

JavaScript. Уровень 3. React и JSX

<https://codepen.io/anon/pen/bQober?editors=0010>

Модуль 1. Введение в React и JSX

Что такое React?

- библиотека построения UI

Какие задачи решает React?

- Декларативный подход
- Компоненто-ориентированность
- Однонаправленный поток данных
- Использование ES2015
- делает быстро

Полезные ссылки

- <https://reactjs.org/>
- <https://codepen.io/>
- <http://htmlab.ru/react/>

Установка библиотеки, ES6 и JSX

- Babel <https://babeljs.io/docs/setup/#installation>

Встраиваемые выражения

```
const element = ( <h1> Hello, {formatName(user)}! </h1> );
```

Определение атрибутов в JSX

```
const element = <div tabIndex="0"></div>;  
const element = <img src={user.avatarUrl}></img>;
```

Указание дочерних элементов

```
const element = (  
  <h1 className="greeting">  
    Hello, world!  
  </h1>  
);
```

React-элементы

```
const element = React.createElement(  
  'h1',  
  {className: 'greeting'},  
  'Hello, world!'  
);
```

Отрисовка элементов


```
<div id="root"></div>  
  
const element = <h1>Hello, world</h1>;  
ReactDOM.render(  
  element,  
  document.getElementById('root')  
);
```

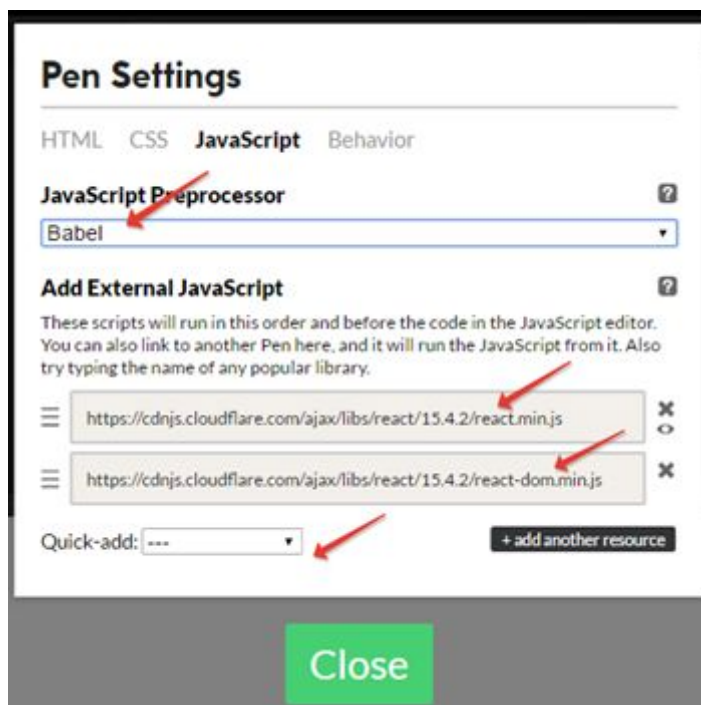
<https://codepen.io/gaearon/pen/rpvgNB?editors=1010>

Практическая работа №0 Первое знакомство с библиотекой React

1. Выберите вариант работы <http://codepen.io> или локальная работа (второй вариант более приоритетен для курса)

codepen.io:

1. Зайдите на ресурс <http://codepen.io/pen/>
2. Выберите в настройках 
3. В пункте «JavaScript Preprocessor» выберите Babel
4. В выпадающем списке «Quick-add» добавьте последовательно:
 - React
 - React DOM



5. Нажмите «Close»
6. В раздел HTML введите:

```
<div id="app"></div>
```

7. В раздел JS(Babel) введите:

```
ReactDOM.render(  
  <h1>Hello, world!</h1>,  
  document.getElementById('root')  
)
```

8. Результат будет выглядеть так:

Hello, world!

