# ЛАБОРАТОРНАЯ РАБОТА N2. 4 КУРС

*Цель:*

*Ход работы:*

*Практика:*

Данная статься была частично взята с офф сайта https://code.mu/ru/javascript/framework/react/book/prime/

# Часть 3 input

## 35. Работа с инпутами в React

Работа с инпутами в React происходит с помощью стейтов. Каждому инпуту назначается свой стейт, содержащий в себе value инпута.

Давайте посмотрим на примере. Пусть у нас есть инпут:

function App() {

  return (

    <div>

      <input />

    </div>

  );

}

Пусть также у нас есть стейт:

function App() {

  const [value, setValue] = useState("text");

  return (

    <div>

      <input />

    </div>

  );

}

Давайте к атрибуту value инпута привяжем нас стейт:

function App() {

  const [value, setValue] = useState("text");

  return (

    <div>

      <input value={value} />

    </div>

  );

}

В таком случае получится, что *при изменении стейта, реактивно поменяется и текст инпута*.

Это, однако, дает интересный побочный эффект: теперь при запуске кода в браузере в инпуте невозможно поменять текст! Почему: потому что при вводе текста в инпут не меняется стейт value, соответственно, и текст в инпуте не должен меняться.

Попробуйте сами. Скопируйте мой код и запустите у себя. Попробуйте поизменять текст в инпуте - у вас ничего не получится. Откройте консоль браузера - вы увидите в ней предупреждение React. Это предупреждение указывает нам, что мы привязали стейт к инпуту, но тем самым заблокировали инпут.

Давайте сделаем так, чтобы в наш инпут можно было вводить текст. Для этого нужно сделать так, чтобы при вводе изменялся наш стейт на текущее значение инпута.

Для начала для этого нужно навесить на инпут событие onChange:

function App() {

  const [value, setValue] = useState("text");

  return (

    <div>

      <input value={value} onChange={handleChange} />

    </div>

  );

}

Данное событие в React ведет себя по-другому по сравнению с чистым JS. В React оно *срабатывает сразу же по изменению инпута*. То есть при вводе или удалении символа.

Давайте теперь добавим обработчик нашего события:

function App() {

  const [value, setValue] = useState("text");

  function handleChange() {}

  return (

    <div>

      <input value={value} onChange={handleChange} />

    </div>

  );

}

В этом обработчике мы должны прочитать текущий текст инпута и установить его в стейт с помощью функции setValue.

Проблема в том, что this данной функции не будет указывать на наш инпут - такова особенность React. Чтобы получить элемент, в котором случилось событие, нам необходимо использовать [event.target](https://code.mu/ru/javascript/manual/event/target/):

function App() {

  const [value, setValue] = useState("text");

  function handleChange(event) {

    console.log(event.target); // ссылка на DOM элемент инпута

  }

  return (

    <div>

      <input value={value} onChange={handleChange} />

    </div>

  );

}

Выведем с помощью event.target текущий текст инпута:

function App() {

  const [value, setValue] = useState("text");

  function handleChange(event) {

    console.log(event.target.value); // текущий текст инпута

  }

  return (

    <div>

      <input value={value} onChange={handleChange} />

    </div>

  );

}

А теперь запишем текст инпута в наш стейт:

function App() {

  const [value, setValue] = useState("text");

  function handleChange(event) {

    setValue(event.target.value);

  }

  return (

    <div>

      <input value={value} onChange={handleChange} />

    </div>

  );

}

Теперь мы сможем вводить текст в инпут. При этом стейт value всегда будет содержать текущий текст инпута.

