|  |  |
| --- | --- |
| **1. Руководство для начинающих по React Introduction** | **1. A Beginners Guide to React Introduction** |
| Добро пожаловать в Руководство по React для новичков! Я рада, что ты здесь. Я рекомендую вам просмотреть все эти видео один раз, не касаясь клавиатуры. Просто делайте заметки. Затем вы можете посмотреть их во второй раз и следить за ними, изменить / сломать код и просмотреть те части, которые вам нужны. Я покажу вам, как я налаживаю все, но не стесняйтесь копировать / вставлять код в среду, в которой вы чувствуете себя комфортно. | Welcome to The Beginner’s Guide to React! I’m happy you’re here. I recommend that you watch all of these videos through once without touching the keyboard. Just take notes. Then you can watch them a second time and follow along and change/break the code and review the bits that you need. I’ll show you how I get things up and running, but feel free to copy/paste the code to an environment where you feel comfortable. |
| Кент С. Доддс: [0:00](https://egghead.io/lessons/react-a-beginners-guide-to-react-introduction) Как дела, ребята? Меня зовут Кент С. Доддс, и я очень рад быть вашим учителем здесь, на сайте egghead.io, где я собираюсь познакомить вас с Руководством по React JS для начинающих. Мы собираемся начать с самого начала.  [0:11](https://egghead.io/lessons/react-a-beginners-guide-to-react-introduction) Если вы занимаетесь React в течение длительного времени, вы просто хотите отточить основы, или вы никогда раньше не занимались React, и вам просто любопытно, это будет идеальное место для вас потому что мы начнем с чистого листа.  [0:24](https://egghead.io/lessons/react-a-beginners-guide-to-react-introduction) Когда мы используем React, мы используем множество других инструментов в экосистеме вокруг него. У вас есть маршрутизатор, у вас есть управление состоянием, у вас есть инструменты для сборки, такие как webpack или парсер, или что-то еще, что вы используете.  [0:35](https://egghead.io/lessons/react-a-beginners-guide-to-react-introduction) Когда вы используете все эти инструменты, может быть довольно сложно изолировать, что такое React, и что он делает для меня. Если вы не изолируете его эффективно, вы не сможете использовать его очень эффективно.  [0:46](https://egghead.io/lessons/react-a-beginners-guide-to-react-introduction) Мы начинаем с буквального пустого файла index.html. Мы пишем в нем немного HTML. Мы пишем JavaScript прямо в этом HTML-файле. Тогда пуф. Внезапно мы пишем JavaScript для создания узлов DOM. Мы вставляем эти узлы DOM на страницу. Мы делаем его динамичным. Это круто.  [1:03](https://egghead.io/lessons/react-a-beginners-guide-to-react-introduction) Затем мы переключаемся с обычного JavaScript на React, чтобы вы получили хорошее представление о том, какую часть этой головоломки решает React и как я могу наиболее эффективно использовать ее при создании приложений.  [1:15](https://egghead.io/lessons/react-a-beginners-guide-to-react-introduction) Затем мы переходим на JSX. Я покажу вам некоторые хитрости JSX. Мы полностью проходим формы и HTTP, обработку ошибок и все такое. Вам это понравится.  [1:27](https://egghead.io/lessons/react-a-beginners-guide-to-react-introduction) Когда мы дойдем до самого конца, мы воспользуемся некоторыми замечательными инструментами, такими как CodeSandbox, GitHub и Netlify, для развертывания нашего приложения в мире. Вы можете показать его своим друзьям, семье, своей собаке и тому, кому вы хотите это показать. Вы будете так взволнованы, что создали что-то, что доступно во всемирной паутине с помощью React. Это супер, супер круто.  [1:50 Вперед](https://egghead.io/lessons/react-a-beginners-guide-to-react-introduction) . Мы углубимся в материалы семинара, чтобы вы могли видеть, где находятся файлы, как их запустить и запустить, если вы хотите продолжить или просто хотите повозиться с вещами в процессе. Затем мы можем перейти на этот потрясающий семинар, который я для вас приготовил.  [2:06](https://egghead.io/lessons/react-a-beginners-guide-to-react-introduction) Мастерская доступна на GitHub. Вы найдете его на моем GitHub, kentcdodds / beginners-guide-to-react. Рекомендую сменить ветку на яйцеголовый. Таким образом, вы убедитесь, что материал, с которым вы работаете, точно такой же, как я записал в курсе, и если мы внесем какие-либо изменения в будущем, они будут в основной ветке. Ветка-яйцеголовый всегда будет именно такой, как вы видите, как я делаю в курсе.  [2:32](https://egghead.io/lessons/react-a-beginners-guide-to-react-introduction) Курс - это просто набор файлов HTML, плюс этот стартовый скрипт, который мы рассмотрим. Вы можете загрузить его с помощью Git, если вам это удобно, или вы можете просто щелкнуть Загрузить ZIP, и файл ZIP будет загружен на ваш компьютер. Вы можете открыть это и извлечь, и это то, что вы найдете в конце дня.  [2:49](https://egghead.io/lessons/react-a-beginners-guide-to-react-introduction) Я использую VS Code. Вы можете свободно использовать Sublime или текстовый редактор или что угодно, но я рекомендую VS Code, и это здорово. У нас есть все уроки прямо здесь в виде файлов HTML, которые вы можете пройти.  [3:04](https://egghead.io/lessons/react-a-beginners-guide-to-react-introduction) HTML-файл, доступный здесь, представляет собой законченную версию нашего кода. Вы можете проработать каждый из них после просмотра видео, немного поработать с ним.  [3:15](https://egghead.io/lessons/react-a-beginners-guide-to-react-introduction) Чтобы начать работу, вы можете просто открыть это, чтобы мы могли нажать здесь на «Показать в Finder». Мы можем дважды щелкнуть по нему, и он просто откроет его прямо в браузере для вас. Это сработает для многих из них, но для некоторых из этих уроков требуется, чтобы у вас был настоящий сервер.  [3:35](https://egghead.io/lessons/react-a-beginners-guide-to-react-introduction) Есть несколько способов сделать это, если вы сами поймете, как это сделать. У меня есть стартовый сценарий, и я его использую. Мы используем инструмент под названием Browsersync. Мы используем npx для этого. npx поставляется вместе с Node. Если у вас установлен Node.js, вы можете использовать npx.  [3:51](https://egghead.io/lessons/react-a-beginners-guide-to-react-introduction) Вы можете просто скопировать и вставить это в окно терминала. Это запустит Browsersync, а затем вы сможете открыть localhost: 3000, где будет показан список всех файлов. Мы продолжим и нажмем на этот первый.  [4:04](https://egghead.io/lessons/react-a-beginners-guide-to-react-introduction) Обычно на семинаре я открываю это бок о бок с моим редактором кода. Это будет выглядеть примерно так. Здесь вы заметите, что я могу нажать кнопку сохранения, и он автоматически перезагрузится.  [4:19](https://egghead.io/lessons/react-a-beginners-guide-to-react-introduction) Мы также хотим, чтобы наши инструменты разработчика были открыты. У меня здесь есть наши инструменты разработчика. Вот это у меня внизу. Здесь есть два сценария, которых нет в исходном коде. Это добавлено Browsersync.  [4:34](https://egghead.io/lessons/react-a-beginners-guide-to-react-introduction) Самое замечательное в Browsersync заключается в том, что вам не нужно вручную нажимать кнопку перезагрузки, вы просто меняете что-то, нажимаете «Сохранить», и он автоматически перезагружается за вас. Вот почему мы выбираем Browsersync. Это действительно хорошо.  [4:50](https://egghead.io/lessons/react-a-beginners-guide-to-react-introduction) Вы можете свободно экспериментировать с этим материалом. По ходу дела вы можете очистить этот материал и напечатать его самостоятельно, или вы можете просто подождать, пока я просмотрю материал, а затем вы возитесь с готовым продуктом, или вы можете просто просмотреть все это, делать заметки, а затем пройти через это позже самостоятельно.  [5:08](https://egghead.io/lessons/react-a-beginners-guide-to-react-introduction) Что бы вы ни [выбрали](https://egghead.io/lessons/react-a-beginners-guide-to-react-introduction) , я рекомендую, чтобы в первый раз, когда вы проходите через это, вы просто наблюдали и делали заметки, потому что обычно это приносит больше удовольствия большинству людей, но вы делаете это так, как хотите.  [5:19](https://egghead.io/lessons/react-a-beginners-guide-to-react-introduction) Не могу передать, как я рад, что вы прошли через это. Я упорно трудился , чтобы попытаться сделать это как можно , как доступная, дать вам глубокое понимание основополагающего , не получив слишком много сорняков. Мне не терпится увидеть классные вещи, которые вы создаете с помощью этих знаний, которые вы приобретаете, изучая этот материал.  [5:36](https://egghead.io/lessons/react-a-beginners-guide-to-react-introduction) Пожалуйста, дайте мне знать, что вы создаете и что создаете, используя знания React. Не могу дождаться, когда ты это испытаешь. Давайте продолжим и прыгнем на курс. | Kent C. Dodds: [0:00](https://egghead.io/lessons/react-a-beginners-guide-to-react-introduction) What is up, folks? My name is Kent C. Dodds, and I am super excited to be your teacher here on egghead.io, where I'm going to take you through the Beginner's Guide to React JS. We're going to start from the very, very beginning.  [0:11](https://egghead.io/lessons/react-a-beginners-guide-to-react-introduction) Whether you've been doing React for a long time, you just want to sharpen yourself on the fundamentals or you've never done any React before and you're just curious, this is going to be a perfect place for you because we're going to start with a totally blank slate.  [0:24](https://egghead.io/lessons/react-a-beginners-guide-to-react-introduction) When we're using React, we're using a bunch of other tools in the ecosystem around it. You've got a router, you've got state management, you've got the building tools like webpack or parser or whatever it is that you're using.  [0:35](https://egghead.io/lessons/react-a-beginners-guide-to-react-introduction) When you're using all these tools, it can be pretty difficult to isolate what is React and what is it doing for me. If you don't isolate it effectively, then you can't use it very effectively.  [0:46](https://egghead.io/lessons/react-a-beginners-guide-to-react-introduction) We start out with a literal blank index.html file. We write some HTML in it. We write some JavaScript directly in that HTML file. Then, poof. All of a sudden, we're writing some JavaScript to create DOM nodes. We're inserting those DOM nodes into the page. We are making it dynamic. It's awesome.  [1:03](https://egghead.io/lessons/react-a-beginners-guide-to-react-introduction) Then, we switch over from regular JavaScript to React, so you get a good sense of what piece of this puzzle is React solving and how can I use it most effectively in building applications.  [1:15](https://egghead.io/lessons/react-a-beginners-guide-to-react-introduction) Then, we upgrade to JSX. I'll show you some tricks of JSX. We go all the way through forms and HTTP, error handling, all of that stuff. You're going to love this.  [1:27](https://egghead.io/lessons/react-a-beginners-guide-to-react-introduction) When we get down to the very end, we're going to use some awesome tools like CodeSandbox, GitHub, and Netlify to deploy our application to the world. You can show it to your friends, family, your dog, and whoever it is that you want to show this to. You'll be so excited that you've built something that is available on the World Wide Web with React. It's super, super cool.  [1:50](https://egghead.io/lessons/react-a-beginners-guide-to-react-introduction) Let's go ahead. We'll dive into the workshop stuff so that you can see where the files are, how to get things up and running if you want to follow along, or if you just want to tinker with things as we go. Then, we can get into this awesome workshop that I've prepared for you.  [2:06](https://egghead.io/lessons/react-a-beginners-guide-to-react-introduction) The workshop is available to you on GitHub. You'll find it on my GitHub, kentcdodds/beginners-guide-to-react. I recommend that you switch the branch to egghead. That way, you make sure that the stuff that you're working with is exactly the same stuff that I recorded in the course and if we make any changes in the future, that will have them on the master branch. The egghead branch will always be exactly what you see me do in the course.  [2:32](https://egghead.io/lessons/react-a-beginners-guide-to-react-introduction) The course is just a bunch of HTML files, plus this start script that we'll take a look at. You can download this using Git if you're comfortable with that or you can just click Download ZIP and that will download the ZIP file to your computer. You can open that up and extract that and this is what you're going to find at the end of the day.  [2:49](https://egghead.io/lessons/react-a-beginners-guide-to-react-introduction) I am using VS Code. You can feel free to use Sublime or text editor or whatever it is that you want to, but I recommend VS Code and it's great. We've got each one of the lessons right here as HTML files that you can go through.  [3:04](https://egghead.io/lessons/react-a-beginners-guide-to-react-introduction) The HTML file that's available here is the finished version of our code. You can work through each one of these after watching the video, tinker around with it a little bit.  [3:15](https://egghead.io/lessons/react-a-beginners-guide-to-react-introduction) To get things up and going, you can just open this up, so we can click on Reveal in Finder here. We can double click on that and it will just open it right up in the browser for you. That will work for lots of this, but some of these lessons require that you're running a real server.  [3:35](https://egghead.io/lessons/react-a-beginners-guide-to-react-introduction) There are a couple ways that you can do that if you can figure out how to make that work yourself. I have the start script and that's what I'm using. We're using a tool called Browsersync. We're using npx to get that running. npx comes bundled with Node. If you get Node.js installed, then you can use npx.  [3:51](https://egghead.io/lessons/react-a-beginners-guide-to-react-introduction) You can just copy and paste this in a terminal window. That will get Browsersync up and running and then you can open up localhost:3000, where that will show you a listing of all the files. We'll go ahead and click on this first one.  [4:04](https://egghead.io/lessons/react-a-beginners-guide-to-react-introduction) Typically, in the workshop, I've got this open up side-by-side with my code editor. You're going to have it look a little bit like this. Here, you'll notice that I can hit save key and it automatically reloads.  [4:19](https://egghead.io/lessons/react-a-beginners-guide-to-react-introduction) We'll also want to have our developer tools open. I've got our developer tools right here. I have that up here at the bottom. We have two scripts in here that you don't find in the source code. That's added by Browsersync.  [4:34](https://egghead.io/lessons/react-a-beginners-guide-to-react-introduction) The cool thing about Browsersync is that you don't have to manually hit the reload button, you just go ahead and change something, hit save and it will automatically reload for you. That's why we're going with Browsersync. It's really nice for this.  [4:50](https://egghead.io/lessons/react-a-beginners-guide-to-react-introduction) You can feel free to play around with this stuff. As we go, you can clear this stuff out and type it out yourself or you can just wait until I go through the material and then you tinker around with the finished product, or you can just watch through the whole thing, take notes and then go through it again later on your own.  [5:08](https://egghead.io/lessons/react-a-beginners-guide-to-react-introduction) Whatever works best for you, I do recommend that the first time you go through this, you just watch and take notes because that typically ends up in a better experience for most people, but you do it however you want to.  [5:19](https://egghead.io/lessons/react-a-beginners-guide-to-react-introduction) I can't tell you how excited I am for you to go through this. I've worked hard to try to make this as approachable as possible, give you a deep foundational understanding without getting too much in the weeds. I can't wait to see the cool things that you build with this knowledge that you acquire through going through this material.  [5:36](https://egghead.io/lessons/react-a-beginners-guide-to-react-introduction) Please do let me know what you build and what you create with this knowledge of React. I can't wait for you to experience this. Let's go ahead and jump into the course. |
| 2. Создайте пользовательский интерфейс с помощью ванильного JavaScript и DOM | 2. Create a User Interface with Vanilla JavaScript and DOM |
| Вы можете создать простой пользовательский интерфейс в Интернете, используя HTML и CSS. Но как только вы захотите сделать свое приложение интерактивным, вам нужно будет использовать JavaScript для управления DOM (объектной моделью документа), чтобы прослушивать пользовательские события и вносить обновления в пользовательский интерфейс. В этом уроке мы узнаем, как создать <div>элемент, используя необработанный JavaScript и API браузера. | You can create a simple user interface on the web using HTML and CSS. But as soon as you want to make your application interactive, you need to use JavaScript to manipulate the DOM (Document Object Model) to listen to user events and make updates to the user interface. In this lesson we'll learn how to create a <div> element using raw JavaScript and browser APIs. |
|  |  |
| Кент С. Доддс: [0:00](https://egghead.io/lessons/react-create-a-user-interface-with-vanilla-javascript-and-dom) Для начала нам понадобится body. Внутри у нас будет div с идентификатором root. Вот где будет смонтировано наше приложение. Тогда у нас будет сценарий. Нашим типом для нашего скрипта будет текст / JavaScript. Здесь нам нужно получить доступ к этому div. Я собираюсь создать переменную с именем rootElement. Это будет document.getElementById root.  [0:29](https://egghead.io/lessons/react-create-a-user-interface-with-vanilla-javascript-and-dom) Затем я возьму этот rootElement и (добавлю) appendChild элемент. Нам нужно добавить к этому элемент. Давайте создадим этот элемент с помощью document.createElement. Мы сделаем это div. Для этого элемента мы хотим, чтобы текстовое содержимое было «Hello, world». Просто для удовольствия, у нас будет равный контейнер element.className. Мы сохраним это и обновим здесь. Теперь у нас есть «Привет, мир».  [0:56](https://egghead.io/lessons/react-create-a-user-interface-with-vanilla-javascript-and-dom) Если мы посмотрим на вывод, у нас есть кое-что из этого Browsersync, но вот наше приложение в корне. Это созданный нами div с именем класса контейнера и текстовым содержимым «Hello, world». Тогда вот наш код JavaScript, который мы написали.  [1:11 Итак](https://egghead.io/lessons/react-create-a-user-interface-with-vanilla-javascript-and-dom) , чтобы создать пользовательский интерфейс на JavaScript, вам понадобится место, куда вы будете добавлять элементы DOM, созданные с помощью JavaScript. Мы собираемся получить доступ к этому элементу из API документов. Затем мы создадим наш собственный элемент. Мы добавим этому элементу некоторые свойства. Затем мы добавим этот элемент в наш rootElement. | Kent C. Dodds: [0:00](https://egghead.io/lessons/react-create-a-user-interface-with-vanilla-javascript-and-dom) To start, we're going to need a body. Inside of here, we're going to have a div with an ID of root. That's where our application is going to be mounted. Then we'll have a script. Our type for our script is going to be text/JavaScript. Inside of here, we need to get access to this div. I'm going to make a variable called rootElement. That'll document.getElementById root.  [0:29](https://egghead.io/lessons/react-create-a-user-interface-with-vanilla-javascript-and-dom) Then I'm going to take this rootElement and append a child. We need to append an element to this. Let's create that element with document.createElement. We'll make it a div. For this element, we want the text content to be "Hello, world." Just for fun, we'll have the element.className equal container. We'll save that and get a refresh here. Now we have "Hello, world."  [0:56](https://egghead.io/lessons/react-create-a-user-interface-with-vanilla-javascript-and-dom) If we look at the output, we have one thing from that Browsersync, but then here is our application at the root. That's the div we created with the class name of container and the text content of "Hello, world." Then here's our JavaScript code that we wrote.  [1:11](https://egghead.io/lessons/react-create-a-user-interface-with-vanilla-javascript-and-dom) In review, to create a user interface in JavaScript, you're going to need to have a place where you append your JavaScript-generated DOM elements. We're going to get access to that element from the document APIs. Then we'll create our own element. We'll add some properties onto that element. Then we'll append that element to our rootElement. |
| 3. Создайте пользовательский интерфейс с помощью React createElement API | 3. Create a User Interface with React’s createElement API |
| React использует те же API-интерфейсы для управления и обновления DOM, которые мы использовали на предыдущем уроке. Но он предлагает столько возможностей для таблицы, что мы собираемся ввести его в проект и переписать то, что мы написали, в код React. Вместо того, чтобы создавать элементы DOM, мы создадим элементы React, а затем передадим их react-domдля обработки, превращая их в элементы DOM и помещая их на страницу.  Если вы когда-либо изучали или использовали React раньше, вы, вероятно, больше знакомы с JSX, чем с createElementAPI React , но важно createElementсначала понять API, чтобы вы поняли магию и могли сами стать волшебником кода. Не волнуйтесь, мы скоро перейдем к JSX! | React uses the same APIs to control and update the DOM that we did in the previous lesson. But it offers so much to the table that we’re going to introduce it into the project and rewrite what we’ve written to React code. Instead of creating DOM elements, we’ll create React elements and then hand those off to react-dom to handle turning those into DOM elements and putting them into the page.  If you’ve ever learned or used React before, you’re probably more familiar with JSX than React’s createElement API, but it’s important to understand the createElement API first so you understand the magic and can become a code magician yourself. Not to worry, we’ll get to JSX soon enough! |
| Кент С. Доддс: [0:00](https://egghead.io/lessons/react-create-a-user-interface-with-react-s-createelement-api) Чтобы использовать React для создания этого пользовательского интерфейса, нам понадобится React на странице. Мы можем получить React из npm, но мы собираемся использовать сервис под названием unpackage. Если я пойду на unpackage.com, это покажет мне, как использовать unpackage для получения любого файла, который распространяется на npm. В нем есть эти файлы для React. Если я перейду к этому файлу, то увижу уменьшенную версию React.  [0:25](https://egghead.io/lessons/react-create-a-user-interface-with-react-s-createelement-api) Давайте продолжим и добавим здесь несколько тегов скрипта на нашу страницу, чтобы мы получили React на странице. С помощью React и ReactDOM на странице я теперь могу создавать элементы React, а затем использовать ReactDOM для визуализации этих элементов на странице.  [0:41](https://egghead.io/lessons/react-create-a-user-interface-with-react-s-createelement-api) Я собираюсь прокомментировать все это, а затем с помощью ReactDOM я собираюсь визуализировать свой элемент в корневой элемент, но элемент, который я предоставляю, не будет элементом DOM, который я создаю с документом . Вместо этого это будет React.createElement, который я создам с помощью React.  [1:01](https://egghead.io/lessons/react-create-a-user-interface-with-react-s-createelement-api) API не совсем такой, как тот, который мы получаем с документом. Мы можем указать тип элемента в качестве первого аргумента, но вместо того, чтобы получить элемент и прикрепить к нему свойства, мы указываем эти свойства при создании как объект.  [1:16](https://egghead.io/lessons/react-create-a-user-interface-with-react-s-createelement-api) Еще одно отличие - вместо текстового содержимого мы указываем дочерние элементы. Здесь мы укажем наших детей как Hello World, а затем имя класса будет контейнером. Это даст мне мой элемент, который я визуализирую с помощью ReactDOM для корневого элемента. Если мы сохраним это, мы получим обновление страницы. Возможно, вы даже не заметили этого, потому что он отображает то же самое, что и раньше.  [1:44](https://egghead.io/lessons/react-create-a-user-interface-with-react-s-createelement-api) Если мы перейдем сюда к root, мы увидим наш div с контейнером класса и текстовым содержимым Hello World. Мы также увидим те два тега скрипта, которые мы добавили для добавления React и ReactDOM на страницу, а затем наш собственный JavaScript, который мы написали для создания этого пользовательского интерфейса.  [2:00](https://egghead.io/lessons/react-create-a-user-interface-with-react-s-createelement-api) Давайте посмотрим, что это за элемент. Если это не узел DOM, то что это? Мы сохраним console.log (element), а затем перейдем к нашей консоли. Мы увидим, что у нас есть тип недвижимости $$. Это символ, и React должен знать, что этот объект является допустимым элементом React.  [2:19](https://egghead.io/lessons/react-create-a-user-interface-with-react-s-createelement-api) Мы знаем, что это тип div, и есть пара других свойств, которые используются React»ом внутри. Затем этот props - объект, это то, что мы передали в качестве второго аргумента здесь с дочерними элементами и именем класса.  [2:31](https://egghead.io/lessons/react-create-a-user-interface-with-react-s-createelement-api)  есть кое-что интересное в этом children prop, и я хочу вам здесь показать. Вы можете записать то же самое в качестве третьего аргумента в React.createElement, просто указав значение для свойства children. Здесь мы можем предоставить Hello World. Они функционально эквивалентны.  [2:51](https://egghead.io/lessons/react-create-a-user-interface-with-react-s-createelement-api) Если мы здесь посмотрим на элемент React, когда вы посмотрите на наши props, и мы все еще получаем children и это Hello World. Затем, если вы предоставите другой аргумент, это будет дополнительный дочерний элемент. Можно сказать запятую, Goodbye World. Сохраните это, мы получим обновление, и мы увидим Hello World, Goodbye World, и мы получим свойства нашего элемента как дочерние элементы в виде массива.  [3:14](https://egghead.io/lessons/react-create-a-user-interface-with-react-s-createelement-api) Мы можем и впердь действовать через children. Или мы могли бы взять это их и использовать children prop прямо вот так, указав его как массив самостоятельно. Вы увидите, что результат такой же.  [3:30](https://egghead.io/lessons/react-create-a-user-interface-with-react-s-createelement-api) Это также дает нам возможность создавать дополнительные элементы React, которые будут дочерними. Здесь я мог бы сказать React.createElement("span"). Мы могли бы сказать, что нам не нужны реквизиты для этого диапазона, и мы хотим, чтобы дочерние элементы были Hello и World.  [3:47](https://egghead.io/lessons/react-create-a-user-interface-with-react-s-createelement-api) У нас есть тип диапазона, свойства - null, дочерние элементы - это массив «Hello» и «World». Если мы расширим это, мы увидим props, children, а children - это просто еще один элемент React, который сам имеет props. И эти дочерние элементы представляют собой массив из двух строк.  [4:05 Итак](https://egghead.io/lessons/react-create-a-user-interface-with-react-s-createelement-api) , чтобы создать элементы React и отобразить их на странице, вам необходимо включить React и ReactDOM, React для создания элементов и ReactDOM для отображения этих элементов на странице.  [4:16](https://egghead.io/lessons/react-create-a-user-interface-with-react-s-createelement-api) Затем нам все еще нужно получить доступ к некоторому элементу на странице, чтобы мы могли визуализировать наши элементы в этом узле DOM. Затем мы создаем наши элементы React и используем ReactDOM.render для рендеринга нашего элемента в корневой узел DOM. | Kent C. Dodds: [0:00](https://egghead.io/lessons/react-create-a-user-interface-with-react-s-createelement-api) To use React to create this user interface, we're going to need to have React on the page. We can get React from npm, but there's a service called unpackage that we're going to use. If I go to unpackage.com that will show me how to use unpackage to get any file that is distributed on npm. It has these files for React. If I go to that file, then I'll see a minified version of React.  [0:25](https://egghead.io/lessons/react-create-a-user-interface-with-react-s-createelement-api) Let's go ahead and add some script tags to our page here so that we get React on the page. With React and ReactDOM both on the page, I can now create React elements and then use ReactDOM to render those elements to the page.  [0:41](https://egghead.io/lessons/react-create-a-user-interface-with-react-s-createelement-api) I'm going to comment all of this out and then with ReactDOM I'm going to render my element to the root element, but the element that I provide is not going to be a DOM element that I create with the document. Instead, it's going to be a React.createElement that I create with React.  [1:01](https://egghead.io/lessons/react-create-a-user-interface-with-react-s-createelement-api) The API is not exactly the same as the one that we get with document. We do get to specify the type of element as the first argument, but instead of getting the element and attaching properties to it, we specify those properties on creation as an object.  [1:16](https://egghead.io/lessons/react-create-a-user-interface-with-react-s-createelement-api) Another difference is instead of text content, we specify children. Here we'll specify our children as Hello World and then the class name will be a container. This is going to give me my element, which I will render with ReactDOM to the root element. If we save this, we'll get a page refresh. You may not have even noticed that because it's rendering exactly the same thing that we had before.  [1:44](https://egghead.io/lessons/react-create-a-user-interface-with-react-s-createelement-api) If we come down here to root, we'll see our div with a class container and the text content Hello World. We'll also see those two script tags that we added to add React and ReactDOM onto the page then our own JavaScript that we wrote to create this user interface.  [2:00](https://egghead.io/lessons/react-create-a-user-interface-with-react-s-createelement-api) Let's take a look at what this element is. If it's not a DOM node, then what is it? We'll console.log(element), save that, and then we'll come over here to our console. We'll see that we have a $$ type of property. That's a symbol and that's something for React to know that this object is a legitimate React element.  [2:19](https://egghead.io/lessons/react-create-a-user-interface-with-react-s-createelement-api) We know that the type is a div and there are a couple other properties that are used by React internally. Then this props object is what we passed as our second argument here with the children and the class name.  [2:31](https://egghead.io/lessons/react-create-a-user-interface-with-react-s-createelement-api) There's something interesting about this children prop that I want to show you here. You can write this same thing as a third argument to React.createElement by simply providing the value for the children prop. Here we can provide Hello World. Those are functionally equivalent.  [2:51](https://egghead.io/lessons/react-create-a-user-interface-with-react-s-createelement-api) If we look at the React element here, when you look at our props, and we still are getting children is Hello World. Then if you provide another argument, that will be an additional child. We can say comma, Goodbye World. Save that, we'll get a refresh, and we see Hello World, Goodbye World, and we get our element's props to be children as an array.  [3:14](https://egghead.io/lessons/react-create-a-user-interface-with-react-s-createelement-api) We can continue going on through children forever and ever. Or we could take these and use the children prop directly like this, specifying it as an array ourselves. You'll see that the result here is the same.  [3:30](https://egghead.io/lessons/react-create-a-user-interface-with-react-s-createelement-api) This also gives us the flexibility of creating additional React elements to be children. Here I could say React.createElement("span"). We could say we want no props for this span, and we want the children to be Hello and World.  [3:47](https://egghead.io/lessons/react-create-a-user-interface-with-react-s-createelement-api) We have the type of span, the props are null, the children are an array of "Hello" and "World". If we expand this, we'll see props, children, and children is just another React element that itself has props. And that children is an array of two strings.  [4:05](https://egghead.io/lessons/react-create-a-user-interface-with-react-s-createelement-api) In review, to create React elements and render them to the page, you need to include React and ReactDOM, React for creating the elements and ReactDOM for rendering those elements to the page.  [4:16](https://egghead.io/lessons/react-create-a-user-interface-with-react-s-createelement-api) Then we still need to get access to some element that's on the page so we can render our elements to that DOM node. Then we create our React elements and we use ReactDOM.render to render our element to the root DOM node. |
| 4. Создайте пользовательский интерфейс с синтаксисом React JSX | 4. Create a User Interface with React’s JSX syntax |
| Замечательно, что мы можем напрямую взаимодействовать с createElementAPI React для создания элементов React, а затем передавать их в React DOM, чтобы они отображались на экране. Но это не самый эргономичный API для написания нашего кода пользовательского интерфейса. Трудно отследить, когда один элемент начинается, а следующий заканчивается.  Вот почему команда React придумала JSX. Это расширение языка JavaScript для поддержки синтаксиса, похожего на HTML, который вы бы написали для создания этих элементов DOM (есть [несколько отличий](https://reactjs.org/docs/dom-elements.html) ). JSX дает нам выразительный синтаксис для представления нашего пользовательского интерфейса без потери преимуществ и возможностей написания нашего пользовательского интерфейса на JavaScript. Лучший способ воспользоваться этим - узнать, как JSX компилируется для обычных вызовов JavaScript, поэтому мы тоже немного рассмотрим это. | It’s awesome that we can interact directly with React’s createElement API to create React elements and then hand those off to React DOM to get them to show up on the screen. But it’s not the most ergonomic API to write our UI code in. It’s hard to track when one element starts and the next one ends.  This is why the React team came up with JSX. It’s an extension to the JavaScript language to support syntax that looks similar to the HTML that you would write to create these DOM elements (there are [a handful of differences](https://reactjs.org/docs/dom-elements.html)). JSX gives us an expressive syntax for representing our UI, without losing the benefits and powers of writing our UI in JavaScript. The best way to take advantage of this is to learn how JSX is compiled to regular JavaScript calls, so we’ll look into that a bit as well. |
| Кент С. Доддс: [0:00](https://egghead.io/lessons/react-create-a-user-interface-with-react-s-jsx-syntax) Использование React для создания элементов таким образом работает отлично, но не совсем эргономично. Большинство участников сообщества не так создают элементы React. Большая часть сообщества использует JSX, который представляет собой HTML-подобный синтаксис в нашем JavaScript.  [0:16](https://egghead.io/lessons/react-create-a-user-interface-with-react-s-jsx-syntax) Если бы я создал тот же элемент React, используя JSX, я бы сделал это так. Мы скажем, что element - это div. Наши children - Hello World. Затем у нас было className, которое является a prop. Это будет атрибутом нашего div здесь с className = {'container').  [0:34](https://egghead.io/lessons/react-create-a-user-interface-with-react-s-jsx-syntax) Если мы это сохраним, мы получим обновление. Здесь мы получим белый экран. Это потому, что в нашем JavaScript есть синтаксическая ошибка. Это потому, что это не код JavaScript. Это JSX. Браузер изначально этого не понимает. Это должно быть скомпилировано из этого во что-то, что может понять браузер. Вот где приходит Babel.  [0:54](https://egghead.io/lessons/react-create-a-user-interface-with-react-s-jsx-syntax) Babel - это компилятор JavaScript, поддерживающий следующее поколение JavaScript, а также нестандартные функции, такие как JSX. Если мы перейдем на эту страницу «Попробовать», а затем перейдем сюда, скопируем наш код и вставим его сюда, тогда мы увидим, что наш код компилируется во что-то, что нам очень знакомо.  [1:13](https://egghead.io/lessons/react-create-a-user-interface-with-react-s-jsx-syntax) React.createElement. Тип - div. Вот the props, className - это контейнер, затем все дочерние элементы переходят сюда. Я бы порекомендовал вам немного поиграть с этим инструментом. Понимание того, как компилируется JSX, поможет вам более эффективно использовать JSX.  [1:29](https://egghead.io/lessons/react-create-a-user-interface-with-react-s-jsx-syntax) В типичном приложении вы будете использовать инструмент, который будет использовать Babel для компиляции JSX в JavaScript за вас. Для наших целей мы собираемся использовать Babel в браузере, чтобы прямо здесь скомпилировать наш JavaScript без необходимости устанавливать какие-либо другие инструменты.  [1:44](https://egghead.io/lessons/react-create-a-user-interface-with-react-s-jsx-syntax) Я собираюсь добавить сюда script tag, чтобы включить автономный Babel. Это скомпилирует любой тег скрипта, имеющий тип text/babel. Затем он создаст новый тег сценария со скомпилированным кодом с типом text/JavaScript, таким образом браузер сможет его определить.  [2:00 Итак](https://egghead.io/lessons/react-create-a-user-interface-with-react-s-jsx-syntax) , если я сохраню это, мы загрузим Babel. Он скомпилирует наш JavaScript и позволит браузеру определить его, чтобы мы могли использовать JSX в браузере. Мы также получим это предупреждение о том, что мы используем преобразователь Babel в браузере, что не является тем, что вы хотели бы делать in production, но оно позволяет нам быстро выполнять итерацию прямо здесь, в этом index.html файле.  [2:20](https://egghead.io/lessons/react-create-a-user-interface-with-react-s-jsx-syntax) Если мы посмотрим на элементы, мы увидим код, который мы написали в этом сценарии text/babel. Затем, если мы перейдем к заголовку, мы увидим здесь новый тег скрипта, который является скомпилированной версией нашего кода.  [2:31](https://egghead.io/lessons/react-create-a-user-interface-with-react-s-jsx-syntax) В обзоре мы хотели сделать так, чтобы мы могли создавать наш пользовательский интерфейс без использования React createElement API и вместо этого использовать JSX, который представляет собой HTML-подобный синтаксис для создания пользовательского интерфейса.  [2:44](https://egghead.io/lessons/react-create-a-user-interface-with-react-s-jsx-syntax) Для этого нам понадобится компилятор вроде Babel. Мы установили автономный Babel, который будет компилировать наш код на лету в браузере. Нам также пришлось обновить наш тип до text/babel. Babel будет искать все сценарии с этим типом, компилировать код, который находится внутри них, и вставлять новый сценарий с скомпилированным кодом, чтобы браузер мог его оценить.  [3:05](https://egghead.io/lessons/react-create-a-user-interface-with-react-s-jsx-syntax) В типичном приложении вы не будете использовать Babel автономно. Вместо этого вы будете использовать другой инструмент, который, вероятно, использует Babel для компиляции вашего кода. Таким образом, вы получите гораздо более удобный синтаксис для создания пользовательского интерфейса с помощью React. | Kent C. Dodds: [0:00](https://egghead.io/lessons/react-create-a-user-interface-with-react-s-jsx-syntax) Using React to create elements in this way works out just fine, but it's not entirely ergonomic. It's not the way that most of the community creates React elements. Most of the community is using JSX, which is an HTML-like syntax in our JavaScript.  [0:16](https://egghead.io/lessons/react-create-a-user-interface-with-react-s-jsx-syntax) If I were to create this same React element using JSX, I would make it like this. We'll say element is a div. Our children are Hello World. Then we had a className, which is a prop. That'll go as an attribute on our div here, with className={'container').  [0:34](https://egghead.io/lessons/react-create-a-user-interface-with-react-s-jsx-syntax) If we save that, we'll get a refresh. We'll get a white screen here. That's because we have a syntax error in our JavaScript. That's because this is not JavaScript code. This is JSX. The browser does not understand this natively. It needs to be compiled from this to something that the browser can understand. That's where Babel comes in.  [0:54](https://egghead.io/lessons/react-create-a-user-interface-with-react-s-jsx-syntax) Babel is a JavaScript compiler supporting the next generation of JavaScript as well as non-standard features like JSX. If we go to this Try It Out page and then go over here and copy our code and paste it in here, then we'll see that our code is being compiled to something that's very familiar to us.  [1:13](https://egghead.io/lessons/react-create-a-user-interface-with-react-s-jsx-syntax) React.createElement. The type is div. Here are the props, className is container, then all of the children come hereafter. I would recommend that you spend some time playing around in this tool. Understanding how JSX is compiled will make you more effective at using JSX.  [1:29](https://egghead.io/lessons/react-create-a-user-interface-with-react-s-jsx-syntax) In a typical application, you're going to be using a tool that will use Babel to compile JSX to JavaScript for you. For our purposes, we're going to use Babel in the browser to get this compiling our JavaScript right here without having to install any other tools.  [1:44](https://egghead.io/lessons/react-create-a-user-interface-with-react-s-jsx-syntax) I'm going to add a script tag here to include Babel standalone. That will compile any script tag that has the type of text/babel. Then it will create a new script tag with the compiled code with the type as text/JavaScript so that the browser can evaluate it.  [2:00](https://egghead.io/lessons/react-create-a-user-interface-with-react-s-jsx-syntax) With that, if I save this, we'll get Babel downloaded. It will compile our JavaScript and allow the browser to evaluate it so we can use JSX in the browser. We'll also get this warning that we're using the in-browser Babel transformer, which is not something that you'd want to do in production, but it makes it easy for us to iterate quickly right here in this index.html file.  [2:20](https://egghead.io/lessons/react-create-a-user-interface-with-react-s-jsx-syntax) If we take a look at the elements, we're going to see the code that we wrote in this text/babel script. Then if we go to the head, we'll see a new script tag in here which is the compiled version of our code.  [2:31](https://egghead.io/lessons/react-create-a-user-interface-with-react-s-jsx-syntax) In review, what we wanted to do here was make it so that we can author our UI without using the React createElement API and instead use JSX, which is an HTML-like syntax for authoring UI.  [2:44](https://egghead.io/lessons/react-create-a-user-interface-with-react-s-jsx-syntax) To do this, we need a compiler like Babel. We installed Babel standalone, which will compile our code on the fly in the browser. We also had to update our type to text/babel. Babel will look for all scripts that have that type, compile the code that's inside them, and insert a new script that has the compiled code so the browser can evaluate it.  [3:05](https://egghead.io/lessons/react-create-a-user-interface-with-react-s-jsx-syntax) In a typical application, you will not be using Babel standalone. Instead, you'll be using another tool which probably is using Babel to compile your code. That way, you get a much more friendly syntax for authoring your UI with React. |
