Массивы в React

Массивы в React также обладают реактивностью. При чем React реактивно реагирует на все изменения массива: на добавление, удаление, изменение элементов, а также на изменение их порядка.
При этом правилами React запрещено изменять сам массив из стейта, он должен быть иммутабельным.
Правильным подходом в React является создание нового массива на основе предыдущего и перезапись стейта новым массивом.

Пусть в стейте notes хранится массив

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

Выведем каждый элемент этого массива в отдельном абзаце

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

  const result = notes.map((note, index) => {
    return <p key={index}>{note}</p>;
  });

  return <div>
    {result}
  </div>;
}
```

Добавим в массив notes еще один элемент. Через метод push это делать неправильно, так как мы изменяем сам масси

```
notes.push(6); // так не правильно
setNotes(notes);
```

Чтобы не изменять сам массив, можно сделать его копию, применить к ней метод push, и затем перезаписать стейт на массив из копии:

```
const copy = Object.assign([], notes);
copy.push(6); // так правильно
setNotes(copy);
```

А можно использовать прием с деструктуризацией `setNotes([...notes, 6]);`

Удалим из массива элемент по его номеру. Пусть этот номер хранится в переменной index. Через метод splice удаление делать неправильно, так как изменятся массив:

```
notes.splice(index, 1); // так не правильно
setNotes(notes);
```

```
const copy = Object.assign([], notes);
copy.splice(index, 1); // так правильно
setNotes(copy);
```

С деструктуризацией

```
setNotes([...notes.slice(0, index), ...notes.slice(index + 1)]);
```

Изменим значение некоторому элементу массива. Пусть номер для изменения хранится в переменной inde

```
notes[index] = '!'; // так не правильно
setNotes(notes);
```

```
let copy = Object.assign([], notes);
copy[index] = '!'; // так правильно
setNotes(copy);
```

Используя деструктуризацию

```
setNotes([...notes.slice(0, index), '!',...notes.slice(index + 1)]);
```

Сортировать массив также будет неправильным, так как функция сортировки изменяет массив:

```
notes.sort(); // так не правильно
setNotes(notes);
```

Правильным подходом будет выполнить сортировку копии

```
let copy = Object.assign([], notes);
copy.sort(); // так правильно
setNotes(copy);
```

Задания

1. Сделайте кнопку, по нажатию на которую будет происходить добавление нового элемента в массив.
2. Сделайте кнопку, по нажатию на которую будет происходить удаление элемента из массива. Пусть номер элемента для удаления хранится в переменной.
3. Сделайте кнопку, по нажатию на которую будет происходить изменение элемента массива. Пусть номер элемента для изменения хранится в переменной.
4. Сделайте кнопку, по нажатию на которую будет происходить переворот элементов массива в обратном порядке.

Форма для добавления элементов в массив

Пусть в стейте notes хранится массив и элементы этого массива выводятся в абзацах:

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

  const result = notes.map((note, index) => {
    return <p key={index}>{note}</p>;
  });

  return <div>
    {result}
  </div>;
}
```

Добавим инпут и кнопку с помощью которых можно будет реактивно добавлять новые абзацы. Мы будем вводить текст в input, жать на кнопку и после этого должен появится новый абзац с введенным нами текстом.
Для этого необходимо добавить новый элемент в стейт с массивом.
После этого HTML код автоматически изменится.

```
function App() {
  const [notes, setNotes] = useState([1, 2, 3, 4, 5]);
  const [value, setValue] = useState('');

  const result = notes.map((note, index) => {
    return <p key={index}>{note}</p>;
  });

  function addItem() {
    setNotes([...notes, value]);
  }

  function changeInput(event) {
    setValue(event.target.value);
  }

  return <div>
    {result}

    <input value={value} onChange={changeInput} />
    <button onClick={addItem}>add</button>
  </div>;
}
```

Можно переписать в короткой форме:

```
function App() {
  const [notes, setNotes] = useState([1, 2, 3, 4, 5]);
  const [value, setValue] = useState('');

  const result = notes.map((note, index) => {
    return <p key={index}>{note}</p>;
  });

  return <div>
    {result}

    <input value={value} onChange={event => setValue(event.target.value)} />
    <button onClick={() => setNotes([...notes, value])}>add</button>
  </div>;
}
```