Мы можем легко убедится в этом. Выведем содержимое нашего текста в абзац. В этом случае при вводе текста в инпут введенный текст будет автоматически появляться в абзаце:

function App() {

  const [value, setValue] = useState("");

  function handleChange(event) {

    setValue(event.target.value);

  }

  return (

    <div>

      <input value={value} onChange={handleChange} /> <p>text: {value}</p>

    </div>

  );

}

Можем переписать на более компактный вариант с анонимной стрелочной функцией:

function App() {

  const [value, setValue] = useState("");

  return (

    <div>

      <input value={value} onChange={(event) => setValue(event.target.value)} />

      <p>text: {value}</p>

    </div>

  );

}

Таким образом, для работы любого инпута нам нужно следующее: создать стейт для этого инпута, привязать стейт к атрибуту value инпута, навесить событие onChange на инпут, в обработчике события менять стейт инпута на его текст.

Данные операции нужно будет проводить с каждым инпутом. То есть, если у вас два инпута, то у вас будет два стейта и две функции-обработчика события onChange.

### Задание №1

Сделайте два инпута. Пусть текст первого инпута выводится в первый абзац, а текст второго инпута - во второй абзац.

## 36. Изменение данных инпута при их выводе в React

Пусть мы будем вводить в инпут число. Давайте сделаем так, чтобы по мере ввода числа в инпут в абзац выводился квадрат вводимого числа:

function App() {

  const [value, setValue] = useState(0);

  function handleChange(event) {

    setValue(event.target.value);

  }

  return (

    <div>

      <input value={value} onChange={handleChange} /> <p>{value \*\* 2}</p>

    </div>

  );

}

### Задание №1

Дан инпут. Дан абзац. Сделайте так, чтобы при вводе текста в инпут, в абзаце выводилось количество введенных в инпут символов.

## 37. Изменение данных с помощью функции в React

Не обязательно совершать некие операции над стейтом прямо на выводе. Можно воспользоваться функцией:

function square(num) {

  return num \*\* 2;

}

function App() {

  const [value, setValue] = useState(0);

  function handleChange(event) {

    setValue(event.target.value);

  }

  return (

    <div>

      <input value={value} onChange={handleChange} /> <p>{square(value)}</p>

    </div>

  );

}

## 38. Работа с несколькими инпутами в React

Пусть у нас есть два инпута, в которые будут вводится числа. Давайте сделаем так, чтобы по мере ввода в абзац выводилась сумма двух инпутов:

function App() {

  const [value1, setValue1] = useState(0);

  const [value2, setValue2] = useState(0);

  function handleChange1(event) {

    setValue1(+event.target.value);

  }

  function handleChange2(event) {

    setValue2(+event.target.value);

  }

  return (

    <div>

      <input value={value1} onChange={handleChange1} />

      <input value={value2} onChange={handleChange2} /> <p>result: {value1 + value2}</p>

    </div>

  );

}

### Задание №1

Даны 5 инпутов. Сделайте так, чтобы при вводе чисел в наши инпуты в абзац выводилось среднее арифметическое введенных чисел.

## 39. Обработка данных формы по нажатию на кнопку в React

В предыдущем уроке мы делали так, чтобы при вводе символа в инпут в абзаце мгновенно появлялся результат. Это, конечно же, смотрится красиво, но имеет недостаток.

Представим себе, что нам нужно сделать некоторую "тяжелую", ресурсоемкую операцию. Не очень оптимально делать ее на каждый ввод символа - лучше дождаться окончательного ввода данных и потом выполнить нужную операцию один раз и результат вывести в абзац.

Для этого нам нужно ввести кнопку, по нажатию на которую будет совершаться наша ресурсоемкая операция. В таком случае у нас опять каждому инпуту будет соответствовать свой стейт, плюс еще один стейт нам нужен для записи результата операции и отображения его на экран.

Посмотрим на примере. Пусть у нас есть два инпута и кнопка. По нажатию на кнопку давайте найдем сумму чисел, введенных в инпуты.