| 5. Эффективное использование JSX с React | 5. Use JSX effectively with React |
| JSX - это не совсем другой язык, но это своего рода расширение языка, поэтому знание того, как вы могли бы выразить определенные вещи JavaScript в синтаксисе JSX, важно для эффективного использования JSX.  В этом уроке мы рассмотрим несколько разных вещей, которые вы можете делать с JSX, например, интерполяцию, композицию и распространение свойств (props).  И мы сравним эти синтаксисы с тем, что генерируется babel, чтобы иметь хорошее представление о том, как это переводится в обычные вызовы JavaScript в React.createElement. | JSX is not an entirely different language, but it is a bit of an extension to the language, so knowing how you would express certain JavaScript things within the JSX syntax is important to using JSX effectively.  In this lesson we’ll explore a few different things you can do with JSX, like interpolation, composition, and spreading props.  And we’ll compare these syntaxes with what is generated by babel so we have a good understanding of how this translates to regular JavaScript calls into React.createElement. |
| Кент С. Доддс: [0:00](https://egghead.io/lessons/react-use-jsx-effectively-with-react) Если мы хотели создать переменную для Hello World, и мы хотели бы интерполировать значение этой переменной в положение дочерних элементов, то я собираюсь скопировать это. Мы сделаем это нашими children.  [0:12](https://egghead.io/lessons/react-use-jsx-effectively-with-react) Затем мы подойдем сюда и для интерполяции воспользуемся фигурными скобками и поместим сюда children. Все, что вы помещаете между этими двумя фигурными скобками, может быть выражением JavaScript. Независимо от того, что это выражение оценивает, это значение, которое помещается в эту позицию для вызова React.createElement.  [0:30](https://egghead.io/lessons/react-use-jsx-effectively-with-react) Мы можем сделать то же самое для нашего className. Если я скопирую контейнер и сделаю переменную className, и вместо кавычек здесь мы поставим фигурные скобки, чтобы предложить компилятору Babel, что мы хотим, чтобы это значение оценивалось как выражение, тогда мы передадим переменную className . Если мы сохраним это, то получим тот же результат, что и раньше.  [0:53](https://egghead.io/lessons/react-use-jsx-effectively-with-react) Вы можете увидеть скомпилированную версию кода здесь, в нашем теге скрипта, где вы видите, что у нас есть дочерние элементы как переменная, а наш className - как переменная. Наш элемент - это присвоение React.createElement нашему div. Затем наш объект, в котором у нас есть className, назначается переменной className.  [1:09](https://egghead.io/lessons/react-use-jsx-effectively-with-react) Мы можем переименовать это как угодно. Мы могли бы сказать myClassName, а затем скопировать это и вставить туда. Если мы сохраним это, мы получим обновление. Вот, myClassName присваивается свойству className нашего объекта здесь.  [1:24](https://egghead.io/lessons/react-use-jsx-effectively-with-react) То же самое с children. Мы могли бы сказать myChildren. Сохраните это. Обновимся, и вот, myChildren передаются как мои children. Когда вы используете фигурные скобки, это, по сути, говорит Бабелю: «Оставьте это в покое и передайте его напрямую как свойство, которое вы передаете для props здесь и для дочерних аргументов здесь».  [1:47](https://egghead.io/lessons/react-use-jsx-effectively-with-react) Если бы мы хотели разделить их, мы могли бы сделать это с помощью worldChild и сделать worldChild. Поместите их рядом, и мы получим myChildren и worldChild. За исключением того, что вам нужно правильно писать слова. Это важно в программировании. Вот и все.  [2:08](https://egghead.io/lessons/react-use-jsx-effectively-with-react) Это позволяет нам выразить то, как мы создаем пользовательский интерфейс. Если мы хотим добавить сюда еще один элемент React, все, что нам нужно сделать, это добавить . Можно поздороваться (say Hello). Тогда мы могли бы сделать **World** . Затем избавьтесь от них, сохраните это, и поехали. Получаем Hello World. Babel управляет, компилируя это до JavaScript, который браузер может выполнять с помощью React API.  [2:37](https://egghead.io/lessons/react-use-jsx-effectively-with-react) Есть еще один полезный трюк с JSX, который я хочу вам показать. Вернемся к нашим children: «Hello World». Мы просто назовем это className. Мы избавимся от этого. Мы изменим это на className. Затем мы вставим сюда children.  [2:56](https://egghead.io/lessons/react-use-jsx-effectively-with-react) Важно помнить, что children - это просто особая опора, которую мы можем предоставить прямо здесь. Дети есть дети. При этом он работает точно так же, как и раньше, за исключением того, что теперь вместо компиляции для добавления дополнительных аргументов он просто использует здесь ярлык в качестве одного из реквизитов.  [3:16](https://egghead.io/lessons/react-use-jsx-effectively-with-react) Кроме того, поскольку это JSX, а не HTML, мы можем иметь самозакрывающиеся теги для div. Мы можем спасти это, и это самозакрывающийся.  [3:24](https://egghead.io/lessons/react-use-jsx-effectively-with-react) Еще мы можем сделать, если я создам здесь объект, где у нас есть наши props, мы сделаем children и className частью этих свойств (props). Затем я могу подойти прямо сюда, удалить их все и сказать: «Эй, Babel, я хочу, чтобы ты взял все эти props и передал их вызову React.createElement, который ты создаешь для этого div». Интерполировать на позицию этих div's prop's распространение этого prop's object.  [3:50](https://egghead.io/lessons/react-use-jsx-effectively-with-react) Если мы сохраним это, мы сможем взглянуть на скомпилированную версию этого. Получаем наш props object. У нас есть React.createElement. Затем он просто проходит непосредственно через props. Затем я могу добавить дополнительные props, если захочу. Я мог бы сказать id="app-root", сохранить это. Затем в этом случае мы получим the extends helper, который по сути является object.assign.  [4:10](https://egghead.io/lessons/react-use-jsx-effectively-with-react) Затем Babel расширит the props, которые мы поставили вместе с the props, которые мы распространяем. Мы получаем единый комбинированный объект для второго аргумента вызова React.createElement. Из-за того, как работает object.assign, если бы мы хотели переопределить одно из свойств в этом props object, тогда мы могли бы сделать это, просто предоставив его после того, как мы разместим эти props в позиции props для этого элемента.  [4:35](https://egghead.io/lessons/react-use-jsx-effectively-with-react) Можно сказать className="notContainer". Сохраните это. Затем, если мы посмотрим на скомпилированную версию, мы увидим наши расширения для props, которые идут до распространения. Затем мы увидим этот объект, а затем мы увидим другой объект для the props, который идет после распространения.  [4:51](https://egghead.io/lessons/react-use-jsx-effectively-with-react) Затем, если мы посмотрим на результаты в корневом каталоге, мы увидим, что у нас есть the id of app root и класс notContainer. App root выглядит как есть, потому что он не перезаписывается.  [5:04](https://egghead.io/lessons/react-use-jsx-effectively-with-react) Мы получаем children такими, какие они есть, потому что children не перезаписываются, но затем мы получаем className of notContainer, потому что мы перезаписываем className, которое находится в props, которые идут перед className. Все, что будет последним, является победителем в случае конфликта, подобного нашему.  [5:22](https://egghead.io/lessons/react-use-jsx-effectively-with-react) В обзоре мы узнали, что вы можете интерполировать значения с помощью этих фигурных скобок, поместив любое выражение между фигурными скобками и передав это выражение в API React.createElement.  [5:34](https://egghead.io/lessons/react-use-jsx-effectively-with-react) Мы также узнали, что вы можете распространять свойства в позиции props элемента JSX, и эти свойства будут объединены с другими свойствами, которые предоставляются этому элементу декларативным и детерминированным способом. | Kent C. Dodds: [0:00](https://egghead.io/lessons/react-use-jsx-effectively-with-react) If we wanted to create a variable for Hello World and we wanted to interpolate that variable's value into the children position, then I'm going to go ahead and copy this. We'll make this our children.  [0:12](https://egghead.io/lessons/react-use-jsx-effectively-with-react) Then we'll come in here and to interpolate we're going to use curly braces and put children in here. Anything you put between these two curly braces can be a JavaScript expression. Whatever that expression evaluates to is the value that gets placed into this position for the React.createElement call.  [0:30](https://egghead.io/lessons/react-use-jsx-effectively-with-react) We can do the same thing for our className. If I copy the container and make a className variable, and instead of the quotes here, we'll put curly braces to suggest to the Babel compiler that we want this value to be evaluated as an expression, then we'll pass the className variable. If we save this, then we'll get the exact same result as we had before.  [0:53](https://egghead.io/lessons/react-use-jsx-effectively-with-react) You can see the compiled version of the code here in our script tag where you see we have our children as a variable, our className as a variable. Our element is an assignment to React.createElement with our div. Then our object where we have className being assigned to the variable className.  [1:09](https://egghead.io/lessons/react-use-jsx-effectively-with-react) We can rename this to whatever we want. We could say myClassName and then copy this and paste it in there. If we save this, we'll get a refresh. There it is, myClassName is being assigned to the className property of our object here.  [1:24](https://egghead.io/lessons/react-use-jsx-effectively-with-react) Then children is the same thing. We could say myChildren. Save that. We'll get a refresh and there it is, myChildren being passed as my children. When you use the curly braces, that's basically telling Babel, "Leave this alone and pass it directly as the property that you pass for the props here and for the children arguments here."  [1:47](https://egghead.io/lessons/react-use-jsx-effectively-with-react) If we wanted to split these up, then we could do so with worldChild and do worldChild. Put them side by side just like that and we'll get myChildren and worldChild. Except to get it to render you need to spell things correctly. That's important in programming. There we go.  [2:08](https://egghead.io/lessons/react-use-jsx-effectively-with-react) This allows us to be expressive with the way that we're building our UI. If we want to add another React element in here, all we need to do is add a . We can say Hello. Then we could do a **World**. Then get rid of these, save that, and here we go. We get Hello World. Babel is managing, compiling that down to JavaScript that the browser can execute using React APIs.  [2:37](https://egghead.io/lessons/react-use-jsx-effectively-with-react) There's another useful JSX trick that I want to show you. Let's come back to our children: 'Hello World'. We'll just call this className. We'll get rid of this. We'll change this to className. Then we'll interpolate children here.  [2:56](https://egghead.io/lessons/react-use-jsx-effectively-with-react) One important thing to remember is that children is just a special prop that we could provide right here. Children is children. With that it works exactly the same way as it did before, except now instead of being compiled to add additional arguments, it's just using the shortcut here as one of the props.  [3:16](https://egghead.io/lessons/react-use-jsx-effectively-with-react) In addition, because this is JSX and not HTML, we can have self-closing tags for divs. We can save that, and this is self-closing.  [3:24](https://egghead.io/lessons/react-use-jsx-effectively-with-react) Another thing we can do is if I make an object here where we have our props, we're going to make children and className part of those props. Then I can come right here, remove all of those, and I can say, "Hey, Babel, I want you to take all of these props and pass them to the React.createElement call that you create for this div." Interpolate onto this div's prop's position a spread of that prop's object.  [3:50](https://egghead.io/lessons/react-use-jsx-effectively-with-react) If we save that, we can take a look at the compiled version of this. We get our props object. We have React.createElement. Then it just passes the props directly. Then I can add additional props if I want to. I could say id="app-root", save that. Then in this case we're going to get the extends helper, which is basically object.assign.  [4:10](https://egghead.io/lessons/react-use-jsx-effectively-with-react) Then Babel is going to extend the props that we provided with the props that we're spreading. We get a single combined object for the second argument of the React.createElement call. Because of the way that object.assign works, if we wanted to override one of the properties in this props object, then we could do so simply by providing it after we spread those props in the props position for this element.  [4:35](https://egghead.io/lessons/react-use-jsx-effectively-with-react) We can say className="notContainer". Save that. Then if we look at the compiled version, we'll see our extends for the props that come before the spread. Then we'll see that object and then we'll see another object for the props that come after the spread.  [4:51](https://egghead.io/lessons/react-use-jsx-effectively-with-react) Then if we look at the results under our root, we'll see we have the id of app root and the class of notContainer. App root because it's not getting overwritten, so it appears as is.  [5:04](https://egghead.io/lessons/react-use-jsx-effectively-with-react) We get the children as they are because the children are not being overwritten, but then we get the className of notContainer because we are overwriting the className that's in the props which comes before a className. Whatever comes last is the winner in the event of a conflict like we have here.  [5:22](https://egghead.io/lessons/react-use-jsx-effectively-with-react) In review, what we learned here is that you can interpolate values with these curly braces by putting any expression between the curly braces and have that expression passed along to the React.createElement API.  [5:34](https://egghead.io/lessons/react-use-jsx-effectively-with-react) We also learned that you can spread props in the props position of a JSX element and those props will be combined with the other props that are provided to that element in a declarative and deterministic way. |
| 6. Рендеринг двух элементов бок о бок с помощью React Fragments | 6. Render two elements side-by-side with React Fragments |
| В React нельзя отображать два элемента React бок о бок ( <span>Hello</span><span>World</span>). Они должны быть заключены в другой элемент (например, a <div>). Это может показаться странным ограничением, но когда вы думаете о том, что JSX скомпилирован для React.createElementвызовов, это имеет смысл. В этом уроке мы рассмотрим это и как обойти это ограничение с помощью React Fragments. | In React, you can’t render two React elements side-by-side (<span>Hello</span><span>World</span>). They have to be wrapped in another element (like a <div>). This may seem like an odd limitation, but when you think about the fact that JSX is compiled to React.createElement calls, it makes sense. In this lesson we’ll take a look at that and how to side-step this limitation with React Fragments. |
| КентС.Доддс: [00:00](https://egghead.io/lessons/react-render-two-elements-side-by-side-with-react-fragments#t=0) Предположим, что вместо рендеринга div с контейнером имени класса и затем «Hello, World» мы хотели визуализировать два промежутка рядом, один из которых говорит «Hello», а другой - "Мир." Давайте сначала сделаем это с помощью React createElement, а затем посмотрим, как это можно сделать с помощью JSX.  [00:15](https://egghead.io/lessons/react-render-two-elements-side-by-side-with-react-fragments#t=15) Я собираюсь сделать свой элемент React.createElement span. Реквизит нам не нужен. Затем мы скажем: «Привет». Нам понадобится еще один элемент. Мы назовем это нашим worldElement. Назовем его нашим helloElement. Этот скажет: «Мир».  [00:36](https://egghead.io/lessons/react-render-two-elements-side-by-side-with-react-fragments#t=36) Затем нам нужно отрендерить их рядом. Я не знаю, как мы это делаем. Скажем helloElement и worldElement. Нет, похоже, это не сработает.  [00:51](https://egghead.io/lessons/react-render-two-elements-side-by-side-with-react-fragments#t=51) Как поставить две переменные рядом и передать их обе в качестве первого аргумента? Это невозможно, но если бы мы сделали это в HTML, это было бы довольно просто. Мы бы просто сказали Привет и Мир. Это сработает нормально.  [01:05](https://egghead.io/lessons/react-render-two-elements-side-by-side-with-react-fragments#t=65) Это просто сделать в HTML, но невозможно сделать с React, поэтому команда React создала особый тип элемента, названный React fragment. Можно сказать React.createElement. React fragment - это тип. Для этого не нужен any props.  [01:22](https://egghead.io/lessons/react-render-two-elements-side-by-side-with-react-fragments#t=82) Тогда мы можем обеспечить обоих наших children. Давай запишем это сюда. Мы возьмем наш helloElement, передадим его как первого потомка, а worldElement, передадим его как второго потомка. Затем он получит наш элемент, который мы собираемся визуализировать.  [01:37 Итак](https://egghead.io/lessons/react-render-two-elements-side-by-side-with-react-fragments#t=97) , у нас есть ожидаемый результат. Мы также могли бы добавить сюда дополнительного child для пробела. Опять же, никто не хочет напрямую использовать React.createElement API. Прокомментируем это и посмотрим, как бы это выглядело, если бы мы захотели сделать это с помощью JSX.  [01:53](https://egghead.io/lessons/react-render-two-elements-side-by-side-with-react-fragments#t=113) Наш корневой элемент - это фрагмент React. Мы сделаем открытую скобку и скажем React.Fragment. Мы закроем это с помощью React.Fragment. Затем мы хотим создать a span для приветствия и далее Hello . Затем мы сделаем пробел, а затемМир и закрой это. Если мы сохраним это, мы получим точно такой же результат, и мы будем использовать наш элемент React.Fragment.  [02:22](https://egghead.io/lessons/react-render-two-elements-side-by-side-with-react-fragments#t=142) Давайте посмотрим на скомпилированный JSX здесь. Вы заметили, что мы получаем наш элемент с помощью React.createElement. Он просто передает React.Fragment в качестве первого аргумента React.createElement, тогда как это, имя тега, передается в виде строки.  [02:36](https://egghead.io/lessons/react-render-two-elements-side-by-side-with-react-fragments#t=156) Фрагмент React позволяет нам размещать элементы рядом без необходимости иметь какой-то контейнерный элемент, такой как div. Это может быть полезно, когда вы создаете такие вещи, как таблицы, которые имеют определенную структуру.  [02:49](https://egghead.io/lessons/react-render-two-elements-side-by-side-with-react-fragments#t=169) Поскольку это обычное дело, JSX имеет специальный синтаксис для фрагментов React. То есть просто удалить фрагмент React и получить открывающую и закрывающую угловую скобку(<>…</>). Это функционально эквивалентно тому, что было у нас раньше. Этот синтаксис я обычно использую всякий раз, когда мне нужны фрагменты React. | Kent C. Dodds: [00:00](https://egghead.io/lessons/react-render-two-elements-side-by-side-with-react-fragments#t=0) Let's say that instead of rendering a div with the class name container and then "Hello, World," we wanted to render two spans side by side, one that says "Hello" and the other that says "World." Let's first do this with React createElement, and then we'll see how we could do this with JSX.  [00:15](https://egghead.io/lessons/react-render-two-elements-side-by-side-with-react-fragments#t=15) I'm going to make my element React.createElement span. We don't need any props. Then we'll say, "Hello." We'll need another element. We'll call this our worldElement. Let's call this one our helloElement. This one will say, "World."  [00:36](https://egghead.io/lessons/react-render-two-elements-side-by-side-with-react-fragments#t=36) Then we need to render those side by side. I don't know how we do that. Let's say helloElement and worldElement. No, it doesn't look like that's going to work very well.  [00:51](https://egghead.io/lessons/react-render-two-elements-side-by-side-with-react-fragments#t=51) How do you put two variables side by side and pass them both as the first argument? It's impossible, but if we were to do this in HTML, it would be pretty straightforward. We'd just say Hello and World. That would work just fine.  [01:05](https://egghead.io/lessons/react-render-two-elements-side-by-side-with-react-fragments#t=65) This is straightforward to do in HTML, but it's not possible to do with React, which is why the React team created a special type of element called a React fragment. We can say React.createElement. React fragment is a type. We don't need any props for it.  [01:22](https://egghead.io/lessons/react-render-two-elements-side-by-side-with-react-fragments#t=82) Then we can provide both of our children. Let's put this down here. We'll take our helloElement, pass that as the first child, and worldElement, pass that as the second child. Then this is going to get our element that we're going to render.  [01:37](https://egghead.io/lessons/react-render-two-elements-side-by-side-with-react-fragments#t=97) With that, we have the output that we're expecting. We could also add an extra child in here to give us an extra space. Again, nobody wants to use the React.createElement API directly. Let's comment this out and see what this would look like if we wanted to do it with JSX.  [01:53](https://egghead.io/lessons/react-render-two-elements-side-by-side-with-react-fragments#t=113) Our root element is this React fragment. We'll do an open bracket and say React.Fragment. We'll close that off with React.Fragment. Then we'll want to create a span for hello, so Hello. Then we'll do a space and then World and close that off. If we save this, we're going to get the exact same output, and we're using our React.Fragment element.  [02:22](https://egghead.io/lessons/react-render-two-elements-side-by-side-with-react-fragments#t=142) Let's take a look at the compiled JSX here. You notice that we get our element with React.createElement. It's just passing React.Fragment as a first argument to React.createElement, whereas this, the tag name, is being passed as a string.  [02:36](https://egghead.io/lessons/react-render-two-elements-side-by-side-with-react-fragments#t=156) The React fragment allows us to put elements side by side without having to have some sort of container element like a div. This can be useful when you're creating things like tables that have a specific structure to them.  [02:49](https://egghead.io/lessons/react-render-two-elements-side-by-side-with-react-fragments#t=169) Because this is so common to do, JSX has a special syntax for React fragments. That is to simply remove the React fragment and have an open and closing angle bracket. This is functionally equivalent to what we had before. This syntax is typically what I use whenever I need React fragments. |
| 7. Создайте простой повторно используемый компонент React | 7. Create a Simple Reusable React Component |
| Одним из самых больших сдвигов парадигмы, который React предложил экосистеме пользовательского интерфейса, стала компонентная модель. Он позволяет упаковать всю логику, стили и макет единицы пользовательского интерфейса в блок, который вы можете легко перемещать и повторно использовать, не раскрывая никаких деталей реализации компонента. Вам не нужно понимать, как компонент работает внутри, чтобы использовать его эффективно. С идиоматической компонентной архитектурой рендеринг компонента в одном месте не повлияет на другие части вашего приложения.  В этом уроке мы узнаем, как создать компонент React и как его использовать. Вы узнаете, почему все работает именно так, и как использовать это в своих интересах. | One of the biggest paradigm shifts that React offered to the UI ecosystem was the component model. It allows you to package up all the logic, styling, and layout of a unit of UI into a box that you can easily move around and reuse without exposing any of the implementation details of the component. You don’t have to understand how a component works internally to use it effectively. With idiomatic component architecture, rendering a component in one place won’t impact other parts of your app.  In this lesson we’ll learn how to create a React component and how to use it. You’ll learn why things work the way they do and how to use this to your advantage. |
| Кент: [0:00](https://egghead.io/lessons/react-create-a-simple-reusable-react-component-b1ac5ff9) Итак, у нас есть этот div с именем класса "container", и у нас есть этот div с именем класса "message" здесь, который говорит "Hello, World!"  [0:06](https://egghead.io/lessons/react-create-a-simple-reusable-react-component-b1ac5ff9) Допустим, я хотел иметь, не знаю, может быть, два из них. Итак, теперь у нас есть "Hello, World!" и "Hello, World!" и тогда я не хочу повторяться. Так что я собираюсь сделать здесь постоянное сообщение: Hello, World!" и мы поместим это здесь вместо «message», просто вставим это туда. Сейчас я не повторяюсь, и это хорошо.  [0:27](https://egghead.io/lessons/react-create-a-simple-reusable-react-component-b1ac5ff9) Но что, если бы я хотел, чтобы один из них сказал: «Привет, мир!» а другой сказать: «Прощай, мир!» Что ж, нам нужно как-то параметризовать это, и когда мы делаем это в JavaScript, мы создаем функцию. Я сделаю здесь стрелочную функцию, и мы примем объект. Мы назовем его «props», а затем, каким бы ни было значение props.msg или props.message, мы будем вставлять его в свойство children для этого создаваемого div.  [0:52](https://egghead.io/lessons/react-create-a-simple-reusable-react-component-b1ac5ff9) Теперь вместо того, чтобы просто помещать туда «сообщение», мы собираемся вызвать его с помощью объекта и использовать msg как «Hello, World!» а затем мы сделаем это сообщение "Goodbye, World!" Сохраните это. И получаем "Hello, World!" и "Прощай, мир!" что потрясающе.  [1:10](https://egghead.io/lessons/react-create-a-simple-reusable-react-component-b1ac5ff9) Но опять же, это не совсем эргономично. Это выглядит не очень хорошо, и одно из преимуществ использования JSX - способность нашего пользовательского интерфейса напоминать декларативную природу HTML.  [1:21](https://egghead.io/lessons/react-create-a-simple-reusable-react-component-b1ac5ff9) Что я действительно хотел бы сделать, так это сказать здесь «сообщение», а затем «Hello, World!» и «сообщение». Но если я попытаюсь это сделать, вы заметите, что мы действительно получаем "Hello, World!" но здесь мы получим предупреждение консоли: «Сообщение тега не распознается в этом браузере. Если вы намеревались визуализировать компонент React, начните его имя с заглавной буквы».  [1:42](https://egghead.io/lessons/react-create-a-simple-reusable-react-component-b1ac5ff9) О, это интересно. Большое спасибо. Итак, давайте узнаем об этом больше.  [1:46](https://egghead.io/lessons/react-create-a-simple-reusable-react-component-b1ac5ff9) Если мы посмотрим на скомпилированную версию нашего кода, мы увидим нашу функцию сообщения, которая возвращает элемент React, создающий div. А затем, если мы спустимся сюда, мы увидим сообщение React.createElement в виде строки.  [1:57](https://egghead.io/lessons/react-create-a-simple-reusable-react-component-b1ac5ff9) Это проблематично, потому что мы не хотим иметь сообщение в виде строки для создания элемента DOM, который является элементом типа сообщения. Мы действительно хотели использовать эту функцию, чтобы мы могли использовать эту функцию для создания дополнительных элементов React.  [2:11](https://egghead.io/lessons/react-create-a-simple-reusable-react-component-b1ac5ff9) Итак, как следует из предупреждения, нам нужно использовать заглавную букву, поэтому, если вместо этого мы говорим заглавную «Message», тогда мы используем заглавную «Message» здесь, и давайте убедитесь, что мы не получаем ошибок. Вот. Поэтому мы скажем для них заглавную «Message», и мы можем сохранить это.  [2:25](https://egghead.io/lessons/react-create-a-simple-reusable-react-component-b1ac5ff9) Теперь мы больше не получаем этого предупреждения. Если мы посмотрим на наш скомпилированный код, мы увидим здесь заглавное «Message», а когда мы перейдем к React.createElement, мы получим заглавное Message прямо здесь, вне строки.  [2:38](https://egghead.io/lessons/react-create-a-simple-reusable-react-component-b1ac5ff9) Итак, мы передаем ссылку на функцию непосредственно в React.createElement (). А затем, если мы в конечном итоге посмотрим, что отрисовывается в DOM, мы спустимся сюда, увидим наш идентификатор root, наш контейнер, а затем мы увидим div с классом сообщения.  [2:52](https://egghead.io/lessons/react-create-a-simple-reusable-react-component-b1ac5ff9) Теперь давайте посмотрим, что на самом деле представляет собой этот элемент сообщения. Это console.log, этот элемент сообщения. Мы посмотрим на нашу консоль и увидим, что это элемент React, и его тип вместо того, чтобы быть div, как у нас раньше, давайте посмотрим на это для сравнения, console.log div "Hello, World ! " Сохраните это, и мы получим два журнала консоли, поэтому первый будет нашим div. Тип - это строка div, а тип здесь - наша функция сообщения.  [3:19](https://egghead.io/lessons/react-create-a-simple-reusable-react-component-b1ac5ff9) Здесь мы получаем props: { children: “Hello, World!"} и у нас есть props: { children: “Hello, World!"} в этом тоже.  [3:25](https://egghead.io/lessons/react-create-a-simple-reusable-react-component-b1ac5ff9) Итак, когда вы указываете заглавную букву в JSX, это то же самое, что вызывать React.createElement с функцией, на которую вы ссылаетесь. Это позволяет вам создавать собственный компонент, который можно использовать для многократного использования кода.  [3:39](https://egghead.io/lessons/react-create-a-simple-reusable-react-component-b1ac5ff9) Давайте [вырежем](https://egghead.io/lessons/react-create-a-simple-reusable-react-component-b1ac5ff9) это и перепишем, используя JSX. Мы скажем сообщение, а затем воспользуемся опорой msg, потому что это то, что мы ожидаем. А потом вернемся сюда, сохраним, и все работает без ошибок.  [3:55](https://egghead.io/lessons/react-create-a-simple-reusable-react-component-b1ac5ff9) Если мы посмотрим на нашу скомпилированную версию нашего кода, то увидим, что мы вызываем React.createElement с этой функцией сообщения.  [4:03](https://egghead.io/lessons/react-create-a-simple-reusable-react-component-b1ac5ff9) Если бы мы хотели сделать его красивым и декларативным, чтобы мы могли вкладывать эти вещи, как если бы это была children prop, тогда мы просто принимаем детей в качестве имени prop.  [4:12](https://egghead.io/lessons/react-create-a-simple-reusable-react-component-b1ac5ff9) Затем мы можем переименовать его в «children», и эти два функционально эквивалентны. Если мы сохраним это, то все равно получим "Hello World!" и "Прощай, мир!" Но, честно говоря, один из них мне кажется немного приятнее, поэтому я собираюсь переключить оба, чтобы использовать этот синтаксис.  [4:26](https://egghead.io/lessons/react-create-a-simple-reusable-react-component-b1ac5ff9) Что в этом круто, так это то, что я также могу вкладывать их вместе и передавать все, что может отобразить React, включая строку здесь, или дополнительные компоненты здесь, или дополнительные элементы здесь.  [4:40](https://egghead.io/lessons/react-create-a-simple-reusable-react-component-b1ac5ff9) В обзоре того, что мы здесь сделали, у нас был код, который дублировался, и мы хотели повторно использовать этот код в нескольких местах. Итак, мы создали наш собственный компонент пользовательской функции, который представляет собой функцию, которая принимает объект props и возвращает больше элементов React.  [4:54](https://egghead.io/lessons/react-create-a-simple-reusable-react-component-b1ac5ff9) Мы должны были убедиться, что наш функциональный компонент начинается с заглавной буквы, чтобы Babel скомпилировал его для передачи самой функции в React.createElement, а не строковое сообщение в React.createElement. Таким образом, когда React отображает наш элемент, он знает, какую функцию вызывать. | Kent: [0:00](https://egghead.io/lessons/react-create-a-simple-reusable-react-component-b1ac5ff9) So, here we have this div with the class name "container," and we have this div with the class name "message" in here that says "Hello, World!"  [0:06](https://egghead.io/lessons/react-create-a-simple-reusable-react-component-b1ac5ff9) Let's say that I wanted to have, I don't know, maybe two of those. So now we have "Hello, World!" and "Hello, World!" and then I don't want to repeat myself. So I'm going to make a constant message here that is that "Hello, World!" and we'll put that in place of here "message," just interpolate that there. Now I'm not repeating myself, which is good.  [0:27](https://egghead.io/lessons/react-create-a-simple-reusable-react-component-b1ac5ff9) But what if I wanted one of these to say, "Hello, World!" and the other to say, "Goodbye, World!" Well, we need to parameterize this somehow, and when we do that in JavaScript, we create a function. I'll make an arrow function here, and we'll accept an object. We'll call it "props," and then whatever that props.msg or props.message value is, is what we'll interpolate into the children prop for this div that we create.  [0:52](https://egghead.io/lessons/react-create-a-simple-reusable-react-component-b1ac5ff9) Now, instead of just putting "message" in there directly, we're going to call it with an object, and we'll use msg as "Hello, World!" and then we'll do this one msg "Goodbye, World!" Save that. And we get "Hello, World!" and "Goodbye, World!" which is awesome.  [1:10](https://egghead.io/lessons/react-create-a-simple-reusable-react-component-b1ac5ff9) But again, this is not really ergonomic. It doesn't look very good, and one of the benefits to using JSX is the ability for our UI to resemble the declarative nature of HTML.  [1:21](https://egghead.io/lessons/react-create-a-simple-reusable-react-component-b1ac5ff9) What I'd really like to be able to do is say "message" here and then "Hello, World!" and "message." But if I try to do that, you'll notice we actually do get the "Hello, World!" but we're going to get a console warning here saying, "The tag message is unrecognized in this browser. If you meant to render a React component, start its name in an upper case letter."  [1:42](https://egghead.io/lessons/react-create-a-simple-reusable-react-component-b1ac5ff9) Oh, that's interesting. Thank you very much. So let's learn more about that.  [1:46](https://egghead.io/lessons/react-create-a-simple-reusable-react-component-b1ac5ff9) If we look at the compiled version of our code, we're going to see our message function that's returning a React element, that's creating a div. And then if we come down here, we're going to see React.createElement message as a string.  [1:57](https://egghead.io/lessons/react-create-a-simple-reusable-react-component-b1ac5ff9) That's problematic because we don't want to have the message as a string to create a DOM element that is a message type element. We actually wanted to use this function, so that we can use this function to create additional React elements.  [2:11](https://egghead.io/lessons/react-create-a-simple-reusable-react-component-b1ac5ff9) So, as the warning suggests, we need to use an upper case letter, and so if instead, we say capital "Message," then we use capital "Message" here, and let's make sure we don't get errors here. So we'll say capital "Message" for these, and we can save that.  [2:25](https://egghead.io/lessons/react-create-a-simple-reusable-react-component-b1ac5ff9) Now, we're no longer getting that warning. If we look at our compiled code, we're going to see that capital "Message" here, and when we get past to React.createElement, we're going to get a capital Message right here, outside of a string.  [2:38](https://egghead.io/lessons/react-create-a-simple-reusable-react-component-b1ac5ff9) So, we're passing a reference to the function directly into the React.createElement(). And then if we ultimately look at what's rendered to the DOM, we come down here, we see our ID of root, our container, and then we see a div with the class of message.  [2:52](https://egghead.io/lessons/react-create-a-simple-reusable-react-component-b1ac5ff9) Now, let's take a look at what this message element really is. It's console.log that Message element. We'll look at our console, and we'll see that it is a React element, and its type instead of being a div like we had before, let's take a look at that to compare, console.log div "Hello, World!" Save that and we'll get two console logs, so the first one being our div. The type is a string of div, whereas the type here is our message function.  [3:19](https://egghead.io/lessons/react-create-a-simple-reusable-react-component-b1ac5ff9) Here we get the props: { children: “Hello, World!"} and we have props: { children: “Hello, World!"} in this one as well.  [3:25](https://egghead.io/lessons/react-create-a-simple-reusable-react-component-b1ac5ff9) So, when you give a capital letter in JSX, that's the same as calling React.createElement with the function that you're referencing. That allows you to create a custom component that you can use to reuse code over and over again.  [3:39](https://egghead.io/lessons/react-create-a-simple-reusable-react-component-b1ac5ff9) Let's go ahead and cut this out, and we'll rewrite this using JSX. We'll say message, and then use the msg prop, because that's what we're expecting. And then we'll come back here, save that, and everything's working without any errors.  [3:55](https://egghead.io/lessons/react-create-a-simple-reusable-react-component-b1ac5ff9) If we look at our compiled version of our code, then we'll see that we're calling React.createElement with that message function.  [4:03](https://egghead.io/lessons/react-create-a-simple-reusable-react-component-b1ac5ff9) If we wanted to make it nice and declarative so that we could nest to these things as if it's the children prop, then, we just accept children for the name of the prop.  [4:12](https://egghead.io/lessons/react-create-a-simple-reusable-react-component-b1ac5ff9) Then we can rename this one to "children," and those two are functionally equivalent. If we save this, then we'll still get "Hello World!" and "Goodbye, World!" But honestly, one of those looks a little bit nicer to me, so I'm going to switch both of these to use that syntax.  [4:26](https://egghead.io/lessons/react-create-a-simple-reusable-react-component-b1ac5ff9) What's cool about this is I'm also able to nest these together and pass anything that React can render, which includes the string here, or additional components here, or additional elements here.  [4:40](https://egghead.io/lessons/react-create-a-simple-reusable-react-component-b1ac5ff9) In review of what we did here was we had some code that was being duplicated, and we wanted to reuse that code in multiple places. So, we created our own custom function component which is a function that accepts a props object and returns more React elements.  [4:54](https://egghead.io/lessons/react-create-a-simple-reusable-react-component-b1ac5ff9) We had to make sure that our function component started with a capital letter so that Babel would compile this to pass the function itself to React.createElement rather than the string message to React.createElement. That way, when React renders our element, it knows what function to call. |
| 8. Проверка свойств настраиваемого компонента React с помощью PropTypes | 8. Validate Custom React Component Props with PropTypes |
| Когда вы создаете повторно используемые компоненты React, вы хотите убедиться, что люди используют их правильно. Лучший способ сделать это - использовать TypeScript в вашей кодовой базе, чтобы вы могли проверять ваш код во время компиляции. Но если вы не используете TypeScript, вы все равно можете использовать PropTypes для проверки во время выполнения. Между нами, когда я еще работал над AngularJS, я так завидовал этой функции, что [попытался встроить ее в AngularJS](https://github.com/angular/angular.js/issues/11657) .  В этом уроке мы узнаем, как работают PropTypes, почему они не включены в производственной среде и как использовать предварительно созданный пакет [prop-types](https://www.npmjs.com/package/prop-types) от команды React. | When you create reusable React components, you want to make sure that people use them correctly. The best way to do this is to use TypeScript in your codebase to give you compile-time checking of your code. But if you’re not using TypeScript, you can still use PropTypes to get runtime validation. Between you and me, when I was still doing AngularJS work, I was so envious of this feature, that [I tried to build it into AngularJS](https://github.com/angular/angular.js/issues/11657).  In this lesson we’ll learn how PropTypes work, why they’re not enabled in production, and how to use the pre-built [prop-types](https://www.npmjs.com/package/prop-types) package from the React team. |
