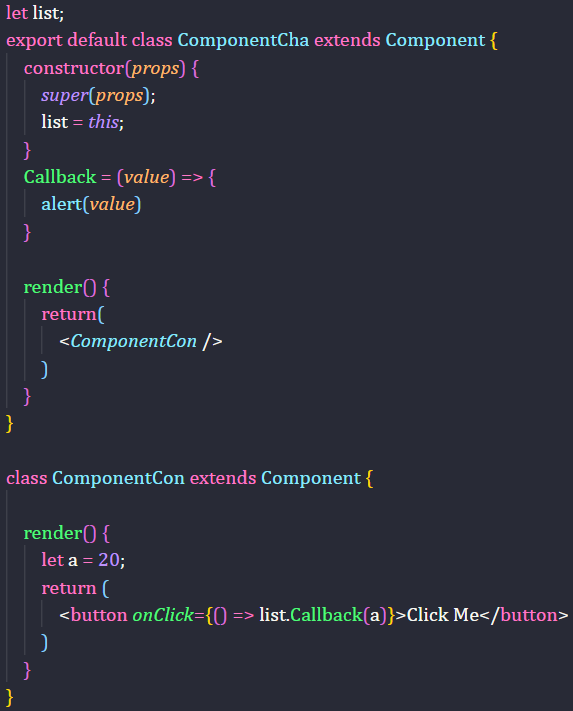
Thông thường hàm callback được sử dụng trong 1 component hoặc từ các component khác nhau nhưng có liên quan đến nhau. Ví dụ trong 1 component:



Khi khác component thì được truyền sử dụng hàm static như sau:



Hoặc ta có thể dùng gán đối tượng:



Trong JSX khi gọi hàm không sử dụng cặp dấu () ví dụ đúng:

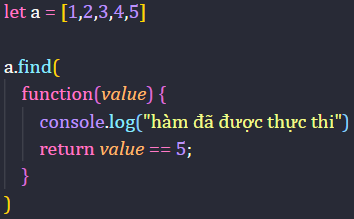


Ví dụ sai:



Hàm số gần giống như 1 biến, nó chỉ khác ở chỗ là nó có thể thực thi các dòng lệnh.

Khi khởi tạo 1 hàm bên trong 1 hàm được ngôn ngữ dựng sẵn như hàm setState(), find(), … khi chạy hàm lớn thì hàm nhỏ được khởi tạo đó cũng thực thi luôn không. Ví dụ:

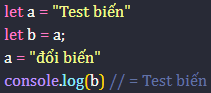


Mở console.log() lên thì hàm đã được thực thi.

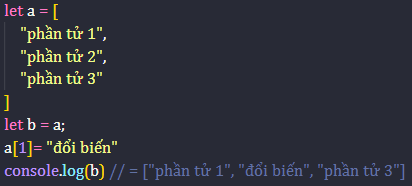
Còn đối với hàm tự khởi tạo bên trong hàm tự khởi tạo khác, khi thực thi hàm lớn thì hàm nhỏ không thực thi.

Khi 1 hàm được thực thi thì nó mới nhận được giá trị của hàm đó và thực thi các đoạn code bên trong.

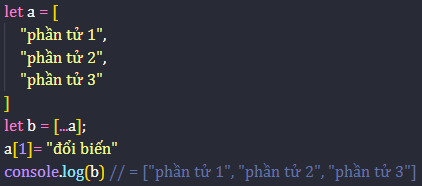
Trong JS khi các biến là number, string, boolean, … và đặt biến khác bằng biến đó thì biến khác kia là biến tham trị. Ví dụ:



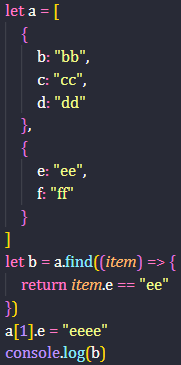
Nếu các biến là array, object và đặt biến khác bằng biến đó thì biến khác kia là biến tham chiếu. Ví dụ:



Để chuyển đổi những biến này thành tham trị thì ta làm như sau:

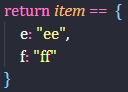


Cho ví dụ đoạn code sau:



Thì chương trình biên dịch từ trên xuống mà khi thực hiện xong hàm find thì trả về giá trị thỏa mãn cho b dạng tham chiếu về nơi sản xuất. Sau đó thay đổi giá trị của a[1].e thành “eeee” nhưng hàm find không phải thực hiện lại lần nữa nếu không kết quả sẽ undefined.

