Давайте разберем работу с циклами и условиями if подробнее и научимся применять их на практике.

Примечание

Когда мы изучали вставку значений переменных в кусочки HTML кода, я писал о том, что в этих фигурных скобках можно выполнить произвольный JavaScript код. На самом деле там можно выполнить не любой код, а только самый примитивный.

К примеру, обычный **if** там написать нельзя - это не будет работать. Но иногда бывает нужно. В этом случае нам на помощь придет тернарный оператор - его использовать можно.

Пусть у нас в стейте есть элемент **hello**, который может принимать значение true или false. Давайте сделаем следующее: выведем на экран слово 'привет', если **this.state.hello** равен true, и 'пока', если **this.state.hello** равен false. Воспользуемся тернарным оператором:

```
class App extends React.Component {
	constructor() {
		super();
		this.state = {hello: true};
	}

	render() {
		//Выведет или 'привет', или 'пока':
		return <div>
			{this.state.hello ? 'привет' : 'пока'}
		</div>;
	}
}
```

Запустите этот код - вы увидите слово 'привет'. А теперь поменяйте this.state.hello на false в строчке **this.state = {hello: true}** и запустите код еще раз - вы увидите слово 'пока'.

Давайте теперь сделаем кнопку, по нажатию на которую 'привет' поменяется на 'пока':

```
class App extends React.Component {
	constructor() {
		super();
		this.state = {hello: true};
	}

	//Меняем состояние this.state.hello на false:
```

```
	changeText() {
		this.setState({hello: false});
	}

	render() {
		return <div>
			{this.state.hello ? 'привет' : 'пока'}
			<button onClick={this.changeText.bind(this)}>нажми на
меня</button>
		</div>;
	}
}
```

Запустите этот код и нажмите на кнопку - слово 'привет' мгновенно поменяется на 'пока'.

А теперь сделаем кнопку, по нажатию на которую 'привет' будет меняться на 'пока' и наоборот:

```
class App extends React.Component {
	constructor() {
		super();
		this.state = {hello: true};
	}

	//Меняем состояние this.state.hello на противоположное:
	toggleText() {
		this.setState({hello: !this.state.hello});
	}

	render() {
		return <div>
			{this.state.hello ? 'привет' : 'пока'}
			<button onClick={this.toggleText.bind(this)}>нажми на
меня</button>
		</div>;
	}
}
```

Запустите этот код и понажимайте на кнопку - слово 'привет' будет на 'пока' и наоборот.

Как это работает: вся магия в строке **this.setState({hello: !this.state.hello})**. С помощью **this.setState** мы в **this.state.hello** запишем обратное тому, что там есть сейчас. То есть: если там сейчас true - то запишем false и наоборот.

Так как у нас действует **реактивность**, то любые изменения с **this.state.hello**, производимые через **this.setState** будут мгновенно отображаться на экране - и мы будем видеть то 'привет', то 'пока'.

Запустите этот код и убедитесь в этом сами.

Показ элемента по нажатию на кнопку

Пусть у нас есть переменная **message** с текстом ** Привет! **. Давайте выведем ее значение внутри тега **div**:

```
class App extends React.Component {
	render() {
		const message = <p>Привет!</p>;

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

Пусть теперь у нас в **this.state** есть элемент **show**, который может принимать два значения: или **true**, или **false**:

```
class App extends React.Component {
	constructor() {
		super();
		this.state = {show: true};
	}
}
```

Давайте добавим **if**, который в зависимости от значения **this.state.show** будет или показывать, или скрывать блок текста, который хранится в переменной **message**:

```
class App extends React.Component {
	constructor() {
		super();
		this.state = {show: true};
	}

	render() {

		//Показ или сокрытие блока:
		if (this.state.show) {
			var message = <p>Привет!</p>;
		}

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

Для определения переменной **message** используем **var**, так как **const** и **let** ограничивают область видимости переменной только внутри ифа. Здесь также можно использовать **let**, если вначале определить переменную над ифом.

