React 스터디
8장 Hook

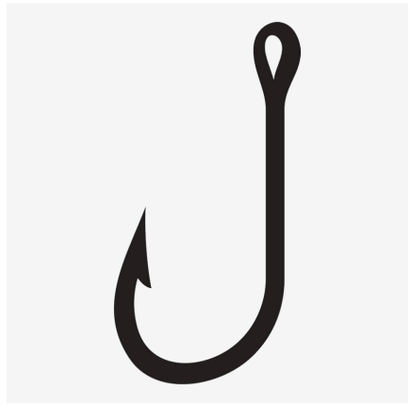
2020 / 03 / 12 목요일
이 상 아

Why we use HOOKS?

Hooks and FC

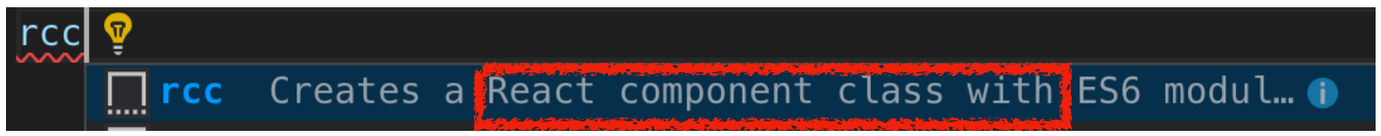
- **Hooks !== Functional Component**
- Functional Component는 예전부터 존재했던 개념이었음
  - 상태관리를 할 수 없었음
  - 때문에 단순하게 component를 return 하는데만 사용했었음

Hooks and FC

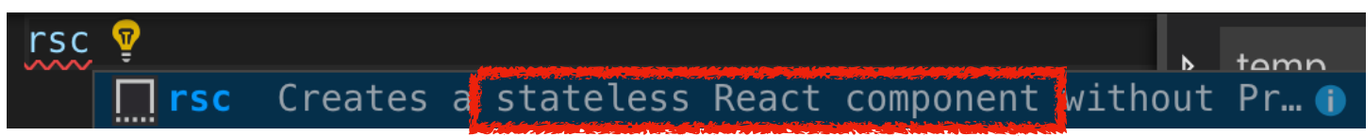


- **상태관리**를 할 수 있도록 도와주는 기능을 **hook** 해서 사용 할 수 있음
    - useState, useEffect 등 …

CC, FC




```
import React, { Component } from 'react';

class temp extends Component {
    render() {
        return (
            <div>

            </div>
        );
    }
}

export default temp;
```




```
import React from 'react';

const temp = () => {
    return (
        <div>

        </div>
    );
};

export default temp;
```

Why we need HOOKS?