Реализуем:

function App() {

  const [value1, setValue1] = useState(0);

  const [value2, setValue2] = useState(0);

  const [result, setResult] = useState(0);

  function handleChange1(event) {

    setValue1(event.target.value);

  }

  function handleChange2(event) {

    setValue2(event.target.value);

  }

  function handleClick() {

    setResult(Number(value1) + Number(value2));

  }

  return (

    <div>

      <input value={value1} onChange={handleChange1} />

      <input value={value2} onChange={handleChange2} /> <button onClick={handleClick}>btn</button>

      <p>result: {result}</p>

    </div>

  );

}

Можно использовать сокращенный вариант:

function App() {

  const [value1, setValue1] = useState(0);

  const [value2, setValue2] = useState(0);

  const [result, setResult] = useState(0);

  return (

    <div>

      <input value={value1} onChange={(event) => setValue1(event.target.value)} />

      <input value={value2} onChange={(event) => setValue2(event.target.value)} />

      <button onClick={() => setResult(Number(value1) + Number(value2))}>btn</button>

      <p>result: {result}</p>

    </div>

  );

}

## 40. Работа с textarea в React

Давайте теперь научимся работать с многострочным полем ввода [textarea](https://code.mu/ru/markup/manual/html/tag/textarea/). В React работа с ним, для удобства, сделана похожей на работу с текстовым инпутом. В отличие от чистого JS, в React в textarea не нужен закрывающий тег, а его текст следует размещать в атрибуте value.

Смотрите пример:

function App() {

  const [value, setValue] = useState("");

  function handleChange(event) {

    setValue(event.target.value);

  }

  return (

    <div>

      <textarea value={value} onChange={handleChange} /> <p>{value}</p>

    </div>

  );

}

В сокращенной форме:

function App() {

  const [value, setValue] = useState("");

  return (

    <div>

      <textarea value={value} onChange={(event) => setValue(event.target.value)} /> <p>{value}</p>

    </div>

  );

}

## 41. Работа с чекбоксами в React

Работа с [чекбоксами](https://code.mu/ru/markup/manual/html/tag/input/checkbox/) также осуществляется по схожему принципу, только вместо атрибута value мы указываем атрибут checked. Если в этот атрибут передать true - то чекбокс будет отмечен, а если false - не будет отмечен:

function App() {

  return (

    <div>

      <input type='checkbox' checked={true} /> отмечен <input type='checkbox' checked={false} /> не

      отмечен

    </div>

  );

}

Обычно в атрибут checked передается стейт, содержащий логическое значение:

function App() {

  const [checked, setChecked] = useState(true);

  return (

    <div>

      <input type='checkbox' checked={checked} />

    </div>

  );

}

Так же, как и при работе с инпутами, если жестко задать значение атрибута checked - состояние чекбокса нельзя будет сменить. Для корректной работы будем по изменению чекбокса менять его стейт на противоположное ему значение:

function App() {

  const [checked, setChecked] = useState(true);

  function handleChange() {

    setChecked(!checked); // инвертируем стейт

  }

  return (

    <div>

      <input type='checkbox' checked={checked} onChange={handleChange} />

    </div>

  );

}

Можем упростить:

function App() {

  const [checked, setChecked] = useState(true);

  return (

    <div>

      <input type='checkbox' checked={checked} onChange={() => setChecked(!checked)} />

    </div>

  );

}

Давайте выведем состояние чекбокса в абзац, используя тернарный оператор:

function App() {

  const [checked, setChecked] = useState(true);

  return (

    <div>

      <input type='checkbox' checked={checked} onChange={() => setChecked(!checked)} />

      <p>состояние: {checked ? "отмечен" : "не отмечен"}</p>

    </div>

  );

}

## 42. Чекбоксы и условный рендеринг в React

Давайте сделаем так, чтобы в зависимости от отметки чекбокса, на экран выводился либо один кусочек кода, либо другой. Используем для этого условный рендеринг:

function App() {

  const [checked, setChecked] = useState(true);

  let message;

  if (checked) {

    message = <p>сообщение 1</p>;

  } else {

    message = <p>сообщение 2</p>;

  }

  return (

    <div>

      <input type='checkbox' checked={checked} onChange={() => setChecked(!checked)} />