Как это работает: если мы попали вовнутрь ифа, то в переменную **message** запишется текстовый блок. Если же мы не попали в иф - то переменная **message** будет пуста и тут - ** {message} ** - вместо **message** ничего не выведется и никакой ошибки при этом не будет.

То есть: *можно выводить на экран переменные, значение которых не задано*. В этом случае вместо них ничего не вставится и ошибки не будет.

Давайте теперь сделаем метод **hideText**, который будет изменять **this.state.show** на **false**:

```
class App extends React.Component {
	constructor() {
		super();
		this.state = {show: true};
	}

	//Вызов метода изменит this.state.show на false:
	hideText() {
		this.setState({show: false});
	}

	render() {
		...
	}
}
```

Что будет после вызова метода **hideText**: свойство **this.state.show** изменится на false, что заставит метод **render** автоматически перезапуститься, при перезапуске переменная **message** станет пустой и наш текстовый блок пропадет.

Давайте сделаем кнопку, по нажатию на которую должен будет скрываться текст, и привяжем к ней метод **hideText**:

```
class App extends React.Component {
	constructor() {
		...
	}

	render() {
		if (this.state.show) {
			var message = <p>Привет!</p>;
```

```
            }

            //Сделаем кнопку:
            return <div>
                  {message}
                  <button onClick={this.hideText.bind(this)}>скрыть
текст</button>
            </div>;
      }
}
```

Соберем все вместе - теперь по нажатию на кнопку текст будет скрываться:

```
class App extends React.Component {
      constructor() {
            super();
            this.state = {show: true};
      }

      //Меняет this.state.show на false:
      hideText() {
            this.setState({show: false});
      }

      render() {

            //Показываем текст, только если this.state.show равен true:
            if (this.state.show) {
                  var message = <p>Привет!</p>;
            }

            return <div>
                  {message}
                  <button onClick={this.hideText.bind(this)}>скрыть
текст</button>
            </div>;
      }
}
```

Запустите этот код и вы увидите следующее: изначально текст из переменной **message** будет на экране, а по нажатию на кнопку - пропадет.

Сокрытие и показ элемента

Давайте теперь сделаем так, чтобы по первому нажатию на кнопку элемент скрывался, а по второму нажатию - показывался.

Для этого сделаем метод **toggleText**, который будет изменять **this.state.show** на противоположное значение:

```
class App extends React.Component {
	constructor() {
		super();
		this.state = {show: true};
	}

	//Меняем состояние this.state.show на противоположное:
	toggleText() {
		this.setState({show: !this.state.show});
	}
}
```

Как работает эта конструкции: в строчке **show: !this.state.show** прочитывается текущее значение **this.state.show** и меняется на противоположное с помощью восклицательного знака.

Давайте внесем правки в наш код:

```
class App extends React.Component {
	constructor() {
		super();
		this.state = {show: true};
	}

	//Меняем состояние this.state.show на противоположное:
	toggleText() {
		this.setState({show: !this.state.show});
	}

	render() {
		//Показываем текст, только если this.state.show равен true:
		if (this.state.show) {
			var message = <p>Привет!</p>;
		}

		return <div>
			{message}
			<button onClick={this.toggleText.bind(this)}>нажми на
меня</button>
		</div>;
	}
}
```

Запустите этот код и понажимайте на кнопку - текст будет то появлятся, то скрываться, как мы и задумали.

Стейты и циклы

Пусть у нас в стейте хранится массив:

```
class App extends React.Component {
	constructor() {
		super();

		//Дан массив:
		this.state = {items: [1, 2, 3, 4, 5]};
	}
}
```

Давайте переберем эти элементы этого массива циклом и запишем каждый из них в свою **li**, которые разместим в общем в теге **ul**.

То есть мы хотим получить следующее:

```
<ul>
	<li>1</li>
	<li>2</li>
	<li>3</li>
	<li>4</li>
	<li>5</li>
</ul>
```

Для этого наш массив следует перебрать циклом, например циклом map, сделать в этом цикле набор лишек и записать их в какую-либо переменную, назовем ее **list**:

```
const list = this.state.items.map(function(item, index) {
	return <li key={index}>{item}</li>;
});
```

Обратите внимание на атрибут **key**, который мы обязаны ввести для корректной работы React.