| Кент: [00:00](https://egghead.io/lessons/react-validate-custom-react-component-props-with-proptypes-2f1d4768#t=0) Когда вы начнете создавать пользовательские функциональные компоненты, у вас будут люди, использующие эти функциональные компоненты, и они могут использовать это неправильно. Например, здесь у нас есть это: «Сказать привет!» это принимает объект props, который мы деструктурируем прямо здесь.  [00:13](https://egghead.io/lessons/react-validate-custom-react-component-props-with-proptypes-2f1d4768#t=13) Мы получаем prop для имени и фамилии и ожидаем, что сможем их отобразить, но способ ее использования заключается в том, что для prop имени назначается значение false, и мы не предоставляем prop фамилия. Конечный результат - это не совсем то, что мы ищем.  [00:28](https://egghead.io/lessons/react-validate-custom-react-component-props-with-proptypes-2f1d4768#t=28) Команда React разработала пакет под названием PropTypes, который позволяет добавлять во время выполнения проверку свойств, передаваемых вашим компонентам. Вот почему React поддерживает функцию, называемую типами свойств, которая позволяет вам проверять типы свойств, которые передаются вашим компонентам при их рендеринге.  [00:45](https://egghead.io/lessons/react-validate-custom-react-component-props-with-proptypes-2f1d4768#t=45) Чтобы добавить поддержку этого, мы собираемся сделать SayHello.PropTypes равным этому объекту. У нас будет ключ для prop, который мы хотим проверить. Это будет функция, которая принимает the prop name и component name. Каждый раз, когда наш компонент SayHello визуализируется, мы можем проверить, правильно ли передано prop first name.  [01:07](https://egghead.io/lessons/react-validate-custom-react-component-props-with-proptypes-2f1d4768#t=67) Мы можем сказать: «Если тип prop в имени prop не равен строке, то мы можем вернуть новую ошибку, которая говорит:« Эй, имя компонента требует, чтобы имя prop было строкой, но вы передали тип prop по имени prop. '"  [01:35](https://egghead.io/lessons/react-validate-custom-react-component-props-with-proptypes-2f1d4768#t=95) Если мы сохраним это, то увидим, что предупреждение появляется. Неудачный prop-type. Привет, компонент "Сказать привет!" необходимо, чтобы prop first name было строкой, но вы передали логическое значение.  [01:46](https://egghead.io/lessons/react-validate-custom-react-component-props-with-proptypes-2f1d4768#t=106) Мы могли бы сделать то же самое для last name prop. Что я собираюсь сделать, так это вытащить это. Здесь мы создадим last name prop, и я собираюсь создать здесь объект типа опоры, в котором у нас есть строка, которая является только что созданным валидатором. Тогда здесь мы можем сказать, что имя - PropTypes.string и фамилия - PropTypes.string.  [02:08](https://egghead.io/lessons/react-validate-custom-react-component-props-with-proptypes-2f1d4768#t=128) Мы могли бы добавить еще несколько валидаторов к типам свойств, для чисел и всего, что вы можете себе представить.  [02:14 Таким образом](https://egghead.io/lessons/react-validate-custom-react-component-props-with-proptypes-2f1d4768#t=134) , мы получаем предупреждение для обоих. Это настолько распространено, что команда React создала пакет на npm, который мы можем использовать, и я собираюсь извлечь его из распаковки.  [02:24](https://egghead.io/lessons/react-validate-custom-react-component-props-with-proptypes-2f1d4768#t=144) Я вставлю это прямо сюда, и это называется prop-types. При этом он фактически создает здесь для нас глобальную переменную, называемую prop-types, когда мы используем ее как подобный скрипт. У него тот же API, который мы только что создали, какой форсайт.  [02:37](https://egghead.io/lessons/react-validate-custom-react-component-props-with-proptypes-2f1d4768#t=157) Мы это сохраним. Мы получим обновление, и все работает немного по-другому. Раньше мы получали предупреждение как для имени, так и для фамилии, но теперь мы получаем предупреждение только для имени.  [02:48](https://egghead.io/lessons/react-validate-custom-react-component-props-with-proptypes-2f1d4768#t=168) Причина в том, что проп-типы по умолчанию не требуются. Поскольку фамилия не была указана, в этом случае мы не проверяем ее. Это хорошо работает, если вы укажете для фамилии значение по умолчанию, например, если бы мы сказали здесь неизвестно.  [03:04](https://egghead.io/lessons/react-validate-custom-react-component-props-with-proptypes-2f1d4768#t=184) Но если вы хотите убедиться, что это требуется, вы можете добавить «точка требуется на обоих из них». Сохраните это, и теперь вы получите предупреждение о том, что имя неправильного типа, а фамилия обязательна, но не указана.  [03:18](https://egghead.io/lessons/react-validate-custom-react-component-props-with-proptypes-2f1d4768#t=198) Одна из разумных проблем, которые могут возникнуть у вас по поводу типов свойств, заключается в том, что они добавляют изрядное количество кода, который необходимо запускать всякий раз, когда React отображает ваши компоненты, что может повлиять на производительность.  [03:28](https://egghead.io/lessons/react-validate-custom-react-component-props-with-proptypes-2f1d4768#t=208) Давайте взглянем на production.minified версию React, и мы заметим, что все работает точно так же, как и раньше, но мы больше не получаем эти предупреждения. Это потому, что в производстве проп-типы не применяются.  [03:44](https://egghead.io/lessons/react-validate-custom-react-component-props-with-proptypes-2f1d4768#t=224) Если вы хотите пойти дальше, вы можете получить плагин Babel, который удалит типы свойств из вашего исходного кода для производства. Этот плагин называется Babel-plugin-transform-remove-prop-types. Вы можете установить это и использовать в своей производственной сборке, или многие наборы инструментов устанавливают и используют это по умолчанию.  [04:02](https://egghead.io/lessons/react-validate-custom-react-component-props-with-proptypes-2f1d4768#t=242) Если вы хотите узнать больше о доступных prop-types, вы можете перейти на страницу prop-types на npm. Существует множество различных типов, которые вы можете использовать для своих компонентов, чтобы убедиться, что люди правильно используют ваши компоненты. | Kent: [00:00](https://egghead.io/lessons/react-validate-custom-react-component-props-with-proptypes-2f1d4768#t=0) When you start creating custom function components, you're going to have people using those function components and they may not use it quite right. For example, here, we have this, "Say Hello!" that's accepting a props object we're destructuring right here.  [00:13](https://egghead.io/lessons/react-validate-custom-react-component-props-with-proptypes-2f1d4768#t=13) We're getting a first name and a last name prop and expecting to be able to render those out, but the way that it's being used is we have that first name prop being assigned to false and we're not providing the last name prop at all. The end result is not exactly what we're looking for.  [00:28](https://egghead.io/lessons/react-validate-custom-react-component-props-with-proptypes-2f1d4768#t=28) The React team has developed a package called PropTypes that allows you to add run time validation of the props that are passed to your components. This is why React supports a feature called prop-types that allows you to validate the types of props that are being passed to your components when they're rendered.  [00:45](https://egghead.io/lessons/react-validate-custom-react-component-props-with-proptypes-2f1d4768#t=45) To add support for that, we're going to have SayHello.PropTypes equals this object. We'll have a key for the prop that we want to validate. That's going to be a function that takes the prop name and the component name. Any time our SayHello component is rendered, we can check that the first name prop was passed properly.  [01:07](https://egghead.io/lessons/react-validate-custom-react-component-props-with-proptypes-2f1d4768#t=67) We can say, "If type of props at prop name is not equal to a string, then we can return a new error that says, 'Hey, the component name needs the prop name to be a string, but you passed a type of props at prop name.'"  [01:35](https://egghead.io/lessons/react-validate-custom-react-component-props-with-proptypes-2f1d4768#t=95) If we save that, then we can see that warning shows up. Failed prop-type. Hey, the component "Say, Hello!" needs the prop first name to be a string, but you passed a boolean.  [01:46](https://egghead.io/lessons/react-validate-custom-react-component-props-with-proptypes-2f1d4768#t=106) We could do the same thing for our last name prop. What I'm going to do is pull this out. We'll create a last name prop here and I'm going to make a prop-types object here where we have a string that is that validator that we just created. Then here, we can say first name is PropTypes.string and last name is PropTypes.string.  [02:08](https://egghead.io/lessons/react-validate-custom-react-component-props-with-proptypes-2f1d4768#t=128) We could add a bunch more validators to the prop-types, for numbers and whatever else you could imagine.  [02:14](https://egghead.io/lessons/react-validate-custom-react-component-props-with-proptypes-2f1d4768#t=134) With that, we get a warning for both of these. This is so common that the React team created a package on npm that we can use and I'm going to pull it in from unpackage.  [02:24](https://egghead.io/lessons/react-validate-custom-react-component-props-with-proptypes-2f1d4768#t=144) I'll paste that right in here and it's called prop-types. With that, it actually creates a global variable here for us called prop-types when we're using it as a script like that. It has the same API that we just created, what foresight.  [02:37](https://egghead.io/lessons/react-validate-custom-react-component-props-with-proptypes-2f1d4768#t=157) We'll save that. We'll get a refresh and things are working a little differently. Before, we were getting a warning for both the first name and the last name, but now we're just getting one for the first name.  [02:48](https://egghead.io/lessons/react-validate-custom-react-component-props-with-proptypes-2f1d4768#t=168) The reason for that is that prop-types by default are not required. Because the last name wasn't provided, we're not getting the last name validated in this case. That works out nicely if you provide a default for the last name, like if we were to say unknown here.  [03:04](https://egghead.io/lessons/react-validate-custom-react-component-props-with-proptypes-2f1d4768#t=184) But if you want to make sure that it's required, then you can add "a dot is required on both of these." Save that and now you'll get a warning saying that the first name is the wrong type and the last name is required but wasn't provided.  [03:18](https://egghead.io/lessons/react-validate-custom-react-component-props-with-proptypes-2f1d4768#t=198) One reasonable concern you might have about prop-types is that it adds a fair amount of code that needs to be run whenever React is rendering your components which may impact performance.  [03:28](https://egghead.io/lessons/react-validate-custom-react-component-props-with-proptypes-2f1d4768#t=208) Let's take a look at the production.minified version of React and we'll notice that everything is working exactly as it was before but we're no longer getting those warnings. That's because in production prop-types are not applied.  [03:44](https://egghead.io/lessons/react-validate-custom-react-component-props-with-proptypes-2f1d4768#t=224) If you want to take things a step further, then you can actually get a Babel plugin that will remove the prop-types from your source code for production. That plugin is called Babel-plugin-transform-remove-prop-types. You could install that and use that in your production build, or many tool kits install and use this by default.  [04:02](https://egghead.io/lessons/react-validate-custom-react-component-props-with-proptypes-2f1d4768#t=242) If you want to learn more about the prop-types that are available, you can go to the prop-types page on npm. There are a lot of different types that you can use for your components to validate that people are using your components properly. |
| 9. Понять и использовать интерполяцию в JSX | 9. Understand and Use Interpolation in JSX |
| Теперь, когда у нас есть хорошая основа для элементов React, JSX, настраиваемых элементов и свойств, давайте напишем компонент React с некоторой условной логикой для исследования характеристик интерполяции синтаксиса JSX. Мы узнаем, что интерполяция не уникальна для JSX, и мы узнаем ограничения и возможности интерполяции в JSX. Обладая этими знаниями, мы будем более продуктивными при использовании JSX. | Now that we have a good foundation on React elements, JSX, custom elements, and props, let’s write a React component that has some conditional logic in it to explore the interpolation characteristics of JSX syntax. We’ll learn that interpolation is not unique to JSX and we’ll learn the limitations and capabilities of interpolation in JSX. With this knowledge, we’ll be more productive using JSX. |
| Инструктор: [00:00](https://egghead.io/lessons/react-understand-and-use-interpolation-in-jsx#t=0) Здесь у нас есть элемент React, который является фрагментом React. В качестве его дочерних элементов у нас есть счетчик символов с текстом «hello world» и счетчик символов с текстом «пустой строки». В конечном счете, мы хотим, чтобы это отображало текст «hello world имеет 11 символов» и текст «пустая строка не имеет символов».  [00:18](https://egghead.io/lessons/react-understand-and-use-interpolation-in-jsx#t=18) Давайте продолжим и реализуем это. У нас уже есть наш функциональный компонент. Мы принимаем text prop и тут же ее разрушаем. Что мы собираемся сделать, так это обернуть это в div, а затем мы хотим сказать, что the text и текст содержат text.length characters. Это для первого случая.  [00:39 Здесь](https://egghead.io/lessons/react-understand-and-use-interpolation-in-jsx#t=39) нам нужно кое-что вставить. Для этого текста прямо здесь, я думаю, было бы проще всего интерполировать все это как литерал шаблона, поэтому мы сделаем это. Мы поместим это в фигурные скобки внутри литерала шаблона. После этого мы добавим дополнительный пробел.  [00:56](https://egghead.io/lessons/react-understand-and-use-interpolation-in-jsx#t=56) Что касается длины текста, мы можем интерполировать это прямо сейчас. Мы сохраним это, и здесь у нас есть текст «hello world имеет 11 символов». Давайте продолжим и поместим этот текст в a strong элемент, чтобы он был полужирным. Мы скажем strong, длина текста, strong, за исключением этого. Теперь он выделен жирным шрифтом.  [01:16](https://egghead.io/lessons/react-understand-and-use-interpolation-in-jsx#t=76) Мы не хотим указывать длину текста для нулевых символов. Мы хотим, чтобы не было символов. Давайте продолжим и скажем, что наша длина равна, если есть длина текста, мы скажем длину текста. В противном случае мы скажем нет. Затем мы можем вставить это прямо сюда. Теперь у нас есть 11 characters и нет characters.  [01:38](https://egghead.io/lessons/react-understand-and-use-interpolation-in-jsx#t=98) Благодаря фигурным скобкам, допускающим интерполяцию, мы могли взять это выражение целиком, вставить его прямо туда и полностью избавиться от этой переменной. Это тоже сработает. Теперь предположим, что мы не хотим, чтобы внутри этого strong было «нет». Мы только хотим, чтобы в strong это было обернуто вокруг действительного числа. Это потребует от нас немного изменить ситуацию.  [01:58](https://egghead.io/lessons/react-understand-and-use-interpolation-in-jsx#t=118) Я добавлю это прямо здесь. Мы скажем text.length, поэтому, если есть длина, мы хотим, чтобы она была strong с text.length. В противном случае мы просто будем использовать строку no. С этим давайте продолжим и интерполируем эту символьную строку как строку, чтобы у нас было явное пространство прямо здесь. Тогда мы это сохраним. Теперь мы получаем 11 полужирным шрифтом и «нет» как не выделенным в полужирный.  [02:23](https://egghead.io/lessons/react-understand-and-use-interpolation-in-jsx#t=143) Похоже, (Интерпретатор?) Prettier добавил сюда дополнительное место. На самом деле это не обязательно, поэтому я удалю это, и мы получим здесь тот же результат.  [02:30](https://egghead.io/lessons/react-understand-and-use-interpolation-in-jsx#t=150) Причина, по которой я хотел вам это показать, заключалась в том, чтобы продемонстрировать, каково это - переходить между областью JSX и областью JavaScript и обратно. Давайте проследим за этим синтаксисом. Когда мы начинаем с функции, это область JavaScript. Мы можем выполнять функции JavaScript, операторы if и все такое. Затем, когда мы выполняем возврат и открывающие круглые скобки, это все еще область JavaScript.  [02:49](https://egghead.io/lessons/react-understand-and-use-interpolation-in-jsx#t=169) Мы входим в область JSX, создавая эту открывающую скобку с тегом. Теперь мы находимся в области JSX, и у нас есть синтаксис prop, у нас есть дочерний синтаксис, а затем у нас есть закрывающий тег, и мы снова входим в область JavaScript. Внутри открывающего и закрывающего тегов мы находимся в области JSX. Как только мы сделаем открывающую фигурную скобку, мы снова окажемся в области JavaScript.  [03:12](https://egghead.io/lessons/react-understand-and-use-interpolation-in-jsx#t=192) Это ограниченная область JavaScript, потому что мы можем делать только выражения. Нам не разрешено делать такие вещи, как циклы for, операторы if или что-либо подобное. Это должно быть выражение JavaScript, которое дает некоторое значение. Строка, подобная этой, или подобное тернарное выражение, это выражения, и они разрешены в фигурных скобках.  [03:34](https://egghead.io/lessons/react-understand-and-use-interpolation-in-jsx#t=214) Если вы остановитесь и [задумаетесь](https://egghead.io/lessons/react-understand-and-use-interpolation-in-jsx#t=214) , для чего этот JSX скомпилирован, все это должно иметь смысл. Давайте посмотрим на код JavaScript, который Babel генерирует для нас. Здесь у нас есть character counter (счетчик). Мы вызываем react.createElement. Мы передаем div, представляющий этот div, прямо здесь. Он не принимает никаких props, поэтому мы получаем null. Затем в качестве третьего аргумента мы получаем эту строку.  [03:55](https://egghead.io/lessons/react-understand-and-use-interpolation-in-jsx#t=235) Давайте продолжим и напишем это здесь. У нас есть react.createElement div, null для свойств, а затем строка, текст и так далее. Если бы мы захотели вместо передачи этой строки выполнить какой-нибудь оператор if, это бы вообще не сработало. Нет никакого смысла передавать оператор в качестве аргумента функции.  [04:16](https://egghead.io/lessons/react-understand-and-use-interpolation-in-jsx#t=256) По той же причине вы не можете использовать операторы внутри фигурных скобок JSX, потому что фигурные скобки в основном говорят: «Эй, Babel, когда ты перейдешь к этой части, я хочу, чтобы ты взял все, что находится между этими двумя фигурными скобками, а затем просто вставь их в этот аргумент вызова createElement ". Хорошее понимание того, как Babel компилирует ваш JSX, поможет вам использовать JSX более эффективно.  [04:37](https://egghead.io/lessons/react-understand-and-use-interpolation-in-jsx#t=277) Как только мы введем эту закрывающую фигурную скобку, мы вернемся в область JSX. Затем мы возвращаемся к фигурной скобке, и теперь мы снова в области JavaScript. У нас есть это выражение. Пока мы находимся в середине этой области JavaScript с тернарным выражением, мы фактически снова входим в область JSX, так что мы находимся на нескольких уровнях в этом.  [04:55](https://egghead.io/lessons/react-understand-and-use-interpolation-in-jsx#t=295) Начнем с JavaScript. Затем мы переходим в JSX, затем в JavaScript, затем снова в JSX, а затем в JavaScript. Мы выходим из JavaScript, мы входим в JSX, выходим из JSX и снова возвращаемся в JavaScript. Затем выходим из JavaScript и переходим в JSX. Перепады между JavaScript и JSX действительно распространены.  [05:20](https://egghead.io/lessons/react-understand-and-use-interpolation-in-jsx#t=320) Этот тип интерполяции характерен не только для JSX. Фактически, у нас есть пара других примеров интерполяции прямо здесь. Этот файл представляет собой HTML-файл, и мы начнем с HTML. Мы все еще просматриваем HTML, пока не дойдем до конца открывающего тега скрипта. Теперь мы находимся в области JavaScript. У нас также есть теги стиля, и теперь мы находимся на поле CSS. В HTML всегда была такая интерполяция.  [05:46 В](https://egghead.io/lessons/react-understand-and-use-interpolation-in-jsx#t=346) JavaScript также есть встроенная интерполяция, когда мы говорим о шаблонных литералах. Прямо здесь у нас есть этот шаблонный литерал. Мы в области строки. Затем, когда мы выполняем $ {, теперь мы находимся в области JavaScript. Фактически, это имеет те же ограничения, что и JSX для своих интерполяций.  [06:04 В](https://egghead.io/lessons/react-understand-and-use-interpolation-in-jsx#t=364) частности, здесь нельзя использовать операторы if, циклы for или переключатели. Все, что вы здесь вкладываете, должно быть выражением. Это выражение может быть сложным. Мы можем вставить туда тернарные. Мы могли бы даже добавить выражение функции, вызываемой немедленно, хотя я этого не советую.  [06:20 Хорошее](https://egghead.io/lessons/react-understand-and-use-interpolation-in-jsx#t=380) понимание JSX и интерполяция выражений JavaScript - ключ к правильному использованию JSX. Это то, в чем вы со временем усовершенствуетесь, если будете использовать JSX. | Instructor: [00:00](https://egghead.io/lessons/react-understand-and-use-interpolation-in-jsx#t=0) Here we have a React element that is a React fragment. As its children, we have a character count with the text of "hello world," and a character count with the text of "empty string." Ultimately, we want this to render the text "hello world has 11 characters," and the text "empty string has no characters."  [00:18](https://egghead.io/lessons/react-understand-and-use-interpolation-in-jsx#t=18) Let's go ahead and implement this. We already have our function component here. We're accepting the text prop and destructuring it right there. What we're going to do is I want to wrap this in a div, and then we want to say the text and text has text.length characters. That's for this first case.  [00:39](https://egghead.io/lessons/react-understand-and-use-interpolation-in-jsx#t=39) We need to interpolate some stuff here. For this text right here, I think it would be easiest to interpolate all of this as a template literal, so we'll do that. We'll put this inside of curly braces inside of that template literal. Then we'll put an extra space after this.  [00:56](https://egghead.io/lessons/react-understand-and-use-interpolation-in-jsx#t=56) For the text length, we can interpolate that right there. We'll save this, and here we have the text "hello world has 11 characters." Let's go ahead and put this text length in a strong element so that it's bold. We'll say strong, text length, strong, save that. Now it's bolded.  [01:16](https://egghead.io/lessons/react-understand-and-use-interpolation-in-jsx#t=76) We don't want to say text length for zero characters. We want it to say no characters. Let's go ahead and say our length is equal to, if there is a text length, then we'll say the text length. Otherwise, we'll say no. Then we can interpolate that directly into here. Now we're getting 11 characters and no characters.  [01:38](https://egghead.io/lessons/react-understand-and-use-interpolation-in-jsx#t=98) Thanks to this curly braces allowing interpolation, we could take this whole expression, paste it directly into there, and get rid of that variable altogether. That will work just as well. Now let's say we don't want "no" to be inside of this strong. We only want this strong to be wrapped around an actual number. That will require us to change things up a little bit.  [01:58](https://egghead.io/lessons/react-understand-and-use-interpolation-in-jsx#t=118) I'll add that right here. We'll say text.length, so if there is a length, then we want that in a strong with text.length. Otherwise, we'll just use the string no. With that, let's go ahead and interpolate this character string as a string so we can have an explicit space right there. Then we'll save this. Now we get 11 as bolded, and "no" as not bolded.  [02:23](https://egghead.io/lessons/react-understand-and-use-interpolation-in-jsx#t=143) It looks like Prettier added this extra space here. It's not really necessary, so I'll go ahead and remove that, and we'll get the same result here.  [02:30](https://egghead.io/lessons/react-understand-and-use-interpolation-in-jsx#t=150) The reason that I wanted to show you this was to demonstrate what it's like to go between JSX land and JavaScript land and back again. Let's follow the flow of this syntax. When we start in the function, this is JavaScript land. We can do JavaScript functions, if statements, all of that stuff. Then when we do the return and this open parentheses, that's still JavaScript land.  [02:49](https://egghead.io/lessons/react-understand-and-use-interpolation-in-jsx#t=169) We enter JSX land by doing this open-angle bracket with the tag. Now we're in JSX land, and we have prop syntax, we have children syntax, and then we have our closing tag, and we enter JavaScript land again. Inside of the opening and closing tag, we're in JSX land. Once we do an opening curly brace, we're now in JavaScript land again.  [03:12](https://egghead.io/lessons/react-understand-and-use-interpolation-in-jsx#t=192) This is a limited JavaScript land because we can only do expressions. We're not allowed to do things like for loops, if statements, or any of that. It has to be an expression of JavaScript that evaluates to some value. A string like this or a ternary like this, those are expressions, and those are allowed within the curly braces.  [03:34](https://egghead.io/lessons/react-understand-and-use-interpolation-in-jsx#t=214) If you stop to consider what this JSX is compiled to, this should all make sense. Let's take a look at the JavaScript code that Babel generates for us. Here we have our character counter. We're calling react.createElement. We pass the div that's representing this div right here. It's not taking any props, so we get null. Then as the third argument, we get this string.  [03:55](https://egghead.io/lessons/react-understand-and-use-interpolation-in-jsx#t=235) Let's go ahead and write this over here. We have react.createElement div, null for the props, and then the string, the text, and so on. If we wanted to, instead of passing this string, do some sort of if statement, that wouldn't work at all. It doesn't make any sense to pass a statement as an argument to a function.  [04:16](https://egghead.io/lessons/react-understand-and-use-interpolation-in-jsx#t=256) For the same reason, you can't use statements inside of the curly braces of JSX because the curly braces basically are saying, "Hey, Babel, as you get to this part, I want you to take everything between these two curly braces and then just stick it in this argument of the createElement call." Having a good understanding of how Babel is compiling your JSX will help you use JSX more effectively.  [04:37](https://egghead.io/lessons/react-understand-and-use-interpolation-in-jsx#t=277) Once we enter that closing curly brace, we're back in JSX land. Then we come back to this curly brace, and we're now in JavaScript land again. We have this expression. While we're in the middle of this JavaScript land with the ternary expression, we actually enter JSX land again, so we're a couple layers into this.  [04:55](https://egghead.io/lessons/react-understand-and-use-interpolation-in-jsx#t=295) We start out in JavaScript. Then we go into JSX, then we go into JavaScript, then we go into JSX again, and then we go into JavaScript. We exit JavaScript, we enter JSX, we exit JSX, and we're back into JavaScript. Then we exit JavaScript and go into JSX. This back and forth between JavaScript and JSX is really common.  [05:20](https://egghead.io/lessons/react-understand-and-use-interpolation-in-jsx#t=320) This type of interpolation is not unique to just JSX. In fact, we have a couple other examples of interpolation right here. This file is an HTML file, and we start with HTML all right here. We're still going through HTML until we hit the end of this opening script tag. Now we're in JavaScript land. We also have style tags, and now we're in CSS land. HTML has always had this type of interpolation.  [05:46](https://egghead.io/lessons/react-understand-and-use-interpolation-in-jsx#t=346) JavaScript also has built-in interpolation when we're talking about template literals. Right here, we have this template literal. We're in string land. Then when we do the $ {, now we're in JavaScript land. In fact, this has the same limitations that JSX has for its interpolations.  [06:04](https://egghead.io/lessons/react-understand-and-use-interpolation-in-jsx#t=364) Specifically, you can't use if statements, for loops, or switch statements in here. Whatever you put in here has to be an expression. That expression can be complex. We can put ternaries in there. We could even put an immediately invoked function expression, though I don't advise that.  [06:20](https://egghead.io/lessons/react-understand-and-use-interpolation-in-jsx#t=380) Having a strong understanding of JSX and interpolating JavaScript expressions is key to using JSX well. It's something that you'll get better with over time as you're using JSX. |
| 10. Повторно отрендерить приложение React | **10. Rerender a React Application** |
| Приложения на самом деле не являются приложениями, если они не изменяются с течением времени, отражая изменения в приложении с течением времени. Обычно в React вы используете состояние для управления этим, но прежде чем мы перейдем к этому, мы просто вызовем ReactDOM.render для того же элемента, чтобы вы понимали, что React делает для вас. Итак, мы узнаем, как React работает с новыми элементами, которые вы ему предоставляете, сравним его с предыдущими элементами и внесем хирургические обновления в DOM, чтобы дать вам максимально быстрый и лучший пользовательский интерфейс (поскольку обновление DOM обычно является самой медленной частью. в целом процессе). | Applications aren’t really applications if they don’t change over time to represent changes in the application over time. Normally in React you’ll use state to manage this, but before we get to that, we’ll just call ReactDOM.render on the same element so you get an understanding of what React is doing for you. So we’ll learn how React deals with the new elements you give it, compare it to the previous elements, and make surgical updates to the DOM to give you the fastest and best user experience possible (because updating the DOM is typically the slowest part in the whole process). |
| Инструктор: [0:00 Мы Сделали](https://egghead.io/lessons/react-rerender-a-react-application-0bf050ac) приложение часов. Здесь оно отображает время, и мы получаем это время с new date toLocaleTimeString, а затем мы помещаем это в div и визуализируем. Проблема в том, что он устаревает, как только вы обновляете страницу. Каждый раз, когда вы обновляете страницу или обновляете ее, но вам нужно постоянно обновлять страницу, и какая это боль.  [0:19](https://egghead.io/lessons/react-rerender-a-react-application-0bf050ac) Вместо этого я хочу поместить это в функцию и вызывать эту функцию каждую секунду. Давайте продолжим и создадим эту функцию под названием tick. Мы поместим все это в tick, а затем вызовем tick. Мы просто реорганизовали, просто добавили что-то в функцию. Это вполне нормально. Теперь я скажу установить интервал, поставить tick каждые 1000 миллисекунд. Я получаю это обновление.  [0:48](https://egghead.io/lessons/react-rerender-a-react-application-0bf050ac) Я хочу показать вам, как это работает, если мы посмотрим в тело, а затем перейдем к нашему приложению. Мы увидим это обновление в нашей строке прямо здесь. Если бы я добавил что-то здесь, например, мы поместили еще один div, hello и еще один div прямо здесь, а затем вставили это время туда, тогда мы заметили бы, что единственное, что обновляется, это время этих часов, а не этот div.  [1:14](https://egghead.io/lessons/react-rerender-a-react-application-0bf050ac) Давайте посмотрим, чем это будет отличаться, если мы просто превратим это в строку HTML, и мы интерполируем это. Затем вместо ReactDOM.render мы скажем, что root element.innerHTML равен этому элементу.  [1:31](https://egghead.io/lessons/react-rerender-a-react-application-0bf050ac) Мы получаем то же поведение, что и раньше, за исключением того, как оно работает: каждый раз обновляется все содержимое нашего приложения, начиная с этого root element. Это не оптимально, потому что вызывает несколько проблем. Давайте посмотрим на пример.  [1:47](https://egghead.io/lessons/react-rerender-a-react-application-0bf050ac) Если я сделаю ввод и установлю значение на это время, и давайте продолжим, и мы поместим сюда два, то мы сохраним это, и теперь у нас есть два из них, и они обновляются каждую секунду , но когда я щелкаю здесь, мое внимание уходит. Это потому, что старые элементы, которые были входными данными, полностью удаляются из DOM, а новые элементы помещаются на их место. У React такой проблемы нет.  [2:13](https://egghead.io/lessons/react-rerender-a-react-application-0bf050ac) Давайте продолжим и снова превратим это в JSX. Просто уберите это, уберите это и заключительные кавычки, а затем мы удалим это и закомментируем. Теперь мы вернулись, и все, что мы получаем, - это обновление значений, которые действительно имеют значение для нашего приложения, которые являются свойствами для обоих этих входов.  [2:36](https://egghead.io/lessons/react-rerender-a-react-application-0bf050ac) Это имеет большое значение для производительности наших приложений, а также для доступности, потому что React отслеживает, на что мы ориентированы.  [2:44](https://egghead.io/lessons/react-rerender-a-react-application-0bf050ac) Это не обычный способ рендеринга нового приложения. Как правило, при изменении состояния вам не нужно повторно визуализировать все приложение, но я хотел показать вам это, чтобы вы понимали, что React действительно делает для вас.  [2:56](https://egghead.io/lessons/react-rerender-a-react-application-0bf050ac) Когда вы создаете эти элементы React, вы передаете их ReactDOM.render или запускаете повторную визуализацию компонента, и React сравнивает элементы, которые вы вернули на этот раз, с элементами, которые вы вернули в прошлый раз.  [3:09](https://egghead.io/lessons/react-rerender-a-react-application-0bf050ac) Он собирается выполнить div из этих двух элементов, а затем хирургическим путем обновит DOM, чтобы обновить только те вещи, которые были разными между прошлым и этим разом, когда вы вернули JSX. | Instructor: [0:00](https://egghead.io/lessons/react-rerender-a-react-application-0bf050ac) We made a clock application. Here, it's rendering out the time and we get that time with new date toLocaleTimeString, and then we put that inside of a div and render that. The problem is that it gets out of date as soon as you refresh the page. Every time you refresh the page or get that updated, but you have to keep on refreshing the page, and what a pain that is.  [0:19](https://egghead.io/lessons/react-rerender-a-react-application-0bf050ac) What I want to do instead is I'm going to put this into a function and have that function called every second. Let's go ahead and make that function called tick. We'll put all this inside of tick, and then we'll call tick. We just refactored, just put something in a function. That's a very normal thing to do. Now I'm going to say set interval, tick, every 1,000 milliseconds. I'm getting that update.  [0:48](https://egghead.io/lessons/react-rerender-a-react-application-0bf050ac) The thing that I want to show you about the way that this is working is if we look in the body and then come down here to our app. We're going to see that update happening on our string right here. If I were to add something in here, like we put another div, hello, and another div right there, and then paste that time in there, then we're going to notice that the only thing that's updating is this clock time and not this div.  [1:14](https://egghead.io/lessons/react-rerender-a-react-application-0bf050ac) Let's see how this differs if we were to just turn this into an HTML string, and we'll interpolate that. Then instead of ReactDOM.render, we're going to say root element.innerHTML equals that element.  [1:31](https://egghead.io/lessons/react-rerender-a-react-application-0bf050ac) We're getting that same behavior that we had before, except the way that it works is it's updating the entire contents of our application every time, starting from this root element. That's not optimal, because it causes a couple of problems. Let's take a look at an example.  [1:47](https://egghead.io/lessons/react-rerender-a-react-application-0bf050ac) If I make an input and set the value to that time, and let's go ahead and we'll put two in here, then we'll save that, and now we have two of those, and they are updating every second, but as I click in here, my focus is going away. That's because the old elements that were the inputs there are getting totally removed from the DOM, and the new elements are getting put in their place. React doesn't have this problem.  [2:13](https://egghead.io/lessons/react-rerender-a-react-application-0bf050ac) Let's go ahead and turn this back into JSX. Just take away that, take away that, and the closing quotes, and then we'll remove this and comment that out. Now we're back, and all that we're getting is an update to the values that actually matter for our application, which are the properties for both of these inputs.  [2:36](https://egghead.io/lessons/react-rerender-a-react-application-0bf050ac) This has great implications for the performance of our applications, as well as the accessibility, because React is keeping track of our focus for us.  [2:44](https://egghead.io/lessons/react-rerender-a-react-application-0bf050ac) This isn't how you normally re-render new application. Typically, whenever state changes, you don't have to re-render your entire application, but I wanted to show you this so you have an understanding of what React is really doing for you.  [2:56](https://egghead.io/lessons/react-rerender-a-react-application-0bf050ac) When you create these React elements, you give that to ReactDOM.render or you trigger a re-render of a component, and React is going to compare the elements that you returned this time with the elements that you returned last time.  [3:09](https://egghead.io/lessons/react-rerender-a-react-application-0bf050ac) It's going to do a div of those two elements, and then it will update the DOM surgically to only update the things that were different between the last time and this time you returned JSX. |
| **11. Стилизуйте компоненты React с помощью className и встроенных стилей** | **11. Style React Components with className and inline Styles** |
| Макет приложения - это только одна часть уравнения пользовательского интерфейса. Другая часть - это стиль. В этом уроке мы исследуем classNameи style prop для стилизации наших компонентов. Этот урок идет гораздо глубже.  Мы узнаем, почему classNameэто classNameа не class(как в HTML), и мы узнаем, почему style prop принимает объект стилей camelCase, а не строку css (как в HTML). Мы пойдем дальше, создавая повторно используемый компонент , который инкапсулирует эти стили и компонует данные classNameи style prop вместе. Вам понравится этот.  Наконец, когда вы действительно создаете полноценное приложение, вы найдете бесценным инструментом [Tailwind.css](https://tailwindcss.com/) . | Application layout is only one part of the user interface equation. Another part is styling. In this lesson we’ll explore the className and style prop for styling our components. This lesson goes much deeper than that.  We’ll learn why className is className and not class (like it is in HTML), and we’ll learn why the style prop accepts an object of camelCase styles rather than a string of css (like it does in HTML). We’ll take it even further by creating a reusable component that encapsulates these styles and composes the given className and style prop together. You’re going to love this one.  Finally, when you actually build a full application, a tool you’ll find invaluable is [Tailwind.css](https://tailwindcss.com/). |