      <div>{message}</div>

    </div>

  );

}

### Задание №1

Дан чекбокс. С помощью чекбокса спросите у пользователя, если ли ему уже 18 лет. Если чекбокс отмечен, покажите пользователю следующий блок кода:

<div>

  <h2>Ура, вам уже есть 18</h2>

  <p> здесь расположен контент только для взрослых </p>

</div>;

А если чекбокс не отмечен - то следующий:

<div>

  <p> увы, вам еще нет 18 лет:( </p>

</div>;

## 43. Работа с селектами в React

Давайте теперь займемся выпадающими списками [select](https://code.mu/ru/markup/manual/html/tag/select/). Работа с ними также практически не отличается от работы с инпутами и чекбоксами.

Пусть у нас есть вот такой селект:

function App() {

  return (

    <div>

      <select>

        <option>text1</option> <option>text2</option> <option>text3</option> <option>text4</option>

      </select>

    </div>

  );

}

Давайте обеспечим его работу средствами React:

function App() {

  const [value, setValue] = useState("");

  function handleChange(event) {

    setValue(event.target.value);

  }

  return (

    <div>

      <select value={value} onChange={handleChange}>

        <option>text1</option> <option>text2</option> <option>text3</option> <option>text4</option>

      </select>

      <p> ваш выбор: {value} </p>

    </div>

  );

}

### Задание №1

Сделайте выпадающий список городов. Сделайте также абзац, в который будет выводиться выбор пользователя.

## 44. Пункты селекта из массива в React

Пусть у нас есть массив с текстами пунктов выпадающего списка:

function App() {

  const texts = ["text1", "text2", "text3", "text4"];

  const [value, setValue] = useState("");

  return <div> ... </div>;

}

Давайте сформируем наши теги option в цикле:

function App() {

  const texts = ["text1", "text2", "text3", "text4"];

  const [value, setValue] = useState("");

  const options = texts.map((text, index) => {

    return <option key={index}>{text}</option>;

  });

  return (

    <div>

      <select value={value} onChange={(event) => setValue(event.target.value)}>

        {options}

      </select>

      <p> ваш выбор: {value} </p>

    </div>

  );

}

## 45. Атрибуты value в селекте в React

Пусть теперь у нас в тегах option есть атрибуты value:

function App() {

  return (

    <div>

      <select>

        <option value='1'>text1</option> <option value='2'>text2</option>

        <option value='3'>text3</option>

      </select>

    </div>

  );

}

В таком случае из-за наличия атрибутов value в стейт будут попадать именно их значения, а не тексты тегов option. Можно убедится в этом, выведя результат выбора в абзац:

function App() {

  const [value, setValue] = useState("");

  return (

    <div>

      <select value={value} onChange={(event) => setValue(event.target.value)}>

        <option value='1'>text1</option> <option value='2'>text2</option>

        <option value='3'>text3</option>

      </select>

      <p> ваш выбор: {value} </p>

    </div>

  );

}

Разделить текст option и его значение может быть удобно: текст тега мы можем менять как нам угодно, при этом в нашем скрипте результат выбора будет обрабатываться по значению атрибута value, которые останется неизменным.

Смотрите пример:

function App() {

  const [value, setValue] = useState("");

  return (

    <div>

      <select value={value} onChange={(event) => setValue(event.target.value)}>

        <option value='1'>text1</option> <option value='2'>text2</option>

        <option value='3'>text3</option>

      </select>

      <p>

        {value === "1" && "вы выбрали первый пункт"} {value === "2" && "вы выбрали второй пункт"}

        {value === "3" && "вы выбрали третий пункт"}

      </p>

    </div>

  );

}

Теперь, если мы изменим тексты оптионов, то работа скрипта не нарушится - ведь она привязана к значению атрибута value.

### Задание №1

С помощью выпадающего списка предложите пользователю выбрать к какой возрастной группе он относится: от 0 до 12 лет, от 13 до 17, от 18 до 25, либо старше 25 лет.