9. Примечание: В этой и последующих работах считается, что HTML –код всегда состоит из `<div id="app"></div>`

Локальная работа

1. Создайте/откройте файл **index.html**
2. Подключите React и Babel

```
<script
src='https://cdnjs.cloudflare.com/ajax/libs/react/16.2.0/umd/react.devel
opment.js'></script>
<script
src='https://cdnjs.cloudflare.com/ajax/libs/react-dom/16.2.0/umd/react-d
om.development.js'></script>
<script
src="https://unpkg.com/babel-standalone@6/babel.min.js"></script>
```

3. Подключите библиотеку контроля типов prop-types.js

```
<script src="https://unpkg.com/prop-types@15.6/prop-types.js"></script>
```

4. Подключите набор данных с описанием книг

```
<script src="js/data.js"></script>
```

5. Создайте блок для работы со скриптом с учётом babel

```
<script type="text/babel">
ReactDOM.render(
  <h1>Hello, world!</h1>,
  document.getElementById('app')
);
</script>
```

или

```
<script type="text/babel" src="js/Б А Ш С К Р И П Т .js"></script>
```

Практическая работа №1. Знакомство с синтаксическим расширением JavaScript (syntax extension to JavaScript)

Упражнение 1.1 Встраивание выражений в JSX

1. Вместо «**Hello, world!**» вставьте фразу «**Book 1**»
2. Поменяйте **h1** на **h3**
3. Убедитесь, что вместо «**Hello, world!**» теперь выводится «**Book 1**»
4. Перенесите JSX `<h3>Book 1</h3>` в константу **book**
5. Константу **book** пропишите первым аргументом метода **ReactDOM.render()**
6. Убедитесь, что на экране отображается «**Book 1**»
7. Введите содержимое константы в текстовое поле на сайте транспайлера <https://babeljs.io/repl>
8. Познакомьтесь с получившимся кодом
9. Убедитесь, что в файле подключенном файле **data.js** существует константа **dataBook** с данными типа:

```
const dataBook = [  
  {id:1, title:"Book 1",author:"Author 1", price: 500},  
  {id:5, title:"Book 2",author:"Author 2", price: 1200},  
];
```

10. Встройте название первой книги в константу **book** так, чтобы вывод на экране не поменялся:

```
const book = <h3>{dataBook[0]["title"]}</h3>;  
ReactDOM.render(  
  book,  
  document.getElementById('app')  
);
```

11. Измените JSX константы **book** так, чтобы выводилась также информация об авторе и цене книги. Примечание: автор и цена должны выводиться в параграфах с подписями, например `<p>Автор: имяАвтора</p>`, где **имяАвтора** – значение поля **author**.
12. Убедитесь, что браузер отображает описание книги. Примечание: все характеристики книги должны быть заключены в обертывающий элемент с классом **book**
13. Если есть ошибки – исправьте их

Упражнение 1.2 JSX – выражения

1. Добавьте в **dataBook** новую книгу с полем **price** равным **null**:

```
const dataBook = [  
  {id:1, title:"Book 1",author:"Author 1", price: 500},  
  {id:5, title:"Book 2",author:"Author 2", price: 1200},  
  {id:6, title:"Book 3",author:"Author 3", price: null}  
];
```

2. Создайте функцию `formatPrice(price)`, которая при значении `price` равным `null` будет ставить прочерк в цене:

```
function formatPrice(price) {  
  return price ? <strong>{price}</strong> : <del>&nbsp;</del>;  
}
```

3. Выведите третью книгу из **dataBook** (книга с идентификатором **6**):

```
const book = (  
  <div className="book">  
    <h3>{dataBook[2]["title"]}</h3>  
    <p>А в т о р : {dataBook[2]["author"]}</p>  
    <p>Ц е н а : {formatPrice(dataBook[2]["price"])} р у б .</p>  
  </div>  
)
```

