How React.memo works?

In this lesson, we'll discuss how React.memo works.

WE'LL COVER THE FOLLOWING

`React.memo`

`React.memo` is the perfect replacement for the class's `PureComponent`. All you do is wrap your functional component in the `React.memo` function call and you get the exact behavior `PureComponent` gives.

Here's a quick example:

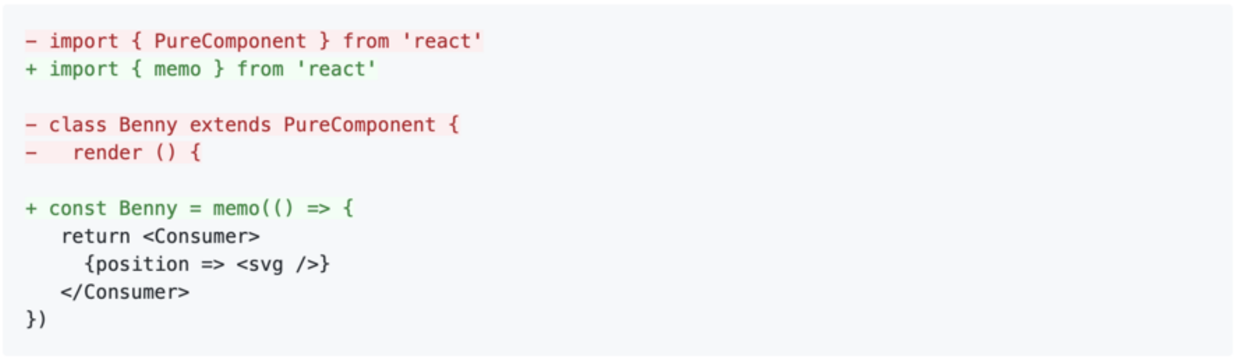
```
// before
import React from 'react'
function MyComponent ({name}) {
    return ( <div>
        Hello {name}.
     </div>
    )
}
export default MyComponent
// after
import React, { memo } from 'react'
export default  memo(function MyComponent ({name}) {
    return ( <div>
        Hello {name}.
     </div>
    )
})
```

So simple, it couldn't get any easier.

Technically, `React.memo` is a higher-order function that takes in a functional component and returns a new memoized component.

So, if props do not change, `react` will skip rendering the component and just use the previously memoized value.

With this newfound information, John refactors the functional component, `Benny`, to use `React.memo`.



```diff
- import { PureComponent } from 'react'
+ import { memo } from 'react'

- class Benny extends PureComponent {
-   render () {

+ const Benny = memo(() => {
    return <Consumer>
      {position => <svg />}
    </Consumer>
})
```

In the next lesson, we'll discuss how to handle a component when nested props are used.