## 46. Атрибуты value селекта из массива в React

Пусть у нас опять пункты списка хранятся в массиве:

function App() {

  const texts = ["text1", "text2", "text3", "text4"];

  const [value, setValue] = useState("");

}

Давайте сформируем с помощью этого массива теги option, добавив им в качестве атрибутов value значения элементов массива:

function App() {

  const texts = ["text1", "text2", "text3", "text4"];

  const [value, setValue] = useState("");

  const options = texts.map((text, index) => {

    return (

      <option key={index} value={index}>

        {text}

      </option>

    );

  });

}

Используя сформированные теги создадим выпадающий список:

return (

  <div>

    <select value={value} onChange={(event) => setValue(event.target.value)}>

      {options}

    </select>

  </div>

);

Выведем в абзац номер выбранного пункта:

return (

  <div>

    <select value={value} onChange={(event) => setValue(event.target.value)}>

      {options}

    </select>

    <p> ваш выбор: {value} </p>

  </div>

);

А теперь выведем текст выбранного пункта, используя его номер и массив с текстами:

return (

  <div>

    <select value={value} onChange={(event) => setValue(event.target.value)}>

      {options}

    </select>

    <p> ваш выбор: {texts[value]} </p>

  </div>

);

Соберем все вместе и получим следующий код:

function App() {

  const texts = ["text1", "text2", "text3", "text4"];

  const [value, setValue] = useState("");

  const options = texts.map((text, index) => {

    return (

      <option key={index} value={index}>

        {text}

      </option>

    );

  });

  return (

    <div>

      <select value={value} onChange={(event) => setValue(event.target.value)}>

        {options}

      </select>

      <p> ваш выбор: {texts[value]} </p>

    </div>

  );

}

## 47. Работа с radio в React

Работа с [радиокнопочками](https://code.mu/ru/markup/manual/html/tag/input/radio/) radio несколько отличается, к примеру, от тех же чекбоксов. Проблема в том, что у нескольких радио будет один и тот же стейт, но разные value.

Поэтому работа происходит следующем образом: каждой радиокнопке в атрибут value записывают свое значение, а в атрибут checked - специальное условие, которое проверяет, равен ли стейт определенному значению. Если равен - радиокнопочка станет отмеченной, а если не равен - будет не отмеченной.

Вот реализация описанного:

function App() {

  const [value, setValue] = useState(1);

  function changeHandler(event) {

    setValue(event.target.value);

  }

  return (

    <div>

      <input

        type='radio'

        name='radio'

        value='1'

        checked={value === "1" ? true : false}

        onChange={changeHandler}

      />

      <input

        type='radio'

        name='radio'

        value='2'

        checked={value === "2" ? true : false}

        onChange={changeHandler}

      />

      <input

        type='radio'

        name='radio'

        value='3'

        checked={value === "3" ? true : false}

        onChange={changeHandler}

      />

    </div>

  );

}

## 48. Значения по умолчанию в React

Иногда может потребоваться сделать так, чтобы по умолчанию в инпуте уже было какое-то значение. При этом нужно, чтобы из стейта было взято только начальное значение инпута, а сам инпут не был бы привязан к этому стейту.

Для решения такой задачи нужно воспользоваться атрибутом defaultValue:

function App() {

  const [value, setValue] = useState("text");

  return (

    <div>

      <input defaultValue={value} />

    </div>

  );

}

Для чекбоксов существует аналогичный атрибут defaultChecked, с помощью которого можно задать начальное:

function App() {

  const [checked, setChecked] = useState(true);

  return (

    <div>

      <input type='checkbox' defaultChecked={checked} />

    </div>

  );

}

## 49. Привязка инпутов к массиву в React

Пусть в стейте notes хранится массив:

function App() {

  const [notes, setNotes] = useState([1, 2, 3]);

  return <div> </div>;

