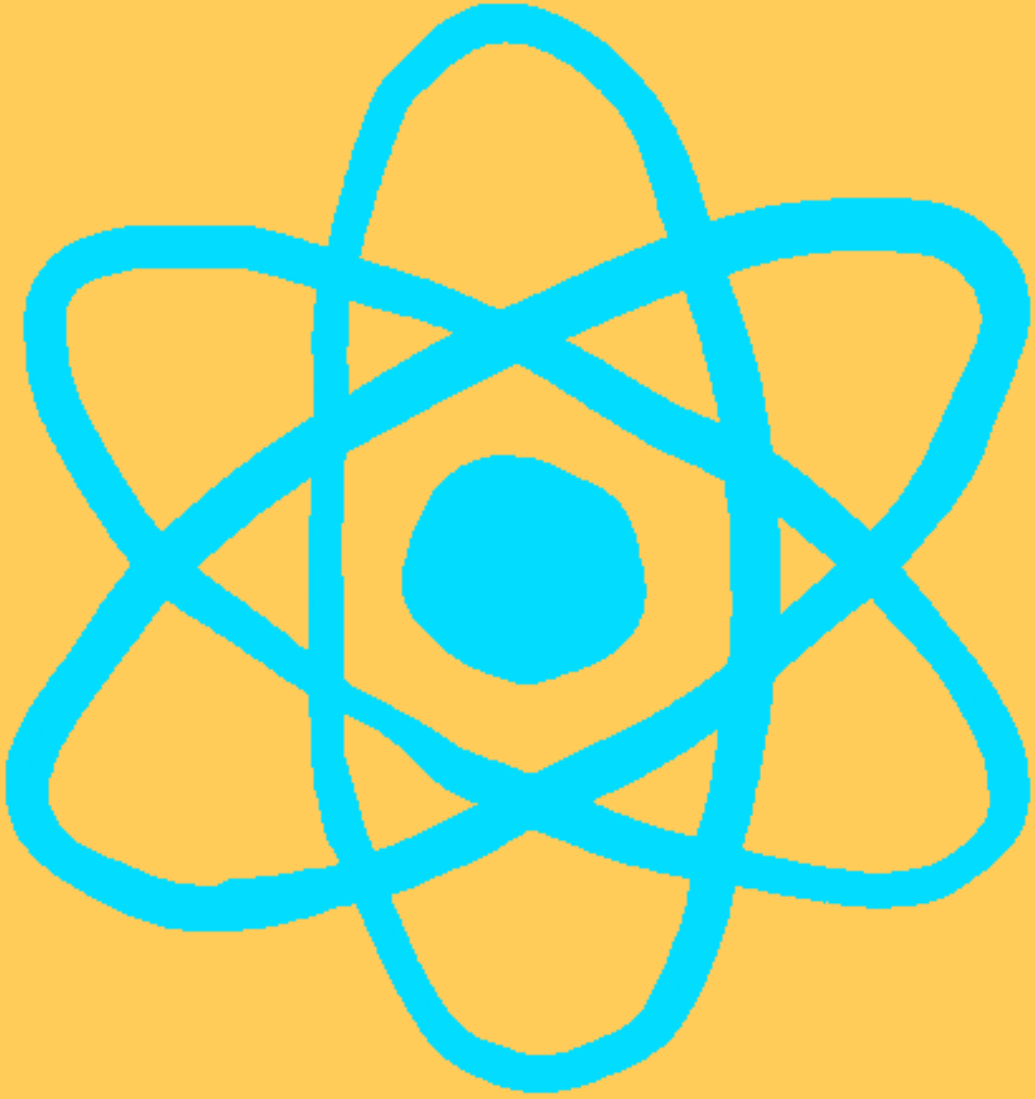
FRONTEND. REACT. ЛЕКЦІЯ 4

HOOKS.