4. Убедитесь, что для выводимой книги ставится прочерк в цене

Упражнение 1.3 Определение атрибутов с JSX

1. Добавьте в вывод изображение книги. Примечание: чтобы упростить работу, воспользуйтесь ссылками вида <http://placeholder.it/100x120>:

```
const book = (  
  <div className="book">  
    <h3>{dataBook[2]["title"]}</h3>  
    <img  
      src={`http://placeholder.it/100x120?text='+dataBook[2]['title']}<br/>  
      alt="" />  
  </div>  
)
```

```
<p>А в т о р : {dataBook[2]["author"]}</p>  
<p>Ц е н а : {formatPrice(dataBook[2]["price"])} р у б .</p>  
</div>  
>;
```

Упражнение 1.4 Вывод всех данных (*Если будет время)

1. Напишите JSX, который выведет информацию обо всех объектах **dataBook**

Обновление элементов

- виртуальный DOM

```
function tick() {  
  const element = (  
    <div>  
      <h1>Hello, world!</h1>  
      <h2>It is {new Date().toLocaleTimeString()}.</h2>  
    </div>  
  );  
  ReactDOM.render(  
    element,  
    document.getElementById('root')  
  );  
}  
setInterval(tick, 1000);
```

<https://codepen.io/gaearon/pen/gwoJZk?editors=0010>

Тест

<https://goo.gl/forms/5zIF6SQWchRRG2Tm1>

Модуль 2. React-компоненты, состояния и жизненный цикл

Функциональные и классовые компоненты

- stateless
- классы ES2015

Stateless-синтаксис

```
function Welcome(props) {  
  return <h1>Hello, {props.name}</h1>;  
}  
<Welcome name="В а с и л и й" />
```

Компонент с классами ES2015

```
class Welcome extends React.Component {  
  render() {  
    return <h1>Hello, {this.props.name}</h1>;  
  }  
}  
<Welcome name="В а с и л и й" />
```

Значения props по умолчанию

```
Welcome.defaultProps = {  
  name: 'Г О С Т Ь'  
}
```

PropTypes

```
optionalArray: PropTypes.array,  
optionalBool: PropTypes.bool,  
optionalFunc: PropTypes.func,  
optionalNumber: PropTypes.number,  
optionalObject: PropTypes.object,
```

```
optionalString: PropTypes.string,  
optionalSymbol: PropTypes.symbol
```

```
PropTypes.func.isRequired
```

<https://reactjs.org/docs/typechecking-with-proptypes.html>
<https://www.npmjs.com/package/prop-types>

Пример использования proptypes

```
Welcome.propTypes = {  
  name: PropTypes.string  
};
```

Композиция элементов

```
function App() {  
  return dataBook.map( item => <Book  
    title={item["title"]}   
    author={item["author"]}   
    price={item["price"]}   
  />);  
}  
ReactDOM.render(  
  <App />,  
  document.getElementById("app")  
)
```

<http://htmlab.ru/react-composition-vs-inheritance/>

Извлечение компонентов

- изначальное описание компонентов
- выделение из страницы
- целесообразность выделения компонентов

Понятие состояния

- состояние
- доступность состояний в компонентах

Преобразование функции в класс

- <http://htmlab.ru/react-state/>

```
class Clock extends React.Component {  
  render() {  
    return (  
      <div>  
        <h1>Hello, world!</h1>  
        <h2>It is {this.props.date.toLocaleTimeString()}.</h2>  
      </div>  
    );  
  }  
}
```

<https://codepen.io/gaearon/pen/zKRGpo?editors=0010>

Локальное состояние класса

```
class Clock extends React.Component {  
  constructor(props) {  
    super(props);  
    this.state = {date: new Date()};  
  }  
  render() {  
    ...  
  }  
}
```