}

Пусть у нас также есть вспомогательная функция, находящая сумму элементов массива:

function getSum(arr) {

  let sum = 0;

  for (const elem of arr) {

    sum += +elem;

  }

  return sum;

}

function App() {}

Давайте найдем и выведем сумму элементов нашего массива из стейта, использовав для этого нашу вспомогательную функцию:

function App() {

  const [notes, setNotes] = useState([1, 2, 3]);

  return <div> {getSum(notes)} </div>;

}

Давайте теперь сделаем три инпута и в value каждого инпута запишем один из элементов массива:

function App() {

  const [notes, setNotes] = useState([1, 2, 3]);

  return (

    <div>

      <input value={notes[0]} /> <input value={notes[1]} /> <input value={notes[2]} />

      {getSum(notes)}

    </div>

  );

}

Давайте теперь добавим событие onChange нашим инпутам. При этом сделаем *одну общую функцию-обработчик* этого события:

function App() {

  const [notes, setNotes] = useState([1, 2, 3]);

  function changeHandler(index, event) {

    // общая функция-обработчик

  }

  return (

    <div>

      <input value={notes[0]} onChange={(event) => changeHandler(0, event)} />

      <input value={notes[1]} onChange={(event) => changeHandler(1, event)} />

      <input value={notes[2]} onChange={(event) => changeHandler(2, event)} /> {getSum(notes)}

    </div>

  );

}

Как вы видите, функция changeHandler первым параметром принимает номер того элемента массива, который редактирует данный инпут.

По этому номеру мы можем заменить элемент массива на содержимое инпута.

Сделаем это:

function changeHandler(index, event) {

  setNotes([...notes.slice(0, index), event.target.value, ...notes.slice(index + 1)]);

}

Теперь можно будет поредактировать любой инпут, при этом реактивно будет изменяться массив и, соответственно, пересчитываться сумма его элементов.

Давайте соберем весь наш код вместе:

function App() {

  const [notes, setNotes] = useState([1, 2, 3]);

  function changeHandler(index, event) {

    setNotes([...notes.slice(0, index), event.target.value, ...notes.slice(index + 1)]);

  }

  return (

    <div>

      <input value={notes[0]} onChange={(event) => changeHandler(0, event)} />

      <input value={notes[1]} onChange={(event) => changeHandler(1, event)} />

      <input value={notes[2]} onChange={(event) => changeHandler(2, event)} /> {getSum(notes)}

    </div>

  );

}

Можно сделать так, чтобы инпуты формировались в цикле:

function App() {

  const [notes, setNotes] = useState([1, 2, 3]);

  function changeHandler(index, event) {

    setNotes([...notes.slice(0, index), event.target.value, ...notes.slice(index + 1)]);

  }

  const result = notes.map((note, index) => {

    return <input key={index} value={note} onChange={(event) => changeHandler(index, event)} />;

  });

  return (

    <div>

      {result} {getSum(notes)}

    </div>

  );

}

### Задание №1

Дан следующий массив:

[1, 2, 3, 4, 5, 6, 7, 8, 9];

Выведите на экран среднее арифметическое элементов этого массива. В цикле сделайте инпуты для редактирования элементов.

## 50. Привязка инпутов к объекту в React

Пусть в стейте хранится объект:

const initObj = { prop1: "value1", prop2: "value2", prop3: "value3" };

function App() {

  const [obj, setObj] = useState(initObj);

  return (

    <div>

      {obj.prop1}-{obj.prop2}-{obj.prop3}

    </div>

  );

}

Давайте выведем каждое свойство нашего объекта в отдельном инпуте:

function App() {

  const [obj, setObj] = useState(initObj);

  return (

    <div>

      <input value={obj.prop1} /> <input value={obj.prop2} /> <input value={obj.prop3} /> <br />

      {obj.prop1}-{obj.prop2}-{obj.prop3}

    </div>

  );

}

Давайте теперь к каждому инпуту привяжем событие onChange. В качестве обработчика назначим одну общую функцию:

function App() {

  const [obj, setObj] = useState(initObj);

  return (

    <div>

      <input value={obj.prop1} onChange={(event) => handleChange("prop1", event)} />

      <input value={obj.prop2} onChange={(event) => handleChange("prop2", event)} />

      <input value={obj.prop3} onChange={(event) => handleChange("prop3", event)} /> <br />

      {obj.prop1}-{obj.prop2}-{obj.prop3}

    </div>

  );

}

Как вы видите, функция handleChange первым параметром принимает имя соответствующего свойства объекта.