Khi sử dụng hàm find trên ta mà sử dụng phép so sánh:



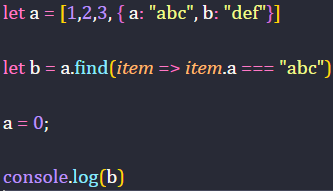
Thì chương trình đang so sánh tham chiếu của item với tham chiếu của đối tượng bên phải do tính chất array và object của JS. Tham chiếu của item là các phần tử trong mảng a còn tham chiếu của object là ngay tại toán tử so sánh. Do đó kết quả là false.

Chú ý: Ngay cả trường hợp sau kết quả vẫn ra false:



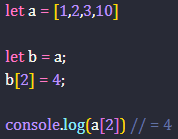
Bởi vì tham chiếu của 2 toán tử so sánh đều tại vị trí nó được định nghĩa. Do vậy kết quả vẫn ra false. Để tránh xảy ra lỗi với các phép so sánh ta nên chuyển sang so sánh các phần tử bên trong nó do các phần tử này là dạng tham trị.

Các hàm xử lý mảng tìm ra phần tử thì phần tử tìm được đó được truyền vào biến dưới dạng tham trị. Ví dụ:



Ta thấy phần tử thỏa mãn hàm find là đối tượng cuối cùng trong mảng. Nếu nó mà dạng tham chiếu thì giá trị của nó được trỏ đến phần tử a[3] và khi ta thay đổi a mà kết quả b không thay đổi.

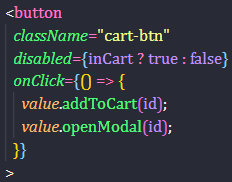
Trường hợp sau:



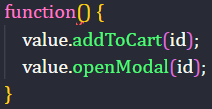
Khi thay đổi giá trị các phần tử trong b thì phần tử trong a cũng thay đổi do cùng chỉ đến phần tử thứ 3 (tương đương index = 2).

Bất cứ thẻ nào cũng có thể onclick được không nhất thiết phải là button.

Cho code sau:



Nếu ta viết hàm onclick như vậy khi click vào thì 2 hàm trên sẽ được thực thi. Vì nếu ta chuyển arrow function thành function bình thường thì hàm bên trong onClick trở thành:

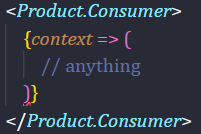


Khi click vào sự kiện đó thì hàm này được thực thi và thực thi các hàm bên trong.

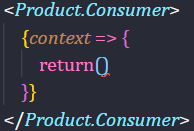
Ta có thể import bootstrap và fontawesome ở App.js mà không cần import ở các component con vì ở các component con dù không được import thì nó cũng chẳng ảnh hưởng đến code và khi import vào App component thì nó mới nhận được các thuộc tính của các class do bootstrap và fontawesome tạo nên.

Đối với các import khác như React, Component và các component khác ta phải import vì nếu thiếu import sẽ không tạo nên được component cha mà gây ra lỗi.

Trường hợp sử dụng Consumer mà nhiều thành phần cần xử lý thì ta sử dụng ngoặc () nếu không sử dụng return và sử dụng {} nếu sau { là return() để gom phần tử:



Hoặc:



Nó cũng tương tự như tính chất của arrow function. Đối với cách viết 1 thì không cần dùng return vì phần bên trong được gom trong dấu ngoặc () nên chương trình hiểu là return phần được gom đó. Đối với cách viết 2 thì sử dụng {} nên phải dùng return.

Trong styled component khi định dạng 1 thẻ nào đó thì những class bên trong cặp thẻ đó mới có thể được định dạng trong styled đó còn class ứng với thẻ đó không thể định dạng được mà phải định dạng trực tiếp.

1 thành phần được định dạng position: fixed thì nó được chuyển thành inline-block. Nếu ta định dạng cho nó cách top, bottom, left, right bằng 1 giá trị nào đó thì thành phần đó sẽ cách các lề tương ứng với giá trị tương ứng. Đây là cách để xác định chiều rộng và chiều cao của thành phần được fixed đó.