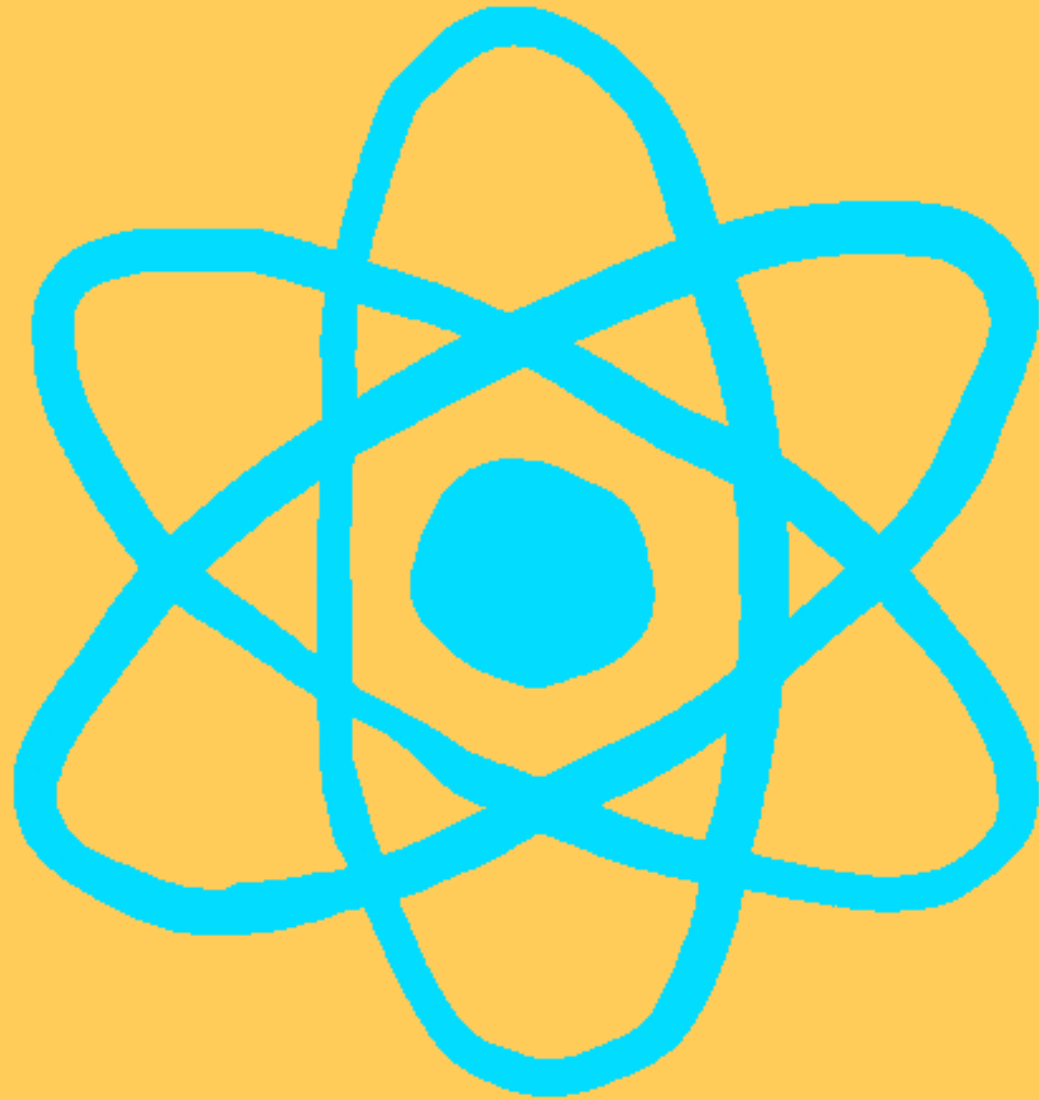


FRONTEND. REACT. ЛЕКЦІЯ 4

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

```
1 export function Price(props) {
2   const { price, unit, colorT } = props;
3   let [color, setColor] = useState(colorT);
4   const [stateSmth, setStateSmth] = useState(0);
5
6   const changeColor = () => {
7     const randomColor = `#${Math.floor(Math.random()*16777215).toString(16).padStart(6, '0')}`;
8     setColor(randomColor);
9     setStateSmth(stateSmth + 1);
10    setStateSmth(stateSmth + 1);
11  }
12
13  return (<div style={{backgroundColor: " " color}}=" ">
14    {price}{unit}{stateSmth}
15    <button onClick="{changeColor}">Change color {color} with {stateSmth}
16    </div>);
17 }
```

USESTATE

```
1 export function Price(props) {
2   const { price, unit, colorT } = props;
3   let [color, setColor] = useState(colorT);
4   const [stateSmth, setStateSmth] = useState(0);
5
6   const changeColor = () => {
7     const randomColor = `#${Math.floor(Math.random()*16777215).toString(16).padStart(6, '0')}`;
8     setColor(randomColor);
9     setStateSmth(stateSmth + 1);
10    setStateSmth(stateSmth + 1);
11  }
12
13  return (<div style={{backgroundColor:" color"}}=" ">
14    {price}{unit}{stateSmth}
15    <button onclick="{changeColor}">Change color {color} with {stateSmth}
16    </div>);
17 }
```