ЗАГАЛЬНЕ ПРО ХУКИ

ЗАГАЛЬНЕ ПРО ХУКИ

- Використуються у функціональних компонентах

ЗАГАЛЬНЕ ПРО ХУКИ

- Використуються у функціональних компонентах
- Прийшли з реакту 16.8

ЗАГАЛЬНЕ ПРО ХУКИ

- Використуються у функціональних компонентах
- Прийшли з реакту 16.8
- Викликаються лише на верхньому рівні компонента

ЗАГАЛЬНЕ ПРО ХУКИ

- Використуються у функціональних компонентах
- Прийшли з реакту 16.8
- Викликаються лише на верхньому рівні компонента
- Викликаються лише з реакт функцій

МОТИВАЦІЯ. ПРОБЛЕМИ

МОТИВАЦІЯ. ПРОБЛЕМИ

- Важке перевикористання логіки між компонентами

МОТИВАЦІЯ. ПРОБЛЕМИ

- Важке перевикористання логіки між компонентами
- Важкі компоненти є нечитабельні

МОТИВАЦІЯ. ПРОБЛЕМИ

- Важке перевикористання логіки між компонентами
- Важкі компоненти є нечитабельні
- Класи важкі а повільні для JS розробки

USESTATE

USESTATE

```
export function Price(props) {
        const { price, unit, colorT } = props;
        let [color, setColor] = useState(colorT);
        const [stateSmth, setStateSmth] = useState(0);

        const changeColor = () => {
          const randomColor = `#${Math.floor(Math.random()*16777215).toString
          setColor(randomColor);
          setStateSmth(stateSmth + 1);
          setStateSmth(stateSmth + 1);
        }

        return (<div style="{{backgroundColor:" color}}="">
          {price}{unit}{stateSmth}
          <button onclick="{changeColor}">Change color {color} with {stateSmt
          </div>);
  }
```

USESTATE

```
export function Price(props) {
        const { price, unit, colorT } = props;
        let [color, setColor] = useState(colorT);
        const [stateSmth, setStateSmth] = useState(0);

        const changeColor = () => {
          const randomColor = `#${Math.floor(Math.random()*16777215).toString
          setColor(randomColor);
          setStateSmth(stateSmth + 1);
          setStateSmth(stateSmth + 1);
        }

        return (<div style="{{backgroundColor:" color}}="">
          {price}{unit}{stateSmth}
          <button onclick="{changeColor}">Change color {color} with {stateSmt
          </div>);
  }
```

USESTATE

```
export function Price(props) {
        const { price, unit, colorT } = props;
        let [color, setColor] = useState(colorT);
        const [stateSmth, setStateSmth] = useState(0);

        const changeColor = () => {
          const randomColor = `#${Math.floor(Math.random()*16777215).toString
          setColor(randomColor);
          setStateSmth(stateSmth + 1);
          setStateSmth(stateSmth + 1);
        }

        return (<div style="{{backgroundColor:" color}}="">
          {price}{unit}{stateSmth}
          <button onclick="{changeColor}">Change color {color} with {stateSmt
          </div>);
  }
```

USESTATE

```
export function Price(props) {
        const { price, unit, colorT } = props;
        let [color, setColor] = useState(colorT);
        const [stateSmth, setStateSmth] = useState(0);

        const changeColor = () => {
          const randomColor = `#${Math.floor(Math.random()*16777215).toString
          setColor(randomColor);
          setStateSmth(stateSmth + 1);
          setStateSmth(stateSmth + 1);
        }

        return (<div style="{{backgroundColor:" color}}="">
          {price}{unit}{stateSmth}
          <button onclick="{changeColor}">Change color {color} with {stateSmt
          </div>);
  }
```

USESTATE

USESTATE

- const [value, setter] = useState(defaultValue);

useState

- const [value, setter] = useState(defaultValue);
- const. Виконання коду асинхронне.

USESTATE

- const [value, setter] = useState(defaultValue);
- const. Виконання коду асинхронне.
- setter(). Відбувається рендеринг компоненту.

useState

- const [value, setter] = useState(defaultValue);
- const. Виконання коду асинхронне.
- setter(). Відбувається рендеринг компоненту.
- Для кожної зміни стану новий useState.

useState

- const [value, setter] = useState(defaultValue);
- const. Виконання коду асинхронне.
- setter(). Відбувається рендеринг компоненту.
- Для кожної зміни стану новий useState.
- Best practice. useState зверху функції.