Реализация удаления элементов из массива

Пусть есть массив, каждый элемент которого выводится в абзаце:

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

  const result = notes.map((note, index) => {
    return <p key={index}>{note}</p>;
  });

  return <div>
    {result}
  </div>;
}
```

Сделаем так, чтобы по клику на абзац происходило его удаление. Согласно идеологии React для этого нам нужно будет удалять соответствующий элемент массива. В этом случае React автоматически в соответствии с изменениями массива внесет изменения в HTML код.

Для начала давайте к каждому абзацу привяжем событие, в котором будет вызываться функция remItem для удаления:

```
const result = notes.map((note, index) => {
  return <p key={index} onClick={remItem}>{note}</p>;
});
```

Функция remItem должна знать номер абзаца для удаления. Это значит, что этот номер нужно передать параметром при вызове функции.
Номер каждого абзаца последовательно попадает в переменную index от цикла map. Передадим параметром функции эту переменную

```
const result = notes.map((note, index) => {
  return <p key={index} onClick={() => remItem(index)}>
    {note}
  </p>;
});
```

код для удаления элемента массива будет выглядеть следующим образом

```
function remItem(index) {
  setNotes([...notes.slice(0, index), ...notes.slice(index + 1)]);
}
```

Весь код

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

  const result = notes.map((note, index) => {
    return <p key={index} onClick={() => remItem(index)}>
      {note}
    </p>;
  });

  function remItem(index) {
    setNotes([...notes.slice(0, index), ...notes.slice(index + 1)]);
  }

  return <div>
    {result}
  </div>;
}
```

Задания

1. Дан массив:

```
const notes = ['a', 'b', 'c', 'd', 'e'];
```

Выведите элементы этого массива в виде списка ul. Добавьте инпут для добавления новых пунктов списка. Пусть добавление происходит по потери фокуса в инпуте.

2. Модифицируйте предыдущую задачу так, чтобы при добавлении новой li текст инпута очищался.

3. Дан массив:

```
const notes = ['a', 'b', 'c', 'd', 'e'];
```

Выведите элементы этого массива в виде списка ul. Сделайте так, чтобы в конце каждой li стояла кнопка для ее удаления.

Привязка инпутов к массиву

Пусть в стейте notes хранится массив:

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

И 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;
}
```

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>;
}
```

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>;
}
```

Добавим событие onChange 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>;
}
```

Функция changeHandler первым параметром принимает номер того элемента массива, который редактирует данный инпут.
По этому номеру можно заменить элемент массива на содержимое инпута.

```
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.

```
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>;
}
```

Задания

1. Дан 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.

Редактирование массива

Пусть есть массив, каждый элемент которого выводится в абзаце

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

  const result = notes.map((note, index) => {
    return <p key={index}>{note}</p>;
  });

  return <div>
    {result}
  </div>;
}
```

Сделаем возможность редактирования элементов массива. Для этого под абзацами сделаем input. Пусть по клику на абзац его текст появляется в этом инпуте. При редактировании текст Input и одновременно будет изменяться текст абзаца.
Для начала сделаем стейт editNum, хранящий номер редактируемого в данный момент элемента массива. Если же в данный момент ничего не редактируется пусть этот editNum имеет значение null.

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

Сделаем так, чтобы по клику на абзац в key записывался номер элемента массива, соответствующий этом

```
const result = notes.map((note, index) => {
  return <p key={index} onClick={() => setEditNum(index)}>
    {note}
  </p>;
});
```

Сделаем так, чтобы в input выводился текст редактируемого элемента массива:

```
<input value={notes[editNum]} />
```

Если editNum равен null, то мы получим ошибку. В этом случае в value инпута попадет undefined, что не разрешено React. Для решения проблемы можно использовать тернарный оператор:

```
<input value={editNum ? notes[editNum] : ''} />
```

Добавим Input событие onChange `<input value={editNum ? notes[editNum] : ''} onChange={changeItem} />`

Реализуем обработчик этого события. В нем мы будем заменять редактируемый элемент массива на текст из input

```
function changeItem(event) {
  setNotes([...notes.slice(0, editNum), event.target.value, ...notes.slice(editNum + 1)]);
}
```