| Инструктор: [00:00](https://egghead.io/lessons/react-style-react-components-with-classname-and-inline-styles#t=0) У нас есть набор CSS в нашем HTML-документе, и мы хотим стилизовать этот div с помощью этого CSS. Чтобы применить этот CSS, мы собираемся добавить className of Box, чтобы обвести этот прямоугольник вокруг этого небольшого светло-голубого прямоугольника. Поскольку мы хотим, чтобы он был маленьким, давайте также добавим box--small, и мы ее сохраним.  [00:26](https://egghead.io/lessons/react-style-react-components-with-classname-and-inline-styles#t=26) Тогда я хочу, чтобы он был голубым, но у нас нет CSS для голубого. Мы могли бы добавить его, но я хочу показать вам the style prop, которая принимает объект, а не строку стилей. Мы собираемся передать этот объект, и это будут те же имена свойств, которые вы получите, если вы используете getComputedStyle для этого элемента, то есть свойства CSS, которые имеют camel-cased, а не kebab-cased.  [00:51](https://egghead.io/lessons/react-style-react-components-with-classname-and-inline-styles#t=51) Чтобы получить цвет фона, мы назовем backgroundСolor, а затем нам понадобится lightblue. Таким образом, мы получаем light-blue box. Давайте также добавим здесь курсивный шрифт. Теперь у нас есть маленькая голубая рамка, выделенная курсивом.  [01:10](https://egghead.io/lessons/react-style-react-components-with-classname-and-inline-styles#t=70) У нас также есть CSS для больших и средних блоков, так что давайте продолжим и создадим еще несколько таких блоков. Мы сделаем его средним, и, используя здесь несколько наших курсоров, мы будем использовать для него большой. Вместо голубого для нашей среды давайте сделаем розовый. Вместо голубого нашего большого мы будем использовать оранжевый. Мы сохраним это, и у нас есть stack коробок.  [01:35](https://egghead.io/lessons/react-style-react-components-with-classname-and-inline-styles#t=95) В них довольно много дублирования, поэтому я собираюсь создать новый функциональный компонент под названием Box. Это потребует несколько props, а затем вернет div, и мы spread все props. Прямо сейчас этот компонент Box в основном функционирует как div. Мы могли бы заменить все эти div с несколькими курсорами на поле. Мы сохраняем это и получаем точно такой же результат.  [02:02](https://egghead.io/lessons/react-style-react-components-with-classname-and-inline-styles#t=122) Теперь давайте сделаем этот блок немного более полезным, взяв повторение каждого из них и поместив его в компонент Box. Во-первых, у нас есть className, а затем у него есть box--, но размер другой. Давайте продолжим и просто возьмем поле className. Мы поместим className Box, мы его сохраним.  [02:26](https://egghead.io/lessons/react-style-react-components-with-classname-and-inline-styles#t=146) Как только мы сохраним это - и мы сможем избавиться от них - мы заметим, что Box className больше не применяется, потому что мы не получаем эту границу. Мы можем убедиться в этом, проверив это, посмотрев наше приложение. Мы видим небольшой класс Box, но здесь мы не видим Box className. Подумайте на мгновение, почему это могло быть.  [02:48](https://egghead.io/lessons/react-style-react-components-with-classname-and-inline-styles#t=168) Причина может быть немного яснее, если я возьму этот набор реквизитов и помещу его с другой стороны от className. Мы сохраняем это, и теперь мы получаем противоположную проблему, когда у нас есть граница, которую предоставляет поле className, но у нас нет применений small, medium или large classNames.  [03:05](https://egghead.io/lessons/react-style-react-components-with-classname-and-inline-styles#t=185) Причина в том, что из-за такого разброса свойств мы получаем переопределение свойства className, которое предоставляется компоненту Box. Что мы можем сделать, так это объединить classNames в одно className, которое будет работать для всех наших блоков. Вместо того, чтобы просто взять объект props как есть, я собираюсь деструктурировать его на месте, и мы уберем className отсюда.  [03:28](https://egghead.io/lessons/react-style-react-components-with-classname-and-inline-styles#t=208) Тогда я поставлю ...rest, чтобы сказать, что это остальные props. Затем мы удалим это и поставим spread в конце. Затем мы вручную объединим className, которое мы хотим предоставить для boxes в целом, с className, которое предоставляется нам через props.  [03:46 Здесь](https://egghead.io/lessons/react-style-react-components-with-classname-and-inline-styles#t=226) мы сделаем шаблон буквально. Мы скажем Box, а затем добавим className. Мы это сохраним, и теперь все работает как надо. Одна из проблем заключается в том, что если я уберу className на одном из этих блоков, то я получу «box undefined».  [04:03](https://egghead.io/lessons/react-style-react-components-with-classname-and-inline-styles#t=243) Это, вероятно, не такая уж большая проблема, но ее довольно просто исправить, добавив значение по умолчанию для этого className в виде пустой строки. Теперь у нас есть Box with a space. Я не так беспокоюсь о пространстве. Если бы это было так, то вы могли бы добавить .trim прямо здесь, и это избавило бы от нашего пространства. Я не думаю, что это большое дело, так что мы избавимся от этого. Давайте вернем этой маленькой коробке былую славу.  [04:26 Еще одна](https://egghead.io/lessons/react-style-react-components-with-classname-and-inline-styles#t=266) общая черта всех этих блоков - это курсив. Давайте предоставим это здесь. Мы скажем стиль fontStyle italic. Теперь мы можем удалить стиль шрифта из всего этого. Мы сохраним это, и мы сделали вещи намного более краткими для пользователей этого компонента коробки.  [04:47](https://egghead.io/lessons/react-style-react-components-with-classname-and-inline-styles#t=287) Вы могли заметить, что у нас больше нет шрифта курсива. Причина этого та же, что и в случае с className. Если я перемещу этот …rest сюда перед стилем, теперь мы получим курсив, но мы не получим настраиваемые стили, которые пользователь нашего компонента коробки пытается указать.  [05:04](https://egghead.io/lessons/react-style-react-components-with-classname-and-inline-styles#t=304) Опять же, нам нужно скомпоновать их вручную. Я собираюсь вырвать the style prop из наших props здесь, и мы объединим опору стиля из тех, с опорой стиля, которую мы указываем, используя синтаксис распространения объекта. Затем я снова поставлю здесь …rest внизу.  [05:23](https://egghead.io/lessons/react-style-react-components-with-classname-and-inline-styles#t=323) Мы это сохраним, и мы получим курсив и особый цвет, а это именно то, что мы ищем. Теперь любой, кто хочет иметь box, может использовать этот компонент, и для него будет автоматически применяться Box className и курсив.  [05:39](https://egghead.io/lessons/react-style-react-components-with-classname-and-inline-styles#t=339) Одна из полезных функций создания подобных компонентов заключается в том, что мы можем использовать ее для отделения нашего пользовательского кода от проблем стилизации нашего конкретного компонента. Вместо className было бы лучше принять size prop. Можно сказать, размер маленький, средний или большой.  [05:57](https://egghead.io/lessons/react-style-react-components-with-classname-and-inline-styles#t=357) Давайте заставим это работать. Я удалю здесь className. Мы это сохраним. Прокрутите вверх, и мы увидим, что наш small, light-blue box уже не маленький. Теперь я возьму size prop, и мы собираемся сгенерировать размер className. Если указана size prop, то мы скажем box--size. В противном случае мы сделаем пустую строку.  [06:24](https://egghead.io/lessons/react-style-react-components-with-classname-and-inline-styles#t=384) Затем мы можем поместить этот size className в наш список className. Если мы сохраним это, то получим a small, light-blue box, и пользователям нашего box component больше не нужно будет заботиться о className, которое нужно применить для создания small box.  [06:41](https://egghead.io/lessons/react-style-react-components-with-classname-and-inline-styles#t=401) Этот Box component может иметь дело с деталями реализации того, что нужно для создания small box. Это означает, что мы можем изменить используемые классы classNames. Мы могли бы использовать стороннюю библиотеку CSS и изменить эту библиотеку CSS. Любой компонент, использующий коробку, просто должен следовать API, создаваемому для него коробкой.  [06:59](https://egghead.io/lessons/react-style-react-components-with-classname-and-inline-styles#t=419) Мы можем отделить наши компоненты от classNames, используемых для их стилизации. Тем не менее, мы все еще можем применять наши собственные имена классов, если захотим, и мы можем применять наши стили, если захотим, потому что они будут объединяться с тем, что происходит в коробке. Давайте продолжим и обновим classNames для обоих из них, потому что мне больше нравится the size prop.  [07:20](https://egghead.io/lessons/react-style-react-components-with-classname-and-inline-styles#t=440) Мы сохраним это, и теперь у нас есть a small, light-blue box, a medium pink box, and a large orange box с многоразовым Box component, который объединяет className, the style prop, вперед по всем props, например, child prop, который мы передаем для каждого из них, и принимает свойство специального размера, чтобы вы могли установить соответствующий размер поля.  [07:43](https://egghead.io/lessons/react-style-react-components-with-classname-and-inline-styles#t=463) Я хочу отметить одну вещь, касающуюся className prop в HTML. Обычно мы использовали бы class вместо className. ClassName - это свойство dom, которое вы используете для доступа к атрибуту класса на узлах dom. Если мы пойдем сюда и скажем $0, который будет обращаться к элементу, который мы в данный момент выбрали на нашей вкладке элементов, тогда мы можем сказать .className, и это даст нам этот список className.  [08:09](https://egghead.io/lessons/react-style-react-components-with-classname-and-inline-styles#t=489) Кроме того, если бы мы взяли все наши props и создали объект props здесь, мы просто переместим все это прямо сюда, конвертируем его в синтаксис JavaScript, а не JSX, а затем распространим эти свойства. Это тоже сработает. Тогда предположим, что это был class вместо className.  [08:29](https://egghead.io/lessons/react-style-react-components-with-classname-and-inline-styles#t=509) Если бы мы хотели извлечь это и использовать здесь сокращенный синтаксис, мы бы сделали его переменной класса. Это не сработает, потому что класс - зарезервированное слово. Кроме того, если бы мы хотели оставить это здесь, а затем деструктурировать свойство класса из реквизита, это также не сработало бы, потому что класс также является зарезервированным ключевым словом. По этой и другим причинам это свойство className prop, а не the class prop.  [08:56](https://egghead.io/lessons/react-style-react-components-with-classname-and-inline-styles#t=536) Одна вещь, которая мне действительно нравится в этой style prop, являющейся объектом стилей, а не строкой стилей, заключается в том, что намного проще комбинировать объекты стилей. С className у нас есть строка. Я не возражаю против этого, потому что порядок, в котором я применяю эти имена классов, не имеет отношения к тому, какой из них будет применяться.  [09:13](https://egghead.io/lessons/react-style-react-components-with-classname-and-inline-styles#t=553) Однако порядок, в котором я применяю эти стили, действительно влияет на то, что я буду применять. Если бы я применил это первым, то у меня, как у пользователя компонента box, не было бы возможности изменить это, чтобы контролировать, какие стили на самом деле будут применяться.  [09:27](https://egghead.io/lessons/react-style-react-components-with-classname-and-inline-styles#t=567) Давайте [вернем](https://egghead.io/lessons/react-style-react-components-with-classname-and-inline-styles#t=567) все на место и рассмотрим. У нас был набор CSS, который мы хотели применить к нашим div здесь, чтобы создать small, light-blue box, a medium pink box, and a large orange box. Мы сделали это вручную, создав div для каждого из них и применив соответствующие classNames и значение стиля.  [09:45](https://egghead.io/lessons/react-style-react-components-with-classname-and-inline-styles#t=585) Затем, чтобы уменьшить дублирование, мы создали этот функциональный компонент под названием Box, который принимает className, объединяет его с className, необходимым для создания блока, принимает опору стиля и объединяет это со стилем, необходимым для создания поле с курсивным шрифтом.  [10:02](https://egghead.io/lessons/react-style-react-components-with-classname-and-inline-styles#t=602) Также принимая a size prop, чтобы отделить код для пользователей компонента Box от кода, необходимого для создания a box этого указанного стиля. Затем мы берем rest of the props и распространяем их на базовый div, чтобы пользователи могли предоставить дополнительные реквизиты, такие как ID is root, а также дочерние реквизиты для содержимого бокса.  [10:24 И](https://egghead.io/lessons/react-style-react-components-with-classname-and-inline-styles#t=624) последнее, что я хочу вам показать, это Tailwind CSS, на который я рекомендую вам взглянуть для создания согласованных пользовательских интерфейсов. Определенно взгляните на это при создании и стилизации ваших приложений. Вы можете применить те же принципы, которые мы узнали, для создания пользовательских компонентов, использующих эти имена классов. | Instructor: [00:00](https://egghead.io/lessons/react-style-react-components-with-classname-and-inline-styles#t=0) We have a bunch of CSS in our HTML document, and we want to style this div with that CSS. To apply that CSS, we're going to add a className of Box to get that box around this small, light-blue box. Because we want it to be small, let's also add a box--small, and we'll save that.  [00:26](https://egghead.io/lessons/react-style-react-components-with-classname-and-inline-styles#t=26) Then, I want it to be light blue, but we don't have any CSS for light blue. We could add it, but I want to show you the style prop which accepts an object rather than a string of styles. We're going to pass this object, and it will be the same property names that you get if you use getComputedStyle for this element, that is, CSS properties that are camel-cased instead of kebab-cased.  [00:51](https://egghead.io/lessons/react-style-react-components-with-classname-and-inline-styles#t=51) For us to get a background color, we're going to say background color, and then we want lightblue. With that, we get a small, light-blue box. Let's also add a font style italic right there. Now we have a small, light-blue box in italics.  [01:10](https://egghead.io/lessons/react-style-react-components-with-classname-and-inline-styles#t=70) We also have CSS for large and medium boxes, so let's go ahead and create some more of these boxes. We'll make this one be medium, and using our multiple cursors here, we'll use large for that one. Instead of light blue for our medium, let's do pink. Instead of the light blue of our large, we'll use orange. We'll save that, and we have this stack of boxes.  [01:35](https://egghead.io/lessons/react-style-react-components-with-classname-and-inline-styles#t=95) There's a fair amount of duplication in these, so what I'm going to do is make a new function component called Box. This is going to take some props, and then will return a div, and we'll spread all the props. Right now, this Box component basically functions as a div. We could replace all of these divs, with multiple cursors here, with a box. We save that, and we get the exact same result.  [02:02](https://egghead.io/lessons/react-style-react-components-with-classname-and-inline-styles#t=122) Now, let's make this box a little bit more useful by taking the repetition from each one of these and putting it within the Box component. First off, we have a className, and then it has a box--, but then the size is different. Let's go ahead and just take the className Box. We'll put className Box, we'll save that.  [02:26](https://egghead.io/lessons/react-style-react-components-with-classname-and-inline-styles#t=146) Once we save that -- and we can get rid of these as well -- then we'll notice that the Box className is no longer being applied because we're not getting that border. We can verify that by inspecting this, looking our application. We see class Box small, but we don't see the Box className here. Think for a moment why that might be.  [02:48](https://egghead.io/lessons/react-style-react-components-with-classname-and-inline-styles#t=168) The reason might be a little more clear if I take this spread of props and put it on the other side of the className. We save that, and now we get the opposite problem where we have the border the className Box provides, but we don't have the small, medium, or large classNames applies.  [03:05](https://egghead.io/lessons/react-style-react-components-with-classname-and-inline-styles#t=185) The reason is, because of this spread of props, we're getting an override of the className prop that's being provided to the Box component. What we can do is combine the classNames into a single className that will work for all of our boxes. Instead of just taking the props object as it is, I'm going to destructure it in-place, and we'll take the className out of here.  [03:28](https://egghead.io/lessons/react-style-react-components-with-classname-and-inline-styles#t=208) Then I'll put a ...rest to say that these are the rest of the props. Then we'll remove this, and we'll put the spread at the end. Then we'll manually combine the className that we want to provide for boxes in general with the className that's provided to us through the props.  [03:46](https://egghead.io/lessons/react-style-react-components-with-classname-and-inline-styles#t=226) We'll make a template literal here. We'll say Box, and then we'll put className. We'll save that, and now everything's working as it should be. One problem here is that if I take off the className on one of these boxes, then I'm going to get "box undefined."  [04:03](https://egghead.io/lessons/react-style-react-components-with-classname-and-inline-styles#t=243) That's probably not areally big issue, but it's pretty simple to fix by adding a default value for this className to be an empty string. Now we get Box with a space. I'm not as worried about the space. If you were, then you could add a .trim right here, and that would get rid of our space. I don't think that's a big deal, so we'll get rid of that. Let's restore this small box to its former glory.  [04:26](https://egghead.io/lessons/react-style-react-components-with-classname-and-inline-styles#t=266) The other thing that's common across all these boxes is the font style italic. Let's have that be provided here. We'll say style fontStyle italic. With that, we can now remove the font style from all of these. We'll save that, and we've made things a lot more terse for users of this box component.  [04:47](https://egghead.io/lessons/react-style-react-components-with-classname-and-inline-styles#t=287) You might notice we no longer have that font style of italic. The reason that's happening is the exact same as with the className. If I move this rest up here before the style, now we're going to get the italic, but we won't get the custom styles that the user of our box component is trying to specify.  [05:04](https://egghead.io/lessons/react-style-react-components-with-classname-and-inline-styles#t=304) Again, we do need to compose these manually. I'm going to pluck off the style prop from our props here, and we'll combine the style prop from those with the style prop that we specify by using object spread syntax. Then I'm going to put rest at the bottom here again.  [05:23](https://egghead.io/lessons/react-style-react-components-with-classname-and-inline-styles#t=323) We'll save that, and we get our italics and we get our special color, which is exactly what we're looking for. Now anyone that wants to have a box can use this component, and it will have the Box className and italics applied automatically for them.  [05:39](https://egghead.io/lessons/react-style-react-components-with-classname-and-inline-styles#t=339) One useful feature of creating components like this is we can use it to separate our user code from the styling concerns of our specific component. Rather than a className, it would actually be better to accept a size prop. We could say the size is small, medium, or large.  [05:57](https://egghead.io/lessons/react-style-react-components-with-classname-and-inline-styles#t=357) Let's make that work. I'll remove the className here. We'll save this. Let's scroll up to the top, and we'll see that our small, light-blue box is no longer small. Now I'm going to take the size prop, and we're going to generate a size className. If the size prop is specified, then we'll say box--size. Otherwise, we'll do an empty string.  [06:24](https://egghead.io/lessons/react-style-react-components-with-classname-and-inline-styles#t=384) Then we can put that size className inside of our className list here. If we save that, then we get a small, light-blue box, and users of our box component no longer need to concern themselves with the className that needs to be applied to create a small box.  [06:41](https://egghead.io/lessons/react-style-react-components-with-classname-and-inline-styles#t=401) This Box component can deal with the implementation details of what it takes to make a small box. This means that we could change the classNames that we use. We could use a third-party CSS library, and we could change that CSS library. Any component that's using the box just needs to follow the API that the box creates for it.  [06:59](https://egghead.io/lessons/react-style-react-components-with-classname-and-inline-styles#t=419) We can separate our components from the classNames used to style them. That said, we can still apply our own classNames if we want to, and we can apply our styles if we want to because those are going to be combined with whatever is happening in the box. Let's go ahead and update the classNames for both of these because I like the size prop better.  [07:20](https://egghead.io/lessons/react-style-react-components-with-classname-and-inline-styles#t=440) We'll save that, and now we have a small, light-blue box, a medium pink box, and a large orange box with a reusable Box component that merges the className, the style prop, forwards along all of the props, like the children prop that we're passing for each one of these, and accepts a special size prop so that you can size the box appropriately.  [07:43](https://egghead.io/lessons/react-style-react-components-with-classname-and-inline-styles#t=463) One thing that I want to note here about the className prop in HTML. We would normally use class instead of className. ClassName is the dom property that you use to access the class attribute on dom nodes. If we go here and say $0 which will access the element that we have currently selected in our elements tab, then we can say .className, and that will give us that className list.  [08:09](https://egghead.io/lessons/react-style-react-components-with-classname-and-inline-styles#t=489) In addition, if we were to take all of our props and make a props object here, we just move all of this right here, convert it to JavaScript rather than JSX syntax, and then spread those props. That will work just as well. Then let's say that this were class instead of className.  [08:29](https://egghead.io/lessons/react-style-react-components-with-classname-and-inline-styles#t=509) If we wanted to extract this and use shorthand syntax here, then we'd make it class variable. This wouldn't work because class is a reserved word. In addition, if we wanted to leave this here and then destructure the class property from the props, then this also wouldn't work because class is a reserved keyword as well. For that reason and others, it is the className prop rather than the class prop.  [08:56](https://egghead.io/lessons/react-style-react-components-with-classname-and-inline-styles#t=536) One thing that I really like about this style prop being an object of styles rather than a string of styles is because it's a lot easier to combine objects of styles. With className, we have a string. I don't mind that because the order in which I apply these classNames has no bearing on which one of these is going to be applied.  [09:13](https://egghead.io/lessons/react-style-react-components-with-classname-and-inline-styles#t=553) However, the order in which I apply these styles does have a bearing in what's going to be applied. If I were to apply this first, then there'd be no way for me, as a user of the box component, to override that to control what styles are actually going to be applied.  [09:27](https://egghead.io/lessons/react-style-react-components-with-classname-and-inline-styles#t=567) Let's put everything back where it was and review. We had a bunch of CSS that we wanted to apply to our divs here to make a small, light-blue box, a medium pink box, and a large orange box. We did that manually by making divs for each one of these and applying the appropriate classNames and style value.  [09:45](https://egghead.io/lessons/react-style-react-components-with-classname-and-inline-styles#t=585) Then, in an effort to reduce duplication, we created this function component called Box, which accepts a className, combines that with the className necessary to make a box, accepts a style prop, and combines that with the style necessary to make a box with a font style of italic.  [10:02](https://egghead.io/lessons/react-style-react-components-with-classname-and-inline-styles#t=602) Also accepting a size prop to separate the code for the users of the Box component from the code necessary to make a box of that specified style. Then we take the rest of the props, and spread that on the underlying div so that users can provide additional props like ID is root as well as the children prop for the contents of the box.  [10:24](https://egghead.io/lessons/react-style-react-components-with-classname-and-inline-styles#t=624) One last thing that I want to show you is Tailwind CSS, which I recommend you take a look at for building consistent user interfaces. Definitely give this a look for building and styling your applications. You can apply the same principles that we learned to building custom components that utilize these classNames. |
| **12. Используйте обработчики событий с React** | **12. Use Event Handlers with React** |
| Приложения можно красиво оформить и стилизовать, но если они не реагируют на действия пользователя, то это просто веб-страницы, а не приложения. Давайте познакомимся с обработчиками событий в React. Существует множество поддерживаемых событий, которые вы можете найти [в документации](https://reactjs.org/docs/events.html#supported-events) . Нам все еще не нужно констатировать, поэтому мы реализовали наш собственный небольшой способ управления состоянием и повторного рендеринга нашего компонента, чтобы мы могли поиграть с обработчиками событий.  Вы должны знать, что события в React очень похожи на работу с событиями в обычном DOM. В React есть реализация оптимизации поверх системы событий под названием [SyntheticEvents](https://reactjs.org/docs/events.html) , но в большинстве случаев вы не заметите никакой разницы с этими событиями от обычных событий DOM (и вы всегда можете получить доступ к собственному событию с помощью nativeEventсвойства). | Application’s can be laid out and styled pretty, but if they don’t respond to interactions from the user then they’re just web pages, not apps. Let’s get an introduction to event handlers with React. There are a ton of supported events that you can find [on the docs](https://reactjs.org/docs/events.html#supported-events). We still haven’t gotten to state yet, so we’ve implemented our own little way of managing state and re-rendering our component so we can play around with event handlers.  One thing you’ll want to know is that events with React are very similar to working with events in regular DOM. React does have an optimization implementation on top of the event system called [SyntheticEvents](https://reactjs.org/docs/events.html), but most of the time you won’t observe any difference with those events from regular DOM events (and you can always get access to the native event using the nativeEvent property). |
| Кент Си Доддс: [0:00](https://egghead.io/lessons/react-use-event-handlers-with-react-05e315dc) Здесь у нас есть компонент приложения, который отображает здесь некоторый JSX, потому что там были события, щелкните меня, вы набрали, а затем введите здесь ввод. У нас есть состояние, в котором запущен этот компонент приложения. Если я обновлю счетчик событий до 10, мы увидим, что было 10 событий.  [0:16](https://egghead.io/lessons/react-use-event-handlers-with-react-05e315dc) Если я изменю имя пользователя на «Hello there» и сохраню его, то мы увидим, что вы набрали Hello there. Значение ввода не обновляется, и на самом деле не было 10 событий, поэтому давайте продолжим и подключим вещи, чтобы мы могли обновить наше состояние и повторно отрендерить приложение.  [0:32](https://egghead.io/lessons/react-use-event-handlers-with-react-05e315dc) У нас уже есть функция с именем setState, которая обновит нашу переменную состояния, а затем отобразит приложение. Это просто снова вызовет React.render в нашем корне. Обычно это не то, как вы повторно визуализируете приложение или управляете состоянием в приложении React, но это будет работать для наших целей изучения того, как события работают в React.  [0:51](https://egghead.io/lessons/react-use-event-handlers-with-react-05e315dc) Сначала [разберемся](https://egghead.io/lessons/react-use-event-handlers-with-react-05e315dc) с этой кнопкой. У кнопки есть событие onClick, которое мы можем вызвать. Мы собираемся предоставить функцию. Я могу сделать это встроенной стрелочной функцией здесь.  [1:03](https://egghead.io/lessons/react-use-event-handlers-with-react-05e315dc) Мы просто скажем setState, вызывая эту функцию здесь с новым состоянием объекта, eventCount. Мы хотим, чтобы это eventCount было текущим состоянием. eventCount + 1. Если мы сохраним это, а затем я перейду сюда и щелкну, мы будем получать приращение каждый раз, когда я щелкаю.  [1:21](https://egghead.io/lessons/react-use-event-handlers-with-react-05e315dc) Есть много разных событий, которые мы можем использовать здесь. Мы можем иметь onFocus и сохранять его, и каждый раз, когда я фокусирую ввод, мы собираемся получить приращение для этого события. Или мы могли бы сказать onMouseOver, сохранить это, и тогда каждый раз, когда я наведу указатель мыши, мы будем получать приращение этого количества событий. React поддерживает гораздо больше событий.  [1:43](https://egghead.io/lessons/react-use-event-handlers-with-react-05e315dc) Давайте вернем его в onCount, а затем мы также можем извлечь это в handleClick. Мы могли бы поместить это прямо здесь. Функция handleClick (), а затем мы вызовем там setState. Это тоже сработает. Это сработало бы, если бы мы использовали onClick вместо onCount. Вот и все.  [2:03](https://egghead.io/lessons/react-use-event-handlers-with-react-05e315dc) Тогда давайте посмотрим на этот ввод. Мы хотим, чтобы это вы ввели, чтобы получать обновления, когда мы вводим новое имя пользователя. Я скажу onBlur, а мы скажем setState('username'). Затем нам нужно указать здесь значение.  [2:18](https://egghead.io/lessons/react-use-event-handlers-with-react-05e315dc) Нам нужно получить значение входных данных, которые получают событие onBlur. В качестве аргумента обработчику события мы получаем событие. Затем мы можем использовать это событие для получения цели события, которая будет нашим входом. Затем мы получим это значение из ввода.  [2:36](https://egghead.io/lessons/react-use-event-handlers-with-react-05e315dc) Если мы это сохраним, то сможем ввести Kent C Dodds и затем blur. Наше состояние обновляется. Если мы хотим, чтобы это обновлялось при вводе, мы можем изменить это на onChange. По мере того, как пользователь вводит этот ввод, мы получим это обновление для нашего состояния и повторно отрендерим приложение.  [2:54](https://egghead.io/lessons/react-use-event-handlers-with-react-05e315dc) Давайте продолжим и вытащим обработчик onChange. Мы скажем handleChange. Мы поднимем это здесь. Функция handleChange примет событие. Давайте посмотрим на это событие. Мы скажем console.log (событие), сохраните это.  [3:10](https://egghead.io/lessons/react-use-event-handlers-with-react-05e315dc) Мы откроем здесь наши инструменты разработчика, посмотрим на консоль и введем один символ. Мы заметим, что здесь мы получаем SyntheticEvent объект. Это не native event. Собственное событие - это входное событие свойства nativeEvent. React серьезно оптимизирует производительность наших событий, и в этом вся суть синтетических событий.  [3:33 В](https://egghead.io/lessons/react-use-event-handlers-with-react-05e315dc) большинстве случаев вам не нужен доступ к собственному событию, но если он когда-либо понадобится, вы можете сказать event.nativeEvent, и мы нажмем «Сохранить». Мы введем символ и получим это собственное событие ввода. Но SyntheticEvent имеет почти все те же свойства, что и обычное событие, поэтому вы обычно собираетесь его использовать.  [3:54](https://egghead.io/lessons/react-use-event-handlers-with-react-05e315dc) Одна вещь, которая мне нравится в системе событий React, - это то, что вы передаете обработчик событий непосредственно элементу, к которому хотите прикрепить событие, так что отслеживание потока событий и обновлений состояния становится действительно простым. | Kent C Dodds: [0:00](https://egghead.io/lessons/react-use-event-handlers-with-react-05e315dc) Here we have an app component that is rendering some JSX here for there have been events, click me, you typed, and then an input here. We have some state that is running this application component. If I update the event count to 10, then we're going to see there have been 10 events.  [0:16](https://egghead.io/lessons/react-use-event-handlers-with-react-05e315dc) If I update the username to 'Hello there' and save that, then we're going to see you typed Hello there. The value of the input isn't getting updated and there have actually not been 10 events, so let's go ahead and wire up things so we can update our state and re-render the app.  [0:32](https://egghead.io/lessons/react-use-event-handlers-with-react-05e315dc) We already have a function here called setState, which will update our state variable and then render the app. That will just call React.render on our root again. This is not typically how you re-render an application or manage state in a React application, but this will work for our purposes of learning how events work in React.  [0:51](https://egghead.io/lessons/react-use-event-handlers-with-react-05e315dc) First let's deal with this button. We have on button's an onClick event that we can call. This we're going to provide a function. I can make this an inline arrow function here.  [1:03](https://egghead.io/lessons/react-use-event-handlers-with-react-05e315dc) We'll just say setState, calling this function down here with the new state of an object, eventCount. We want that eventCount to be the current state. eventCount + 1. If we save that and then I go over here and click, we're going to get an increment every single time I click.  [1:21](https://egghead.io/lessons/react-use-event-handlers-with-react-05e315dc) There are a lot of different events that we can use here. We can have an onFocus and save that and every time I focus the input we're going to get an increment on that event. Or we could say onMouseOver, save that, and then every time I mouse over, we'll get an increment on that event count. There are a lot more events that are supported by React.  [1:43](https://egghead.io/lessons/react-use-event-handlers-with-react-05e315dc) Let's set this back to onCount and then we can also extract this to handleClick. We could put that right here. Function handleClick() and then we'll call setState there. That will work just as well. It would work if we used onClick instead of onCount. There we go.  [2:03](https://egghead.io/lessons/react-use-event-handlers-with-react-05e315dc) Then let's take a look at this input. We want this you typed to get updated when we enter some new username. I'm going to say onBlur and we'll say setState('username'). Then we need to specify a value here.  [2:18](https://egghead.io/lessons/react-use-event-handlers-with-react-05e315dc) What we need is to get the value of the input that is receiving the onBlur event. As an argument to the event handler, we receive the event. Then we can use that event to get the target of the event, which will be our input. Then we'll get that value from the input.  [2:36](https://egghead.io/lessons/react-use-event-handlers-with-react-05e315dc) If we save that, then we can type in Kent C Dodds and then blur. Our state gets updated. If we want this to get updated as we're typing, then we can change this to onChange. As the user types in this input, we'll get that update to our state and we'll re-render the application.  [2:54](https://egghead.io/lessons/react-use-event-handlers-with-react-05e315dc) Let's go ahead and pull this onChange handler out. We'll say handleChange. We'll bring it up here. Function handleChange will take the event. Let's go ahead and take a look at that event. We'll say console.log(event), save that.  [3:10](https://egghead.io/lessons/react-use-event-handlers-with-react-05e315dc) We'll pop open our Dev Tools here, look at the console, and then type in one character. We'll notice we get a SyntheticEvent object here. That's not the native event. The native event is this property nativeEvent input event. React does some serious performance optimizations for our events and that's what this synthetic event thing is all about.  [3:33](https://egghead.io/lessons/react-use-event-handlers-with-react-05e315dc) Most of the time you don't need access to the native event, but if you ever do then you can say event.nativeEvent and we'll hit save. We'll type in a character and we'll get that native input event. But the synthetic event has pretty much all the same properties as a regular event, so that's what you're typically going to use.  [3:54](https://egghead.io/lessons/react-use-event-handlers-with-react-05e315dc) One thing I like about React's event system is that you pass the event handler directly to the element that you want to attach the event to, so it makes following the flow of events and state updates really straightforward. |
| **13. Управляйте состоянием в React Component с помощью хука useState** | **13. Manage state in a React Component with the useState hook** |
| Приложение, которое реагирует на ввод данных пользователем, ценно, но что нам делать с данными, которые нам предоставил пользователь? Здесь и появляется состояние компонента. Нам нужно место для размещения данных, которые могут измениться в нашем приложении, и нам нужно сообщить React, когда это состояние изменится, чтобы он мог обновлять (или повторно отображать) наше приложение для нас.  В React состояние связано с компонентами, и при изменении состояния компонент обновляется. Чтобы получить доступ к этому состоянию и обновить его, мы используем так называемый «React Hook», который позволяет нам вызывать React изнутри нашего компонента и сообщать ему, что нам нужно управлять некоторым состоянием. В этом уроке вы узнаете, как использовать для этого useState hook. | An application that responds to user input is valuable, but what do we do with that data the user has given us? This is where component state comes in. We need a place to put data that can change in our application, and we need to let React know when that state changes so it can update (or re-render) our app for us.  In React, state is associated to components and when the state changes, the component is updated. To get access to this state and to update it, we use what is called a “React Hook” which allows us to call into React from within our component and let it know that we need to manage some state. In this lesson, you’ll learn how to use the useState hook to do this. |
| Кент: [0:00](https://egghead.io/lessons/react-manage-state-in-a-react-component-with-the-usestate-hook) Начнем с создания нашей формы. Здесь у нас будет наш div, а затем мы создадим форму. Затем у нас будет метка с именем и нашим вводом, и мы ее сохраним. Получаем такую форму.  [0:15](https://egghead.io/lessons/react-manage-state-in-a-react-component-with-the-usestate-hook) Тогда у нас будет это сообщение: «Пожалуйста, введите свое имя». Затем мы хотим связать эту метку с этим input, чтобы она была правильно связана, и вы могли сказать это, щелкнув the label. Если он не фокусирует input, значит, он не labeled должным образом.  [0:30](https://egghead.io/lessons/react-manage-state-in-a-react-component-with-the-usestate-hook) Давайте добавим ID к нашему input имени и htmlFor имени. htmlFor похож на атрибут for в HTML, но в JSX вам нужно использовать htmlFor. Это одно из немногих отличий JSX от обычного HTML. Сохраним это, вернемся сюда. Когда я нажимаю на имя, он фокусирует input. Отлично.  [0:53](https://egghead.io/lessons/react-manage-state-in-a-react-component-with-the-usestate-hook) Теперь у нас будет некоторое состояние для имени. Если есть имя, мы заставим его сказать strong: "Hello, Name!" В противном случае он скажет: "Please type your name."  Отлично. Тогда, если мы скажем Kent, сохраните это. Он скажет: "Hello, Kent!" По умолчанию он ничего не говорит.  [1:15](https://egghead.io/lessons/react-manage-state-in-a-react-component-with-the-usestate-hook) Теперь давайте подключим onchange handler для этого input. Мы назовем это изменением дескриптора (handle change). Я подниму это здесь и скажу, что изменение дескриптора - это стрелочная функция, которая принимает событие, и ей необходимо обновить состояние имени.  [1:30](https://egghead.io/lessons/react-manage-state-in-a-react-component-with-the-usestate-hook) Нам нужен способ обновить это, и мы не можем просто сказать name = event.target.value. Это не сработает, потому что это не приведет к повторному рендерингу, и даже если бы он вызвал повторный рендеринг, вся эта функция будет вызвана. Тогда имя, которое мы будем устанавливать, будет собрано мусором, и мы создадим новую переменную имени.  [1:52](https://egghead.io/lessons/react-manage-state-in-a-react-component-with-the-usestate-hook) Вместо этого у React есть так называемый перехватчик React (a React hook) для поддержания состояния компонента. Мы собираемся использовать это с react.useState. Мы передадим ему значение по умолчанию пустой строки, и react.useState вернет массив, мы назовем его нашим массивом состояний (state array), и мы получим наше имя из массива состояний с нулевым индексом и имя нашего набора из массива состояний в индекс один.  [2:21](https://egghead.io/lessons/react-manage-state-in-a-react-component-with-the-usestate-hook) Здесь вместо того, чтобы пытаться переназначить эту переменную, мы вызовем имя набора с целевым значением события. Если мы сохраним это, теперь все работает, но никто не хочет писать свой код таким образом каждый раз, когда они хотят использовать состояние, поэтому мы собираемся деструктурировать это, чтобы оно было name и set name. Затем мы можем удалить эти строки и сохранить их, и они все еще работают.  [2:48](https://egghead.io/lessons/react-manage-state-in-a-react-component-with-the-usestate-hook) Вау! React rocks! Итак, чтобы использовать состояние в компоненте функции React, вы используете useState hook from React из React. The useState hook принимает начальное значение, поэтому, когда этот компонент приветствия изначально визуализируется, это будет значение нашей переменной name.  [3:07](https://egghead.io/lessons/react-manage-state-in-a-react-component-with-the-usestate-hook) Каждый раз, когда мы вызываем этот второй элемент массива, наша функция обновления запускает повторный рендеринг всего этого функционального компонента. Когда React useState вызывается снова, он проигнорирует начальное значение и вместо этого даст нам текущее значение этого имени.  [3:23](https://egghead.io/lessons/react-manage-state-in-a-react-component-with-the-usestate-hook) Поскольку React отслеживает порядок их вызова, мы можем добавить второй. Здесь мы назовем это name two и установим name two. Мы сделаем еще одно изменение дескриптора, чтобы обработать изменение два. У нас будет имя набора вызовов два. Затем мы просто продублируем все это, поместим это внутрь другого div, а затем мы будем ссылаться на имя два и называть два. Это будет handle change two.  [3:49](https://egghead.io/lessons/react-manage-state-in-a-react-component-with-the-usestate-hook) Теперь мы можем сказать Kent 1 и Kent 2. Эти состояния управляются независимо друг от друга, потому что React отслеживает порядок, в котором они будут вызываться, что позволяет вам использовать состояние столько, сколько вам нужно для вашего компонента. .  [4:05](https://egghead.io/lessons/react-manage-state-in-a-react-component-with-the-usestate-hook) Здесь могут быть любые значения. Вы можете сделать его логическим, или числом, или объектом, или массивом - все, что имеет смысл для состояния, которым вы пытаетесь управлять. | Kent: [0:00](https://egghead.io/lessons/react-manage-state-in-a-react-component-with-the-usestate-hook) Let's start by making our form. Here we'll have our div and then we'll make a form. Then we'll have a label with the name and our input and we'll save that. We get that form.  [0:15](https://egghead.io/lessons/react-manage-state-in-a-react-component-with-the-usestate-hook) Then we'll have this message, "Please type your name." Then we want to associate this label to this input so that it's properly associated and you can tell that by clicking on the label. If it doesn't focus the input, then it's not properly labeled.  [0:30](https://egghead.io/lessons/react-manage-state-in-a-react-component-with-the-usestate-hook) Let's add an ID to our input of name and an htmlFor of name. htmlFor is like the for attribute in HTML but in JSX you need to use htmlFor. That's one of the few differences between JSX and regular HTML. Let's save that, come back here. When I click on name, it now focuses the input. Great.  [0:53](https://egghead.io/lessons/react-manage-state-in-a-react-component-with-the-usestate-hook) Now, we're going to have some state here for the name. If there is a name, then we'll have it say strong, "Hello, Name!" Otherwise we'll have it say, "Please type your name." Great. Then if we say Kent, save that. It will say, "Hello, Kent!" By default it doesn't say anything.  [1:15](https://egghead.io/lessons/react-manage-state-in-a-react-component-with-the-usestate-hook) Now, let's wire up an onchange handler for this input. We'll call that handle change. I'll bring that up here and say handle change is an arrow function that accepts an event and it needs to update the state of the name.  [1:30](https://egghead.io/lessons/react-manage-state-in-a-react-component-with-the-usestate-hook) We need to have some way to update that and we can't simply say name = event.target.value. That's not going to work because that won't trigger a re-render and, even if it were to trigger a re-render, this whole function would be recalled. Then the name that we would be setting would get garbage collected and we'd create a new name variable.  [1:52](https://egghead.io/lessons/react-manage-state-in-a-react-component-with-the-usestate-hook) Instead, React has what's called a React hook for maintaining state for a component. We're going to use that with react.useState. We'll pass it the default value of an empty string and react.useState returns an array, we'll call this our state array, and we'll get our name from state array at index zero and our set name from state array at index one.  [2:21](https://egghead.io/lessons/react-manage-state-in-a-react-component-with-the-usestate-hook) Here instead of trying to reassign that variable, we'll call set name with the event target value. If we save that, now everything is working, but nobody wants to write their code like this every time they want to use state, so we're going to destructure this to be name and set name. Then we can delete those lines and save that and it's still working.  [2:48](https://egghead.io/lessons/react-manage-state-in-a-react-component-with-the-usestate-hook) Woo! React rocks! In review, to use state in a React function component, you use the useState hook from React. The useState hook accepts the initial value, so when this greeting component is initially rendered, that is going to be the value of our name variable.  [3:07](https://egghead.io/lessons/react-manage-state-in-a-react-component-with-the-usestate-hook) Any time we call this second element of the array, our updater function will trigger a re-render of this entire function component. When React useState is called again, it will ignore the initial value and instead give us the current value of that name.  [3:23](https://egghead.io/lessons/react-manage-state-in-a-react-component-with-the-usestate-hook) Because React keeps track of the order in which these are called, we could add a second one. Here we'll call this name two and set name two. We'll make another handle change to handle change two. We'll have that call set name two. Then we'll just duplicate all this stuff, put that inside of another div here, and then we'll reference name two and name two. This will be handle change two.  [3:49](https://egghead.io/lessons/react-manage-state-in-a-react-component-with-the-usestate-hook) Now we can say Kent 1 and Kent 2. Those states are managed independently of one another because React keeps track of the order in which these are going to be called, allowing you to use state as much as you need for your component.  [4:05](https://egghead.io/lessons/react-manage-state-in-a-react-component-with-the-usestate-hook) The values in here can be anything. You can make it a Boolean or you can make it a number or you can make it an object or an array -- whatever makes sense for the state that you're trying to manage. |