USESTATE

```
1 export function Price(props) {
2     const { price, unit, colorT } = props;
3     let [color, setColor] = useState(colorT);
4     const [stateSmth, setStateSmth] = useState(0);
5
6     const changeColor = () => {
7         const randomColor = `#${Math.floor(Math.random()*16777215).toString(16).padStart(6, '0')}`;
8         setColor(randomColor);
9         setStateSmth(stateSmth + 1);
10        setStateSmth(stateSmth + 1);
11    }
12
13    return (<div style={{backgroundColor:" color"}}=>
14        {price}{unit}{stateSmth}
15        <button onclick="{changeColor}">Change color {color} with {stateSmth}
16        </div>);
17 }
```

USESTATE

```
1 export function Price(props) {
2   const { price, unit, colorT } = props;
3   let [color, setColor] = useState(colorT);
4   const [stateSmth, setStateSmth] = useState(0);
5
6   const changeColor = () => {
7     const randomColor = `#${Math.floor(Math.random()*16777215).toString(16).padStart(6, '0')}`;
8     setColor(randomColor);
9     setStateSmth(stateSmth + 1);
10    setStateSmth(stateSmth + 1);
11  }
12
13  return (<div style={{backgroundColor: " " color}}=" ">
14    {price}{unit}{stateSmth}
15    <button onClick={changeColor}>Change color {color} with {stateSmth}
16    </div>);
17 }
```


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

```
1 export function Price(props) {
2   const { price, unit, colorT } = props;
3   let [color, setColor] = useState(colorT);
4   const [stateSmth, setStateSmth] = useState(0);
5
6   const changeColor = () => {
7     const randomColor = `#${Math.floor(Math.random()*16777215).toS
8     setColor(randomColor);
9     setStateSmth((prevCounter) => prevCounter + 2);
10    setStateSmth((prevCounter) => prevCounter + 1);
11  }
12
13  return (<div style={{backgroundColor:" color}}="">
14    {price}{unit}{stateSmth}
15    <button onclick="{changeColor}">Change color {color} with {sta
16    </div>);
17  }
```

USESTATE

Використання callback з попереднім значенням

```
1 export function Price(props) {
2   const { price, unit, colorT } = props;
3   let [color, setColor] = useState(colorT);
4   const [stateSmth, setStateSmth] = useState(0);
5
6   const changeColor = () => {
7     const randomColor = `#${Math.floor(Math.random()*16777215).toS
8     setColor(randomColor);
9     setStateSmth((prevCounter) => prevCounter + 2);
10    setStateSmth((prevCounter) => prevCounter + 1);
11  }
12
13  return (<div style={{backgroundColor:" color}}="">
14    {price}{unit}{stateSmth}
15    <button onclick="{changeColor}">Change color {color} with {sta
16    </div>);
17  }
```


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="{background-color: " 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="{background-color: " 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="{background-color: " 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

```
1 useEffect(() => {  
2     console.log(`useEffect. Render once on onload`);  
3 }, []);
```

USEEFFECT

componentWillUnmount

```
1 useEffect(() => {  
2     console.log(`useEffect. Render once on onload`);  
3     return () => console.log(`useEffect. Render once on unload`);  
4 }, []);
```

USEEFFECT

componentDidUpdate

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

USEREF

USEREF

- Отримання посилання на DOM елемент

USEREF

- Отримання посилання на DOM елемент
- Зберігання даних без ререндеру

USEREF

- Отримання посилання на DOM елемент
- Зберігання даних без ререндеру
- Отримання попереднього значення елемента

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(0);
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 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 (
19     <input type="text" value={value} ref={inputRef} />
20   );
21 }
```

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

```
4   const [value, setValue] = useState('');
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 (
19     <>
20       <p>Render {renderCount.current}</p>
21       <p>Prev value {prevValue.current}</p>
22       <input type="text" value={value} ref={inputRef} />
23     </>
24   );
```

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

```
1 const memoValue = useMemo(() => {  
2     expensiveOperationHere(data, value)  
3 }, [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 > 1000000000000000) 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

Мемоїзується 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

ИЛИЛИ ХУКИ

`useDeferredValue` only defers the value that you pass to it. If you want to prevent a child component from re-rendering during an urgent update, you must also memoize that component with `React.memo` or `React.useMemo`:

```
function Typeahead() {
  const query = useSearchQuery('');
  const deferredQuery = useDeferredValue(query);

  // Memoizing tells React to only re-render when
  // deferredQuery changes,
  // not when query changes.
  const suggestions = useMemo(() =>
    <SearchSuggestions query={deferredQuery} />,
    [deferredQuery]
  );

  return (
    <>
      <SearchInput query={query} />
      <Suspense fallback="Loading results...">
        {suggestions}
      </Suspense>
    </>
  );
}
```

INSTALLATION ▾

MAIN CONCEPTS ▾

ADVANCED GUIDES ▾

API REFERENCE ▾

HOOKS ^

1. Introducing Hooks
2. Hooks at a Glance
3. Using the State Hook
4. Using the Effect Hook
5. Rules of Hooks
6. Building Your Own Hooks

Створення користувацьких хуків

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

ВСТАНОВЛЕННЯ ▾

ОСНОВНІ ПОНЯТТЯ ▾

ПРОСУНУТІ ТЕМИ ▾

API-ДОВІДКА ▾

ХУКИ ▲

1. Знайомство з хуками
2. Огляд хуків
3. Використання хука ста
4. Використання хука ефекту
5. Правила хуків
- 6. Створення користувацьких хуків**
7. Довідка API хуків

WHO? WHO?