соберем весь код

```
function App() {
  const [notes, setNotes] = useState([1, 2, 3, 4, 5]);
  const [editNum, setEditNum] = useState(null);

  const result = notes.map((note, index) => {
    return <p key={index} onClick={() => setEditNum(index)}>
      {note}
    </p>;
  });

  function changeItem(event) {
    setNotes([...notes.slice(0, editNum), event.target.value, ...notes.slice(editNum + 1)]);
  }

  return <div>
    {result}
    <input value={editNum ? notes[editNum] : ''} onChange={changeItem} />
  </div>;
}
```

Задания

1. Дан массив:

```
const notes = ['a', 'b', 'c', 'd', 'e'];
```

Выведите элементы этого массива в виде списка ul. Под списком реализуйте input для редактирования пунктов списка. Пусть в конце каждой li стоит кнопка, по нажатию на которую будет начинаться редактирование этой li.

2. Модифицируйте предыдущую задачу так, чтобы при потере фокуса в input его текст очищался.

Универсальная форма для изменения массива

```
function App() {
  const [notes, setNotes] = useState([1, 2, 3, 4, 5]);
  const [editNum, setEditNum] = useState(null);
  const [value, setValue] = useState('');

  const result = notes.map((note, index) => {
    return <p key={index} onClick={() => setEditNum(index)}>
      {note}
    </p>;
  });

  function changeItem(event) {
    setNotes([...notes.slice(0, editNum), event.target.value,...notes.slice(editNum + 1)]);
  }
  function stopEdit(event) {
    setEditNum(null);
  }

  function changeValue(event) {
    setValue(event.target.value)
  }
  function addItem(event) {
    setNotes([...notes, value]);
  }
```

```
  let input;
  if (editNum) {
    input = <input
      value={notes[editNum]}
      onChange={changeItem}
      onBlur={stopEdit}
    />
  } else {
    input = <input
      value={value}
      onChange={changeValue}
      onBlur={addItem}
    />
  }

  return <div>
    {result}
    {input}
  </div>;
}
```

Подход второй

```
unction App() {
const [notes, setNotes] = useState([1, 2, 3, 4, 5]);
const [editNum, setEditNum] = useState(null);
const [value, setValue] = useState('');

const result = notes.map((note, index) => {
  return <p key={index} onClick={() => startEdit(index)}>
    {note}
  </p>;
});

function startEdit(index) {
  setEditNum(index);
  setValue(notes[index]);
}
function changeHandler(event) {
  setValue(event.target.value);

  if (editNum) {
    setNotes([...notes.slice(0, editNum), event.target.value,...notes.slice(editNum + 1)]);
  }
}
  function blurHandler(event) {
    if (!editNum) {
      setNotes([...notes, value]);
    } else {
      setEditNum(null);
    }

    setValue('');
  }

  return <div>
    {result}
    <input value={value} onChange={changeHandler} onBlur={blurHandler} />
  </div>;
}
```

Подход третий

При добавлении нового элемента он сразу появляется в виде нового абзаца. И при наборе текста в инпуте в этом абзаце сразу набирается текст нового элемента.

```
function App() {
  const [notes, setNotes] = useState([1, 2, 3, 4, 5]);
  const [editNum, setEditNum] = useState(null);

  const result = notes.map((note, index) => {
    return <p key={index} onClick={() => startEdit(index)}>{note}</p>;
  });

  function startEdit(index) {
    setEditNum(index);
  }
  function editItem(event) {
    setNotes([...notes.slice(0, editNum), event.target.value,...notes.slice(editNum + 1)]);
  }
  function createItem() {
    if (!editNum) {
      const res = [...notes, ''];
      setNotes(res);
      setEditNum(res.length - 1);
    }
  }
  function stopEdit() {
    setEditNum(null);
  }

  return <div>
    {result}

    <input
      value={editNum ? notes[editNum] : ''}
      onChange={editItem}
      onFocus={createItem}
      onBlur={stopEdit}
    />
  </div>;
}
```

Задания

1. Дан массив: `const notes = ['a', 'b', 'c', 'd', 'e'];`

Выведите элементы этого массива в виде списка ul. Под списком реализуйте инпут для редактирования существующих и добавления новых пунктов списка. Решите задачу тремя описанными подходами.