| 14. Управляйте побочными эффектами в React Component с помощью хука useEffect | 14. Manage side-effects in a React Component with the useEffect hook |
| Еще одна часть puzzle веб-приложений - это управление побочными эффектами взаимодействия наших пользователей. В этом уроке мы будем взаимодействовать с localStorageAPI браузера , но то же самое будет применяться, если мы взаимодействуем с внутренним сервером, или API геолокации, или чем-либо еще, что должно произойти при изменении состояния нашего компонента.  Вы узнаете, как использовать React’s useEffect hook для управления побочным эффектом сохранения состояния в localStorage, а также как повторно синхронизировать наше приложение с сохраненным значением в localStorage. [Узнайте больше о просмотре localStorageв Chrome DevTools](https://developers.google.com/web/tools/chrome-devtools/storage/localstorage) . | Another piece to the web application puzzle is managing side-effects of our user’s interactions. In this lesson we’ll be interacting with the browser’s localStorage API, but this same thing would apply if we’re interacting with a backend server, or the geolocation API, or anything else that needs to happen when the state of our component changes.  You’ll learn how to use React’s useEffect hook to manage the side-effect of saving state into localStorage, and also how to re-synchronize our application with the stored value in localStorage. [Learn more about viewing localStorage in the Chrome DevTools](https://developers.google.com/web/tools/chrome-devtools/storage/localstorage). |
| Преподаватель: [0:00](https://egghead.io/lessons/react-manage-side-effects-in-a-react-component-with-the-useeffect-hook) Я хочу иметь возможность ввести здесь какое-то значение и сохранить его в localStorage, чтобы при обновлении страницы это значение было извлечено из localStorage и загружено в input.  [0:11](https://egghead.io/lessons/react-manage-side-effects-in-a-react-component-with-the-useeffect-hook) Это называется побочным эффектом, и для этого вы используете react useEffect. Это еще один хук, подобно useState. Эта функция будет вызываться каждый раз при рендеринге компонента приветствия.  [0:23](https://egghead.io/lessons/react-manage-side-effects-in-a-react-component-with-the-useeffect-hook) Каждый раз, когда компонент приветствия визуализируется, мы будем говорить window.localStorage.setItem - мы назовем его name в localStorage и установим для него значение name. Если мы сохраним это, а затем откроем наши инструменты разработчика и на вкладке приложения здесь, мы сможем перейти в localStorage и увидим, что там ничего нет.  [0:45](https://egghead.io/lessons/react-manage-side-effects-in-a-react-component-with-the-useeffect-hook) Затем мы можем ввести имя, и мы увидим, что имя обновляется в localStorage с любым текущим значением для этой переменной имени.  [0:54](https://egghead.io/lessons/react-manage-side-effects-in-a-react-component-with-the-useeffect-hook) Вы заметите, что если я обновлюсь здесь, я не получу это значение имени, загруженное в мой ввод, и значение в localStorage очищается. Это потому, что мы инициализируем наше состояние пустой строкой и не принимаем во внимание значение localStorage.  [1:09](https://egghead.io/lessons/react-manage-side-effects-in-a-react-component-with-the-useeffect-hook) Затем, поскольку мы отрисовали, мы запускаем этот обратный вызов, обновляя элемент localStorage для имени до этого нового имени пустой строки. Нам нужно инициализировать это значение тем, что находится в localStorage, если оно есть, и пустой строкой, если это не так.  [1:23](https://egghead.io/lessons/react-manage-side-effects-in-a-react-component-with-the-useeffect-hook) Здесь мы скажем window.localStorage.getItem (name), и если это вернет значение null, потому что там ничего нет, тогда мы будем использовать по умолчанию пустую строку. Мы сохраняем это. Набираем имя. Мы нажимаем «обновить» и замечаем, что значение в localStorage по-прежнему согласовано.  [1:43](https://egghead.io/lessons/react-manage-side-effects-in-a-react-component-with-the-useeffect-hook) Мы замечаем, что значение здесь правильное, но во вводе имени его нет. Нам нужно указать, каким должно быть значение для ввода. Мы указываем значение имени.  [1:57](https://egghead.io/lessons/react-manage-side-effects-in-a-react-component-with-the-useeffect-hook) Теперь, когда мы сохраним это, мы получим обновление, и теперь это то же значение, что и в localStorage и в памяти для состояния нашего компонента реакции. Мы заметим, что при изменении и обновлении мы всегда получаем это значение из localStorage, а также обновляем это значение в localStorage.  [2:16](https://egghead.io/lessons/react-manage-side-effects-in-a-react-component-with-the-useeffect-hook) Итак, чтобы все это заработало, мы сначала использовали response useEffect, чтобы установить значение имени в localStorage равным значению имени в нашем состоянии в памяти. Затем, чтобы наше состояние в памяти было инициализировано из localStorage, мы использовали Window.localStorage.getItem (name), и если там ничего нет, мы инициализируем его пустой строкой по умолчанию.  [2:37](https://egghead.io/lessons/react-manage-side-effects-in-a-react-component-with-the-useeffect-hook) Затем, чтобы убедиться, что на входе отображается то же значение для имени, что и для имени в памяти, мы указали свойство value, изменив этот ввод с неконтролируемого на управляемый. | Instructor: [0:00](https://egghead.io/lessons/react-manage-side-effects-in-a-react-component-with-the-useeffect-hook) I want to be able to type in here some value and have that saved in localStorage so that when I refresh the page, that value will be retrieved from localStorage and be loaded into the input.  [0:11](https://egghead.io/lessons/react-manage-side-effects-in-a-react-component-with-the-useeffect-hook) This is called a side effect, and in react, to do this, you use react useEffect. This is another hook, like useState. This function will be called every time the greeting component is rendered.  [0:23](https://egghead.io/lessons/react-manage-side-effects-in-a-react-component-with-the-useeffect-hook) Anytime the greeting component is rendered, we're going to say window.localStorage.setItem -- we'll call it name in localStorage, and we'll set it to the value of name. If we save that, and then open our dev tools and in the application tab here, we can go to localStorage, and we'll see that there's nothing in there.  [0:45](https://egghead.io/lessons/react-manage-side-effects-in-a-react-component-with-the-useeffect-hook) Then we can type a name, and we'll see that the name gets updated in localStorage with whatever the current value for that name variable is.  [0:54](https://egghead.io/lessons/react-manage-side-effects-in-a-react-component-with-the-useeffect-hook) You'll notice that if I refresh here, I'm not getting that name value loaded into my input, and the value in localStorage is getting cleared. This is because we initialize our state to an empty string, and we don't take the localStorage value into account.  [1:09](https://egghead.io/lessons/react-manage-side-effects-in-a-react-component-with-the-useeffect-hook) Then, because we've rendered, we run this callback, updating the localStorage item for name to that new empty string name. We need to initialize that value to whatever's in localStorage if it's there, and an empty string if it's not.  [1:23](https://egghead.io/lessons/react-manage-side-effects-in-a-react-component-with-the-useeffect-hook) Here we'll say window.localStorage.getItem(name), and if that returns null because there's nothing in there, then we'll default that to an empty string. We save that. We type a name. We hit refresh, and we notice that the value in localStorage is still consistent.  [1:43](https://egghead.io/lessons/react-manage-side-effects-in-a-react-component-with-the-useeffect-hook) We notice the value right here is correct, but the name input does not have the name in there. We need to specify what the value for the input should be. We specify a value of name.  [1:57](https://egghead.io/lessons/react-manage-side-effects-in-a-react-component-with-the-useeffect-hook) Now when we save this, we'll get a refresh, and now the value is the same value that's in localStorage and in memory for the state of our react component. We'll notice that as we change this and refresh, we always get that value from localStorage, and we keep that value updated in localStorage as well.  [2:16](https://egghead.io/lessons/react-manage-side-effects-in-a-react-component-with-the-useeffect-hook) In review, to make all this work, we first used react useEffect to set the name value in localStorage to the name value in our state in memory. Then, to have our state in memory be initialized from localStorage, we used Window.localStorage.getItem(name), and if there is nothing in there, then we'll initialize it to an empty string as a default.  [2:37](https://egghead.io/lessons/react-manage-side-effects-in-a-react-component-with-the-useeffect-hook) Then, to make sure that the input is showing the same value for the name as a name in memory, we specified a value prop, changing this input from an uncontrolled to a controlled input. |
| 15. Используйте ленивый инициализатор с useState | 15. Use a lazy initializer with useState |
| Важно понимать, что каждый раз, когда вы вызываете функцию обновления состояния (например, setNameфункцию в нашем компоненте), она запускает повторный рендеринг компонента, который управляет этим состоянием ( Greetingкомпонент в нашем примере). Это именно то, что мы *хотим* , но это *может* быть проблемой в некоторых ситуациях, и есть некоторые оптимизации, которые мы можем применить например: useStateспециально для таких проблем.  В нашем случае мы читаем, localStorageчтобы инициализировать значение нашего состояния для первого рендеринга нашего Greetingкомпонента. Но после этого первого рендера нам больше не нужно читать localStorage, потому что теперь мы управляем этим состоянием в памяти (особенно в той nameпеременной, которую React дает нам при каждой рендеринге). Так что считывать localStorageкаждый рендер после первого не нужно. Таким образом, React позволяет нам указывать функцию вместо фактического значения, и тогда он будет вызывать эту функцию только тогда, когда это необходимо - при первоначальной визуализации.  В этом уроке я покажу вам, как это сделать, и продемонстрирую, как это работает. | Something that’s important to recognize is that every time you call the state updater function (like the setName function in our component), that will trigger a re-render of the component that manages that state (the Greeting component in our example). This is exactly what we *want* to have happen, but it *can* be a problem in some situations and there are some optimizations we can apply for useState specifically in the event that it is a problem.  In our case, we’re reading into localStorage to initialize our state value for the first render of our Greeting component. But after that first render, we don’t need to read into localStorage anymore because we’re managing that state in memory now (specifically in that name variable that React gives us each render). So reading into localStorage every render after the first one is unnecessary. So React allows us to specify a function instead of an actual value, and then it will only call that function when it needs to–on the initial render.  In this lesson, I’ll show you how to do this and demonstrate how it works. |
| Кент С. Доддс: [0:00](https://egghead.io/lessons/react-use-a-lazy-initializer-with-usestate) Одна вещь, которую важно знать об хуке useState, заключается в том, что исходное значение, которое вы здесь указываете, действительно важно для первоначального рендеринга нашего компонента, но затем оно игнорируется для рендеринга нашего компонента в дальнейшем.  [0:11](https://egghead.io/lessons/react-use-a-lazy-initializer-with-usestate) Обычно это не проблема, особенно если у вас здесь пустая строка, но здесь мы читаем в localStorage каждый раз, когда наш компонент приветствия повторно визуализируется. Мы можем добавить console.log('rendered') прямо здесь, чтобы увидеть, как часто это происходит.  [0:26](https://egghead.io/lessons/react-use-a-lazy-initializer-with-usestate) Мы открываем нашу консоль и видим, что у нас уже есть рендеринг для нашего начального рендеринга компонента приветствия. По мере ввода мы получаем рендеринг каждый раз, когда вводим символ. Нам нужно только прочитать его в localStorage для инициализации нашего состояния. Нам не нужно, чтобы он считывался в localStorage каждый раз, когда мы повторно отрисовываем.  [0:47](https://egghead.io/lessons/react-use-a-lazy-initializer-with-usestate) В этом нет ничего особенного, потому что чтение в localStorage происходит довольно быстро, и мы ничего не анализируем, но если бы мы анализировали большой объект JSON, это могло бы стать проблемой.  [0:59](https://egghead.io/lessons/react-use-a-lazy-initializer-with-usestate) Для борьбы с этим в React.useState есть функция отложенной инициализации. Вы можете указать функцию в качестве начального значения, и эта функция будет вызываться для получения начального значения. Эта функция будет вызываться только тогда, когда это абсолютно необходимо для получения начального значения.  [1:15](https://egghead.io/lessons/react-use-a-lazy-initializer-with-usestate) Если мы превратим это в стрелочную функцию, которая просто возвращает это начальное значение, затем мы сохраним это, мы заметим, что мы все еще получаем наши рендеры, но эта функция не вызывается, кроме получения начального значения.  [1:30](https://egghead.io/lessons/react-use-a-lazy-initializer-with-usestate) Мы можем доказать это, сделав эту многострочную стрелочную функцию, возвращая это значение и console.log ('hello') здесь. Теперь мы получаем это приветствие, а затем мы получаем его отрисовку, и это важное замечание: эта функция вызывается синхронно, и ожидается, что она будет синхронной.  [1:51](https://egghead.io/lessons/react-use-a-lazy-initializer-with-usestate) По мере того, как мы набираем "привет", вы увидите, что мы не получаем этот console.log ("привет"). Мы экономим на чтении в локальное хранилище при каждом рендере.  [2:03 Итак](https://egghead.io/lessons/react-use-a-lazy-initializer-with-usestate) , проблема, которую мы здесь решаем, заключается в том, что чтение в localStorage не требуется, за исключением первоначального рендеринга нашего компонента. Здесь мы превращаем наш аргумент начального значения в функцию, чтобы React вызывал его только тогда, когда необходимо получить это начальное значение, которое есть только при первом рендеринге. | Kent C. Dodds: [0:00](https://egghead.io/lessons/react-use-a-lazy-initializer-with-usestate) One thing that's important to know about the useState Hook is that the initial value you provide here is really important for the initial render of our component, but then it's ignored for renders of our component thereafter.  [0:11](https://egghead.io/lessons/react-use-a-lazy-initializer-with-usestate) That's normally not a problem, especially if you have just an empty string here, but here we're reading into localStorage every time our greeting component is re-rendered. We can add a console.log('rendered') right here to see how often that is.  [0:26](https://egghead.io/lessons/react-use-a-lazy-initializer-with-usestate) We open up our console and we'll see that we already have a rendered here for our initial render of the greeting component. As we type, we get a render every time we type a character. We only need it to read into localStorage for the initialization of our state. We don't need to have it read into localStorage for every single time we re-render.  [0:47](https://egghead.io/lessons/react-use-a-lazy-initializer-with-usestate) This isn't really a huge deal here because reading into localStorage is pretty fast and we're not parsing anything, but if we were parsing a big JSON object, then this could be a problem.  [0:59](https://egghead.io/lessons/react-use-a-lazy-initializer-with-usestate) To combat this, React.useState has a lazy initialization feature. You can provide a function as the initial value, and that function will be called to retrieve the initial value. That function will only be called when it's absolutely necessary to retrieve the initial value.  [1:15](https://egghead.io/lessons/react-use-a-lazy-initializer-with-usestate) If we turn this into an arrow function, which simply returns that initial value, then we save this, we'll notice that we're still getting our renders, but this function is not getting called, except on the initial value retrieval.  [1:30](https://egghead.io/lessons/react-use-a-lazy-initializer-with-usestate) We can prove this by making this a multiline arrow function, returning that value and console.log('hello') in here. Now we get that hello, and then we get that rendered, and that's an important note, is that this function is called synchronously, and it's expected to be synchronous.  [1:51](https://egghead.io/lessons/react-use-a-lazy-initializer-with-usestate) As we type hello, you'll see that we do not get that console.log('hello'). We save ourselves the expense of reading into local storage on every render.  [2:03](https://egghead.io/lessons/react-use-a-lazy-initializer-with-usestate) In review, the problem that we're solving here is that reading into localStorage is not necessary, except for the initial render of our component. We turn our initial value argument here into a function so that React will call it only when it's necessary to get that initial value, which is only on the first render. |
| 16. Управление массивом зависимостей useEffect | 16. Manage the useEffect dependency array |
| Что действительно важно знать о React’s useEffect hook это то, что он нетерпеливо пытается синхронизировать «состояние мира» (“state of the world”) с состоянием вашего приложения. Это означает, что обратный вызов вашего эффекта будет запускаться каждый раз при визуализации вашего компонента. Обычно это не приводит к ошибкам (на самом деле, он отлично справляется с предотвращением ошибок, которые преследовали приложения React до того, как useEffectбыли доступны), но это определенно может быть неоптимальным (а в некоторых случаях может привести к бесконечному циклу).  В этом уроке мы увидим, что эффекта нашего callback вызывается чаще, чем нужно, и вы узнаете, как добавить массив зависимостей, чтобы он обновлялся только тогда, когда состояние, на которое он полагается, изменяется. А в реальных приложениях вы должны убедиться, что у вас есть и соблюдаются правила плагина [ESLint](https://eslint.org/) : [eslint-plugin-react-hooks](https://www.npmjs.com/package/eslint-plugin-react-hooks) (многие инструменты, такие как Create React App, устанавливают и настраивают его по умолчанию). | Something that’s really important to know about React’s useEffect hook is that it eagerly attempts to synchronize the “state of the world” with the state of your application. That means that your effect callback will run every time your component is rendered. This normally won’t lead to bugs (in fact, it does a great job at preventing bugs that plagued React apps before useEffect was available), but it can definitely be sub-optimal (and in some cases can result in an infinite loop).  In this lesson we’ll observe that our effect callback is getting called more than it needs to be, and you’ll learn how to add a dependency array so it is updated only when the state it relies on changes. And in real applications, you’ll want to make sure you have and follow the rules from the [ESLint](https://eslint.org/) plugin: [eslint-plugin-react-hooks](https://www.npmjs.com/package/eslint-plugin-react-hooks) (many tools like Create React App have this installed and configured by default). |
| Кент С. Доддс: [0:00](https://egghead.io/lessons/react-manage-the-useeffect-dependency-array) Мы добавили здесь немного, чтобы смоделировать еще одну оптимизацию, которую мы хотим провести. Теперь у нас есть состояние подсчета, и оно просто отображается в этой кнопке. Каждый раз, когда мы нажимаем на кнопку, происходит повторная визуализация нашего компонента приложения, что вызывает повторную визуализацию нашего компонента приветствия.  [0:16](https://egghead.io/lessons/react-manage-the-useeffect-dependency-array) Каждый раз, когда наш компонент приветствия повторно отрисовывается, будет вызываться этот useEffect. Если мы console.log('greeting useEffect'), мы сохраним это. Откройте наши DevTools. Мы видим, что получаем приветствие useEffect для начального рендеринга. Каждый раз, когда я щелкаю по нему, мы получаем новое приветствие useEffect.  [0:35](https://egghead.io/lessons/react-manage-the-useeffect-dependency-array) Эффект, который мы выполняем, обновляет localStorage, чтобы установить имя равным значению имени. Этот побочный эффект не нужно запускать, потому что значение имени не изменилось.  [0:46](https://egghead.io/lessons/react-manage-the-useeffect-dependency-array) Нам нужно только синхронизировать значение localStorage со значением имени, которое у нас есть в памяти для состояния этого компонента.  [0:54](https://egghead.io/lessons/react-manage-the-useeffect-dependency-array) React useEffect принимает второй аргумент как оптимизацию для решения этой проблемы. Этот второй аргумент представляет собой массив зависимостей, в который вы передаете все зависимости для своего побочного эффекта (side effect). Здесь вы передаете все, что хотите, чтобы синхронизировать состояние мира с состоянием нашего приложения.  [1:13](https://egghead.io/lessons/react-manage-the-useeffect-dependency-array) В нашем случае единственное, что мы пытаемся синхронизировать здесь, - это состояние localStorage, то есть состояние мира, с состоянием нашего приложения, которым является имя. Для нашего массива зависимостей мы дадим имя.  [1:27](https://egghead.io/lessons/react-manage-the-useeffect-dependency-array) Если мы сохраним это, то заметим, что сначала вызывается приветствие useEffect, потому что useEffect всегда будет вызываться при первоначальном монтировании. В дальнейшем, когда мы нажимаем на эту кнопку, мы не получим приветствие useEffect.  [1:44](https://egghead.io/lessons/react-manage-the-useeffect-dependency-array) Затем, когда мы обновим значение имени, мы получим приветствие useEffect, вызванное, потому что имя изменилось, и состояние мира теперь не синхронизировано с состоянием нашего приложения. React вызовет наш обратный вызов useEffect.  [1:58](https://egghead.io/lessons/react-manage-the-useeffect-dependency-array) Очень важно, чтобы этот массив зависимостей был точным в соответствии с зависимостями, от которых зависит ваша функция обратного вызова. Например, мы полагаемся на имя. Вот почему так важно хранить здесь этот список зависимостей.  [2:11](https://egghead.io/lessons/react-manage-the-useeffect-dependency-array) Если вы не будете вести этот список точным, то вы можете упустить возможность синхронизации состояния мира с изменениями состояния, которые происходят в вашем приложении, и ваши пользователи могут потерять сохраненную работу.  [2:21](https://egghead.io/lessons/react-manage-the-useeffect-dependency-array) Например, если бы я удалил имя из этого массива и сохранил его, мы получили бы вызов этого приветствия useEffect. Когда я введу здесь текст, мы заметим, что состояние моего приложения постоянно обновляется, но я не получаю console.logs.  [2:37](https://egghead.io/lessons/react-manage-the-useeffect-dependency-array) Если я обновлюсь, я вернусь туда, где был раньше. Я, как пользователь этого приложения, потерял работу, потому что мой код неправильно синхронизировал состояние моего приложения с состоянием мира вне моего приложения.  [2:51](https://egghead.io/lessons/react-manage-the-useeffect-dependency-array) Чтобы помочь вам избежать ошибок здесь, команда React создала плагин ESLint под названием eslint-plugin-react-hooks, который вы можете использовать не только для того, чтобы гарантировать, что массив зависимостей постоянно обновляется, но поддерживайте его в актуальном состоянии автоматически, используя функцию исправления ESLint.  [3:09](https://egghead.io/lessons/react-manage-the-useeffect-dependency-array) Правило, которое поможет вам в этом, - это правило исчерпывающей зависимости, и я настоятельно рекомендую вам использовать этот инструмент и следовать этому правилу.  [3:17 Итак](https://egghead.io/lessons/react-manage-the-useeffect-dependency-array) , проблема, которую мы здесь решали, заключалась в том, что обратный вызов нашего эффекта вызывается чаще, чем необходимо. Я хочу особо отметить здесь, что то, что его вызывали больше, чем нужно, не означает, что в нашем приложении есть ошибка, так что это просто оптимизация, чтобы наше приложение работало немного быстрее.  [3:35](https://egghead.io/lessons/react-manage-the-useeffect-dependency-array) Массив зависимостей может быть необходим не во всех случаях, но в нашем случае мы могли бы просто добавить этот массив зависимостей с одной зависимостью, на которую опирался наш обратный вызов эффекта. Этот обратный вызов вызывается только при необходимости. | Kent C. Dodds: [0:00](https://egghead.io/lessons/react-manage-the-useeffect-dependency-array) We've added a little bit here to simulate another optimization that we want to make. Here now we have a count state, and that's just being rendered into this button. Every time we click on the button, it re-renders our app component, which triggers a re-render of our greeting component.  [0:16](https://egghead.io/lessons/react-manage-the-useeffect-dependency-array) Every time our greeting component re-renders, this useEffect is going to be called. If we console.log('greeting useEffect'), we'll save that. Open up our DevTools. We see we get the greeting useEffect for the initial render. Every time I click this, we'll get another greeting useEffect.  [0:35](https://egghead.io/lessons/react-manage-the-useeffect-dependency-array) The effect that we're running is updating localStorage to set the name to the name value. This side effect doesn't need to be run because the name value hasn't changed.  [0:46](https://egghead.io/lessons/react-manage-the-useeffect-dependency-array) We only need to synchronize the localStorage value with the name value that we have in memory for the state of this component.  [0:54](https://egghead.io/lessons/react-manage-the-useeffect-dependency-array) React useEffect accepts a second argument as an optimization to combat this problem. That second argument is a dependency array where you pass all the dependencies for your side effect. This is where you pass anything that you want to make sure you synchronize the state of the world with the state of our application.  [1:13](https://egghead.io/lessons/react-manage-the-useeffect-dependency-array) In our case, the only thing that we're trying to synchronize here is the state of localStorage, which is the state of the world, with the state of our application, which is the name. For our dependency array, we're going to provide the name.  [1:27](https://egghead.io/lessons/react-manage-the-useeffect-dependency-array) If we save that, then we'll notice that we get that greeting useEffect called initially, and that's because useEffect is always going to be called on the initial mount. Hereafter, when we click on this button, we're not going to get the greeting useEffect.  [1:44](https://egghead.io/lessons/react-manage-the-useeffect-dependency-array) Then when we update the name value, we're going to get greeting useEffect called because the name has changed, and the state of the world is now out of sync with the state of our application. React is going to call our useEffect callback.  [1:58](https://egghead.io/lessons/react-manage-the-useeffect-dependency-array) It's very important that you keep this dependency array accurate according to the dependencies that your callback function relies on. For example, we rely on name. That's why it's important that we keep this dependency list in here.  [2:11](https://egghead.io/lessons/react-manage-the-useeffect-dependency-array) If you don't keep this list accurate, then you could be missing out on synchronizing the state of the world with state changes that happen in your app, and your users could lose saved work.  [2:21](https://egghead.io/lessons/react-manage-the-useeffect-dependency-array) For example, if I were to remove the name from this array and save that, then we'll get that greeting useEffect called. As I type in here, we'll notice that the state of my application is being kept updated, but I'm not getting any console.logs.  [2:37](https://egghead.io/lessons/react-manage-the-useeffect-dependency-array) If I refresh, I'm going to be back to where I was before. I, as the user of this application, have lost work, because my code was not properly synchronizing the state of my application with the state of the world outside my application.  [2:51](https://egghead.io/lessons/react-manage-the-useeffect-dependency-array) In an effort to help you avoid making mistakes here, the React team has created an ESLint plugin called eslint-plugin-react-hooks which you can use to not only ensure that the dependency array is kept up-to-date, but keep it up-to-date automatically for you using ESLint's fix feature.  [3:09](https://egghead.io/lessons/react-manage-the-useeffect-dependency-array) The rule that will help you with this is the exhaustive-deps rule, and I strongly advise that you use this tool and follow that rule.  [3:17](https://egghead.io/lessons/react-manage-the-useeffect-dependency-array) In review, the problem that we were solving here is our effect callback was being called more than it needed to be. I want to make a special note here, that just because it was being called more than it needed to be didn't mean we had a bug in our application, so this is just an optimization to make our application run a little faster.  [3:35](https://egghead.io/lessons/react-manage-the-useeffect-dependency-array) A dependency array may not be necessary in all cases, but in our case, we could simply add this dependency array with the one dependency that our effect callback relied on. This callback is only called when necessary. |
| 17. Создавайте многоразовые custom hooks | 17. Create reusable custom hooks |
| Одна из вещей, которые нам нравятся в программировании, - это возможность взять код, поместить его в функцию и повторно использовать в других местах программного обеспечения. Давайте представим сценарий, в котором мы хотим поделиться нашим localStorageкодом с другими компонентами, чтобы другие компоненты могли синхронизировать состояние localStorage(или мы могли бы даже сделать это с разными переменными в одном компоненте). В частности, сделайте шаг назад от React и, учитывая, как повторное использование кода работает в JavaScript в целом, мы можем просто создать функцию, поместить в нее соответствующий код и затем вызвать ее из исходного местоположения. Этот процесс работает точно так же с кодом хуков React, так что давайте сделаем это!  В процессе мы узнаем несколько соглашений для этого и узнаем о некоторых дополнительных проблемах, которые возникают при обобщении нашего кода. Как всегда, следуйте [практике программирования AHA](https://kcd.im/aha) ! | One of the things we love about programming is the ability to take code, place it in a function, and reuse it in other places in the software. Let’s imagine a scenario where we want to share our localStorage code with other components so other components could synchronize state with localStorage (or we could even do it with different variables in the same component). Take a step back from React specifically and considering how code reuse works in JavaScript in general, we can simply make a function, put our relevant code in that function, and then call it from the original location. That process works exactly the same with React hooks code, so let’s do that!  In the process, we’ll learn a few of the conventions for doing this and we’ll learn about some additional challenges that come with generalizing our code. As always, follow [AHA programming practices](https://kcd.im/aha)! |
| Кент С. Доддс: [0:00](https://egghead.io/lessons/react-create-reusable-custom-hooks) Логика, которая у нас есть для сохранения некоторого состояния в localStorage и поддержания его синхронизации, может быть полезна в других областях нашего приложения.  [0:07](https://egghead.io/lessons/react-create-reusable-custom-hooks) К счастью, React Hooks - это довольно ванильный JavaScript, и поделиться этой логикой так же просто, как поделиться любой другой логикой в JavaScript. Что мы собираемся сделать, так это сделать функцию. Мы вызовем это useLocalStorageState (), а затем переместим эти строки кода в эту функцию и заменим их вызовом этой функции.  [0:30](https://egghead.io/lessons/react-create-reusable-custom-hooks) Нам нужно обобщить код нашей функции. Вместо имени, возможно, имеет смысл называть это состояние и setState. Вместо того, чтобы получать элемент со строковым именем, для пользователя этой функции может иметь смысл предоставить нам ключ для localStorage, поэтому мы примем параметр с именем key.  [0:51](https://egghead.io/lessons/react-create-reusable-custom-hooks) Вместо этого в качестве значения по умолчанию людям может иметь смысл предоставить нам свое собственное значение по умолчанию. Мы примем это как еще один параметр. Мы можем по умолчанию использовать пустую строку на тот случай, если они не захотят ее предоставлять.  [1:05](https://egghead.io/lessons/react-create-reusable-custom-hooks) Затем мы заменим это значение по умолчанию. Мы заменим имя строки параметром, который мы принимаем, с именем key. Затем убедитесь, что мы обновили эти две ссылки и на это старое значение состояния.  [1:19](https://egghead.io/lessons/react-create-reusable-custom-hooks) Затем, когда мы вызываем useLocalStorageState, нам нужно будет получить доступ к этой функции stateUpdater и к самому значению состояния. Вернем состояние и setState. Мы можем сделать так, чтобы в нашем настраиваемом хуке был API, аналогичный useState, чтобы он был знаком людям, которые привыкли к хуку useState.  [1:39](https://egghead.io/lessons/react-create-reusable-custom-hooks) Тогда мы можем спуститься сюда. Мы можем получить наше имя и наш setName из useLocalStorageState. Мы укажем наш ключ как имя. Тогда мы закончили. Это довольно простой рефакторинг кода, который у нас был ранее, в пользовательскую функцию. Если мы введем здесь Чак, а затем обновим, мы увидим, что Чак все еще там.  [2:02](https://egghead.io/lessons/react-create-reusable-custom-hooks) Одна вещь, которую я хочу здесь отметить, - это то, что мы поставили перед нашей функцией слово use. Это соглашение, а не требование. Мы можем изменить это на что угодно. Сохраните это. Все будет работать так же хорошо. Мы можем нажать «Обновить». Вот и все.  [2:19](https://egghead.io/lessons/react-create-reusable-custom-hooks) Причина, по которой мы следуем этому соглашению, заключается в том, что eslint-plugin-react-hooks может обеспечивать соблюдение некоторых из тех же рекомендаций и правил для наших пользовательских хуков, что и для встроенных хуков, потому что он действует в предположении, что эти функции, которые начинаются с use, являются пользовательскими хуками.  [2:39](https://egghead.io/lessons/react-create-reusable-custom-hooks) В обзоре мы взяли код, который был в нашем функциональном компоненте. Мы переместили его в отдельную функцию, а затем вызвали его из нашего функционального компонента. Это ничем не отличалось от любого другого обычного рефакторинга JavaScript, который упрощает совместное использование логики между компонентами и даже создание библиотек с открытым исходным кодом, которые предоставляют такие настраиваемые хуки.  [2:59](https://egghead.io/lessons/react-create-reusable-custom-hooks) Мы также узнали о важности использования префикса использования в наших пользовательских хуках, чтобы мы следовали соглашению сообщества, а плагин ESLint может помочь нам избежать ошибок.  [3:09](https://egghead.io/lessons/react-create-reusable-custom-hooks) Между прочим, прямо сейчас происходит одна такая ошибка. В этом файле index.html я не могу запустить ESLint, чтобы показать мне ошибки, которые я делаю. Я сделал ошибку прямо здесь, в этом массиве зависимостей. Интересно, сможете ли вы понять, что это такое? Мне не хватает одной из зависимостей для моего обратного вызова.  [3:28](https://egghead.io/lessons/react-create-reusable-custom-hooks) Раньше этот ключ был жестко запрограммирован как имя. Это никогда не могло измениться. Моему массиву зависимостей это не нужно было включать. Теперь я принимаю ключ в качестве параметра от пользователей настраиваемого хука LocalStorageState. Они потенциально могут изменить это значение. Если да, то мне нужно обязательно отреагировать на это изменение, чтобы обновить локальное хранилище и не потерять работу пользователя.  [3:53](https://egghead.io/lessons/react-create-reusable-custom-hooks) Важно, чтобы в ваших приложениях вы использовали это правило ESLint и следовали ему, чтобы избежать подобных ошибок. Имея это в виду, мы могли бы сделать этот хук useLocalStorageState намного более надежным, удалив элемент в старом ключе, если этот ключ изменился. Я оставлю это как забавное упражнение, которым вы можете заняться позже. | Kent C. Dodds: [0:00](https://egghead.io/lessons/react-create-reusable-custom-hooks) The logic that we have here for storing some state into localStorage and keeping it synchronized could be useful in other areas of our application.  [0:07](https://egghead.io/lessons/react-create-reusable-custom-hooks) Thankfully, React Hooks are pretty Vanilla JavaScript, and sharing that logic is just as straightforward as sharing any other logic in JavaScript. What we're going to do is make a function. We'll call that useLocalStorageState(), and then we'll move these lines of code into that function, and we'll replace them with a call to that function.  [0:30](https://egghead.io/lessons/react-create-reusable-custom-hooks) We need to generalize the code that's in our function. Instead of a name, it might make more sense to call this state and setState. Instead of getting the item with the string name, it might make more sense for the user of this function to provide us a key for localStorage, so we'll accept a parameter called key.  [0:51](https://egghead.io/lessons/react-create-reusable-custom-hooks) Instead of this as a default value, it might make sense for people to provide us their own default value. We'll accept that as another parameter. We can default that to an empty string just in case they don't want to provide it.  [1:05](https://egghead.io/lessons/react-create-reusable-custom-hooks) Then we'll replace that with the default value. We'll replace the string name with a parameter that we accept called key. Then let's make sure we update these two references to that old state value as well.  [1:19](https://egghead.io/lessons/react-create-reusable-custom-hooks) Then when we call useLocalStorageState, we're going to need to get access to that stateUpdater function and to the state value itself. Let's return state and setState. We can make our custom Hook have a similar API to useState so it's familiar to people who are used to the useState Hook.  [1:39](https://egghead.io/lessons/react-create-reusable-custom-hooks) Then we can come down here. We can get our name and our setName from useLocalStorageState. We'll specify our key as name. Then we're done. That's a pretty straight-up refactor from the code that we had before into a custom function. If we type in here Chuck and then refresh, we'll see that Chuck is still in there.  [2:02](https://egghead.io/lessons/react-create-reusable-custom-hooks) One thing I want to call out here is that we prefaced our function with the word use. That's a convention and not required. We could change this to whateverWeWant. Save that. Things will work just as well. We can hit refresh. There we have it.  [2:19](https://egghead.io/lessons/react-create-reusable-custom-hooks) The reason that we follow this convention is because the eslint-plugin-react-hooks is able to enforce some of the same recommendations and rules on our custom Hooks as it is on the built-in Hooks because it acts under the assumption that functions that begin with use are custom Hooks.  [2:39](https://egghead.io/lessons/react-create-reusable-custom-hooks) In review, what we did here is we took some code that was in our function component. We moved it into its own function and then called it from our function component. This was no different from any other regular JavaScript refactor, which makes it easy to share logic across components and even make open source libraries that expose custom Hooks like this.  [2:59](https://egghead.io/lessons/react-create-reusable-custom-hooks) We also learned the importance of using the use prefix on our custom Hooks so that we follow the community convention and the ESLint plugin can help us avoid mistakes.  [3:09](https://egghead.io/lessons/react-create-reusable-custom-hooks) Incidentally, one such mistake is happening right now. In this index.html file, I can't have ESLint running to show me the mistakes that I'm making. I made a mistake right here in this dependency array. I wonder if you can figure out what it is. I'm missing one of the dependencies for my callback.  [3:28](https://egghead.io/lessons/react-create-reusable-custom-hooks) Before, we had this key hardcoded as name. It couldn't ever change. My dependency array didn't need to include that. Now, I'm accepting a key as a parameter from the users of the LocalStorageState custom Hook. They could potentially change that value. If they do, then I need to make sure to respond to that change so that I update local storage and not lose any of the user's work.  [3:53](https://egghead.io/lessons/react-create-reusable-custom-hooks) It's important that in your applications you use and follow that ESLint rule so you can avoid bugs like this. With this in mind, we could make this useLocalStorageState Hook a lot more robust by removing the item at the old key if that key has changed. I'm going to leave that as a fun exercise for you to do later. |