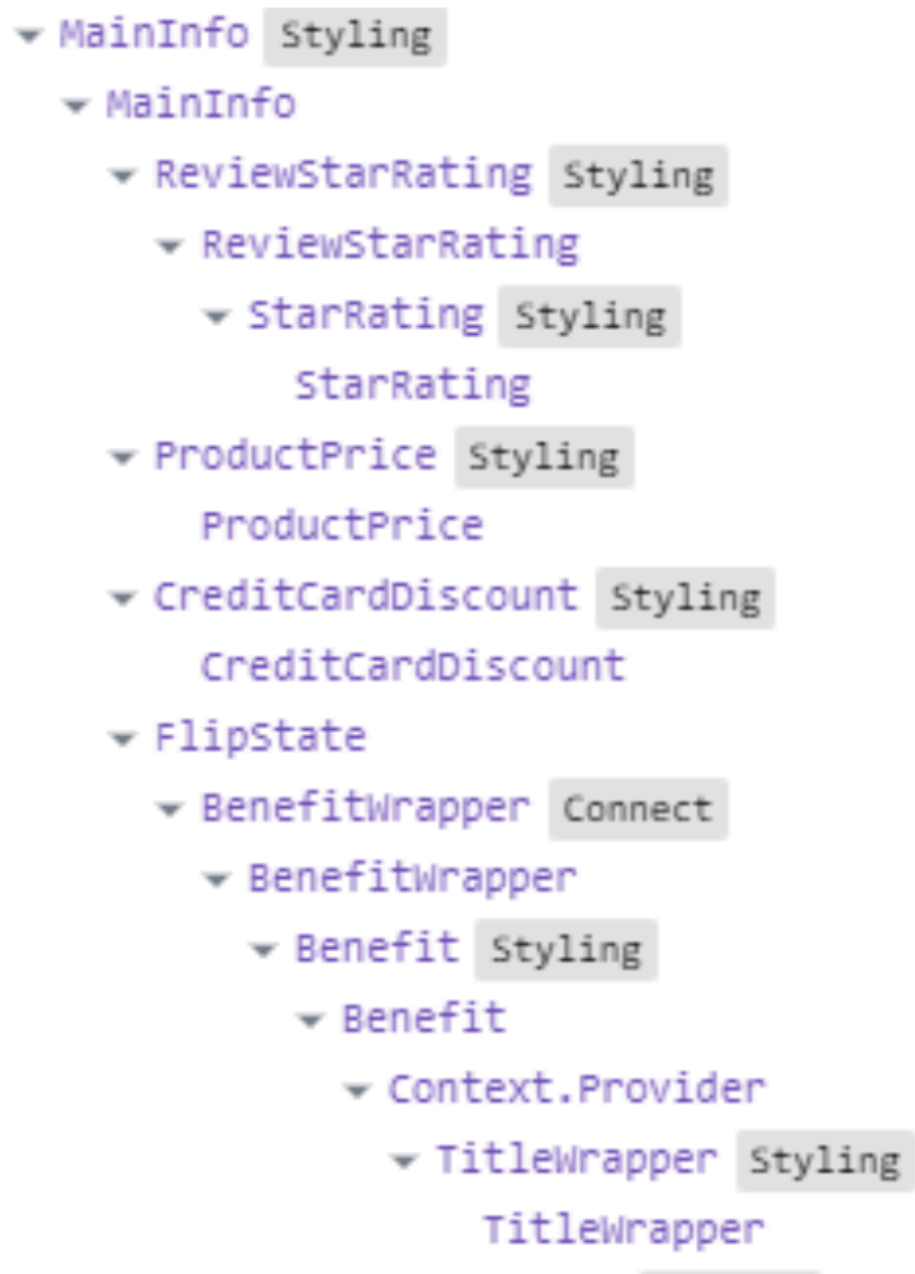
- 코드를 간결하게 만들 수 있음
- HOC (High Order Component)
- Lifecycle method 간 logic 중복

HOC

(High Order Component)

```
function styling(...styles) {
  return function wrapStyling(ComposedComponent) {
    class Styling extends React.Component {
      render() {
        return <ComposedComponent {...this.props} />
      }
    }
    return hoistStatics(Styling, ComposedComponent)
  }
}
```

```
export default styling(s)(Home)
```



출처: https://www.slideshare.net/ShoppingNAVER/react-hooks-177289990

이상아님이 **링크**를 공유했습니다.
13시간

기존 CC와 hook을 도입한 FC의 차이점에 대해서 공부하고있습니다. 이 문장을 봤는데 그림을 봐도 감이 잘 오지 않아 질문드립니다.

React 애플리케이션을 본다면, providers, consumers, HoC, render props 등 래퍼 지옥을 볼 가능성이 높다 (출처: https://medium.com/.../react-hooks-%EC%82%AC%EC%9A%A9%EC%9D%B...)

provider, consumer, HoC, render props에 대해 잘 모르는터라 이해하기가 어렵습니다.

혹시 이에 대해 쉽게 설명된 자료나 내용을 아시는 분이 있으시다면 알려주시면 감사하겠습니다.

Tlfauddksgksekrh Tlqkffhaemfdk 문장 자체는 별 의미 없는 내용입니다. 레퍼지옥이라는건 컴포넌트 안에 컴포넌트를 추가하다보면 트리구조가 깊어지기때문에 나타나는 자연스러운 현상이나 복잡한 어플리케이션일 수록 깊이가 더 깊어질 뿐입니다. 특히 내가 만든 컴포넌트 외에 외부 라이브러리로 감싸진 컴포넌트인 경우에 더 심해지기 쉽고요.

좋아요 · 답글 달기 · 13시간 · 2

서재원 ☕ 저는 그냥 각 개념들이 어디에서 나오는지만 알려드리도록 하겠습니다.