Методы жизненного цикла

- **componentWillMount** (перед добавлением в DOM)
- **componentDidMount** (Вызывается, когда компонент подключен к DOM. Хорошо для выполнения запросов AJAX)
- **componentDidUpdate** (Вызывается сразу после обновления. Хорошо для запросов AJAX, основанных на изменении свойств)
- **componentWillUnmount** (Вызывается НЕМЕДЛЕННО, пока компонент не размонтирован. Хорошо для очистки слушателей)

Рекомендации по работе с состояниями

- меняйте состояние только через **setState()**
- обновление состояния может быть асинхронным
- Обновление части состояния

Нисходящие потоки данных

- локальное состояние доступно только компоненту
- компонент может передать состояние в качестве свойств дочерним компонентам

ref

Что такое **ref**?

- Механизм обращения к реальным DOM-элементам
- Атрибут для любого компонента. Атрибут содержит функцию обратного вызова

Когда используются

- Управление фокусом, выделениями текста
- Обработка описательной анимации (imperative animations)
- Интеграция сторонних DOM-библиотек
- <https://facebook.github.io/react/docs/refs-and-the-dom.html>

Callback refs

```
this.textInput = null;
```

```
this.setTextInputRef = element => {  
  this.textInput = element;  
};
```

```
<input type="text" ref={this.setTextInputRef} />
```

Практическая работа №2

Практическая работа направлена на закрепление навыков создания React-компонентов функциональным и классовым способами, приобретения опыта работы с свойствами (**props**, пропсы) и состоянием (**state**) компонентов

Упражнение 2.1 Описание React-компонента через функцию

1. Вместо константы **book**, создайте функциональный React-компонент **Book()**.

Примечание: название компонента всегда записывается с заглавной буквы, а возвращается JSX:

```
function Book(props) {  
  const price = price ? <strong>price</strong> : <del>&nbsp;</del>;  
  return <div className="book">  
    <h3>{props.title}</h3>  
    <img src={'http://placeholder.it/100x120?text='+props.title} alt="" />  
    <p>А в т о р : {props.author}</p>  
    <p>Ц е н а : {price} р у б .</p>  
  </div> ;  
}
```

2. В метод **ReactDOM.render()** первым аргументом подставьте созданный функциональный React-компонент:

```
<Book title={dataBook[2]["title"]} author={dataBook[2]["author"]} price={dataBook[2]["price"]} />
```

3. Убедитесь, что вывод информации остаётся прежним. Примечание: он будет похож на

4. Обратите внимание как аргументы из **Book** передаются в **props**!

Упражнение 2.2 Компонент App и композиция

1. Создайте функциональный React-компонент **App** и поместите его первым аргументом в **render()**:

```
ReactDOM.render(  
  <App />,  
  document.getElementById('root')  
)
```

2. **App** должен выводить информацию обо всех текущих книгах в **dataBook**
3. Убедитесь, что выводятся все книги из **dataBook**.
4. Если есть ошибки, исправьте их
5. Примечание: код **App** может быть похож на следующий:

```
function App(){
  return <div>
    <Book title={dataBook[0]["title"]} author={dataBook[0]["author"]}
price={dataBook[0]["price"]} />
    <Book title={dataBook[1]["title"]} author={dataBook[1]["author"]}
price={dataBook[1]["price"]} />
    <Book title={dataBook[2]["title"]} author={dataBook[2]["author"]}
price={dataBook[2]["price"]} />
  </div>
}
```

Упражнение 2.3 Конвертация функционального компонента в класс

1. Перепишите компонент **Book** в класс, следуя алгоритму:
 - a. Создайте ES6 класс, расширяющий **React.Component**. У класса должно быть некоторое имя
 - b. Добавьте классу метод **render()**
 - c. Перенесите тело функции в метод **render()**
 - d. Замените **props** на **this.props** в теле **render()**
 - e. Удалите пустое описание функции
2. Перепишите на класс компонент **App**:

```
class App extends React.Component {
  render(){
    return <div>
      <Book title={dataBook[0]["title"]} author={dataBook[0]["author"]}
price={dataBook[0]["price"]} />
      <Book title={dataBook[1]["title"]} author={dataBook[1]["author"]}
price={dataBook[1]["price"]} />
      <Book title={dataBook[2]["title"]} author={dataBook[2]["author"]}
price={dataBook[2]["price"]} />
    </div>
  }
}
```

3. Если появятся ошибки – исправьте их.

