

Тема занятия:

React: Functions,  
state, map JSX

# Передача функций

**( )  $\Rightarrow$  { }**

## Функция внутри onClick:

В этом варианте функция, отвечающая за действие, определяется непосредственно внутри onClick:

```
const ButtonComponent = () => {  
  
  const handleClick = () => {  
    console.log('Click');  
  };  
  
  return (  
    <div>  
      <button onClick={() => console.log('Click')}>Нажми  
меня</button>  
    </div>  
  );  
};
```

## Функция снаружи onClick:

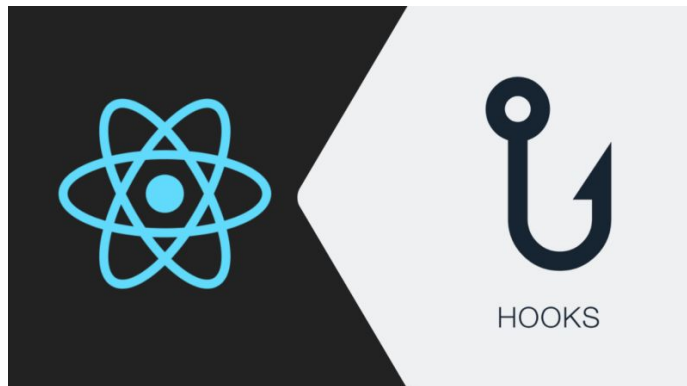
В этом варианте функция, отвечающая за действие, определяется непосредственно внутри onClick:

```
const ButtonComponent = () => {  
  
  const handleClick = () => {  
    console.log('Click');  
  };  
  
  return (  
    <div>  
      <button onClick={handleClick}>Нажми меня</button>  
    </div>  
  );  
};
```

# Передача аргументов

```
const ClickWithParameter = () => {  
  const handleClickWithParameter = (message) => {  
    console.log(`Кнопка была нажата. Сообщение: ${message}`);  
  };  
  
  return (  
    <button onClick={() => handleClickWithParameter('Привет, мир!')}>  
      Нажми меня (с параметром)  
    </button>  
  );  
};
```

# Хуки





Хук — это специальная функция, которая позволяет «подцепиться» к возможностям React.

# REACT HOOKS

- `useId()`
- `useCallback()`
- `useContext()`
- `useReducer()`
- `useMemo()`
- `Custom Hook()`

# Правила хуков

1. Используйте хуки только на верхнем уровне. Не вызывайте хуки внутри циклов, условных операторов или вложенных функций.
1. Вызывайте хуки только из React-функций  
Не вызывайте хуки из обычных функций JavaScript.  
Вместо этого можно:
  -  Вызывать хуки из функционального компонента React.
  -  Вызывать хуки из пользовательского хука
1. Имя хука начинается со слова «use»