| **18. Управляйте DOM с помощью React refs** | **18. Manipulate the DOM with React refs** |
| React действительно хорош в создании и обновлении элементов DOM, но иногда вам нужно работать с ними самостоятельно. Обычно это используется, когда вы используете стороннюю библиотеку, которая не была создана специально для React или с ней. Для этого нам нужно иметь какое-то значение, связанное с нашим компонентом (например, состояние), чтобы хранить ссылку на элемент DOM, но не запускать повторный рендеринг при его обновлении (в отличие от состояния). В React есть что-то специально для этого, и он называется ref.  Вы создаете refобъект с помощью useRef hook, и currentсвойство этого объекта является текущим значением ref. Это может быть что угодно, но если вы передадите этот refобъект компоненту как a prop названный ref, тогда React установит currentсвойство для создаваемого им элемента DOM, чтобы вы могли ссылаться на него и манипулировать им в своем useEffectхуке.  В этом уроке мы увидим, как это работает с классной библиотекой [vanilla-tilt](https://www.npmjs.com/package/vanilla-tilt) . | React is really good at creating and updating DOM elements, but sometimes you need to work with them yourself. A common use case for this is when you’re using a third party library that wasn’t built for or with React specifically. To do this, we need to have some value that’s associated with our component (like state) to store a reference to the DOM element, but doesn’t trigger re-renders when it’s updated (unlike state). React has something specifically for this and it’s called a ref.  You create a ref object with the useRef hook and that object’s current property is the current value of the ref. It can be anything, but if you pass that ref object to a component as a prop called ref, then React will set the current property to the DOM element it creates so you can reference it and manipulate it in your useEffect hook.  In this lesson we’ll get to see how that works with a cool library called [vanilla-tilt](https://www.npmjs.com/package/vanilla-tilt). |
| Кент С. Доддс: [0:00](https://egghead.io/lessons/react-manipulate-the-dom-with-react-refs-cfdb8c6a) У нас есть функциональный компонент под названием Tilt. Благодаря использованию удобных имен классов и некоторого удобного CSS, мы сделали его таким привлекательным, но мы можем заставить его делать что-то необычное, используя включенную мной библиотеку под названием vanilla-tilt. Vanilla-tilt берет ноду DOM и заставляет его реагировать, когда пользователь наводит курсор на эту ноду DOM.  [0:19](https://egghead.io/lessons/react-manipulate-the-dom-with-react-refs-cfdb8c6a) Нода DOM, который мы хотим передать, - это нода DOM, созданный для этого элемента, the tilt-root. Помните, что это элемент React, а не нода DOM, и React берет этот элемент Reactа и отображает его в DOM.  [0:32](https://egghead.io/lessons/react-manipulate-the-dom-with-react-refs-cfdb8c6a) Нам нужно, чтобы React предоставил нам ноду DOM, который создает для этого конкретного элемента React, чтобы мы могли подключить к нему vanilla-tilt. Для этого мы собираемся использовать a ref prop, и нам нужно передать ref, что представляет собой объект с изменяемым текущим свойством.  [0:47](https://egghead.io/lessons/react-manipulate-the-dom-with-react-refs-cfdb8c6a) Давайте продолжим и воспользуемся React.useRef hook. Отсюда мы получим наш tiltRef, а затем мы сможем его скопировать и вставить сюда в нашу ссылку. Тогда tiltRef - это объект, у которого есть текущее свойство. Это текущее свойство является текущим значением для этого объекта ссылки.  [1:05](https://egghead.io/lessons/react-manipulate-the-dom-with-react-refs-cfdb8c6a) В нашем случае, поскольку мы передаем этот ref в div с опорой ref, текущим свойством будет узел DOM, который React создает для этого div. Если мы console.log tiltRef.current и сохраним его, тогда мы должны увидеть узел DOM.  [1:21](https://egghead.io/lessons/react-manipulate-the-dom-with-react-refs-cfdb8c6a) Мы действительно видим undefined. Причина, по которой мы видим это, заключается в том, что во время выполнения этой функции React не создал узел DOM для этого div, поэтому tiltRef.current в настоящее время не определен. Фактически, вы можете инициализировать это текущее значение, передав аргумент их useRef hook.  [1:39](https://egghead.io/lessons/react-manipulate-the-dom-with-react-refs-cfdb8c6a) Как нам получить узел DOM, чтобы мы могли инициализировать на нем vanilla-tilt? Нам нужен код, который запускается после того, как React обновит DOM и установит свойство tiltRef.current. Взаимодействие с DOM - это побочный эффект. Логичным местом для этого был бы хук React.useEffect.  [1:55](https://egghead.io/lessons/react-manipulate-the-dom-with-react-refs-cfdb8c6a) Если мы переместим сюда console.log и сохраним его, то получим обновление, и теперь мы получаем узел DOM. Отсюда мы можем получить наш tiltNode из tiltRef.current, а затем давайте продолжим и создадим здесь несколько vanillaTiltOptions. Мы можем указать, как мы хотим, чтобы vanilla-tilt обрабатывал этот узел DOM.  [2:16](https://egghead.io/lessons/react-manipulate-the-dom-with-react-refs-cfdb8c6a) Мы скажем макс 25, скорость 400, яркость истинная и максимальная яркость 0,5. Теперь мы можем сказать VanillaTilt.init на этом узле tiltNode и с помощью vanillaTiltOptions. Если мы сохраним это, мы получим обновление. Этот код запускается после того, как React обработал DOM и обновил свойство tiltRef.current, чтобы мы могли получить доступ к tiltNode и инициализировать vanilla-tilt.  [2:47](https://egghead.io/lessons/react-manipulate-the-dom-with-react-refs-cfdb8c6a) Теперь, если мы наведем курсор мыши, мы получим действительно крутой эффект. Если мы снимем флажок показывать наклон, это отключит компонент наклона со страницы, удалив со страницы узел DOM. Однако на этом узле DOM все еще есть куча обработчиков событий и несколько ссылок на этот узел DOM из библиотеки vanilla tilt.  [3:07](https://egghead.io/lessons/react-manipulate-the-dom-with-react-refs-cfdb8c6a) Это означает, что сам узел DOM может не существовать на странице, но он все еще существует в памяти, потому что на него есть ссылки в vanilla-tilt. К сожалению, это могло привести к утечке памяти. Если мы будем продолжать монтировать и размонтировать это снова и снова, тогда у нас будет куча узлов DOM, которые будут сидеть в памяти, которые на самом деле больше не нужны пользователю.  [3:28](https://egghead.io/lessons/react-manipulate-the-dom-with-react-refs-cfdb8c6a) Чтобы решить эту проблему, мы можем вернуть функцию, которая будет вызываться при каждом обновлении нашего компонента наклона. В этой функции мы можем сказать tiltNode.vanillaTilt, который является свойством, которое vanilla-tilt добавляет к нашему DOM-узлу .destroy.  [3:48](https://egghead.io/lessons/react-manipulate-the-dom-with-react-refs-cfdb8c6a) Это удалит все ссылки нашего узла DOM из vanilla-tilt и удалит все обработчики событий, чтобы мы могли избежать утечек памяти с помощью нашего компонента tilt. Чего вы не видите, так это того, что этот узел DOM на самом деле был правильно собран мусором. Нам больше не нужно беспокоиться об утечках памяти.  [4:04](https://egghead.io/lessons/react-manipulate-the-dom-with-react-refs-cfdb8c6a) Еще нам следует понять, что useEffect будет вызываться при каждом повторном рендеринге нашего компонента наклона. Это не приведет к каким-либо ошибкам, но это неоптимально, потому что это означает, что мы вызываем destroy с vanilla-tilt, а затем мы вызываем init с vanilla-tilt между каждым рендером нашего компонента.  [4:21](https://egghead.io/lessons/react-manipulate-the-dom-with-react-refs-cfdb8c6a) Нам не нужно этого делать, потому что ни одна из вещей в этой функции не может измениться при повторном рендеринге нашего компонента наклона. Мы собираемся добавить сюда массив зависимостей со всеми зависимостями нашей функции, и поскольку ни одна из переменных в нашем обратном вызове useEffect не может измениться, нам не нужно перечислять здесь какие-либо зависимости.  [4:37 Фактически](https://egghead.io/lessons/react-manipulate-the-dom-with-react-refs-cfdb8c6a) это означает, что я хочу запустить это, когда компонент изначально установлен на странице, а затем я хочу запустить это, когда компонент будет отключен от страницы. Это еще одна полезная оптимизация, которую мы можем сделать для этого случая.  [4:49 Итак](https://egghead.io/lessons/react-manipulate-the-dom-with-react-refs-cfdb8c6a) , что мы хотели сделать, так это заставить этот узел DOM делать какие-то причудливые вещи, и мы включили vanilla-tilt на страницу, чтобы мы могли это сделать. Мы также включили немного CSS от автора vanilla-tilt, чтобы наши узлы DOM выглядели по-настоящему привлекательно.  [5:05](https://egghead.io/lessons/react-manipulate-the-dom-with-react-refs-cfdb8c6a) Чтобы получить доступ к узлам DOM, чтобы мы могли инициализировать vanilla-tilt на нем, мы используем этот проп ref и передаем в div то, что мы сделали, обратно из вызова useRef React. Эта ссылка имеет текущее свойство, которое мы можем использовать для доступа к текущему значению этого объекта.  [5:19](https://egghead.io/lessons/react-manipulate-the-dom-with-react-refs-cfdb8c6a) Причина, по которой useRef - это объект, имеющий текущее свойство, заключается в том, что мы можем изменить текущее свойство на все, что захотим, не вызывая повторной визуализации нашего компонента. Мы можем использовать refs не только для узлов DOM, как здесь. Мы можем использовать его для любого значения, которое хотим отслеживать и изменять с течением времени, не вызывая повторного рендеринга нашего компонента.  [5:39](https://egghead.io/lessons/react-manipulate-the-dom-with-react-refs-cfdb8c6a) После того, как компонент был смонтирован, будет вызван обратный вызов useEffect. Мы получаем узел наклона из нашего текущего свойства tilt ref. Мы создаем несколько опций vanilla-tilt и передаем их в инициализацию нашего узла tilt с помощью vanilla-tilt.  [5:54](https://egghead.io/lessons/react-manipulate-the-dom-with-react-refs-cfdb8c6a) Затем мы возвращаем функцию очистки, чтобы мы могли удалить все ссылки на наш узел DOM в vanilla-tilt и удалить все прослушиватели событий, которые vanilla-tilt поместил на наш DOM-узел. | Kent C. Dodds: [0:00](https://egghead.io/lessons/react-manipulate-the-dom-with-react-refs-cfdb8c6a) We have a function component called Tilt. Thanks to use some handy class names and some handier CSS, we have it styled in this fancy-looking way but we can make it do something fancy by using a library that I've included called vanilla-tilt. Vanilla-tilt takes a DOM node and makes it react to when the user mouses over that DOM node.  [0:19](https://egghead.io/lessons/react-manipulate-the-dom-with-react-refs-cfdb8c6a) The DOM node we want to give it is the DOM node that's created for this element, the tilt root. Remember this is a React element, not a DOM node and React takes that React element and renders it to the DOM.  [0:32](https://egghead.io/lessons/react-manipulate-the-dom-with-react-refs-cfdb8c6a) We need React to give us the DOM node that creates for this particular React element so we can wire up vanilla-tilt to it. To do this, we're going to use a ref prop and we need to pass a ref which is an object that has a mutable current property.  [0:47](https://egghead.io/lessons/react-manipulate-the-dom-with-react-refs-cfdb8c6a) Let's go ahead and use the React.useRef hook. From that we'll get our tiltRef and then we can copy that, paste it here to our ref. Then the tiltRef is an object that has a current property. That current property is the current value for this ref object.  [1:05](https://egghead.io/lessons/react-manipulate-the-dom-with-react-refs-cfdb8c6a) In our case, because we're passing this ref to a div with a ref prop, that current property will be the DOM node that React creates for this div. If we console.log tiltRef.current, and we save that then we should be seeing the DOM node.  [1:21](https://egghead.io/lessons/react-manipulate-the-dom-with-react-refs-cfdb8c6a) We actually see undefined. The reason we see that is because at the time that this function runs, React has not created the DOM node for this div, so the tiltRef.current is currently undefined. In fact, you can initialize that current value by passing an argument to their useRef hook.  [1:39](https://egghead.io/lessons/react-manipulate-the-dom-with-react-refs-cfdb8c6a) Here, how do we get the DOM node so we can initialize vanilla-tilt on it? We need to have some code that runs after React has updated the DOM and set our tiltRef.current property. Interacting with the DOM is a side effect. The logical place for this would be in a React.useEffect hook.  [1:55](https://egghead.io/lessons/react-manipulate-the-dom-with-react-refs-cfdb8c6a) If we move that console.log in here and we save that, then we get a refresh and now, we are getting the DOM node. From here, we can get our tiltNode from tiltRef.current and then let's go ahead and make some vanillaTiltOptions here. We can specify how we want vanilla-tilt to treat this DOM node.  [2:16](https://egghead.io/lessons/react-manipulate-the-dom-with-react-refs-cfdb8c6a) We'll say max 25, speed 400, glare true, and max glare .5. With that now, we can say VanillaTilt.init on that tiltNode and with the vanillaTiltOptions. If we save that, we get a refresh. This code runs after React has rendered to the DOM and updated the tiltRef.current property so that we can get access to the tiltNode and initialize vanilla-tilt.  [2:47](https://egghead.io/lessons/react-manipulate-the-dom-with-react-refs-cfdb8c6a) Now, if we hover, we get that really cool effect. If we unclick the show tilt checkbox, that will unmount the tilt component from the page removing the DOM node from the page. However, there's still a bunch of event handlers on that DOM node and several references to that DOM node from within the vanilla tilt library.  [3:07](https://egghead.io/lessons/react-manipulate-the-dom-with-react-refs-cfdb8c6a) That means that the DOM node itself may not exist on the page but it does still exist in memory because there're references to it in vanilla-tilt. Unfortunately, this could lead to a memory leak. If we keep on mounting and unmounting this over and over and over again, then we're going to have a bunch of DOM node sitting around in memory that really aren't needed by the user anymore.  [3:28](https://egghead.io/lessons/react-manipulate-the-dom-with-react-refs-cfdb8c6a) To combat this problem, we can return a function which will be called for every update of our tilt component. In this function, we can say tiltNode.vanillaTilt which is a property that vanilla-tilt is adding to our DOM node .destroy.  [3:48](https://egghead.io/lessons/react-manipulate-the-dom-with-react-refs-cfdb8c6a) This will remove all references of our DOM node from vanilla-tilt and remove all event handlers so that we can avoid memory leaks with our tilt component. What you can't see is that that DOM node has actually been garbage collected properly. We don't need to worry about memory leaks anymore.  [4:04](https://egghead.io/lessons/react-manipulate-the-dom-with-react-refs-cfdb8c6a) Another thing we should do here is recognize that useEffect is going to be called on every re-render of our tilt component. That's not going to lead to any bugs but it is sub-optimal because what that means is that we're called destroy with vanilla-tilt and then we'll call init with vanilla-tilt between every render of our component.  [4:21](https://egghead.io/lessons/react-manipulate-the-dom-with-react-refs-cfdb8c6a) We don't need to do that because none of the things in this function can change with re-renders of our tilt component. We're going to add a dependency array here with all the dependencies of our function and because none of the variables in our useEffect callback can change, we don't need to list any dependencies here.  [4:37](https://egghead.io/lessons/react-manipulate-the-dom-with-react-refs-cfdb8c6a) What this is effectively saying is I want to run this when the component is initially mounted to the page and then I want to run this when the component is unmounted from the page. That's another useful optimization we can make for this case.  [4:49](https://egghead.io/lessons/react-manipulate-the-dom-with-react-refs-cfdb8c6a) In review, the thing that we wanted to do is make this DOM node do some fancy stuff and we included vanilla-tilt on the page so that we could do that. We also included some CSS from the vanilla-tilt author to make our DOM nodes look really fancy.  [5:05](https://egghead.io/lessons/react-manipulate-the-dom-with-react-refs-cfdb8c6a) To get access to the DOM nodes so we can initialize vanilla-tilt on it, we're using this ref prop on the div and we pass the thing that we did back from a React useRef call. That ref has a current property that we can use to access the current value of this object.  [5:19](https://egghead.io/lessons/react-manipulate-the-dom-with-react-refs-cfdb8c6a) The reason that useRef is an object that has a current property is so that we can mutate the current property to be whatever we want without triggering a re-render of our component. We can use refs for more than just DOM nodes like we're using here. We can use it for any value that we want to keep track of and mutate over time without triggering a re-render of our component.  [5:39](https://egghead.io/lessons/react-manipulate-the-dom-with-react-refs-cfdb8c6a) After a component has been mounted, our useEffect callback is going to be called. We get the tilt node from our tilt ref current property. We create some vanilla-tilt options and pass those to the initialization for our tilt node with vanilla-tilt.  [5:54](https://egghead.io/lessons/react-manipulate-the-dom-with-react-refs-cfdb8c6a) Then we return a cleanup function so that we can remove all references of our DOM node in vanilla-tilt and remove all the event listeners that vanilla-tilt put on our DOM node. |
| **19. Понять, как работает React Hook**  **Комент: надо отдельно посвятить пару часов этой теме** | **19. Understand the React Hook Flow** |
| Понимание порядка, в котором вызываются React hooks, может быть действительно полезным для эффективного использования React hooks. [Эта диаграмма](https://github.com/donavon/hook-flow) может быть действительно полезной для понимания этого, и в этом уроке мы исследуем жизненный цикл функционального компонента с помощью хуков с красочными console log statements, чтобы мы знали, когда начинается одна фаза и когда она заканчивается.  Я рекомендую вам посмотреть это медленно и смотреть его столько раз, сколько вам нужно. Также определенно поиграйте с кодом самостоятельно, пока вы не почувствуете себя относительно комфортно. Понимание всего этого **не** критично для вашего успеха в использовании React, и в большинстве случаев вам вообще не нужно об этом думать, но понимание этого может иногда вам помочь. | Understanding the order in which React hooks are called can be really helpful in using React hooks effectively. [This chart](https://github.com/donavon/hook-flow) can be really helpful in understanding this, and in this lesson we’ll explore the lifecycle of a function component with hooks with colorful console log statements so we know when one phase starts and when it ends.  I recommend you watch this one slowly and watch it as many times as you need to. Also definitely play around with the code yourself until you feel relatively comfortable with this. Understanding all of this is **not** critical to your success with using React, and most of the time you won’t need to think about this at all, but understanding it can help you at times. |
| Кент С. Доддс: [0:00](https://egghead.io/lessons/react-understand-the-react-hook-flow) Иногда бывает полезно понять порядок, в котором будет выполняться ваш код, при использовании React Hooks. Я сделал это небольшое приложение, в котором есть флажок для отображения дочернего элемента, и внутри этого поля он будет отображаться, когда этот флажок установлен. Когда вы нажимаете на эту кнопку, счетчик увеличивается.  [0:17](https://egghead.io/lessons/react-understand-the-react-hook-flow) Это работает следующим образом: у нас есть дочерний компонент, который поддерживает countState, а затем здесь есть целая куча useEffects, которые просто регистрируются в консоли при вызове callback и регистрируются в консоли в функция очистки, которую он предоставляет.  [0:32](https://egghead.io/lessons/react-understand-the-react-hook-flow) Затем мы создаем наш элемент React для рендеринга на страницу, а затем регистрируем в консоли, что рендеринг завершен, а затем возвращаем созданный нами элемент React.  [0:42](https://egghead.io/lessons/react-understand-the-react-hook-flow) Мы делаем то же самое для нашего компонента App, за исключением того, что здесь мы поддерживаем showChild Boolean state. У нас есть все эти useEffects, а затем мы визуализируем здесь несколько элементов React для визуализации этого пользовательского интерфейса.  [0:54](https://egghead.io/lessons/react-understand-the-react-hook-flow) Давайте продолжим и посмотрим, что происходит, когда мы изначально загружаем эту страницу. Я собираюсь обновить, я открою наши DevTools, и я прокручу здесь до приложения, чтобы мы могли следовать в коде с тем, что мы видим в консоли.  [1:08](https://egghead.io/lessons/react-understand-the-react-hook-flow) Первое, что мы видим, это начало рендеринга приложения. Это первое, что происходит, когда мы вызываем ReactDom.render в нашем приложении. Он вызывает наше приложение Function Component. Следующее, что происходит, - мы вызываем response.useState, и React немедленно вызывает эту функцию, чтобы получить начальное состояние для нашего дочернего состояния show. Вот почему мы вызываем это приложение useState callback.  [1:31](https://egghead.io/lessons/react-understand-the-react-hook-flow) Затем мы вызываем все эти React useEffects, но вы заметите, что журналы из них - не следующее, что появляется в нашей консоли. Вместо этого мы фактически создаем этот элемент, а затем получаем журнал в консоли для завершения рендеринга приложения. Как только это произойдет, React фактически обновит DOM. Затем, позже асинхронно, он будет вызывать наши useEffect callbacks, по одному в том порядке, в котором они были вызваны.  [1:55](https://egghead.io/lessons/react-understand-the-react-hook-flow) Мы подошли к началу, в основном, приложение «useEffect to no deps» - первое, что появляется. Мы не получаем нашу очистку, потому что в ней еще нет необходимости, потому что сейчас мы просто монтируем компонент, а у нас еще нет обновлений.  [2:08](https://egghead.io/lessons/react-understand-the-react-hook-flow) Далее мы получаем пустое значение deps useEffects, прямо здесь. У нас есть пустой список зависимостей, а затем используйтеEffect с dep. Это состояние show child's state. Это следующее, что здесь вызывают.  [2:21](https://egghead.io/lessons/react-understand-the-react-hook-flow) Теперь давайте посмотрим, что происходит, когда мы кликаем на «show child». Помните, что это последний console log, который мы видели при первоначальном монтировании компонента. Если я нажму «show child», мы начнем рендеринг приложения. Когда мы нажимаем «show child», это срабатывает при изменении, чтобы установить для параметра «Показать дочерний элемент» установленное значение нашего поля флажка.  [2:44](https://egghead.io/lessons/react-understand-the-react-hook-flow) Этот дочерний элемент set show будет запускать повторный рендеринг нашего приложения, поэтому мы запускаем рендеринг нашего приложения. Поднимись сюда снова на вершину. Мы скажем начало рендеринга приложения и пройдемся по всему этому коду так же, как и в предыдущем рендере, за исключением того, что на этот раз вы заметите, что у нас нет обратного вызова состояния использования приложения, мы переходим прямо из приложения от начала рендеринга до конца рендеринга приложения.  [3:06](https://egghead.io/lessons/react-understand-the-react-hook-flow) Это потому, что React уже получил значение начального состояния для нашего дочернего состояния show, и ему не нужно снова получать это значение. Каждый раз, когда вы используете обратный вызов функции для состояния использования, эта функция будет вызываться только тогда, когда этот компонент изначально визуализируется в течение оставшейся части времени существования этого компонента.  [3:24](https://egghead.io/lessons/react-understand-the-react-hook-flow) Мы снова проходим через все эти причины useEffect, мы создаем наш элемент, а затем фиксируем на консоли, что рендеринг приложения завершился. Затем React вызывает нашего дочернего элемента, чтобы начать рендеринг этого ребенка. Я хочу здесь подчеркнуть одну вещь: мы создаем наш элемент, который включает создание дочернего элемента прямо здесь.  [3:42](https://egghead.io/lessons/react-understand-the-react-hook-flow) Вы заметите, что мы переходим к этой строке кода до того, как начнем рендеринг дочернего элемента. Здесь важно помнить, что то, что вы создаете элемент React, не означает, что функция элемента React будет вызвана, потому что вы не вызываете функцию, а React.  [3:58](https://egghead.io/lessons/react-understand-the-react-hook-flow) React вызовет эту функцию только тогда, когда этот компонент будет фактически отрисован. Это действительно важно, потому что вы могли бы сказать, что Const UI я не буду рендерить, и рендеринг целую кучу из них весь день, и если эти вещи не будут добавлены в какой-то UI, который действительно будет рендериться, все, что вы создаете, - это куча объектов элемента React, и вы фактически не будете вызывать эту дочернюю функцию.  [4:25](https://egghead.io/lessons/react-understand-the-react-hook-flow) Давайте продолжим и прокрутим до этого дочернего рендера. Начните здесь, а затем мы получим обратный вызов нашего состояния использования, потому что это первый раз, когда это состояние использования будет вызываться, поэтому ему необходимо получить начальное значение, равное нулю.  [4:38](https://egghead.io/lessons/react-understand-the-react-hook-flow) Затем мы вызываем все эти useEffects, точно так же, как в приложении, мы создаем элемент, а затем получаем журнал для этого конца рендеринга. Затем, после обновления всей DOM, React начнет вызывать наш useEffects. Затем он вызывает в том порядке, в котором они вызываются, но начиная с дочернего компонента.  [4:56](https://egghead.io/lessons/react-understand-the-react-hook-flow) Мы получаем дочерний useEffect без вызова deps, мы получаем дочерний useEffect, вызывается пустой deps, а затем мы получаем дочерний эффект с вызовом dep, а здесь dep - это значение нашего счетчика. Затем мы начнем вызывать обратные вызовы useEffect приложения.  [5:14](https://egghead.io/lessons/react-understand-the-react-hook-flow) Вы заметите, что на самом деле мы сначала вызываем очистку, а затем - настройку. Здесь у нас есть очистка, а затем настройка. Мы также делаем то же самое для очистки, когда у нас есть зависимость, и затем для настройки, когда у нас есть зависимость. Вы заметите, что очистка для обоих из них вызывается перед настройкой для обоих, и обе очистки вызываются в том порядке, в котором они появляются, точно так же, как настройки.  [5:41](https://egghead.io/lessons/react-understand-the-react-hook-flow) Вы также заметите, что этот обратный вызов useEffect и его очистка не вызывались. Это потому, что обратные вызовы useEffect вызываются только в том случае, если у них нет перечисленных зависимостей или если у них есть указанная зависимость, и одна из этих зависимостей изменяется.  [5:55](https://egghead.io/lessons/react-understand-the-react-hook-flow) У нас есть список зависимостей, но он пуст, поэтому никакие зависимости не меняются, потому что у него нет никаких зависимостей. Этот обратный вызов useEffect и его очистка не будут вызываться при обновлениях.  [6:07](https://egghead.io/lessons/react-understand-the-react-hook-flow) Давайте продолжим и нажмем на эту кнопку. Мы будем помнить, что это последний журнал, который мы видели для последнего обновления. Когда я нажимаю на эту кнопку, мы вызываем setCount и предоставляем функцию обновления функции, в которой мы берем предыдущее значение состояния и возвращаем новое значение нашего состояния.  [6:24](https://egghead.io/lessons/react-understand-the-react-hook-flow) Здесь мы просто возьмем предыдущий счет и прибавим к нему 1. Это вызовет повторную визуализацию. Давайте посмотрим, что здесь происходит. Я нажимаю на это. Вы заметите, что приложение не вызывает ни одного из журналов. Это потому, что обновление состояния на самом деле находится только в этом компоненте, поэтому React знает, что этот компонент - единственный, который нужно повторно отрисовать.  [6:45](https://egghead.io/lessons/react-understand-the-react-hook-flow) Следуем за console.logs. Сначала мы начнем с этого дочернего начала рендеринга. Мы вызываем это снова, и мы не получаем журнал для нашего обратного вызова useState, потому что этот компонент уже был отрисован, и мы уже получили начальное значение, и нам больше не нужно это начальное значение, поэтому React не беспокоится о вызове этой функции больше.  [7:01](https://egghead.io/lessons/react-understand-the-react-hook-flow) Затем мы вызываем все эти хуки useEffect. Мы создаем элемент React для нашего пользовательского интерфейса, а затем вызываем этот дочерний элемент рендеринга. Затем вызываются наши очистки useEffect, если нужно повторно запустить эти конкретные useEffect. В этом случае мы не указали никаких зависимостей, поэтому это будет повторяться при каждом рендеринге этого компонента, поэтому мы проведем очистку там.  [7:25](https://egghead.io/lessons/react-understand-the-react-hook-flow) Это перечисляет массив зависимостей, и нет никаких зависимостей, которые могут измениться, поэтому он не будет вызываться. У этого есть массив зависимостей, и эта зависимость изменилась, поэтому здесь вызывается очистка.  [7:39](https://egghead.io/lessons/react-understand-the-react-hook-flow) Затем мы начинаем с настроек для useEffect, у которого нет зависимостей, и для useEffect, у которого есть измененная зависимость. Это последний, который вызвали. Если мы щелкнем по этому еще раз, мы увидим точно такой же порядок вызовов, а затем, если мы снимем флажок показывать дочерний элемент ... Давайте продолжим и выделим его, чтобы мы знали, какой из них был последним, который мы вызывали, а затем снимаем флажок показать ребенку.  [8:04](https://egghead.io/lessons/react-understand-the-react-hook-flow) Это вызовет повторный рендеринг нашего компонента приложения, потому что мы изменили этот showChildState с помощью setShowChild. Здесь начинается рендеринг. Мы проходим весь этот код, снова пропуская обратный вызов useState.  [8:21](https://egghead.io/lessons/react-understand-the-react-hook-flow) Мы спускаемся сюда, создаем наши элементы, а затем рендерим конец прямо здесь. Затем мы выполняем очистку всех дочерних элементов, поскольку дочерний элемент удаляется со страницы, поэтому теперь мы рендерим null.  [8:34](https://egghead.io/lessons/react-understand-the-react-hook-flow) React замечает, что предыдущий JSX, который вы мне предоставили, включал дочерний элемент, а следующий JSX, который вы мне предоставили, не включает ребенка. Это означает, что мне нужно удалить дочерний элемент со страницы, поэтому я собираюсь размонтировать компонент и вызвать все очистки для всех useEffects, которые были у этого дочернего элемента.  [8:50](https://egghead.io/lessons/react-understand-the-react-hook-flow) Давайте [подойдем](https://egghead.io/lessons/react-understand-the-react-hook-flow) сюда и увидим, что это происходит в том порядке, в котором они появляются в коде. Мы видим, что у нас нет очистки от депс. Мы увидим наш запуск очистки пустых хранилищ. Несмотря на то, что мы перечислили зависимости, но ни одна из них не изменилась, потому что у нас нет зависимостей, наша очистка будет запущена, потому что мы размонтируем этот компонент, а затем мы получим очистку эффекта с зависимостями.  [9:13](https://egghead.io/lessons/react-understand-the-react-hook-flow) Даже если это значение счетчика не изменилось, оно будет вызвано, потому что этот компонент отключается.  [9:18](https://egghead.io/lessons/react-understand-the-react-hook-flow) Затем, поскольку наше состояние изменилось, в приложении мы собираемся выполнить некоторые очистки для релевантных useEffects. Здесь вызывается эта очистка, потому что она не имеет зависимостей. Эта очистка не вызывается, потому что у нее пустой массив зависимостей. Затем вызывается эта очистка, потому что изменилась зависимость.  [9:40](https://egghead.io/lessons/react-understand-the-react-hook-flow) Затем мы запускаем релевантные эффекты: useEffect без зависимостей и useEffect с зависимостью, которая изменилась.  [9:50](https://egghead.io/lessons/react-understand-the-react-hook-flow) Я рекомендую вам поиграть с этим примером кода. Также вам может помочь диаграмма от Донована. Вы переходите к donovan⁄hookflow на GitHub и прокручиваете его вниз, и вы видите диаграмму, созданную Донованом.  [10:05](https://egghead.io/lessons/react-understand-the-react-hook-flow#t=605) Получаем ленивые инициализаторы. Это наш обратный вызов useState. Мы получаем нашу функцию рендеринга, которая завершается после вызова всех ленивых инициализаторов. React обновит DOM.  [10:16 В](https://egghead.io/lessons/react-understand-the-react-hook-flow#t=616) React также есть хук useLayoutEffect, который будет запускаться здесь. Это работает очень похоже на useEffect. Браузер обновляет экран на основе обновлений DOM, сделанных React, а затем будут вызваны наши эффекты. Это именно то, что мы наблюдали в нашем примере.  [10:33](https://egghead.io/lessons/react-understand-the-react-hook-flow#t=633) Когда происходит обновление состояния, мы вызываем наш рендер, наши ленивые инициализаторы не вызываются. React обновит DOM, мы очистим эффекты макета, а затем наши эффекты макета будут запущены. Браузер обновляет экран, затем мы очищаем наши эффекты, а затем наши эффекты запускаются. Когда компонент размонтирован, мы очищаем все наши эффекты.  [10:54](https://egghead.io/lessons/react-understand-the-react-hook-flow#t=654) Для вашей эффективности с React совсем не обязательно иметь четкое представление о порядке их вызова. Это может помочь вам в некоторых ситуациях, поэтому я рекомендую вам поиграть с этим. | Kent C. Dodds: [0:00](https://egghead.io/lessons/react-understand-the-react-hook-flow) Sometimes it could be useful to understand the order in which your code is going to be run when you're using React Hooks. I've made this little app that has a checkbox for showing a child, and inside of this box this will be rendered when that checkbox is checked. When you click on this button, it's going to increment the count.  [0:17](https://egghead.io/lessons/react-understand-the-react-hook-flow) The way that this works is we have a child component here that's maintaining a countState, and then that has a whole bunch of useEffects here, which are simply logging to the console when the callback is called and logging to the console in a cleanup function that it provides.  [0:32](https://egghead.io/lessons/react-understand-the-react-hook-flow) Then we create our React element for rendering to the page, and then we log to the console that our render is finished and then we return that React element we created.  [0:42](https://egghead.io/lessons/react-understand-the-react-hook-flow) We do the same thing for our App component, except here we're maintaining a showChild Boolean state. We have all those useEffects, and then we render a few React elements here for rendering this UI.  [0:54](https://egghead.io/lessons/react-understand-the-react-hook-flow) Let's go ahead and take a look at what happens when we initially load this page. I'm going to refresh, I'll open up our DevTools, and I'll scroll down here to the app, so we can follow along in the code with what we're seeing in the console.  [1:08](https://egghead.io/lessons/react-understand-the-react-hook-flow) The first thing that we see is this app render start. That's the first thing that happens when we call ReactDom.render our app. It calls our app Function Component. The next thing that happens is we call react.useState and immediately React is going to call this function to retrieve the initial state for our show child state. That's why we're getting this app useState callback called.  [1:31](https://egghead.io/lessons/react-understand-the-react-hook-flow) Then, we call all these React useEffects, but you'll notice that the logs in those are not the next thing that appear in our console. Instead, we actually create this element and then we get a log to the console for app render end. Once that happens, React actually is updating the DOM. Then, asynchronously later, it's going to call our useEffect callbacks, one at a time in the order in which they were called.  [1:55](https://egghead.io/lessons/react-understand-the-react-hook-flow) We come up here to the top, mostly, app useEffect to no deps is the first one that appears. We don't get our cleanup because there's no cleanup necessary yet, because right now, we're just mounting the component, and we haven't had any updates yet.  [2:08](https://egghead.io/lessons/react-understand-the-react-hook-flow) Next, we get our useEffects empty deps, right here. We have an empty list of dependencies there and then useEffect with dep. That's our show child's state. That's the next thing that gets called here.  [2:21](https://egghead.io/lessons/react-understand-the-react-hook-flow) Now, let's take a look at what happens when we click on show child. Remember that this is the last console log that we saw when we initially mounted the component. If I click show child, then we're going to get an app render start. When we click show child, that triggers this on change to set show child to the checked value of our check box input.  [2:44](https://egghead.io/lessons/react-understand-the-react-hook-flow) This set show child is going to trigger a re-render of our app, which is why we get our app render start. Come up here to the top again. We'll say app render start, and we'll go through all of this code just like we had at the previous render, except this time you'll notice we don't have an app use state callback, we go straight from app render start to app render end.  [3:06](https://egghead.io/lessons/react-understand-the-react-hook-flow) This is because React has already retrieved the initial state value for our show child state, and it doesn't need to retrieve that value again. Any time you use a function call back for use state, that function is only going to be called when this component is initially rendered for the rest of the lifetime of that component.  [3:24](https://egghead.io/lessons/react-understand-the-react-hook-flow) We go through all these useEffect cause again, we create our element, and then we lock to the console that the app render has finished. Then React calls our child to start rendering of that child. One thing that I want to stress here is that we're creating our element which includes creating the child right here.  [3:42](https://egghead.io/lessons/react-understand-the-react-hook-flow) You'll notice that we get to this line of code before we start rendering the child. The important thing to remember here is that just because you create a React element, doesn't mean that React element's function is going to get called, because you're not calling the function, React is.  [3:58](https://egghead.io/lessons/react-understand-the-react-hook-flow) React will only call that function when that component is actually going to be rendered. That's really important because you could say Const UI I will not render, and render a whole bunch of these all day long, and unless those things are added to some UI that's actually going to get rendered, all that you're creating is a bunch of React element objects, and you won't actually be calling into that child function.  [4:25](https://egghead.io/lessons/react-understand-the-react-hook-flow) Let's go ahead and scroll up to this child render. Start here, and then we get our use state call back, because this is the first time that this use state is going to be called, so it needs to retrieve the initial value of zero.  [4:38](https://egghead.io/lessons/react-understand-the-react-hook-flow) Then we call all of these useEffects, just like in an app, we create an element, and then we get a log for this render end. Then after the entire DOM has been updated, React is going to start calling our useEffects. It calls then in the order in which they are called, but starting at the child component.  [4:56](https://egghead.io/lessons/react-understand-the-react-hook-flow) We get child useEffect no deps is called, we get the child useEffect empty deps is called, and then we get the child effect with dep is called, and the dep here is our count value. Then we're going to start calling the app useEffect callbacks.  [5:14](https://egghead.io/lessons/react-understand-the-react-hook-flow) You'll notice that we actually call the cleanup first, and then we call the setup. Here we have the cleanup and then the setup. We also do the same for the cleanup when we have a dependency, and then the setup when we have a dependency. You'll notice that the cleanup for both of these is called before the setup for both of these, and the cleanups are both called in the order in which the appear, just like the setups.  [5:41](https://egghead.io/lessons/react-understand-the-react-hook-flow) You'll also notice that this useEffect callback and its cleanup were not called. That's because useEffect callbacks are only called if they have no dependency listed or if they have a dependency listed, and one of those dependencies is changed.  [5:55](https://egghead.io/lessons/react-understand-the-react-hook-flow) We have a dependency list, but it's empty, and so therefore, none of the dependency is changed, because it doesn't have any dependencies. This useEffect callback and its cleanup will not be called on updates.  [6:07](https://egghead.io/lessons/react-understand-the-react-hook-flow) Let's go ahead and click on this button. We'll remember that this is the last log that we saw for that last update. When I click on that button, we're going to call setCount, and we're providing a function updater function where we take the previous value of the state and return the new value of our state.  [6:24](https://egghead.io/lessons/react-understand-the-react-hook-flow) Here, we're just going to take the previous count and add 1 to it. This is going to trigger a re-render. Let's take a look at what happens here. I click on that. You'll notice that the app has none of its logs called. That's because the state update actually only resides within this component, so React knows that this component is the only one that needs to be re-rendered.  [6:45](https://egghead.io/lessons/react-understand-the-react-hook-flow) Let's follow the console.logs. First we start with this child render start. We call this again, and we don't get a log for our useState callback, because this component's already been rendered, and we already retrieved the initial value and we no longer need that initial value, so React doesn't bother calling this function anymore.  [7:01](https://egghead.io/lessons/react-understand-the-react-hook-flow) Then we call all these useEffect hooks. We create our React element for our UI, and then we call this child render end. Then our useEffect cleanups are called in order if those particular useEffects need to be rerun. In this case, we haven't listed any dependencies, so this will be rerun on every render of this component, so we'll get the cleanup there.  [7:25](https://egghead.io/lessons/react-understand-the-react-hook-flow) This one is listing a dependency array, and there are no dependencies that can change, so it's not going to be called. This one does have a dependency array, and that dependency did change, so the cleanup is called here.  [7:39](https://egghead.io/lessons/react-understand-the-react-hook-flow) Then we start with the setups for the useEffect that has no dependencies and the useEffect that has a dependency that changed. This is the last one that was called. If we click this again, we'll see the exact same order of calls, and then if we uncheck show child...Let's go ahead and highlight that, so we know which one was the last that we called, and then we uncheck show child.  [8:04](https://egghead.io/lessons/react-understand-the-react-hook-flow) That's going to trigger a re-render of our app component, because we changed that showChildState using setShowChild. That triggers a render start here. We go through all of this code, again skipping the useState callback.  [8:21](https://egghead.io/lessons/react-understand-the-react-hook-flow) We come down here, create our elements, and then render end right here. Then we're doing a cleanup on all of the children, because the child is being removed from the page, so we're now rendering null.  [8:34](https://egghead.io/lessons/react-understand-the-react-hook-flow) React notices the previous JSX that you gave me included the child, and this next JSX that you gave me does not include the child. That means I need to remove the child from the page, and so I'm going to unmount the component and call all of the cleanups for all of the useEffects that that child had going.  [8:50](https://egghead.io/lessons/react-understand-the-react-hook-flow) Let's come up here, and we'll see that that happens in the order in which they appear in the code. We see our no deps cleanup runs. We'll see our empty deps cleanup run. Even though we have listed dependencies but none of them changed because we have no dependencies, our cleanup is going to run, because we're unmounting this component, and then we get a cleanup of the effect with dependencies.  [9:13](https://egghead.io/lessons/react-understand-the-react-hook-flow) Even though this count value didn't change, it's going to be called, because this component is getting unmounted.  [9:18](https://egghead.io/lessons/react-understand-the-react-hook-flow) Then because our state changed, in the app, we're going to run some cleanups for the useEffects that are relevant. Here, we have this cleanup is getting called, because it has no dependencies. This cleanup is not getting called, because it has an empty dependency array. Then this cleanup is getting called, because the dependency that it has has changed.  [9:40](https://egghead.io/lessons/react-understand-the-react-hook-flow) Then we go ahead and run the effects that are relevant, the useEffect with no dependencies and the useEffect with a dependency that changed.  [9:50](https://egghead.io/lessons/react-understand-the-react-hook-flow) I encourage you to play around with this code example. Also, something else that might help you is this diagram from Donovan. You go to donovan⁄hookflow on GitHub and scroll down here and you'll see the diagram that Donovan has created.  [10:05](https://egghead.io/lessons/react-understand-the-react-hook-flow#t=605) We get our lazy initializers. That's our useState callback. We get our render function that finishes after the lazy initializers are all called. React will update the DOM.  [10:16](https://egghead.io/lessons/react-understand-the-react-hook-flow#t=616) React also has a useLayoutEffect hook that will be run here. That operates pretty similarly to the useEffect. The browser updates the screen based on the updates to the DOM that React made and then our effects are going to be called. This is exactly what we observed in our example.  [10:33](https://egghead.io/lessons/react-understand-the-react-hook-flow#t=633) When there's a state update, we get our render called, our lazy initializers are not called. React is going to update the DOM, we get a cleanup of layout effects and then our layout effects are run. Browser updates the screen and then we get a cleanup of our effects and then our effects are run. When the component is unmounted, we get the cleanup of all of our effects.  [10:54](https://egghead.io/lessons/react-understand-the-react-hook-flow#t=654) Having a firm understanding on the order in which these things are called is not totally necessary for you to be effective with React. It can help you in some situations, so I encourage you to play around with this. |
| **20. Создавайтебазовые формы с помощью React** | **20. Make Basic Forms with React** |
| Формы - это основной строительный блок Интернета. Каждое веб-приложение использует элементы формы как способ принимать ввод от пользователя. При работе с формами в Интернете следует помнить о нескольких вещах, и в этом уроке мы узнаем об этом, а также о различных способах извлечения значений из элементов формы, а также о нескольких передовых методах, которые следует учитывать. при работе с элементами формы в сети.  Вы можете узнать больше об основных формах в документации React о [неконтролируемых компонентах](https://reactjs.org/docs/uncontrolled-components.html) . | Forms are a basic building block of the web. Every web application uses form elements as a way to accept input from the user. There are a few things to keep in mind with how forms work on the web and in this lesson we’ll learn about those as well as various ways you can retrieve values from elements in the form as well as a few best practices you should consider when working with form elements on the web.  You can learn more about basic forms in the React documentation about [Uncontrolled Components](https://reactjs.org/docs/uncontrolled-components.html). |
| Кент С. Доддс: [0:00](https://egghead.io/lessons/react-make-basic-forms-with-react-d4e00d42) У нас есть эта форма имени пользователя, и сейчас мы просто возвращаемся к todo, но мы хотим вернуть форму. Я собираюсь создать здесь элемент формы, а затем мы создадим div для хранения нашего имени пользователя метки, а затем у нас будет ввод текста типа для ввода нашего имени пользователя, а затем мы добавим кнопку введите submit. Это наша кнопка отправки, и мы скажем отправить. Мы сохраним это, получим обновление, а вот наша форма. Потрясающе.  [0:33](https://egghead.io/lessons/react-make-basic-forms-with-react-d4e00d42) Мы могли бы поместить здесь обработчик onClick на нашу кнопку отправки, и это будет нормально работать, когда мы нажмем на эту кнопку. Формы фактически автоматически отправляются, когда вы нажимаете клавишу ввода в поле ввода в форме. Люди обычно привыкли к такому пользовательскому опыту.  [0:48](https://egghead.io/lessons/react-make-basic-forms-with-react-d4e00d42) Вместо этого я собираюсь добавить сюда обработчик onSubmit, и если кто-то щелкнет по кнопке отправки или нажмет Enter в этом поле ввода, то будет вызван наш обработчик onSubmit.  [0:59](https://egghead.io/lessons/react-make-basic-forms-with-react-d4e00d42) Создадим функцию с именем handleSubmit. Затем мы поместим этот handleSubmit прямо сюда и отправим консольный журнал. Давайте сохраним это, refresh, скажем, Джо, нажми "Отправить" и ... Ага. Это интересно, это уходит.  [1:18](https://egghead.io/lessons/react-make-basic-forms-with-react-d4e00d42) На самом деле, если вы посмотрите внимательно, вы увидите, что он записывается, но затем мы получаем полное обновление страницы. Мы можем доказать это, включив сохранение журнала. Мы снова говорим «Джо», нажимаем «Отправить», мы отправляемся и переходим на эту страницу.  [1:32](https://egghead.io/lessons/react-make-basic-forms-with-react-d4e00d42) Причина, по которой это происходит, заключается в том, что когда вы отправляете форму в браузере, он автоматически отправляет запрос на публикацию по текущему URL-адресу с данными формы. Мы могли убедиться в этом, если взглянули на нашу вкладку сети.  [1:46](https://egghead.io/lessons/react-make-basic-forms-with-react-d4e00d42) Если вы хотите сделать что-то с JavaScript на основе формы, отправленной пользователем, нам не нужно полное обновление страницы. Вместо этого мы можем делать наш собственный JavaScript.  [1:54](https://egghead.io/lessons/react-make-basic-forms-with-react-d4e00d42) Функция handleSubmit как обработчик событий примет событие в качестве аргумента. Тогда мы можем сказать event.preventDefault. При этом у нас больше не будет полного обновления страницы, и мы по-прежнему будем регистрировать отправленную строку в консоли.  [2:09](https://egghead.io/lessons/react-make-basic-forms-with-react-d4e00d42) Наш следующий шаг - получить значение из этого ввода, потому что мы хотим to alert the value, введенное пользователем. Нам нужно, чтобы наше имя пользователя было равно чему-то, и мы скажем предупреждение: «Вы ввели имя пользователя». Здесь мы можем избавиться от журнала консоли. Можно сказать Джо и вы ввели вопросительный знак.  [2:31](https://egghead.io/lessons/react-make-basic-forms-with-react-d4e00d42) Как мы получим это значение имени пользователя? Что ж, один из способов сделать это - сказать document.querySelectorInput.value, сохрани это. Мы говорим Joe, submit that, и мы получаем Джо. Это не сработает. Это не будет очень хорошо масштабироваться в реальном мире, потому что страница может иметь несколько входов.  [2:52](https://egghead.io/lessons/react-make-basic-forms-with-react-d4e00d42) Если бы мы отобразили их несколько, то они могли бы конфликтовать друг с другом. Запрос всего документа нарушает инкапсуляцию этого компонента, потому что это означает, что вы не можете использовать этот компонент в контексте других компонентов в приложении. Мы не хотим запрашивать документ.  [3:09](https://egghead.io/lessons/react-make-basic-forms-with-react-d4e00d42) Еще мы можем получить доступ к элементу формы из события. Мы могли бы сказать console.logEvent.target, и это будет наш элемент формы. Мы говорим Joe, submit that, и мы сразу же получим элемент формы.  [3:25](https://egghead.io/lessons/react-make-basic-forms-with-react-d4e00d42) Это здорово, но давайте взглянем на некоторые свойства этого элемента формы. Chrome отображает это действительно крутое дерево DOM, подобное тому, что мы видим на вкладке элементов, но я хочу увидеть свойства, которые находятся в элементах формы. Я собираюсь использовать console.dir, который выведет из системы свойства этих элементов.  [3:41](https://egghead.io/lessons/react-make-basic-forms-with-react-d4e00d42) Мы скажем, Joe, submit, и вот, у нас есть форма. Здесь есть множество свойств. Мы видим, что для ввода и кнопки есть ноль и один, поэтому мы можем сказать, что имя пользователя - event.target. Нам нужно нулевое значение, мы сохраняем это, получаем Джо и получаем сообщение «Вы вошли Джо».  [4:04](https://egghead.io/lessons/react-make-basic-forms-with-react-d4e00d42) Это прекрасно работает, но я собираюсь получить имя пользователя из другого из этих свойств. Посмотрим, у нас прямо здесь есть элементы, а у elements есть ноль и единица. Мы могли бы сказать event.target.elements.zero.value и сохранить это. Это тоже сработает.  [4:24](https://egghead.io/lessons/react-make-basic-forms-with-react-d4e00d42) Я не в восторге от того, что неявно полагаюсь на порядок, в котором появляются эти элементы формы, потому что если бы у меня был другой ввод прямо здесь, я бы сказал: «Вы ввели undefined. Я не хочу на это положиться ".  [4:39](https://egghead.io/lessons/react-make-basic-forms-with-react-d4e00d42) Есть еще одна вещь, которую мы можем сделать здесь, это правильно связать нашу метку с вводом, имея HTML для ввода имени пользователя. Затем, имея ID ввода имени пользователя прямо здесь. Теперь метка и ввод правильно связаны. Это означает, что я могу щелкнуть метку, и она сосредоточится на вводе, что хорошо для доступности.  [5:03](https://egghead.io/lessons/react-make-basic-forms-with-react-d4e00d42) Когда я говорю, что здесь "Джо", нажмите "Отправить", и мы смотрим на нашу форму в свойстве elements. В дополнение к индексам нуля и единицы для каждого из этих элементов, мы также получаем здесь входное значение имени пользователя, потому что идентификатор этого элемента внутри нашей формы. Фактически вы получаете то же самое, используя атрибут name.  [5:29](https://egghead.io/lessons/react-make-basic-forms-with-react-d4e00d42) Мы смотрим на элементы формы и видим, что ввод имени пользователя все еще существует.  [5:34](https://egghead.io/lessons/react-make-basic-forms-with-react-d4e00d42) Любые элементы, у которых есть имя или идентификатор, будут присоединены с этим именем и идентификатором к свойству этого элемента узла формы. Вы можете ссылаться на них по этим именам и идентификаторам. Это то, что мы предпочтем здесь. Мы скажем username = event.target - это наша форма - .elements.usernameInput.value, чтобы получить значение из этого ввода имени пользователя.  [5:58 Сказав](https://egghead.io/lessons/react-make-basic-forms-with-react-d4e00d42) это, мы можем сказать «Джо», и все будет работать так, как мы хотим, без неявных запросов ко всему документу или зависимости от порядка элементов.  [6:09](https://egghead.io/lessons/react-make-basic-forms-with-react-d4e00d42) Еще мы могли бы сделать ссылку для ввода имени пользователя. Мы скажем usernameInputRef = React.useRef, а затем мы могли бы сказать ref = usernameInputRef, а вместо этого мы могли бы сказать username = usernameInputRef.current.value. Наш usernameInputRef.current будет узлом DOM для ввода, и мы получим от него значение. Это тоже сработает.  [6:43](https://egghead.io/lessons/react-make-basic-forms-with-react-d4e00d42) Есть несколько способов сделать это, и я добавлю username = document.querySelector input.value. Причина, по которой я удалил это, в том, что я действительно хочу, чтобы вы не использовали это. Это нарушает инкапсуляцию наших компонентов. Одна из вещей, которые нам действительно нравятся в React, - это то, что мы можем инкапсулировать логику в наших компонентах.  [7:05](https://egghead.io/lessons/react-make-basic-forms-with-react-d4e00d42) Я не рекомендую этот метод или этот метод, потому что они неявно зависят от порядка, в котором отображаются элементы, и могут легко сломаться, если вы измените этот порядок. Любой из них будет работать нормально.  [7:16](https://egghead.io/lessons/react-make-basic-forms-with-react-d4e00d42) Лично я думаю, что если вам не нужна ссылка для ввода, то будет проще получить ввод из целевых элементов события. В любом случае рекомендуется использовать HTML для идентификатора, чтобы связывать метки с входными данными. Учитывая, что вам уже нужно это сделать, я думаю, что это самый простой способ получить значение из входных данных в вашей форме.  [7:36](https://egghead.io/lessons/react-make-basic-forms-with-react-d4e00d42) Имея это в виду, давайте продолжим и избавимся от всего этого. Мы избавляемся от этого, очищаем это и удаляем исх. Если бы я писал эту форму сегодня, она бы выглядела так.  [7:46](https://egghead.io/lessons/react-make-basic-forms-with-react-d4e00d42) В заключение, мы должны были создать форму, которая имеет единственный ввод и метку, связанную с этим вводом, и кнопку с типом отправки. Очень важно, чтобы любая кнопка, которую вы помещаете внутри формы, имела тип, потому что неявно у нее будет тип отправки. Это не совсем понятно, особенно если вы скажете отменить для содержимого без определенного типа, который дает этой кнопке любой тип отправки, что действительно сбивает с толку.  [8:12](https://egghead.io/lessons/react-make-basic-forms-with-react-d4e00d42) Если вы действительно хотите иметь кнопку отмены или кнопку сброса, вы хотите указать тип кнопки, который, как я знаю, немного избыточен, но если вы не укажете тип, то тип будет неявно be submit, что действительно сбивает с толку ваших пользователей, когда они попытаются нажать кнопку "Отмена", поэтому мы указываем, что это тип submit. Мы указываем здесь значение submit.  [8:34](https://egghead.io/lessons/react-make-basic-forms-with-react-d4e00d42) Затем вместо того, чтобы помещать обработчик onClick на кнопку, мы помещаем обработчик onSubmit в форму. Таким образом, любой другой способ, которым пользователь пытается отправить форму, вызовет наш обработчик отправки.  [8:44](https://egghead.io/lessons/react-make-basic-forms-with-react-d4e00d42) Затем, чтобы избежать полного обновления страницы, мы используем event.preventDefault, а затем извлекаем ввод пользователя, используя цель события, чтобы получить форму, а затем элементы, чтобы получить элементы формы, и usernameInput, чтобы получить ввод по его ID. Затем мы получаем значение из этого ввода, чтобы получить наше имя пользователя, и затем мы предупреждаем об этом наших пользователей, или вы можете отправить это на внутренний сервер. | Kent C. Dodds: [0:00](https://egghead.io/lessons/react-make-basic-forms-with-react-d4e00d42) We have this username form, and right now we're just returning todo, but we want to return a form. I'm going to make a form element here, and then we'll make a div to hold our label username, and then we'll have an input of type text for our username input, and then we'll add a button of type submit. This is our submit button, and we'll say submit. We'll save that, we'll get a refresh, and here's our form. Awesome.  [0:33](https://egghead.io/lessons/react-make-basic-forms-with-react-d4e00d42) We could put an onClick handler on our submit button here, and that would work OK when we click on this button. Forms are actually automatically submitted when you hit enter in an input within the form. People are typically used to that user experience.  [0:48](https://egghead.io/lessons/react-make-basic-forms-with-react-d4e00d42) What I'm going to do instead is add an onSubmit handler here, and if somebody clicks on the submit button, or if they hit enter in this input, then our onSubmit handler is going to be called.  [0:59](https://egghead.io/lessons/react-make-basic-forms-with-react-d4e00d42) Let's make a function called handleSubmit. Then, we'll put that handleSubmit right in here and we'll console log submitted. Let's save that, get a refresh, say, Joe, hit submit and...Huh. That's interesting, it goes away.  [1:18](https://egghead.io/lessons/react-make-basic-forms-with-react-d4e00d42) Actually, if you watch closely, you'll see that it does log but then we get a full page refresh. We can prove that by turning on preserve log. We say Joe again, hit submit, we get submitted and then navigated to that page.  [1:32](https://egghead.io/lessons/react-make-basic-forms-with-react-d4e00d42) Now, the reason that this is happening is because when you submit a form in the browser, it automatically makes a post request to the current URL with the form data. We could see that if we looked at our network tab.  [1:46](https://egghead.io/lessons/react-make-basic-forms-with-react-d4e00d42) If you want to do some JavaScript stuff based on the user's submitted form, then we don't want the full page refresh. Instead, we can do our own JavaScript stuff.  [1:54](https://egghead.io/lessons/react-make-basic-forms-with-react-d4e00d42) The handleSubmit function as an event handler will accept an event as the argument. Then we can say event.preventDefault. With that, we will no longer get a full page refresh and we'll still get the submitted string logged to the console.  [2:09](https://egghead.io/lessons/react-make-basic-forms-with-react-d4e00d42) Our next step is to get the value out of this input because we want to alert the value that the user entered. We need to get our username equals something and we'll say alert, "You entered username." We can get rid of that console log here. We can say Joe and you entered question mark.  [2:31](https://egghead.io/lessons/react-make-basic-forms-with-react-d4e00d42) How are we going to get that username value? Well, one way we could do this, we could say document.querySelectorInput.value, save that. We say Joe, submit that and here we go, we get Joe. That's not going to work very well. It won't scale super well in the real world because the page could have multiple inputs.  [2:52](https://egghead.io/lessons/react-make-basic-forms-with-react-d4e00d42) If we were to render multiples of these, then they could conflict with one another. Querying the entire document breaks the encapsulation of this component because it means that you can't use this component in the context of other components in an application. We don't want to query the document.  [3:09](https://egghead.io/lessons/react-make-basic-forms-with-react-d4e00d42) Another thing we can do is we can access the form element from the event. We could say console.logEvent.target and that is going to be our form element. We say Joe, submit that and we'll get the form element right there.  [3:25](https://egghead.io/lessons/react-make-basic-forms-with-react-d4e00d42) That's great, but let's take a look at some of the properties on that form element. Chrome is rendering out this really cool DOM tree similar to what we see in the elements tab, but I want to see the properties that are on the form elements. I'm going to use console.dir, which will log out the properties of that elements.  [3:41](https://egghead.io/lessons/react-make-basic-forms-with-react-d4e00d42) We'll say Joe, submit and here we go, we have our form. Here, there are a bunch of properties on here. We can see there's a zero and a one for input and button, so we could actually say username is event.target. We want zero.value, we save this and we get Joe and we get our, "You entered Joe," alert showing up.  [4:04](https://egghead.io/lessons/react-make-basic-forms-with-react-d4e00d42) That works out nicely, but I'm going to get the username from another one of these properties. Let's see, we've got elements right here and elements has a zero and one. We could say event.target.elements.zero.value and save that. That will work as well.  [4:24](https://egghead.io/lessons/react-make-basic-forms-with-react-d4e00d42) I'm not super jazzed about relying implicitly on the order in which these form elements appear because if I were to had another input right here, then I'm going to be toast, "You entered undefined. I don't want to rely on that."  [4:39](https://egghead.io/lessons/react-make-basic-forms-with-react-d4e00d42) There's another thing that we can do here and that is by properly associating our label to the input by having an HTML for username input. Then, having an ID of username input right here. Now the label and the input are properly associated. Meaning that I can click on the label and it will focus on the input, which is good for accessibility.  [5:03](https://egghead.io/lessons/react-make-basic-forms-with-react-d4e00d42) When I say Joe in here, hit submit, then we look at our form in our elements property. In addition to the zero and one indexes for each of these elements, we also get this username input value here because the ID of this element inside of our form. You actually get the same thing using the name attribute as well.  [5:29](https://egghead.io/lessons/react-make-basic-forms-with-react-d4e00d42) We look at our form elements, and we see that username input still exists right there.  [5:34](https://egghead.io/lessons/react-make-basic-forms-with-react-d4e00d42) Any elements that have a name or an ID will be attached by that name and ID to this element's property of the form node. You can reference those by those names and IDs. That's what we're going to prefer here. We'll say username = event.target -- that's our form -- .elements.usernameInput.value to get the value out of that username input.  [5:58](https://egghead.io/lessons/react-make-basic-forms-with-react-d4e00d42) With that, we can say Joe and we get everything working exactly as we want without the implicitness of querying the entire document or relying on the order of the elements.  [6:09](https://egghead.io/lessons/react-make-basic-forms-with-react-d4e00d42) Another thing that we could do is make a ref for our username input. We'll say usernameInputRef = React.useRef, and then we could say ref = usernameInputRef, and instead of this, we could say username = usernameInputRef.current.value. Our usernameInputRef.current will be the DOM node for the input, and we'll get the value from that. This will work as well.  [6:43](https://egghead.io/lessons/react-make-basic-forms-with-react-d4e00d42) There are a couple of ways to do this, and I'll go ahead and add username = document.querySelector input.value. The reason that I removed that is I really want you to please not use this. It breaks encapsulation of our components. That's one of the things that we really love about React, is that we can encapsulate logic within our components.  [7:05](https://egghead.io/lessons/react-make-basic-forms-with-react-d4e00d42) I don't recommend this method or this method, because those implicitly rely on the order in which the elements are rendered and could easily break if you change that order. Either one of these will work just fine.  [7:16](https://egghead.io/lessons/react-make-basic-forms-with-react-d4e00d42) Personally, I think that if you don't need a ref for the input, that it's just easier to retrieve the input from the event target elements. It's recommended that you use HTML for an ID to associate labels to inputs anyway. Given that you already have to do that, this, I think, is the simplest way to get the value from the inputs in your form.  [7:36](https://egghead.io/lessons/react-make-basic-forms-with-react-d4e00d42) With that in mind, let's go ahead and get rid of all of this. We get rid of that, clean this up, and remove the ref. If I were writing this form today, this is what it would look like.  [7:46](https://egghead.io/lessons/react-make-basic-forms-with-react-d4e00d42) In review, what we had to do here was we created a form that has the single input and label associated to that input, and a button with the type of submit. It's really important that any button you put inside of a form has a type, because implicitly, it will have a type of submit. That's not entirely clear, especially if you say cancel for the contents with no specified type that gives this button any submit type, which is really confusing.  [8:12](https://egghead.io/lessons/react-make-basic-forms-with-react-d4e00d42) If you do want to have a cancel button or a reset button, then you want to specify the type is button, which I know is a little redundant, but if you don't specify a type, then the type will implicitly be submit, which would really confuse your users when they try to click cancel, so we specify that type is submit. We put in submit for the value here.  [8:34](https://egghead.io/lessons/react-make-basic-forms-with-react-d4e00d42) Then instead of putting an onClick handler on the button, we put an onSubmit handler on the form. That way, any other way that the user tries to submit the form will call our submit handler.  [8:44](https://egghead.io/lessons/react-make-basic-forms-with-react-d4e00d42) Then to avoid a full page refresh, we use event.preventDefault, and then we retrieve the user's input using event target to get the form, and then elements to get the elements of the form and usernameInput to retrieve the input by its ID. Then we get the value from that input to get our username, and then we alert that to our users, or you could submit this to a backend server. |
| 21. Создавайте динамические формы с помощью React | 21. Make Dynamic Forms with React |
| Часто бывает полезно знать, что вводит пользователь, когда он его набирает, и использовать эту информацию для изменения того, что отображается. Это может быть полезно для динамического поиска или фильтрации входных данных, запуска изменений, когда пользователь устанавливает флажок, или множества других вариантов использования. В этом примере мы собираемся динамически показывать сообщение об ошибке, если пользователь вводит что-то недопустимое, поэтому им не нужно ждать, пока они отправят форму, чтобы знать, что они делают что-то не так.  Для этого мы сохраним входное значение в состоянии, а затем используем это состояние для получения сообщения об ошибке, которое будет отображаться в случае ошибки. | Often, it can be useful to know what the user’s input is as they’re typing it and use that information to change what is rendered. This can be good for dynamic search or filter inputs, or triggering changes when a user checks a checkbox, or a myriad of other use cases. In this example, we’re going to dynamically show an error message if the user types something invalid so they don’t have to wait until they submit the form to know they’re doing something wrong.  To do this we’ll store the input’s value in state and then use that state to derive an error message which will be displayed if there is an error. |
| Кент С. Доддс: [0:00](https://egghead.io/lessons/react-make-dynamic-forms-with-react-4b705290) Допустим, эта UsenameForm не может содержать символы ВЕРХНЕГО регистра. Они могут содержать только строчные буквы. Мы могли бы показать пользователю ошибку, когда он нажимает кнопку «Отправить», но было бы намного лучше для пользователя, если бы мы отображали сообщение об ошибке, когда они вносят изменения в это поле ввода, если они изменяют его на что-то недопустимое.  [0:18](https://egghead.io/lessons/react-make-dynamic-forms-with-react-4b705290) Нам нужно знать, что набирает пользователь, когда он это набирает, а не только когда он отправляет это, поэтому мы собираемся добавить обработчик onChange прямо здесь, и здесь мы назовем этот handleChange. Мы создадим функцию handleChange (), которая примет событие. Затем мы можем использовать event.target.  [0:36](https://egghead.io/lessons/react-make-dynamic-forms-with-react-4b705290) Поскольку целью этого handleChange (события) является наш ввод, то вводом будет event.target, а .value - имя пользователя. Нам нужно где-то сохранить это имя пользователя и вызвать повторную визуализацию UsenameForm, чтобы отображалось сообщение об ошибке, если имя пользователя введено неправильно.  [0:55](https://egghead.io/lessons/react-make-dynamic-forms-with-react-4b705290) Давайте продолжим и добавим некоторое состояние в этот UsenameForm. Мы говорим [username, setUsername](https://egghead.io/lessons/react-make-dynamic-forms-with-react-4b705290) = React.useState(''), и мы инициализируем его пустой строкой. Здесь мы вызовем setUsername с event.target.value, и тогда нам больше не нужно будет получать имя пользователя из элементов ввода формы, потому что мы всегда будем поддерживать это имя пользователя в актуальном состоянии независимо от того, указание пользователя.  [1:21](https://egghead.io/lessons/react-make-dynamic-forms-with-react-4b705290) Мы можем избавиться от этого, и теперь это имя пользователя просто ссылается на это.  [1:25](https://egghead.io/lessons/react-make-dynamic-forms-with-react-4b705290) Затем давайте продолжим и определим, является ли это нижним регистром, сказав username === username.toLowerCase (). Если он равен собственной версии toLowerCase, то мы знаем, что это нижний регистр, и мы знаем, что у нас есть ошибка, если это не нижний регистр. Если это строчные буквы, мы просто скажем, что ошибка null, в противном случае мы скажем, что имя пользователя должно быть в нижнем регистре.  [1:52](https://egghead.io/lessons/react-make-dynamic-forms-with-react-4b705290) Тогда нам просто нужно отобразить это сообщение об ошибке. Мы скопируем это, спустимся сюда и сделаем здесь: style = {color: 'red'} просто для удовольствия. Мы поместим сообщение об ошибке прямо туда.  [2:05](https://egghead.io/lessons/react-make-dynamic-forms-with-react-4b705290) Мы также можем отключить кнопку в случае ошибки. Disabled принимает логическое значение, поэтому мы просто скажем Boolean\_error. Если ошибка истинна, мы передадим истинное значение для disabled, а если ошибки нет или это falsy, то мы передадим false value для disabled.  [2:25](https://egghead.io/lessons/react-make-dynamic-forms-with-react-4b705290) Давайте сохраним это. Получаем обновление. Если мы введем символ верхнего регистра, то мы получим, что имя пользователя должно быть в нижнем регистре, кнопка «Отправить» отключена. Если мы введем символ нижнего регистра, то у нас не будет никаких проблем, пока мы не получим там символ верхнего регистра.  [2:40](https://egghead.io/lessons/react-make-dynamic-forms-with-react-4b705290) Если вам когда-либо понадобится точно знать, что вводит пользователь, когда он это набирает, вы можете использовать событие onChange, чтобы получить доступ к значению ввода и обновить его в состоянии вашего компонента, а затем изменить его на это значение вызовет повторную визуализацию вашего компонента. Это значение состояния будет любым, введенным пользователем, что позволит нам создать это сообщение об ошибке в зависимости от того, является ли имя пользователя строчными буквами.  [3:03](https://egghead.io/lessons/react-make-dynamic-forms-with-react-4b705290) Мы показываем сообщение об ошибке красным здесь   Мы отключаем кнопку отправки, если появляется сообщение об ошибке | Kent C. Dodds: [0:00](https://egghead.io/lessons/react-make-dynamic-forms-with-react-4b705290) Let's say that this UsenameForm cannot have UPPER case characters. They can only have lower case characters. We could show the user an error when they hit the Submit button, but it would be a lot better user experience if we display an error message as they're making changes to this input field if they change it to something that's not allowed.  [0:18](https://egghead.io/lessons/react-make-dynamic-forms-with-react-4b705290) We need to know what the user's typing as they're typing it, not just as they submit it, so we're going to add an onChange handler right here and here we'll call this handleChange. We'll make a function handleChange(), and that will take the event. Then we can use event.target.  [0:36](https://egghead.io/lessons/react-make-dynamic-forms-with-react-4b705290) Because the target of this handleChange(event) is our input, then event.target is going to be the input, .value is going to be the username. We need to store that username somewhere and trigger re-render of the UsenameForm so that it displays the error message if the username is typed incorrectly.  [0:55](https://egghead.io/lessons/react-make-dynamic-forms-with-react-4b705290) Let's go ahead and add some state to this UsenameForm. We'll say [username, setUsername](https://egghead.io/lessons/react-make-dynamic-forms-with-react-4b705290) = React.useState('') and we'll initialize that to an empty string. In here, we'll call setUsername with the event.target.value, and then we no longer need to get the username from the form input elements because we're always going to be keeping this username up-to-date with whatever the user's specifying.  [1:21](https://egghead.io/lessons/react-make-dynamic-forms-with-react-4b705290) We can get rid of that, and now this username is just referencing that.  [1:25](https://egghead.io/lessons/react-make-dynamic-forms-with-react-4b705290) Next, let's go ahead and determine whether this isLowerCase by saying username === username.toLowerCase(). If it's equal to its toLowerCase version of itself, then we know that it is lower case, and we know that we have an error if it's not lower case. If it is lower case, we'll just say the error is null, otherwise we'll say the Username must be lower case.  [1:52](https://egghead.io/lessons/react-make-dynamic-forms-with-react-4b705290) Then we just need to display that error message. We'll copy that, come down here and make a  here with a style = { color: 'red' } just for the fun of it. We'll put the error message right in there.  [2:05](https://egghead.io/lessons/react-make-dynamic-forms-with-react-4b705290) We can also make the button disabled if there's an error. Disabled accepts a Boolean, so we'll just say Boolean\_error. If error is truthy, then we'll pass a true value for disabled, and if there is no error or it's falsy, then we'll pass a false value for disabled.  [2:25](https://egghead.io/lessons/react-make-dynamic-forms-with-react-4b705290) Let's save that. We get a refresh. If we type an upper-case character, then we'll get Username must be lower case, the Submit button is disabled. If we type a lower-case character, then we don't get any problem, until we get an upper-case character in there.  [2:40](https://egghead.io/lessons/react-make-dynamic-forms-with-react-4b705290) If you ever need to know exactly what the user's typing as they're typing it, then you can use the onChange event to get access to the value of the input and update that in the state of your component, then changes to that value will trigger a re-render of your component. That state value will be whatever the user's typed, allowing us to create this error message based on whether the username is lower case.  [3:03](https://egghead.io/lessons/react-make-dynamic-forms-with-react-4b705290) We display that error message here in red in this  We disable the submit button if there is an error message. |
| 22. Управление значениями формы с помощью React | 22. Controlling Form Values with React |
| Есть много ситуаций, когда вы хотите программно управлять значением поля формы. Возможно, вы хотите установить значение одного поля на основе взаимодействия пользователя с другим элементом. Или, может быть, вы хотите изменить ввод пользователя, когда он его набирает. В этом примере мы запретим пользователю вводить символы верхнего регистра в наше поле, превратив наш ввод из «Неконтролируемого поля» в «Контролируемое поле».  Вы можете узнать больше о контролируемых полях и компонентах из [документации React](https://reactjs.org/docs/forms.html#controlled-components) . | There are many situations where you want to programmatically control the value of a form field. Maybe you want to set the value of one field based on the user’s interactions with another element. Or maybe you want to change the user’s input as they’re typing it. In this example, we’ll be preventing the user from typing upper case characters into our field by turning our input from an “Uncontrolled field” to a “Controlled field.”  You can learn more about Controlled fields and components from [the React documentation](https://reactjs.org/docs/forms.html#controlled-components). |