Упражнение 2.4 Добавление локального состояния

1. Добавьте классу **Book** конструктор: единственным аргументом будет **prop**; родительские свойства будут наследоваться (**super(props)**)

```
constructor(props) {  
    super(props);  
}
```

2. Добавьте компоненту **Book** начальное состояние **selected**, обозначающее логическую величину – выбрана книга или нет

```
this.state = {selected: false}
```

3. Перепишите **render()** так, чтобы в **<div className="book">** по умолчанию выводился ещё один класс.

- a. **book-default** – если **selected** является **false**
- b. **book-selected** – если **selected** является **true**

Примечание: для проверки работы подсмотрите DOM-структуру документа и/или добавьте CSS-свойства с селекторами указанных классов

4. Убедитесь, что если поменять **this.state.selected** на **false**, элемент приобретёт соответствующие классы **book** и **book-selected**

5. Добавьте в компонент **Book** две гиперссылки «Сравнить» и «В корзину».

Примечание: **href** у ссылок будет содержать «#»

Упражнение 2.5 Добавление методов жизненного цикла

1. Добавьте классу **Book** методы жизненного цикла:

```
componentDidMount() {  
    console.log('--', 'компонент смонтирован')  
}  
componentWillUnmount() {  
    console.log('--', 'компонент будет демонтирован')  
}
```

2. Проверьте сообщение в консоли браузера

Упражнение 2.6 Изменение состояния React-компонента

1. Проверьте начальное состояние **Book**, поле **selected** должно быть **false**

2. Откройте код метода **render()**
3. Перед return измените состояние selected с **false** на **true**
4. Убедитесь, что все выводимые элементы помечены **book-selected**
5. Если есть ошибки – исправьте их

Упражнение 2.7 Создание ключей (key)

При выводе компонентов, пропишите у каждого уникальный атрибут **key**

Тест

<https://goo.gl/forms/KMx4b8WkasIN4e162>

Модуль 3. Условная отрисовка. Формы

Отрисовка с условием

```
flag ? <Book ... /> : <BookWithoutPrice ... />
```

Формы и события

```
<form action="" onSubmit={this.handleSubmit} >
  handleSubmit(ev){
    ev.preventDefault();
    ...
  }
  this.handleSubmit = this.handleSubmit.bind(this);
```

Предотвращение отрисовки компонента

```
return null;
```

Отрисовка нескольких компонентов

```
- map

const numbers = [1, 2, 3, 4, 5];
const listItems = numbers.map((number) =>
  <li>{number}</li>
);
```

Основной компонент списка

```
function NumberList(props) {
  const numbers = props.numbers;
  const listItems = numbers.map((number) =>
    <li key={number.toString()}>
      {number}
    </li>
  );
  return (
    <ul>{listItems}</ul>
  );
}
```

```
    );  
  }  
  
  const numbers = [1, 2, 3, 4, 5];  
  ReactDOM.render(  
    <NumberList numbers={numbers} />,  
    document.getElementById('root')  
  );
```

Ключи

- помогают React обрабатывать дерево
- должны быть уникальны

Извлечение компонентов с ключами

- имеют смысл только в контексте окружающего массива данных
- Например, если вы извлекаете компонент **ListItem**, вы должны сохранить ключ на элементах **<ListItem />** в массиве, а не на корень **** элемента в **ListItem** itself.