Provider/Consumer - react의 context api를 사용할 때 나오는 개념입니다. (provider는 context의 값을 제공하기 위한 컴포넌트, consumer는 그 값을 읽어들이기 위한 컴포넌트입니다)
HoC - High-order component의 약자로, 컴포넌트를 다루는 함수를 말합니다.(react-redux의 connect, styled-component의 styled)
render props - render 라는 프롭으로 컴포넌트를 전달하는, 랜더링 로직을 공유하기 위한 일종의 기술입니다.

좋아요 · 답글 달기 · 10시간 · 수정됨 · 2

Lifecycle method 간 logic 중복

- 1) Lifecycle feature 간 로직 중복

class component

```
class PhotoVideoList extends React.Component {
fetchPhotoVideoItems = async () => {}

async componentDidMount() {
  await this.fetchPhotoVideoItems()
}

async componentDidUpdate(prevProps) {
  if (this.props.photoVideoReviewIds !==
prevProps.photoVideoReviewIds) {
    await this.fetchPhotoVideoItems()
  }
}
}
```

functional component

```
const PhotoVideoList : React.FC => {


useEffect(() => {
  const fetchPhotoVideoItems = async () => {}
  fetchPhotoVideoItems()
}, [photoVideoReviewIds])
}
```

출처: https://www.slideshare.net/ShoppingNAVER/react-hooks-177289990

Lifecycle method 간 logic 중복

- 2) 1개 lifecycle feature에 lifecycle 에 수행되는 로직 몽땅

class component

```
class EventDetail extends React.Component {
componentDidMount() {
   setInitApplyProducts({})
   moveScrollByHistory()
   setApplyProducts({})
}

componentDidUpdate(prevProps) {
   if (prevProps.applyProductsState !==
this.props.applyProductsState) {
   moveScrollByHistory()
   setApplyProducts({})
   }
}
```

functional component

```
const EventDetail : React.FC => {
useEffect(() => {
  setInitApplyProducts({})
}, [])

useEffect(() => {
  setApplyProducts({})
}, [applyProductsState])

useEffect(() => {
  moveScrollByHistory()
}, [applyProductsState])
```

출처: https://www.slideshare.net/ShoppingNAVER/react-hooks-177289990

Lifecycle method 간 logic 중복

- 3) 관련 있는 로직이 다른 lifecycle feature 에 위치

class component

```
class FloatingBanner extends React.Component {
 handleChange = () => {}

 componentDidMount() {
   window.addEventListener('orientationchange', this. handleChange)
 }

 componentDidUpdate() {}

 componentWillUnmount() {
   window.removeEventListener('orientationchange', this.handleChange)
 }
}
```

functional component

```
const FloatingBanner: React.FC = () => {

  useEffect(() => {
    const handleChange = () => {}
    window.addEventListener('orientationchange', handleChange)
    return () => {
      window.removeEventListener('orientationchange', handleChange)}
  }, [])
}
```

출처: https://www.slideshare.net/ShoppingNAVER/react-hooks-177289990

Lifecycle in Hooks

“Render Phase”