# useState

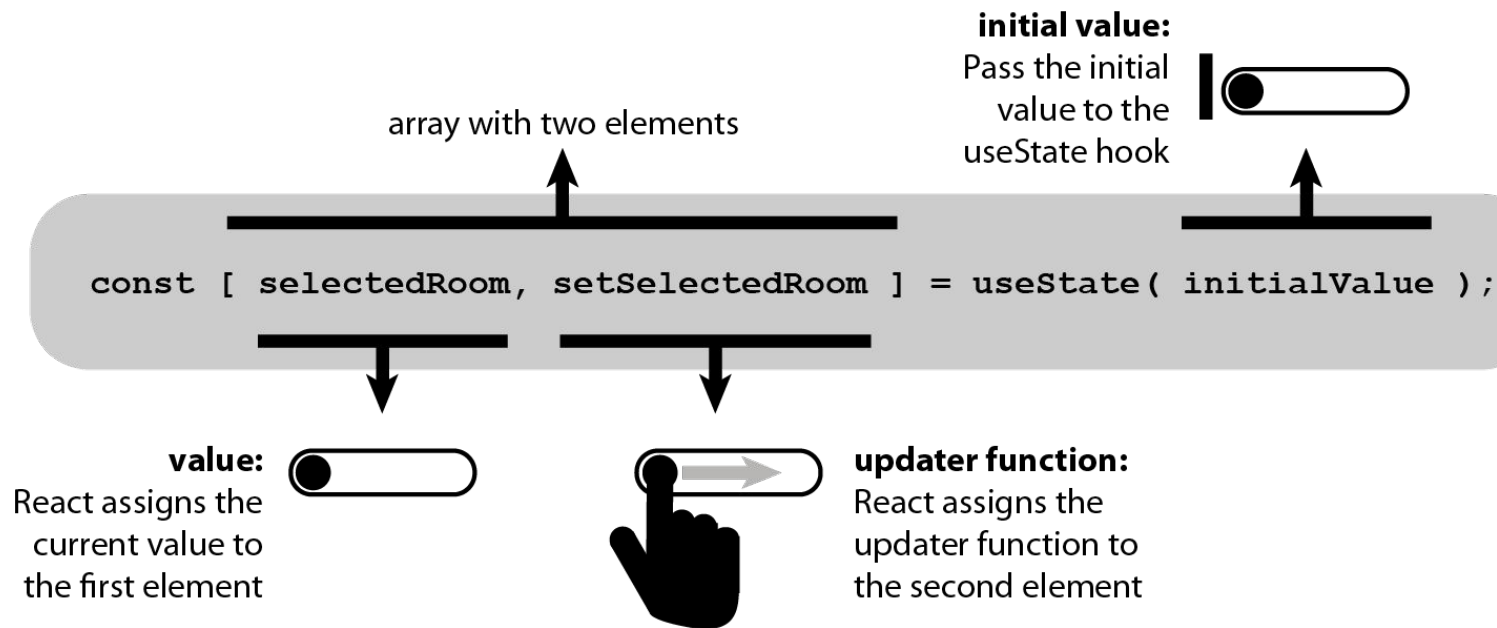


**useState** - это хук (hook) в библиотеке React, который позволяет функциональным компонентам иметь внутреннее состояние.

Состояние в React представляет собой данные, которые могут изменяться в течение времени, и когда состояние изменяется, React обновляет соответствующий компонент.



# СИНТАКСИС useState



# Возможные значения initialValue

## 1. Числа

```
const [count, setCount] = useState(0);
```

## 1. Строки

```
const [name, setName] = useState('John');
```

## 1. Логические значения (boolean)

```
const [isVisible, setIsVisible] = useState(true);
```

# Возможные значения initialValue

## 4. Массивы

```
const [items, setItems] = useState(['apple', 'banana', 'orange']);
```

## 5. Объекты

```
const [person, setPerson] = useState({ name: 'John', age: 25 });
```

## 6. Функции

```
const [callback, setCallback] = useState(() => {  
  // какая-то логика  
  return 'результат';  
});
```

## Возможные значения initialValue

### 7. null или undefined:

```
const [data, setData] = useState(null);
```

Важно выбирать начальное значение в зависимости от типа данных, с которым вы работаете, и от того, что наиболее подходит для вашего конкретного случая использования. В некоторых ситуациях может потребоваться использование функции для вычисления начального значения, особенно если это значение может изменяться в зависимости от предыдущего состояния или других условий.

Для работы с `useState` нужно сначала импортировать его из библиотеки

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

const MyComponent = () => {
  // Деструктуризация, где count - текущее состояние, setCount - функция для обновления
  const [count, setCount] = useState(0);

  // ...
};
```

## Использование значения из useState

Значение из useState может быть вставлена в любое место компонента в качестве переменной

```
<h2>Пример счетчика с функциональным обновлением</h2>  
<p>Текущее значение счетчика: {count}</p>
```

значение count из  
состояния



## Обновление состояния

Второе значение, которое возвращает `useState` - это функция, которую предоставляет `useState` для обновления состояния. Она может принимать новое значение или функцию, которая возвращает новое значение на основе предыдущего состояния.

I

```
<button onClick={() => setCount(count + 1)}>Увеличить</button>
```

II

```
<button onClick={() => setCount(prevCount => prevCount + 1)}>Увеличить</button>
```

The slide features decorative geometric patterns in the corners. The top-left and bottom-right corners contain overlapping blue and gold geometric shapes, including rectangles and triangles, some with a dotted pattern. The text is centered in the white space between these patterns.

# Map method for components

В React метод **map()** обычно используется для обхода массива данных и создания нового массива элементов React. Это особенно полезно при рендеринге списков компонентов.



## Пример использования `map()` для создания компонентов:

Создаём компонент, в котором с помощью props будут передаваться данные для создания списка

```
const DataList = ({ data }) => {  
  return (  
    <ul>  
      {data.map((item, index) => (  
        <li key={index}>{item}</li>  
      ))}  
    </ul>  
  );  
};
```

## Пример использования `map()` для создания компонентов:

Передаём в компонент `DataList` массив через родительский компонент

```
const App = () => {  
  const dataArray = ['Item 1', 'Item 2', 'Item 3'];  
  
  return (  
    <div>  
      <h1>List of Items</h1>  
      <DataList data={dataArray} />  
    </div>  
  );  
};
```

## Key

Когда вы создаете множество компонентов с использованием `map()` или других методов, React ожидает, что каждый компонент будет иметь уникальный **key**. Это помогает React эффективно обновлять и перерисовывать компоненты при изменении данных

Как идентификаторы для ключей, лучше использовать уникальные и стабильные значения. Это могут быть уникальные идентификаторы элементов, предоставляемые, например, сервером или базой данных.

```
const ItemList = ({ items }) => {  
  return (  
    <ul>  
      {items.map((item) => (  
        <li key={item.id}>{item.name}</li>  
      ))}  
    </ul>  
  );  
};
```