USESTATE

Використання callback з попереднім значенням

```
export function Price(props) {
        const { price, unit, colorT } = props;
        let [color, setColor] = useState(colorT);
        const [stateSmth, setStateSmth] = useState(0);

        const changeColor = () => {
                const randomColor = `#${Math.floor(Math.random()*16777215).toS
                setColor(randomColor);
                setStateSmth((prevCounter) => prevCounter + 2);
                setStateSmth((prevCounter) => prevCounter + 1);
        }

        return (<div style="{{backgroundColor:" color}}="">
                {price}{unit}{stateSmth}
                <button onclick="{changeColor}">Change color {color} with {sta
                </div>);
        }
```

USESTATE

Використання callback з попереднім значенням

```
export function Price(props) {
        const { price, unit, colorT } = props;
        let [color, setColor] = useState(colorT);
        const [stateSmth, setStateSmth] = useState(0);

        const changeColor = () => {
                const randomColor = `#${Math.floor(Math.random()*16777215).toS
                setColor(randomColor);
                setStateSmth((prevCounter) => prevCounter + 2);
                setStateSmth((prevCounter) => prevCounter + 1);
        }

        return (<div style="{{backgroundColor:" color}}="">
                {price}{unit}{stateSmth}
                <button onclick="{changeColor}">Change color {color} with {sta
                </div>);
        }
```

useState

Встановлення початкового значення

```
1  const randomNumber = () => Math.floor(Math.random()*100, 1);
2
3  export function Price(props) {
4          const { price, unit, colorT } = props;
5          let [color, setColor] = useState(colorT);
6          const [stateSmth, setStateSmth] = useState(randomNumber);
7          // const [stateSmth, setStateSmth] = useState(randomNumber());
8
9          const randomNumber = () => Math.random();
10
11         const changeColor = () => {
12                 const randomColor = `#${Math.floor(Math.random()*16777215).toS
13                 setColor(randomColor);
14                 setStateSmth((prevCounter) => prevCounter + 2);
15                 setStateSmth((prevCounter) => prevCounter + 1);
16         }
17
18         return (<div style="{{backgroundColor:" color}}="">
```

USESTATE

Встановлення початкового значення

```
1  const randomNumber = () => Math.floor(Math.random()*100, 1);
2
3  export function Price(props) {
4          const { price, unit, colorT } = props;
5          let [color, setColor] = useState(colorT);
6          const [stateSmth, setStateSmth] = useState(randomNumber);
7          // const [stateSmth, setStateSmth] = useState(randomNumber());
8
9          const randomNumber = () => Math.random();
10
11         const changeColor = () => {
12                 const randomColor = `#${Math.floor(Math.random()*16777215).toS
13                 setColor(randomColor);
14                 setStateSmth((prevCounter) => prevCounter + 2);
15                 setStateSmth((prevCounter) => prevCounter + 1);
16         }
17
18         return (<div style="{{backgroundColor:" color}}="">
```

USESTATE

Встановлення початкового значення

```
1  const randomNumber = () => Math.floor(Math.random()*100, 1);
2
3  export function Price(props) {
4          const { price, unit, colorT } = props;
5          let [color, setColor] = useState(colorT);
6          const [stateSmth, setStateSmth] = useState(randomNumber);
7          // const [stateSmth, setStateSmth] = useState(randomNumber());
8
9          const randomNumber = () => Math.random();
10
11         const changeColor = () => {
12                 const randomColor = `#${Math.floor(Math.random()*16777215).toS
13                 setColor(randomColor);
14                 setStateSmth((prevCounter) => prevCounter + 2);
15                 setStateSmth((prevCounter) => prevCounter + 1);
16         }
17
18         return (<div style="{{backgroundColor:" color}}="">
```

USEEFFECT

USEEFFECT

- Можливість виконання сторонніх дій (ефектів)

useEffect

- Можливість виконання сторонніх дій (ефектів)
- componentDidMount + componentDidUpdate + componentWillUnmount

useEffect

- Можливість виконання сторонніх дій (ефектів)
- componentDidMount + componentDidUpdate + componentWillUnmount
- Що компонент має зробити після рендера

useEffect

- Можливість виконання сторонніх дій (ефектів)
- componentDidMount + componentDidUpdate + componentWillUnmount
- Що компонент має зробити після рендера
- useEffect не блокує рендеринг