Контролируемые компоненты и теги

- `<input>`, `<textarea>` и `<select>`
- <https://reactjs.org/docs/forms.html#controlled-components>
- `<input type="file">` - неконтролируемый компонент

Альтернатива контролируемым компонентам

- сложности в использовании управляемых компонентов
- неконтролируемые компоненты
- <https://reactjs.org/docs/uncontrolled-components.html>

Практическая работа 3: Обработка событий и условная отрисовка

В практической работе затрагивается механизм обработки событий на отдельных компонентах. React использует синтетические события, в соответствии с спецификацией W3C, потому вопрос с кроссбраузерностью перед нами не возникает. Примечание: при работе с React не нужно использовать `addEventListener`. Рассматривается ещё одна тема – создание компонента форма и работа с ним.

Упражнение 3.1 Назначение обработчика события

1. Опишите в гиперссылке «Сравнить» обработчик события:

```
onClick={this.handleClick}
```

2. В классе создайте обработчик **handleClick()**. Примечание: он должен принимать аргумент `e`
3. Отмените действие по умолчанию
4. Проверьте работу по нажатию на ссылку. Примечание: можно вызывать **alert()** или **console.log()**
5. Перенесите строку с **this.state()** в **handleClick()**
6. Убедитесь, что при нажатии на «Сравнить» происходит выделение элемента за счет изменения состояния
7. Если событие не срабатывает, проверьте правильно ли указывается `this`:

```
this.handleClick = this.handleClick.bind(this); // в  
конструкторе
```

8. Сделайте так, чтобы при повторном нажатии на «Сравнить», выделение снималось

Упражнение 3.2 Условная отрисовка

1. В классе **App**, в методе **render()**, создайте константу **books** и при помощи **map()** поместите все данные из **dataBook** в **books**
2. В операторе `return` вместо нескольких компонентов **Book** укажите **books** как выражение в JSX

3. Создайте компонент **BookWithoutPrice**, который не будет отображать ссылки, если цена у книги указана как **null**
4. Измените функцию, возвращающую результат в **books** так, чтобы при отсутствии цены возвращался компонент **BookWithoutPrice**
5. Убедитесь, что книга с **price** равным **null**, не содержит ссылок «Сравнить» и «В корзину»

Упражнение 3.3 Работа с элементами форм в React

1. Создайте новый компонент **AddBookForm**:

```
<form action="">
  <div>id <input type="text" name="id" /></div>
  <div>Название <input type="text" name="title" /></div>
  <div>Авторы <input type="text" name="id" /></div>
  <div>Цена <input type="text" name="id" /></div>
  <div><input type="submit" value="Добавить" /></div>
</form>
```

2. В конструкторе **AddBookForm**
 - a. унаследуйте props
 - b. укажите состояние из полей одноимённых названию полей формы

```
this.state = {
  id: 0,
  title: '',
  auhtor: '',
  price: null
}
```

- c. укажите правильную привязку **this** в методах **handleSubmit()** и **handleChange()**. Примечание: методы будут созданы в этом упражнении позже

3. Встройте вывод **AddBookForm** перед списком всех книг
4. Укажите обработчик отправки формы **handleSubmit()** и обработчик изменения каждого поля **handleChange()**
5. Реализуйте в **AddBookForm** метод **handleSubmit()**. Метод должен проверять данные состояния и добавлять новый элемент в **dataBook**. Примечание: отмените действие по умолчанию
6. Проверьте через консоль, что **handleSubmit()** действительно добавляет новый элемент в **dataBook**. Например,

```
isValidBook(book){
  console.log(book) //только для проверки
```

```

    if (book.id && book.title && book.author )
        return true;
    return false;
}
handleSubmit (ev) {
    ev.preventDefault()
    if( this.isValidBook(this.state) ){
        dataBook.push(this.state);
        console.log(dataBook); //только для проверки
    } else alert("Заполните поля корректно")
}