Внутри цикла **map** мы использовали анонимную функцию. Однако, тут более удобным будет использовать стрелочную функцию. Работу с ними мы проходили в уроке про нововведения ES6 (стрелочные функции в конце урока).

Давайте переделаем наш код на стрелочную функцию:

```
const list = this.state.items.map((item, index) => {
	return <li key={index}>{item}</li>;
});
```

Итак, результат работы цикла попадает в переменную **list**. Эту переменную следует вставить в теге **<ul>** после команды **return**:

```
return <ul>
      {list}
</ul>;
```

Давайте соберем все вместе в методе **render**:

```
class App extends React.Component {
      constructor() {
            super();
            this.state = {items: [1, 2, 3, 4, 5]};
      }

      render() {
            const list = this.state.items.map((item, index) => {
                  return <li key={index}>{item}</li>;
            });

            return <ul>
                  {list}
            </ul>;
      }
}
```

Результатом работы этого кода будет следующее:

```
<ul>
      <li>1</li>
      <li>2</li>
      <li>3</li>
      <li>4</li>
      <li>5</li>
</ul>
```

Запустите этот код и убедитесь в этом сами.

Стейты, циклы и события

Давайте теперь сделаем так, чтобы **<ul>** с нашим массивом выводился не сразу, а по нажатию на кнопку.

Начнем со следующего: в стейте сделаем элемент **text**:

```
class App extends React.Component {
      constructor() {
            super();
            this.state = {items: [1, 2, 3, 4, 5], text: ''};
      }
}
```

Затем в методе **render** сделаем место, в котором будем выводить содержимое **this.state.text**:

```
render() {
	return <div>
		{this.state.text}
	</div>;
}
```

Мы хотим сделать так, чтобы по нажатию на кнопку на экран вывелась следующая **<ul>**:

```
<ul>
	<li>1</li>
	<li>2</li>
	<li>3</li>
	<li>4</li>
	<li>5</li>
</ul>
```

Для этого поступим так: сделаем метод (назовем его **showList**), который будет срабатывать по нажатию на кнопку. В этом методе мы будем формировать приведенную выше **<ul>** из массива **this.state.items**. Затем сформированный HTML запишем в **this.state.text**.

Изначально в **this.state.text** ничего не будет, а затем по нажатию на кнопку все перерендерится и вместо **this.state.text** вставится наш список.

Давайте сделаем кнопку, на которую мы будем нажимать, и привяжем к ней метод **showList**:

```
render() {
	return <div>
		{this.state.text}
		<button onClick={this.showList.bind(this)}>показать
список</button>
	</div>;
}
```

Соберем все вместе, оставив пока вместо метода **showList** заготовку:

```
class App extends React.Component {
	constructor() {
		super();
		this.state = {items: [1, 2, 3, 4, 5], text: ''};
	}
```

```
	showList() {
		//тут сформируем this.state.text
	}

	render() {
		return <div>
			{this.state.text}
			<button onClick={this.showList.bind(this)}>показать
список</button>
		</div>;
	}
}
```

Давайте теперь реализуем метод **showList** и получим рабочий код:

```
class App extends React.Component {
	constructor() {
		super();
		this.state = {items: [1, 2, 3, 4, 5], text: ''};
	}

	showList() {

		//Сформируем список:
		const list = this.state.items.map((item, index) => {
			return <li key={index}>{item}</li>;
		});

		//Обновим стейт:
		this.setState({text: <ul>{list}</ul>});
	}

	render() {
		return <div>
			{this.state.text}
			<button onClick={this.showList.bind(this)}>показать
список</button>
		</div>;
	}
}
```

Запустите этот код и нажмите на кнопку - вы мгновенно увидите появившийся список.

Еще раз - как это работает: по нажатию на кнопку вызывается метод **showList**, он формирует список, вызывает **this.setState** для записи этого списка в **this.state.text**, в методе render вставка **{this.state.text}** мгновенно обновляется - и появляется наш список.