USEEFFECT

```
1 useEffect(effect, [deps]);
```

USEEFFECT

componentDidMount + componentDidUpdate

```
1 useEffect(() => {
2         console.log(`useEffect. Execute every time`);
3 });
```

useEffect

componentDidMount

```
useEffect(() => {
        console.log(`useEffect. Render once on onload`);
}, []);
```

USEEFFECT

componentWillUnmount

```
useEffect(() => {
        console.log(`useEffect. Render once on onload`);
        return (() =>  console.log(`useEffect. Render once on unload`));
}, []);
```

useEffect

componentDidUpdate

```
useEffect(() => {
        fetch(`https://jsonplaceholder.typicode.com/${type}/1`)
          .then(response => response.json())
          .then(json => setData(json));
}, [type]);
```

USEREF

useRef

- Отримання посилання на DOM елемент

useRef

- Отримання посилання на DOM елемент
- Зберігання данних без ререндеру

useRef

- Отримання посилання на DOM елемент
- Зберігання данних без ререндеру
- Отримання попереднього значення елементу

useRef

Посилання на елемент

```
1  import {useState, useEffect, useRef} from "react";
2
3  export function Ref() {
4          const [value, setValue] = useState('start');
5          const renderCount = useRef(1);
6          const inputRef = useRef(null);
7          const prevValue = useRef('');
8
9          useEffect(() => {
10                 renderCount.current++;
11         });
12         useEffect(() => {
13                 prevValue.current = value;
14         }, [value]);
15
16         const focus = () => inputRef.current.focus();
17
18         return (
```

useRef

Посилання на елемент

```
 7          const prevValue = useRef( );
 8
 9          useEffect(() => {
10                  renderCount.current++;
11          });
12          useEffect(() => {
13                  prevValue.current = value;
14          }, [value]);
15
16          const focus = () => inputRef.current.focus();
17
18          return (
19                  < >
20                  <p>Render {renderCount.current}</p>
21                  <p>Prev value {prevValue.current}</p>
22                  <input ref="{inputRef}" onchange="{e" ==""> setValue(e.target.
23                  <button onclick="{focus}">Move focus</button>
24                  < />
```

useRef

Посилання на елемент

```
 9          useEffect(() => {
10                  renderCount.current++;
11          });
12          useEffect(() => {
13                  prevValue.current = value;
14          }, [value]);
15
16          const focus = () => inputRef.current.focus();
17
18          return (
19                  < >
20                  <p>Render {renderCount.current}</p>
21                  <p>Prev value {prevValue.current}</p>
22                  <input ref="{inputRef}" onchange="{e" ==""> setValue(e.target.
23                  <button onclick="{focus}">Move focus</button>
24                  < />
25          );
26  }
```

useRef

Зберігання даних без ререндеру

```
1  import {useState, useEffect, useRef} from "react";
2
3  export function Ref() {
4          const [value, setValue] = useState('start');
5          const renderCount = useRef(1);
6          const inputRef = useRef(null);
7          const prevValue = useRef('');
8
9          useEffect(() => {
10                 renderCount.current++;
11         });
12         useEffect(() => {
13                 prevValue.current = value;
14         }, [value]);
15
16         const focus = () => inputRef.current.focus();
17
18         return (
```

useRef

Зберігання даних без рендеру

```
1  import {useState, useEffect, useRef} from 'react';
2
3  export function Ref() {
4          const [value, setValue] = useState('start');
5          const renderCount = useRef(1);
6          const inputRef = useRef(null);
7          const prevValue = useRef('');
8
9          useEffect(() => {
10                 renderCount.current++;
11         });
12         useEffect(() => {
13                 prevValue.current = value;
14         }, [value]);
15
16         const focus = () => inputRef.current.focus();
17
18         return (
```

useRef

Зберігання даних без ререндеру

```
 9          useEffect(() => {
10                  renderCount.current++;
11          });
12          useEffect(() => {
13                  prevValue.current = value;
14          }, [value]);
15
16          const focus = () => inputRef.current.focus();
17
18          return (
19                  < >
20                  <p>Render {renderCount.current}</p>
21                  <p>Prev value {prevValue.current}</p>
22                  <input ref="{inputRef}" onchange="{e" ==""> setValue(e.target.
23                  <button onclick="{focus}">Move focus</button>
24                  < />
25          );
26  }
```