Давайте напишем реализацию нашей функции:

function handleChange(prop, event) {

  const copy = Object.assign({}, obj);

  copy[prop] = event.target.value;

  setObj(copy);

}

Данная реализация рабочая, но ее можно упростить, если воспользоваться вычисляемыми именами свойств объекта:

function handleChange(prop, event) {

  setObj({ ...obj, ...{ [prop]: event.target.value } });

}

Давайте соберем весь код вместе:

function App() {

  const [obj, setObj] = useState(initObj);

  function handleChange(prop, event) {

    setObj({ ...obj, ...{ [prop]: event.target.value } });

  }

  return (

    <div>

      <input value={obj.prop1} onChange={(event) => handleChange("prop1", event)} />

      <input value={obj.prop2} onChange={(event) => handleChange("prop2", event)} />

      <input value={obj.prop3} onChange={(event) => handleChange("prop3", event)} /> <br />

      {obj.prop1}-{obj.prop2}-{obj.prop3}

    </div>

  );

}

### Задание №1

Пусть в стейте хранится объект с датой:

const initDate = { year: 2025, month: 12, day: 31 };

Выведите в абзаце год, месяц и день из даты, хранящейся в стейте, а также день недели, соответствующий ей.

### Задание №2

Модифицируйте предыдущую задачу, добавив три инпута для редактирования даты.

# Часть 4

## 51. Реактивность данных в React

В React мы будем хранить наши данные в различных структурах: в массивах, в объектах, в массивах объектов. Эти структуры в React обладают реактивностью. React реактивно реагирует на все изменения данных: на добавление, удаление, изменение элементов, а также на изменение их порядка.

При этом правилами React *запрещено изменять данные массивов и объектов из стейта*. Нужно делать копию данных, изменять их и записывать обратно в стейт. Это значит, что наши данные должны быть иммутабельными. Читайте про [иммутабельность массивов, объектов и массивов объектов](https://code.mu/ru/javascript/book/supreme/data/immutability/) в соответствующем разделе учебника JavaScript.

## 52. Реактивное добавление в массив в React

Давайте поработаем с реактивным добавлением элементов в массив. Пусть для примера у нас есть массив, который выводится в виде списка ul:

function App() {

  const [notes, setNotes] = useState([1, 2, 3, 4, 5]);

  const result = notes.map((note, index) => {

    return <li key={index}>{note}</li>;

  });

  return (

    <div>

      <ul> {result} </ul>

    </div>

  );

}

### Задание №1

Сделайте кнопку, по нажатию на которую будет добавляться новый элемент в конец массива, тем самым добавляя новый li в конец тега ul.

### Задание №2

Сделайте инпут и кнопку. По нажатию на кнопку пусть текст инпута станет новым тегом li в конец тега ul.

## 53. Реактивные операции над массивами в React

Давайте теперь научимся делать реактивные манипуляции с элементами массива. Для этого мы на каждую li навесим обработчик события, в которой параметром будем передавать номер этой li в массиве:

function App() {

  const [notes, setNotes] = useState([1, 2, 3, 4, 5]);

  const result = notes.map((note, index) => {

    return (

      <li key={index} onClick={() => doSmth(index)}>

        {note}

      </li>

    );

  });

  return (

    <div>

      <ul> {result} </ul>

    </div>

  );

}

Внутри нашей функции мы можем совершить любую операцию над нашим элементом массива:

function App() {

  function doSmth(index) {

    let copy = Object.assign([], notes);

    copy[index] += "!"; // что-то сделаем с элементом

    setNotes(copy);

  }

}

## 54. Реактивное добавление в массив объектов в React

Давайте теперь изучим реактивность массива объектов. В данном случае мы должны будем производить любые изменения, обращаясь к элементам по id, которые хранятся внутри самих объектов.