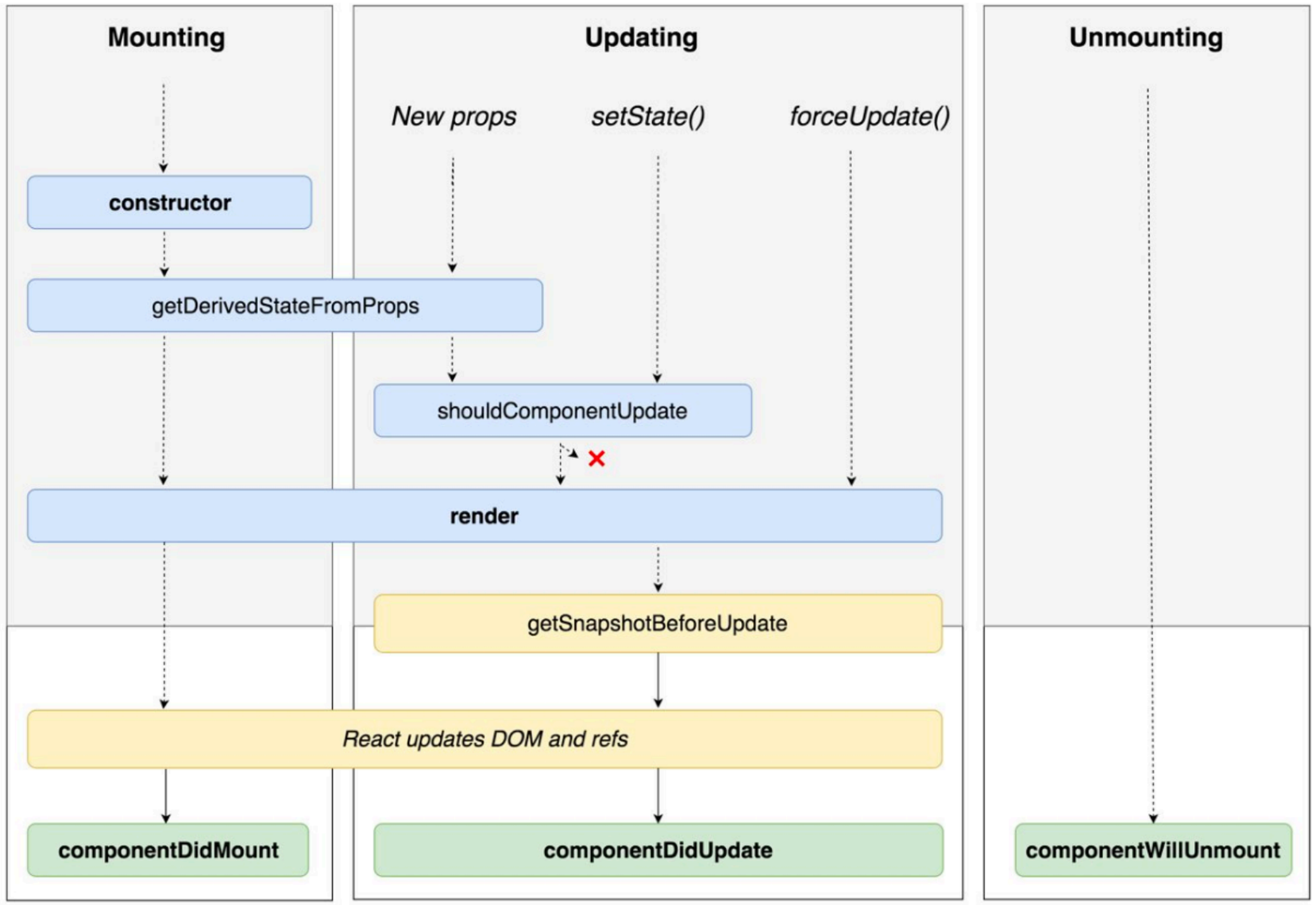
Pure and has no side effects. May be paused, aborted or restarted by React.

“Pre-Commit Phase”

Can read the DOM.

“Commit Phase”

Can work with DOM, run side effects, schedule updates.



useEffect

useMemo

아직 없음

“Render Phase”