```

7. Опишите **handleChange()** так, чтобы поля заполнялись через значения состояния **AddBookForm**, а сам метод менял состояние через **this.setState()**
8. Убедитесь, что происходит корректное добавление новых элементов в **dataBook**.
9. При возникновении ошибок, проверьте свой код и исправьте ошибки

```

handleChange (ev, field) {
    ev.preventDefault()
    const input = {}
    input[ev.target.name] = ev.target.value
    this.setState(input)
    console.log(input)
}

render (){
    return <form action="" onSubmit={this.handleSubmit}>
        <div>id <input type="text" name="id" onChange={this.handleChange}
value={this.state.id?this.state.id:''}/></div>
        <div>Название <input type="text" name="title"
onChange={this.handleChange}
value={this.state.title?this.state.title:''} /></div>
        <div>Авторы <input type="text" name="author"
onChange={this.handleChange}
value={this.state.author?this.state.author:''} /></div>
        <div>Цена <input type="text" name="price" onChange={ev =>
{this.handleChange(ev, 'price')}}
value={this.state.price?this.state.price:''} /></div>
        <div><input type="submit" value="Добавить" /></div>
    </form> }

```

Тест

<https://goo.gl/forms/xQ70rb1tGvqoUBpP2>

Модуль 4. Всплытие состояний

Всплытие состояний

- компоненты могут и/или должны отражать изменения данных
- рекомендуется поднимать общее состояние до ближайшего общего предка
- Пример кода со всплытием: <http://htmlab.ru/react-lifting-state-up/>

Практическая работа №4

React-компоненты часто образуют композицию. В этой практической работе, идёт речь о том, как родительский элемент может узнать об изменении состояния дочернего. В терминологии React это называется «всплытием состояния»

Упражнение 4.1 Организация всплытия состояния

1. Создайте в компоненте **App** состояние **items**
2. В **App** создайте метод **addBasket()**, для добавления книги в **items**
3. В конструкторе **App** укажите корректную привязку привязку **this**

```
constructor(props) {  
  super(props);  
  this.state = {items: {}}  
  this.addBasket = this.addBasket.bind(this);  
}
```

4. Заполните тело метода addBasket():

```
addBasket(id){  
  let items = Object.assign({},this.state.items);  
  items[id] = (id in items) ? items[id]+1 : 1;  
  console.log(items);  
  this.setState({ items: items});  
}
```

5. В методе **render()** компонента **App** укажите свойство (props) **handleAddBasket()** значение которого будет **{this.addBasket}**:

```
render(){  
  const books = dataBook.map(book => {  
    if(book.price)  
    return <Book  
      id={book["id"]}   
      title={book["title"]}   
      author={book["author"]}   
      price={book["price"]}   
      key={book.id}   
      handleAddBasket={this.addBasket}   
    />  
    else  
    return <BookWithoutPrice title={book["title"]}   

```



```

            author={book["author"]}
            key={book.id}
        />
    }
)
return <div>
  <AddBookForm    />
  {books}
</div>
}

```

Примечание: мы передаём через **props** функцию, которую будем вызывать при добавлении товара в **items** при нажатии на «В корзину» в React-компоненте **Book**

6. В компоненте **Book** в **render()** добавьте обработчик нажатия на ссылку «В корзину»:

```
<a href="#" onClick={this.addBasketBook}>В корзину</a>
```

7. Опишите обработчик **addBasketBook()**. Примечание: отменить действие по умолчанию
8. Получите из **props** функцию **handleAddBasket()** и вызовите её с текущим номером книги:

```

addBasketBook(e){
  e.preventDefault();
  const handleAddBasket = this.props.handleAddBasket;
  handleAddBasket(this.props.id)
}

```

9. Убедитесь, что **state.items** в **App** пополняется новыми элементами

Упражнение 4.2 Добавление компонента Basket

1. Создайте React-компонент класс **Basket**:

```

class Basket extends React.Component {
  constructor(props) {
    super(props);
    this.deleteBasketItem = this.deleteBasketItem.bind(this);
  }

  deleteBasketItem(e){
    e.preventDefault()
    const handleRemoveBasket = this.props.handleRemoveBasket