Давайте попробуем. Пусть у нас есть следующий массив объектов:

const initNotes = [

  { id: "GYi9G\_uC4gBF1e2SixDvu", prop1: "value11", prop2: "value12", prop3: "value13" },

  { id: "IWSpfBPSV3SXgRF87uO74", prop1: "value21", prop2: "value22", prop3: "value23" },

  { id: "JAmjRlfQT8rLTm5tG2m1L", prop1: "value31", prop2: "value32", prop3: "value33" },

];

Давайте выведем каждый элемент нашего массива в отдельном теге li:

function App() {

  const [notes, setNotes] = useState(initNotes);

  const result = notes.map((note) => {

    return (

      <li key={note.id}>

        <span>{note.prop1}</span> <span>{note.prop2}</span> <span>{note.prop3}</span>

      </li>

    );

  });

  return (

    <div>

      <ul> {result} </ul>

    </div>

  );

}

### Задание №1

Сделайте кнопку, по нажатию на которую будет добавляться новый элемент в конец массива, тем самым добавляя новый li в конец тега ul.

### Задание №2

Сделайте три инпута и кнопку. По нажатию на кнопку из данных инпута создайте новую li в конце тега ul.

## 55. Реактивные операции над массивами объектов в React

Давайте теперь научимся делать реактивные операции над массивами объектов. В этом случае мы должны передать в функцию id элемента массива, с которым мы собираемся что-то делать:

function App() {

  const [notes, setNotes] = useState(initNotes);

  const result = notes.map((note) => {

    return (

      <li key={note.id}>

        <span>{note.prop1}</span> <span>{note.prop2}</span> <span>{note.prop3}</span>

        <button onClick={() => doSmth(note.id)}> btn </button>

      </li>

    );

  });

  return (

    <div>

      <ul> {result} </ul>

    </div>

  );

}

Для того, чтобы что-то сделать с элементом, мы сначала должны найти его, перебрав весь массив циклом:

  function doSmth(id) {

    setNotes(

      notes.map((note) => {

        if (note.id === id) {

          // что-то делаем с элементом

        }

        return note;

      }),

    );

  }

Давайте для примера поменяем тексты найденного объекта:

function doSmth(id) {

  setNotes(

    notes.map((note) => {

      if (note.id === id) {

        note.prop1 += "!";

        note.prop2 += "!";

        note.prop2 += "!";

        return note;

      }

      return note;

    }),

  );

}

## 56. Реактивный показ данных в React

Пусть у нас есть массив объектов, содержащий в себе названия и описания чего-либо:

const initNotes = [

  { id: 1, name: "name1", desc: "long description 1" },

  { id: 2, name: "name2", desc: "long description 2" },

  { id: 3, name: "name3", desc: "long description 3" },

];

Давайте выведем каждый элемент этого массива в отдельном абзаце:

function App() {

  const [notes, setNotes] = useState(initNotes);

  const result = notes.map((note) => {

    return (

      <p key={note.id}>

        {note.name}, <i>{note.desc}</i>

      </p>

    );

  });

  return <div> {result} </div>;

}

Давайте теперь сделаем так, чтобы описание изначально было скрыто, но в конце каждого абзаца добавим кнопки для показа описания из этого абзаца. Для этого в каждый объект с продуктом добавим свойство show, регулирующее показ описания:

const initNotes = [

  { id: 1, name: "name1", desc: "long description 1", show: false },

  { id: 2, name: "name2", desc: "long description 2", show: false },

  { id: 3, name: "name3", desc: "long description 3", show: false },

];