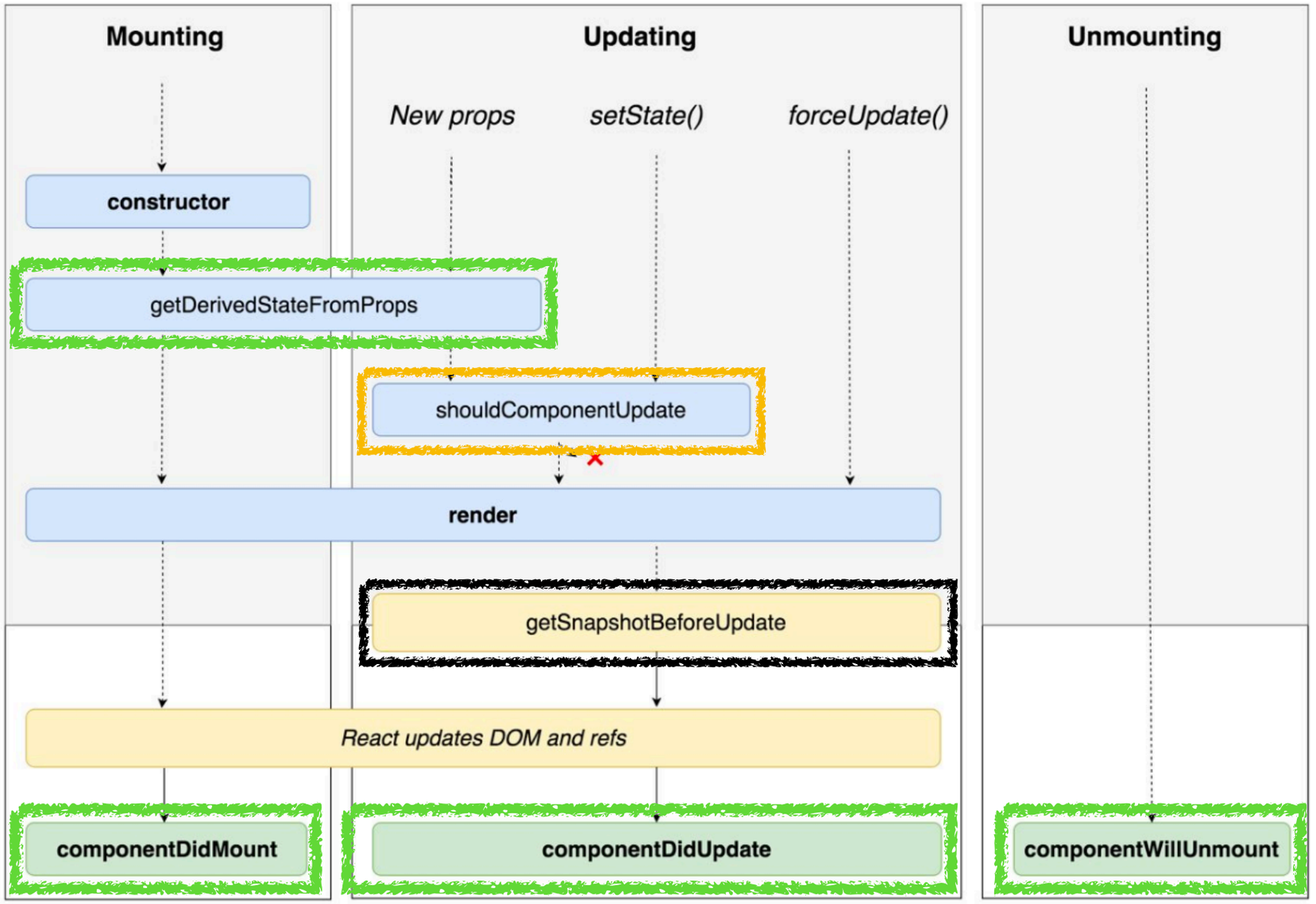
Pure and has no side effects. May be paused, aborted or restarted by React.

“Pre-Commit Phase”

Can read the DOM.

“Commit Phase”

Can work with DOM, run side effects, schedule updates.



useEffect

CC

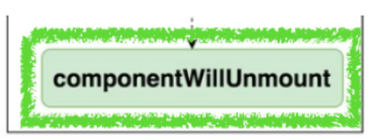
```
componentDidMount() {

}
```

Hooks in FC

```
useEffect(() => {
    // component did mount
}, []);
```


```
componentDidUpdate(prevProps, prevState) {
  console.log(`${JSON.stringify(prevState)}`);
}
```

```
useEffect(() => {
  // component did update
}, [name]);
```

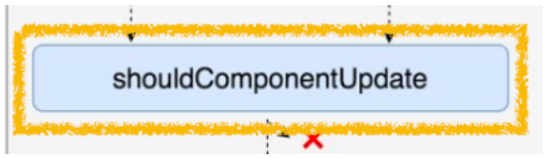



```
componentWillMount() {

}
```

```
useEffect(() => {
  return () => {
    // component will unmount
  }
}, []);
```

useMemo

CC




```
shouldComponentUpdate(nextProps, nextState) {
    // return false 이면 업데이트를 하지 않음
    return this.props.name !== nextState.name;
}
```

Hooks in FC

React.memo

```
export default React.memo(Name);
```

```
import React, { useState } from "react";

const Name = () => {
  const [name, setName] = useState("");
  const onChange = e => {
    const { name, value } = e.target;
    switch (name) {
      case "name":
        setName(value);
        break;

      default:
        break;
    }
  };
  return (
    <div>
      <input placeholder="name" name="name" value={name} onChange={onChange} />
    </div>
  );
};

export default React.memo(Name);
```

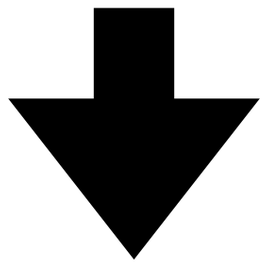
DO IT!

qewrew

qewrew

qrerereqw

qrerereqw



qewreqr

qewreqr

qwrwr

qwrwr

Reference