useRef

Отримання попереднього значення

```
1  import {useState, useEffect, useRef} from "react";
2
3  export function Ref() {
4          const [value, setValue] = useState('start');
5          const renderCount = useRef(1);
6          const inputRef = useRef(null);
7          const prevValue = useRef('');
8
9          useEffect(() => {
10                 renderCount.current++;
11         });
12         useEffect(() => {
13                 prevValue.current = value;
14         }, [value]);
15
16         const focus = () => inputRef.current.focus();
17
18         return (
```

useRef

Отримання попереднього значення

```
const [value, setValue] = useState('start');
const renderCount = useRef(1);
const inputRef = useRef(null);
const prevValue = useRef('');

useEffect(() => {
        renderCount.current++;
});
useEffect(() => {
        prevValue.current = value;
}, [value]);

const focus = () => inputRef.current.focus();

return (
        < >
        <p>Render {renderCount.current}</p>
        <p>Prev value {prevValue.current}</p>
```

useRef

Отримання попереднього значення

```
 9          useEffect(() => {
10                  renderCount.current++;
11          });
12          useEffect(() => {
13                  prevValue.current = value;
14          }, [value]);
15
16          const focus = () => inputRef.current.focus();
17
18          return (
19                  < >
20                  <p>Render {renderCount.current}</p>
21                  <p>Prev value {prevValue.current}</p>
22                  <input ref="{inputRef}" onchange="{e" ==""> setValue(e.target.
23                  <button onclick="{focus}">Move focus</button>
24                  < />
25          );
26  }
```

useMemo

Мемоїзація - це метод оптимізації, який в основному використовується для прискорення комп'ютерних програм шляхом зберігання результатів дорогих викликів функцій та повернення кешованого результату, коли виклики на однакових вхідних даних відбуваються знову.

useMemo

Мемоїзація - це метод оптимізації, який в основному використовується для прискорення комп'ютерних програм шляхом зберігання результатів дорогих викликів функцій та повернення кешованого результату, коли виклики на однакових вхідних даних відбуваються знову.

- Кешування об’єктів

useMemo

Мемоїзація - це метод оптимізації, який в основному використовується для прискорення комп'ютерних програм шляхом зберігання результатів дорогих викликів функцій та повернення кешованого результату, коли виклики на однакових вхідних даних відбуваються знову.

- Кешування об’єктів
- Кешування важких функцій

USEMEMO

```
const memoValue = useMemo(() => {
        expensiveOperationHere(data, value)
}, [data, value]);
```

USEMEMO

```
1  import {useState, useMemo} from "react";
2
3  function randomNumber() {
4  return Math.floor(Math.random()*100, 1);
5  }
6
7  function complex(value) {
8  let i = 0;
9  while (i > 100000000000000) i++;
10         return randomNumber();
11 }
12
13 export function Memo() {
14 const [value, setValue] = useState(0);
15 const [anotherNumber, setAnotherNumber] = useState(0);
16
17 // const recalculate = complex(value);
18
```

USEMEMO

Порівняння об'єктів

```
1 {} === {}
```

useCallback

Мемоізується функція

```
1 const memoValue = useCallback(() => {
2         expensiveFunction(data, value)
3 }, [data, value]);
```

```
1 useCallback(fn, deps) є еквівалентом useMemo(() => fn, deps).
```

USECONTEXT --> CONTEXT API

USEREDUCER --> REDUX

ІНШІ ХУКИ

Створення користувацьких

> **These docs are old and won't be updated. Go to react.dev for the new React docs.**
> These new documentation pages teach modern React and include live examples:
>
> - Reusing Logic with Custom Hooks

Хуки — це новинка в React 16.8. Вони дозволяють вам використовувати стан та інші можливості React без написання класу.

Створення власних хуків дозволить вам винести логіку компонента у функції, придатні для повторного використання.

Коли ми розглядали використання хука ефекту, ми бачили цей компонент з додатку

ШО? ШО?