```

```

    const i = e.target.id
    //if(confirm("Точно хотите удалить?"))
    handleRemoveBasket(i)
  }

  getIndexById(id){
    for(let p in dataBook)
      if (dataBook[p]['id'] == id)
        return p
  }

  render (){
    let items = [], j, sum = 0;
    for(let i in this.props.items){
      //console.log(i + " = " + this.props.items[i] + " (" + this.getIndexById(i) + ") ")
      j = this.getIndexById(i)
      sum += this.props.items[i] * dataBook[j]['price']
      items.push(
        <div className="basket-item">
          <a href="#">«{dataBook[j]['title']}»</a>
          <span>{this.props.items[i]}шт</span>
          <span>{dataBook[j]['price']}руб</span>
          <a href="#" onClick={this.deleteBasketItem} id={i}>Удалить</a>
        </div>
      );
    }

    items.push(
      <div className="basket-item">
        <span>Всего <strong>{sum}</strong> руб.</span>
      </div>
    );

    return <div className='basket'>
      <h3>Корзина</h3>
      {items}
    </div> ;
  }
}

```

2. Внесите изменения в компонент App:

```

class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {items: []}
    this.addBasket = this.addBasket.bind(this);
  }
}

```

```

    this.removeBasket = this.removeBasket.bind(this);
}

addBasket (id){
    let items = this.state.items.slice(0);
    if( id in items )
        items[id]++
    else
        items[id]=1
    this.setState({items: items})
    // console.log(this.state.items)
}

removeBasket (id){
    let items = this.state.items.slice(0), result = []

    for(let i in items )
        if(i !== id)
            result[i] = items[i]
    /*console.log("Удаляем ", id)
    console.log("Исходный ", items)
    console.log("Результирующий ", result)*/
    this.setState({items: result})
}

render(){
    const books = dataBook.map(book => {
        if(book.price)
            return <Book
                id={book["id"]}
                title={book["title"]}
                author={book["author"]}
                price={book["price"]}
                key={book.id}
                handleAddBasket={this.addBasket}
            />
        else
            return <BookWithoutPrice title={book["title"]}
                author={book["author"]}
                key={book.id}
            />
    })

    return <div>
        <Basket items={this.state.items} handleRemoveBasket={this.removeBasket}/>
        <AddBookForm />
    </div>
}

```

```
        {books}  
      </div>  
    }  
  }
```

* Создайте компонент `BasketItem` и измените соответствующую часть метода `render()` класса `Basket`

Тест

<https://goo.gl/forms/bGKykw2yyWAr0tCD3>

Введение в React Router

Что такое React Router?

- React дополнение для маршрутизации в приложении

Подключение React Router

<https://cdnjs.cloudflare.com/ajax/libs/react/16.2.0/umd/react.development.js>
<https://cdnjs.cloudflare.com/ajax/libs/react-dom/16.2.0/umd/react-dom.development.js>
<https://cdnjs.cloudflare.com/ajax/libs/react-router-dom/4.2.2/react-router-dom.js>

Компоненты React Router

- BrowserRouter / HashRouter
- Link, NavLink
- Route, Switch

Пример использования компонентов

```
<HashRouter>
  <div>
    <nav>
      <Link to={` /about`} >О нас</Link>
      <Link to={` /services`} >Услуги</Link>
    </nav>
    <main>
      <Switch>
        <Route path="/about" component={About} />
        <Route path="/services" component={Services} />
      </Switch>
    </main>
  </div>
</HashRouter>
```

Бонус

<https://www.youtube.com/watch?v=OS5s4lLzX8s&list=PLOQDek48BpZGA7ulddqzvQ68vyKdfD6tf&index=6>

<https://www.youtube.com/watch?v=CEiMOwBObNk&list=PLOQDek48BpZGA7ulddgzvQ68vyKdfD6tf&index=7>