| ент С. Доддс: [0:00 Замечательно](https://egghead.io/lessons/react-controlling-form-values-with-react-62f9ac37) , что мы можем отображать сообщение об ошибке, если есть символ верхнего регистра и мы отключили кнопку отправки, но было бы еще круче, если бы мы не позволяли пользователю вводить символы верхнего регистра. на первом месте.  [0:11](https://egghead.io/lessons/react-controlling-form-values-with-react-62f9ac37) Если они попытаются ввести символ верхнего регистра, мы просто сделаем его строчным. Мы могли бы сделать это довольно легко, сказав setUsername.toLowerCase. Теперь имя пользователя, которое хранится внутри нашего состояния, будет в нижнем регистре, поэтому мы можем всегда использовать верхний регистр. Когда мы нажимаем «Отправить», все будет в нижнем регистре.  [0:32](https://egghead.io/lessons/react-controlling-form-values-with-react-62f9ac37) Это не совсем хороший пользовательский опыт - иметь возможность напечатать что-то здесь и получить конечный результат, отличный от того, что я отправил. Что нам нужно сделать, так это контролировать, чтобы входное значение было точно таким же, что и у меня в моем состоянии, потому что я программно изменяю значение, которое я получаю из входа, чтобы установить значение, которое я сохраняю в моем состоянии.  [0:52](https://egghead.io/lessons/react-controlling-form-values-with-react-62f9ac37) Если я хочу, чтобы эти два параметра синхронизировались, мне нужно контролировать значение ввода. Сделать это довольно просто, мы просто добавляем здесь свойство value и передаем имя пользователя.  [1:03](https://egghead.io/lessons/react-controlling-form-values-with-react-62f9ac37) Как только мы передаем это value prop на вход, мы сообщаем React»у, что React больше не нуждается в управлении состоянием этого входа. Теперь мы собираемся контролировать это состояние, и оно не должно изменять это значение ни на что иное, кроме того, что мы вводим здесь, что включает в себя все, что вводит пользователь.  [1:22](https://egghead.io/lessons/react-controlling-form-values-with-react-62f9ac37) Если я сохраню это, вернусь сюда и попытаюсь ввести строку в верхнем регистре, это не сработает. Он автоматически переведет мою строку в нижний регистр. Теперь я не могу напечатать что-то неправильное. Было бы полезно иметь здесь какую-то информацию, которая указывает, почему я не могу вводить здесь символы верхнего регистра.  [1:42](https://egghead.io/lessons/react-controlling-form-values-with-react-62f9ac37) Надеюсь, вы получили общее представление о том, что если вам когда-нибудь понадобится сделать входное значение чем-то отличным от того, что на самом деле набирает пользователь, вы можете сделать это, передав свойство value. Давайте продолжим и очистим кое-что здесь, потому что нам больше не нужно отображать ошибку, потому что сейчас невозможно сделать ошибку.  [1:58](https://egghead.io/lessons/react-controlling-form-values-with-react-62f9ac37) Мы избавляемся от этого представления здесь. Мы избавимся от isLowercase и ошибки прямо здесь, и мы можем сохранить это, и все будет отлично. Вы не можете сказать, но я удерживаю клавишу Shift. Обещаю, я пытаюсь ввести что-нибудь в верхнем регистре.  [2:12](https://egghead.io/lessons/react-controlling-form-values-with-react-62f9ac37) Значение на входе не является прописным, потому что я устанавливаю имя пользователя в виде строчной версии того, что вводится в значение этого ввода. Затем, когда React повторно визуализирует этот компонент, имя пользователя, состоящее только из символов нижнего регистра, передается в значение ввода, и React отправляет это значение ввода в нижнюю версию того, что было. | Kent C. Dodds: [0:00](https://egghead.io/lessons/react-controlling-form-values-with-react-62f9ac37) It's great that we're able to display an error message if there's an uppercase character and we disable the submit button, but it would be even cooler if we didn't allow the user to type uppercase characters in the first place.  [0:11](https://egghead.io/lessons/react-controlling-form-values-with-react-62f9ac37) If they try to type an uppercase character, we just lowercase it for them. We could do that pretty easily by saying setUsername.toLowerCase. Now, the username that's stored inside of our state is going to be lowercase so we can do uppercase forever. When we hit submit, that's all going to be lowercase.  [0:32](https://egghead.io/lessons/react-controlling-form-values-with-react-62f9ac37) It's not exactly a great user experience to be able to type something here and have the end result be different from what I submitted. What we need to do is control the input value to be exactly the same thing that I have in my state because I'm programmatically changing the value that I get from the input to set the value that I store in my state.  [0:52](https://egghead.io/lessons/react-controlling-form-values-with-react-62f9ac37) If I want to keep those two in sync, then I need to control the input's value. To do this is pretty simple, we simply add a value prop here and we'll pass in the username.  [1:03](https://egghead.io/lessons/react-controlling-form-values-with-react-62f9ac37) As soon as we pass this value prop on to an input, we're communicating to React that React no longer needs to manage the state of this input. Now we are going to control that state and it should not change that value to anything other than what we enter in here which includes whatever the user's typing.  [1:22](https://egghead.io/lessons/react-controlling-form-values-with-react-62f9ac37) If I saved this and I come back up here and I try to type an uppercase string, that's not going to work. It's going to lowercase my string for me automatically. Now, it's impossible for me to type something that's incorrect. It might be useful to have some sort of info thing here that indicates why I can't type uppercase characters in here.  [1:42](https://egghead.io/lessons/react-controlling-form-values-with-react-62f9ac37) Hopefully, you get the general idea that if you ever need to make the input value something different from what the user is actually typing, then you can do so by passing a value prop. Let's go ahead and clean some stuff up here because we no longer need to display an error because it's impossible to make an error now.  [1:58](https://egghead.io/lessons/react-controlling-form-values-with-react-62f9ac37) We get rid of that submit here. We'll get rid of the isLowercase and error right here, and we can save that and everything looks great. You can't tell but I'm holding down the shift key. I promise I'm trying to type something uppercase.  [2:12](https://egghead.io/lessons/react-controlling-form-values-with-react-62f9ac37) The value in the input isn't uppercase because I'm setting the username to a lowercase version of what's being typed in that input's value. Then when React re-renders this component, the username which is all the lowercase characters is getting passed into the value of the input and React is sending that input's value to the lowercase version of what it was. |
| 23. Использование React Error Boundaries для обработки ошибок в компонентах React | 23. Using React Error Boundaries to handle errors in React Components |
| Как бы вы ни старались, в конечном итоге код вашего приложения просто не будет вести себя так, как вы ожидаете, и вам придется обрабатывать эти исключения. Если рендеринг запущен и не обработан, ваше приложение будет удалено со страницы, в результате чего у пользователя останется пустой экран ... Немного неловко ...  К счастью для нас, существует простой способ обработки ошибок в вашем приложении с помощью специального компонента, называемого [Error Boundary](https://reactjs.org/docs/error-boundaries.html). К сожалению, в настоящее время нет способа создать компонент Error Boundary с функцией, и вместо этого вы должны использовать компонент класса, но нам повезло, потому что есть потрясающая библиотека с открытым исходным кодом, которую мы можем использовать, вызывая [react-error-boundary](https://www.npmjs.com/package/react-error-boundary).  В этом уроке мы узнаем, как написать нашу собственный простой error boundary, а затем как использовать react-error-boundary. Мы также узнаем о том, где мы размещаем границы ошибок в нашем дереве компонентов React. | No matter how hard you try, eventually your app code just isn’t going to behave the way you expect it to and you’ll need to handle those exceptions. If a render is thrown and unhandled, your application will be removed from the page, leaving the user with a blank screen... Kind of awkward...  Luckily for us, there’s a simple way to handle errors in your application using a special kind of component called an [Error Boundary](https://reactjs.org/docs/error-boundaries.html). Unfortunately, there is currently no way to create an Error Boundary component with a function and you have to use a class component instead, but we got another lucky break because there’s a terrific open source library we can use called [react-error-boundary](https://www.npmjs.com/package/react-error-boundary).  In this lesson, we’ll learn how to write our own simple error boundary and then how to use react-error-boundary instead. We’ll also learn the implications of where we place our Error Boundaries in our React component tree. |
| Кент С. Доддс: [0:00](https://egghead.io/lessons/react-using-react-error-boundaries-to-handle-errors-in-react-components) Как бы усердно вы ни работали над своим приложением, иногда случаются неожиданные вещи. Я не ожидаю, что люди будут выдавать ошибки в теле вашей функции, но они могут вызывать функции, которые не существуют, что приведет к [неслышной](https://egghead.io/lessons/react-using-react-error-boundaries-to-handle-errors-in-react-components) ошибке и приведет к ошибке. Давайте посмотрим, как вы можете управлять этим с помощью React.  [0:16](https://egghead.io/lessons/react-using-react-error-boundaries-to-handle-errors-in-react-components) Здесь у нас есть некоторое разнесенное состояние, в котором мы инициализируем его значением false. Затем у нас есть кнопка, где, когда вы нажимаете на нее, она меняет состояние разнесения на истинное. Затем мы также визуализируем это , где, если есть какое-то состояние взрыва, мы визуализируем эту бомбу, которая вызовет эту ошибку.  [0:33](https://egghead.io/lessons/react-using-react-error-boundaries-to-handle-errors-in-react-components) Если мы посмотрим, как все выглядит прямо сейчас, мы щелкнем по бомбе, и все наше приложение исчезнет. Мы получаем эту Uncaught Error: CABOOM в нашей консоли, и если мы немного прокрутим здесь вниз, React выйдет из трассировки стека компонентов, чтобы мы могли отследить, какой компонент выдал эту ошибку.  [0:52](https://egghead.io/lessons/react-using-react-error-boundaries-to-handle-errors-in-react-components) У нас есть приложение, которое отображается прямо здесь, а затем у нас есть это , который представлен здесь, и тогда у нас есть это , который отображается здесь, а затем у нас есть эта бомба, которая отображается здесь, и эта бомба является причиной ошибки.  [1:05](https://egghead.io/lessons/react-using-react-error-boundaries-to-handle-errors-in-react-components) Это подсказка. «Подумайте о добавлении error boundary в ваше дерево, чтобы настроить поведение обработки ошибок». Это именно то, что мы собираемся делать. Обычно я использую для этого стороннюю библиотеку, но мы собираемся построить нашу собственную небольшую error boundary.  [1:19](https://egghead.io/lessons/react-using-react-error-boundaries-to-handle-errors-in-react-components) Error boundaries должны быть компонентами класса, и чтобы создать компонент класса с помощью React, мы скажем, что класс ErrorBoundary расширяет React.Component. В теле нашего компонента класса React нам понадобится метод рендеринга, и именно он будет рендериться. По сути, это то же самое, что и тело наших обычных функциональных компонентов.  [1:42](https://egghead.io/lessons/react-using-react-error-boundaries-to-handle-errors-in-react-components) Здесь мы не собираемся визуализировать что-либо особенное с помощью ErrorBoundary. Мы просто собираемся вернуть this.props.children, поэтому элементы React, которые мы возвращаем для этой ErrorBoundary, будут теми же элементами React, которые предоставляются этой ErrorBoundary в качестве дочерних элементов. Давайте продолжим и поместим их прямо сюда, внутрь этого.  [2:01](https://egghead.io/lessons/react-using-react-error-boundaries-to-handle-errors-in-react-components) Мы скажем ErrorBoundary, вставим то, что у нас было раньше, а затем закроем эту ErrorBoundary. Если мы сохраним это, мы получим обновление, и когда я нажму на эту бомбу, мы получим точно такое же поведение, что и раньше. Теперь давайте заставим этот ErrorBoundary обработать эту ошибку.  [2:17](https://egghead.io/lessons/react-using-react-error-boundaries-to-handle-errors-in-react-components) Первое, что нам понадобится, это какое-то состояние, поэтому я скажу error: null, а затем у нас будет статический метод getDerivedStateFromError (), который примет ошибку, а затем вернет изменение состояния, которое мы хотим сделать на основе этой ошибки. Мы просто вернем объект со свойством error, который будет присвоен ошибке, которую мы получаем для этого статического метода.  [2:40](https://egghead.io/lessons/react-using-react-error-boundaries-to-handle-errors-in-react-components) Когда это произойдет, мы сделаем повторную визуализацию. Давайте возьмем эту ошибку из нашего состояния, и тогда мы сможем сказать, если есть ошибка, тогда мы вернем что говорит: "Oh no!". Давайте сохраним это, и когда мы щелкнем по нему, мы увидим "Oh no!"  [3:00](https://egghead.io/lessons/react-using-react-error-boundaries-to-handle-errors-in-react-components) Мы по-прежнему будем видеть ошибки, записываемые в консоль, но приложение не выйдет из строя полностью. React просто повторно отрендерит эту ErrorBoundary с возникшей ошибкой, и ErrorBoundary сможет контролировать то, что отрисовывается.  [3:13](https://egghead.io/lessons/react-using-react-error-boundaries-to-handle-errors-in-react-components) Я собираюсь создать здесь функциональный компонент под названием ErrorFallback(), который будет моим общим резервным компонентом для этого приложения. Мы заставим это принять prop с именем error. Он вернет то, что имеет тэг "Что-то пошло не так:", а затем тег с error.message.  [3:40](https://egghead.io/lessons/react-using-react-error-boundaries-to-handle-errors-in-react-components) Как пользователь этого error boundary component я хочу иметь возможность предоставить error boundary резервный компонент, который я хочу отображать при возникновении ошибки. Я собираюсь предоставить ему prop FallbackComponent. Здесь я предоставлю тот компонент, который я только что создал этот ErrorFallback, и вставьте его прямо туда.  [4:01](https://egghead.io/lessons/react-using-react-error-boundaries-to-handle-errors-in-react-components) Чтобы наш error boundary мог принять и отобразить это, у нас будет доступ к резервному компоненту в this.props. Мы можем сказать this.props.fallbackComponent, и мы предоставим the prop error как error.  [4:18](https://egghead.io/lessons/react-using-react-error-boundaries-to-handle-errors-in-react-components) Вы можете вспомнить, что для рендеринга пользовательских компонентов требуется, чтобы первый используемый символ был написан с заглавной буквы, но у компилятора также есть правило, согласно которому, если в имени компонента есть точки, они также будут рассматриваться как пользовательские компоненты, вот почему это работает.  [4:34](https://egghead.io/lessons/react-using-react-error-boundaries-to-handle-errors-in-react-components) Если мы сохраним это, мы нажмем кнопку и получим: "Something went wrong. CABOOM."  Приложение все еще работает, и мы также получаем всю информацию, записываемую в консоль.  [4:45](https://egghead.io/lessons/react-using-react-error-boundaries-to-handle-errors-in-react-components) Как я уже сказал, я никогда не использую компоненты класса даже для error boundaries, потому что я предпочитаю использовать стороннюю библиотеку, которую я собираюсь включить сейчас и которая называется React ErrorBoundary. С the React Error Boundary UMD Export он предоставляет глобальную переменную ReactErrorBoundary и имеет свойство ErrorBoundary. Я собираюсь просто присвоить это ErrorBoundary.  [5:11](https://egghead.io/lessons/react-using-react-error-boundaries-to-handle-errors-in-react-components) Вам будет приятно узнать, что мне не нужно менять какой-либо код, чтобы это работало. Он будет продолжать работать, потому что мы в основном создали простую версию ErrorBoundary из этой библиотеки. Библиотека делает немного больше, чем то, что мы делаем здесь, поэтому я рекомендую вам сделать это вместо этого, но API точно такой же.  [5:30](https://egghead.io/lessons/react-using-react-error-boundaries-to-handle-errors-in-react-components) Еще одна вещь, которую я хочу здесь продемонстрировать, - это то, что этот ErrorBoundary можно отобразить в любом месте дерева. Расположение ErrorBoundary имеет особое значение. ErrorBoundary может обрабатывать любые ошибки, создаваемые его потомками. Также важно отметить, что ErrorBoundary будет отображать что-то вместо всех своих потомков при возникновении ошибки.  [5:50](https://egghead.io/lessons/react-using-react-error-boundaries-to-handle-errors-in-react-components) Это означает, что, хотя мы можем видеть бомбу прямо здесь и в остальной части нашего приложения, если бы мы переместили эту ErrorBoundary вверх, чтобы охватить все наше приложение и сохранить ее, когда мы нажмем эту кнопку, тогда все приложение будет заменен нашим компонентом ErrorFallback.  [6:08](https://egghead.io/lessons/react-using-react-error-boundaries-to-handle-errors-in-react-components) Это может быть, а может и не быть желательным в зависимости от конкретного случая использования, но может быть полезно, чтобы здесь отображалась одна граница ошибки, а затем другие error boundaries отображались во всем приложении с более конкретными ErrorFallbacks.  [6:22](https://egghead.io/lessons/react-using-react-error-boundaries-to-handle-errors-in-react-components) Здесь у нас может быть «что-то пошло не так», и эта ошибка будет обработана ближайшим ErrorBoundary здесь. Затем, если что-то пойдет не так в другом месте нашего приложения, то ErrorBoundary верхнего уровня сможет обработать эту ошибку. Это дает нам точный контроль над тем, какая часть дерева будет заменена тем, что отображается нашим error boundary.  [6:43](https://egghead.io/lessons/react-using-react-error-boundaries-to-handle-errors-in-react-components) Еще одна важная вещь, на которую следует обратить внимание, это то, что our error boundary может обрабатывать только определенные ошибки, в частности ошибки, которые происходят в стеке вызовов React. Он не будет обрабатывать ошибки, которые происходят в обработчиках событий или если есть ошибка в асинхронном обратном вызове, например, в обработчике Promise. Он будет обрабатывать только ошибки, которые происходят в стеке вызовов React, например, в методе рендеринга или обратном вызове React useEffect.  [7:06](https://egghead.io/lessons/react-using-react-error-boundaries-to-handle-errors-in-react-components) В обзоре для обработки ошибок React необходимо создать error boundary или, как я рекомендую, использовать react-error-boundary из npm. С помощью этой error boundary вы можете заключить части вашего кода в оболочку. Ошибки всех потомков этого error boundary будут обрабатываться by the error boundary.  [7:23](https://egghead.io/lessons/react-using-react-error-boundaries-to-handle-errors-in-react-components) Используя error boundary из библиотеки react-error-boundary, вы можете предоставить FallbackComponent. Этот FallbackComponent будет отображаться в случае ошибки, позволяя вам восстановиться после ошибки или просто отображать сообщение об ошибке для чтения пользователем. | Kent C. Dodds: [0:00](https://egghead.io/lessons/react-using-react-error-boundaries-to-handle-errors-in-react-components) As hard as you work on your application, sometimes unexpected things will happen. I don't expect people will be throwing errors in the body of your function, but they could be calling functions that don't exist, which will [inaudible](https://egghead.io/lessons/react-using-react-error-boundaries-to-handle-errors-in-react-components) results in an error. Let's take a look at how you can manage these with React.  [0:16](https://egghead.io/lessons/react-using-react-error-boundaries-to-handle-errors-in-react-components) Here we have some explode state, where we initialize that to false. Then we have a button, where when you click on it, it changes that explode state to true. Then, we also render this , where if there is some explode state, then we'll render that bomb, which will trigger this error.  [0:33](https://egghead.io/lessons/react-using-react-error-boundaries-to-handle-errors-in-react-components) If we take a look at what things look like right now, we click on the bomb and our entire application goes away. We get this Uncaught Error: CABOOM in our console, and if we scroll down here a little bit, React logs out a component stack trace so we can track down which component threw this error.  [0:52](https://egghead.io/lessons/react-using-react-error-boundaries-to-handle-errors-in-react-components) We have our App, and that is rendered right here, and then we have this, which is rendered here, and then we have this , which is rendered here, and then we have this bomb, which is rendered here, and that bomb is the thing that threw the error.  [1:05](https://egghead.io/lessons/react-using-react-error-boundaries-to-handle-errors-in-react-components) It's giving us this tip. "Consider adding an error boundary to your tree to customize error handling behavior." That's just what we're going to do. Normally, I use the third-party library for this, but we're going to build our own little error boundary.  [1:19](https://egghead.io/lessons/react-using-react-error-boundaries-to-handle-errors-in-react-components) Error boundaries have to be class components, and to create a class component with React, we're going to say class ErrorBoundary extends React.Component. In the body of our React class component, we're going to need a render method, and this is what's going to be rendered. It's basically the same thing as the body of our regular function components.  [1:42](https://egghead.io/lessons/react-using-react-error-boundaries-to-handle-errors-in-react-components) Here, we're not going to render anything special with this ErrorBoundary. We're simply going to return this.props.children, so the React elements we return for this ErrorBoundary are going to be the same React elements that are provided to this ErrorBoundary as children. Let's go ahead and put those right here inside of this.  [2:01](https://egghead.io/lessons/react-using-react-error-boundaries-to-handle-errors-in-react-components) We'll say ErrorBoundary, paste in what we had before, and then close off that ErrorBoundary. If we save this, we get a refresh and when I click on this bomb, we get the exact same behavior that we had before. Now let's make this ErrorBoundary handle that error.  [2:17](https://egghead.io/lessons/react-using-react-error-boundaries-to-handle-errors-in-react-components) The first thing that we're going to need is some state, so I'll say error: null, and then we'll have a static method called getDerivedStateFromError() that'll accept an error and then it'll return the state change that we want to make based on this error. We'll just return an object that has an error property and that's going to be assigned to the error that we're getting for this static method.  [2:40](https://egghead.io/lessons/react-using-react-error-boundaries-to-handle-errors-in-react-components) When this happens, we're going to get a re-render. Let's grab that error from our state and then we can say if there's an error, then we'll return a that says, "Oh no!" Let's save that and when we click this, we're going to see "Oh no!"  [3:00](https://egghead.io/lessons/react-using-react-error-boundaries-to-handle-errors-in-react-components) We'll still see errors logged to the console for our benefit, but the application won't completely crash. React will simply re-render this ErrorBoundary with the error that was thrown and the ErrorBoundary gets to control what's being rendered.  [3:13](https://egghead.io/lessons/react-using-react-error-boundaries-to-handle-errors-in-react-components) I'm going to go ahead and make a function component here called ErrorFallback(), which is going to be my generic fallback component for this application. We'll make this accept a prop called error. It will return a that has a tag with "Something went wrong:" and then a tag with the error.message.  [3:40](https://egghead.io/lessons/react-using-react-error-boundaries-to-handle-errors-in-react-components) As the user of this error boundary component, I want to be able to provide to the error boundary the fallback component I want it to render when there's an error. I'm going to provide it the prop FallbackComponent. Here, I'll provide that component that I just created this ErrorFallback, paste that right in there.  [4:01](https://egghead.io/lessons/react-using-react-error-boundaries-to-handle-errors-in-react-components) For our error boundary to accept and render that, we're going to have access to that fallback component on this.props. We can say this.props.fallbackComponent and we'll provide the prop error as error.  [4:18](https://egghead.io/lessons/react-using-react-error-boundaries-to-handle-errors-in-react-components) You may recall that rendering custom components requires that the first character you use is capitalized, but the compiler also has a rule that if there are dots included in the component name, then those will also be treated as custom components, which is why this works.  [4:34](https://egghead.io/lessons/react-using-react-error-boundaries-to-handle-errors-in-react-components) If we save this, we push the button and we're going to get, "Something went wrong. CABOOM." The app is still working, and we get all of the information logged to the console as well.  [4:45](https://egghead.io/lessons/react-using-react-error-boundaries-to-handle-errors-in-react-components) As I said, I never use class components even for error boundaries because I prefer to use a third-party library, which I'm going to include now and that is called React ErrorBoundary. With the React Error Boundary UMD Export, it exposes a global variable called ReactErrorBoundary and it has a property on there called ErrorBoundary. I'm going to simply assign that to ErrorBoundary.  [5:11](https://egghead.io/lessons/react-using-react-error-boundaries-to-handle-errors-in-react-components) You'll be pleased to know that I don't have to change any of my code for this to work. It will to continue to work because we basically built a simple version of the ErrorBoundary from this library. The library is doing a fair a bit more than what we're doing here, which is why I recommend you do that instead, but the API is exactly the same.  [5:30](https://egghead.io/lessons/react-using-react-error-boundaries-to-handle-errors-in-react-components) One other thing I want to demonstrate here is that this ErrorBoundary can be rendered anywhere in the tree. The location of the ErrorBoundary has a special significance. The ErrorBoundary can handle any errors that are thrown by its descendants. It's also important to note that the ErrorBoundary is going to render something in place of all of its descendants when there is an error.  [5:50](https://egghead.io/lessons/react-using-react-error-boundaries-to-handle-errors-in-react-components) That means that while we can see the bomb still right here and the rest of our application, if we were to move this ErrorBoundary up to encompass our entire app and save that, when we click this button, then the entire app is replaced by our ErrorFallback component.  [6:08](https://egghead.io/lessons/react-using-react-error-boundaries-to-handle-errors-in-react-components) This may or may not be desirable based on a specific use case, but it could be useful to have one error boundary rendered up here and then other error boundaries rendered throughout your application with more specific ErrorFallbacks.  [6:22](https://egghead.io/lessons/react-using-react-error-boundaries-to-handle-errors-in-react-components) Here we can have "Something went wrong there," and that error is going to be handled by the nearest ErrorBoundary here. Then if something went wrong somewhere else in our application, then this top-level ErrorBoundary could handle that error. This gives us some fine-grained control over what part of the tree is going to be replaced by what's rendered by our error boundary.  [6:43](https://egghead.io/lessons/react-using-react-error-boundaries-to-handle-errors-in-react-components) Another important thing to note is that our error boundary can only handle certain errors, specifically errors that are happening within the React call stack. It won't handle errors that are happening in event handlers or if there's an error in an asynchronous callback, like a Promise handler. It will only handle errors that happen within a React call stack like the render method or a React useEffect callback.  [7:06](https://egghead.io/lessons/react-using-react-error-boundaries-to-handle-errors-in-react-components) In the review, to handle React errors, you need to create an error boundary or, as I recommend, use react-error-boundary from npm. With this error boundary, you can wrap parts of your code. Any descendants of that error boundary will have its errors handled by the error boundary.  [7:23](https://egghead.io/lessons/react-using-react-error-boundaries-to-handle-errors-in-react-components) With the error boundary from the react-error-boundary library, you can provide a FallbackComponent. That FallbackComponent will be rendered in the event of an error, allowing you to recover from the error or simply display an error message for the user to read. |
| 24. Используйте key prop при рендеринге списка с помощью React | 24.Use the key prop when Rendering a List with React |
| Не займет много времени в React, прежде чем вы захотите отобразить список элементов, и когда вы это сделаете, вы неизбежно столкнетесь с этим предупреждением консоли: «Предупреждение: каждый дочерний элемент в списке должен иметь уникальное ключевое свойство (key prop)». Это предупреждение довольно просто заглушить, предоставив специальную опцию key, но действительно полезно понять, о чем это предупреждение, и об ошибках, которые могут возникнуть, если вы не устраните предупреждение должным образом.  В этом уроке мы увидим демонстрацию этой проблемы и поймем ситуацию, которая может произойти, если мы не решим ее должным образом. В этом примере мы используем входные данные, но то же самое может произойти и с вашими собственными компонентами, которые поддерживают состояние. Вы определенно не хотите игнорировать это предупреждение. | It doesn’t take long working with React before you want to render a list of items and when you do, you’ll inevitably encounter this console warning: “Warning: Each child in a list should have a unique key prop.” This warning is pretty simple to silence by providing the bespoken key prop, but it is really useful to understand what that warning is about and the bugs that can happen if you do not address the warning properly.  In this lesson we’ll see a demo of this problem and understand a situation that can happen when we don’t handle it properly. We’re using inputs in this example, but the same thing can happen for your own components that maintain state. You definitely do not want to ignore this warning. |
| Кент С. Доддс: [0:00 У](https://egghead.io/lessons/react-use-the-key-prop-when-rendering-a-list-with-react-b95caad8) нас есть приложение, которое управляет некоторыми элементами. Мы можем добавлять элементы и удалять элементы. У нас есть фиксированный набор элементов, которые можно добавлять или удалять, и у нас есть кнопка для добавления дополнительных элементов, которая отключается, когда мы добавляем все наши элементы, а затем мы перебираем элементы, которые у нас есть, и визуализируем список для каждого из них.  [0:22](https://egghead.io/lessons/react-use-the-key-prop-when-rendering-a-list-with-react-b95caad8) Здесь у нас есть кнопки удаления для каждой из них, затем само имя элемента и вход с этим элементом в нем. Когда мы визуализируем это, мы получим это ослабление, которое говорит: "Each child in a list should have a unique 'key' prop.".  [0:36](https://egghead.io/lessons/react-use-the-key-prop-when-rendering-a-list-with-react-b95caad8) Речь идет о том, что каждый раз, когда вы визуализируете массив элементов React, в нашем случае мы визуализируем эти <li>s, каждый из этих элементов React должен иметь связанное с ним ключевое свойство, чтобы React мог правильно отслеживать их во времени.  [0:50](https://egghead.io/lessons/react-use-the-key-prop-when-rendering-a-list-with-react-b95caad8) Не обманывайтесь синтаксисом. Здесь нет ничего волшебного. Что мы делаем, так это берем массив строк, сопоставляем этот массив строк и превращаем этот массив строк в массив элементов React. В частности, в нашем случае это <li> элементы.  [1:05](https://egghead.io/lessons/react-use-the-key-prop-when-rendering-a-list-with-react-b95caad8) Это тот случай, когда вам нужно иметь key prop для каждого элемента React в массиве. Мы видим это предупреждение, потому что не видим. Исправить это предупреждение довольно просто. Мы смотрим на элементы, которые повторяем. Вот этот массив прямо здесь. Каждый из этих элементов имеет идентификатор, который однозначно идентифицирует элемент. Мы собираемся использовать этот идентификатор в качестве the key prop.  [1:29](https://egghead.io/lessons/react-use-the-key-prop-when-rendering-a-list-with-react-b95caad8) Здесь мы скажем key = {item.id}. Если мы сохраним это, мы больше не получим этого предупреждения. Я не большой поклонник изменения кода только для того, чтобы избавиться от предупреждений. Мне нравится понимать, почему вообще есть это предупреждение. Вот почему у нас есть этот надуманный пример, который я могу вам показать.  [1:47](https://egghead.io/lessons/react-use-the-key-prop-when-rendering-a-list-with-react-b95caad8) Здесь у нас есть список каждого из этих элементов, где сам элемент является меткой, а значение по умолчанию для ввода также является элементом. Затем мы можем удалить их и добавить. Когда я их удалял, если вы внимательно смотрели, вы могли заметить интересную ошибку.  [2:02](https://egghead.io/lessons/react-use-the-key-prop-when-rendering-a-list-with-react-b95caad8) Если я нажму здесь на «Удалить», а затем удаляю, удаляю и удаляю, все работает нормально. Если я уберу с начала, теперь с яблоком будет ассоциироваться апельсин. Виноград ассоциируется с апельсином. Груша ассоциируется с виноградом. Это интересно.  [2:18](https://egghead.io/lessons/react-use-the-key-prop-when-rendering-a-list-with-react-b95caad8) Если я уберу середину, то теперь груша ассоциируется с апельсином. Апельсин ассоциируется с яблоком. Все в беспорядке. Это ошибка, которую вам поможет избежать key prop.  [2:29](https://egghead.io/lessons/react-use-the-key-prop-when-rendering-a-list-with-react-b95caad8) Если подумать, что происходит, когда мы нажимаем эту кнопку удаления или эту кнопку добавления, тогда это странное поведение будет иметь смысл. Когда мы нажимаем на removeItem, например, если мы пытаемся удалить виноград, это входит в этот код прямо здесь, который вызывает setItems. Это устанавливает для элементов тот же список элементов, который был у нас раньше, за исключением того, что мы отфильтровываем элемент, который вы пытались удалить.  [2:52](https://egghead.io/lessons/react-use-the-key-prop-when-rendering-a-list-with-react-b95caad8) Этот вызов setItems вызовет повторную визуализацию нашего приложения. Приложение будет здесь. Мы собираемся создать эти элементы React. Мы собираемся перебрать этот список элементов, в котором теперь на один элемент меньше, чем было раньше. Мы собираемся передать это React, чтобы он мог соответствующим образом обновить DOM.  [3:10](https://egghead.io/lessons/react-use-the-key-prop-when-rendering-a-list-with-react-b95caad8) React обновляет DOM, так как в нем есть ссылка на элементы JSX, которые вы дали ему при последней визуализации этого компонента приложения. Он сравнивает эти элементы React с новыми элементами React, которые вы только что вернули. Затем он соответствующим образом обновляет DOM.  [3:24](https://egghead.io/lessons/react-use-the-key-prop-when-rendering-a-list-with-react-b95caad8) Когда вы даете ему массив элементов React, если у React нет какого-либо идентификатора, чтобы узнать, какой элемент есть, он не знает, удалили ли вы элемент, или, может быть, вы добавили три и удалили четыре, или, может быть, вы изменил порядок и просто удалил первый.  [3:39](https://egghead.io/lessons/react-use-the-key-prop-when-rendering-a-list-with-react-b95caad8) Он не имеет никакого представления о том, что вы сделали с этим массивом элементов React с момента его последней визуализации до этого раза. Каждый раз, когда вы визуализируете массив элементов React, вам необходимо дать ему ключ, чтобы он мог определить, были ли элементы удалены, добавлены или изменены.  [3:57](https://egghead.io/lessons/react-use-the-key-prop-when-rendering-a-list-with-react-b95caad8) Вот почему предоставление уникального ключа для каждого из этих элементов решит эту проблему. Теперь, если мы нажмем удалить виноград, все введенные данные и их метки верны.  [4:08](https://egghead.io/lessons/react-use-the-key-prop-when-rendering-a-list-with-react-b95caad8) Вы также заметите, что если вы попытаетесь указать какой-то ключ, где это один и тот же ключ для каждого из них, вы получите предупреждение, потому что он встретил двух дочерних элементов с одним и тем же ключом. Ключи должны быть уникальными, чтобы компоненты сохраняли свою идентичность при обновлениях. Неуникальные ключи могут приводить к дублированию и / или пропуску дочерних элементов - поведение не поддерживается и может измениться в будущей версии.  [4:32](https://egghead.io/lessons/react-use-the-key-prop-when-rendering-a-list-with-react-b95caad8) Жесткое кодирование определенного ключа, который дублируется в этих элементах, может привести к очень неожиданному поведению, и мы все еще сталкиваемся с той ошибкой, которая была у нас раньше. Ключ, который вы предоставляете для каждого элемента этого массива элементов React, должен быть уникальным для элемента, который вы визуализируете. Обычно это будет какой-то идентификатор, как в нашем случае.  [4:56](https://egghead.io/lessons/react-use-the-key-prop-when-rendering-a-list-with-react-b95caad8) Еще одна ошибка, которую я иногда вижу, люди делают, что они пытаются использовать индекс в качестве ключа. Пока вы избавляетесь от предупреждения, вы не избавляетесь от ошибок. Это потому, что, поскольку React сравнивает предыдущую версию с новой версией, вы говорите, что элемент, который был в индексе четыре, на самом деле теперь в индексе три, но React не знает этого.  [5:17](https://egghead.io/lessons/react-use-the-key-prop-when-rendering-a-list-with-react-b95caad8) Он поддерживает состояние этого входа, которое было таким же, как и то, которое было в индексе три в последний раз, а не то, которое было в индексе четыре в последний раз. Важно, чтобы вы сохранили ключ как нечто уникальное для элемента, который представляет этот элемент в массиве.  [5:36](https://egghead.io/lessons/react-use-the-key-prop-when-rendering-a-list-with-react-b95caad8) Вот еще одна небольшая демонстрация, которая у меня есть в конце этого файла. Я просто раскомментирую это. Мы это сохраним. Мы получим обновление. В этом примере у нас есть эти предметы. Каждые 1000 миллисекунд у нас есть интервал, который перемешивает эти элементы и запускает повторный рендеринг с этой перемешанной версией элементов. Мы каждый раз рандомизируем предметы.  [5:57](https://egghead.io/lessons/react-use-the-key-prop-when-rendering-a-list-with-react-b95caad8) У нас есть три версии: одна отображает эти элементы как входные данные без ключа, другая отображает эти элементы с входными данными с ключом index, а затем последняя, ​​которая отображает эти элементы с соответствующим ключом. Вы заметите, что все они обновляются правильно, что означает, что они все прыгают по экрану, как должны, но фокус обновляется неправильно.  [6:22](https://egghead.io/lessons/react-use-the-key-prop-when-rendering-a-list-with-react-b95caad8) Сейчас я сосредоточен на яблоке. Когда тот двигается, я не фокусируюсь на яблоке. Кроме того, если я попытаюсь выделить один из них, мое выделение тоже исчезнет. Вы заметите, что с ключом в качестве индекса страдает точно такая же проблема. Несмотря на то, что в консоли не появляется предупреждение, вы все равно не исправляете эту ошибку.  [6:42](https://egghead.io/lessons/react-use-the-key-prop-when-rendering-a-list-with-react-b95caad8) Только когда у вас есть ключ, ваш фокус будет перемещаться по входу, с которым он связан, потому что React может определить, куда переместить фокус при обновлении вашего компонента.  [6:55 Итак](https://egghead.io/lessons/react-use-the-key-prop-when-rendering-a-list-with-react-b95caad8) , в React принято брать массив, сопоставлять этот массив с массивом элементов и отображать его непосредственно в вашем JSX. Когда вы это делаете, важно, чтобы вы добавляли ключ к корневому элементу React каждого элемента в массиве, чтобы React мог отслеживать изменения с течением времени и убедиться, что он сохраняет состояние каждого элемента в массиве и его потомков. | Kent C. Dodds: [0:00](https://egghead.io/lessons/react-use-the-key-prop-when-rendering-a-list-with-react-b95caad8) Here we have an app that's managing some items. We can add items and remove items. We have a fixed set of items that can be added or removed, and we have a button to add additional items, which is disabled when we've added all of our items, and then we iterate over the items that we have and render a list item for each of those.  [0:22](https://egghead.io/lessons/react-use-the-key-prop-when-rendering-a-list-with-react-b95caad8) Here we have remove buttons for each one of these, then the item name itself, and an input with that item in it. When we render this, we're going to get this waning that says, "Each child in a list should have a unique 'key' prop."  [0:36](https://egghead.io/lessons/react-use-the-key-prop-when-rendering-a-list-with-react-b95caad8) What that's talking about is any time you render an array of React elements, in our case we're rendering these <li>s, each one of those React elements must have a key prop associated with it so that React can track these appropriately over time.  [0:50](https://egghead.io/lessons/react-use-the-key-prop-when-rendering-a-list-with-react-b95caad8) Don't be fooled by the syntax. There's nothing really magic going on here. What we're doing is we're taking an array of strings, we're mapping over that array of strings, and turning that array of strings into an array of React elements. Specifically, in our case, these are <li> elements.  [1:05](https://egghead.io/lessons/react-use-the-key-prop-when-rendering-a-list-with-react-b95caad8) That's the case where you need to have a key prop for every React element in the array. We're seeing this warning because we don't. To fix this warning is pretty simple. We look at the items that we're iterating over. That is this array right here. Each one of those items has an id that uniquely identifies the item. We're going to use that id as the key prop.  [1:29](https://egghead.io/lessons/react-use-the-key-prop-when-rendering-a-list-with-react-b95caad8) Here, we'll say key = {item.id}. If we save this, we don't get that warning anymore. I'm not a huge fan of changing my code just to make warnings go away. I like to understand why that warning is there in the first place. That's why we have this contrived example for me to show you.  [1:47](https://egghead.io/lessons/react-use-the-key-prop-when-rendering-a-list-with-react-b95caad8) Here, we have a list of each one of these items, where the item itself is the label and then the default value for the input is the item as well. Then we can remove these and add them. As I removed them, if you watched carefully, you might have noticed an interesting bug.  [2:02](https://egghead.io/lessons/react-use-the-key-prop-when-rendering-a-list-with-react-b95caad8) If I click on remove here and remove and remove and remove, everything works out just fine. If I remove from the beginning, now we're having orange being associated to the apple. Grape is associated to orange. Pear is associated to grape. That's interesting.  [2:18](https://egghead.io/lessons/react-use-the-key-prop-when-rendering-a-list-with-react-b95caad8) If I remove from the middle here, now we have pear is associated to orange. Orange is associated to apple. Things are all kinds of messed up. This is the bug that the key prop helps you to avoid.  [2:29](https://egghead.io/lessons/react-use-the-key-prop-when-rendering-a-list-with-react-b95caad8) If you consider what happens when we click this remove button or this add button, then this strange behavior will make sense. When we click on removeItem, for example, if we try to remove the grape, then that goes into this code right here, which calls setItems. That sets the items to the same list of items that we had before, except we're filtering out the item that you were trying to remove.  [2:52](https://egghead.io/lessons/react-use-the-key-prop-when-rendering-a-list-with-react-b95caad8) This call to setItems is going to trigger a re-render of our app. The app is going to come down here. We're going to create these React elements. We're going to iterate over this list of items, which now has one less item than it had before. We're going to hand that off to React so that it can update the DOM appropriately.  [3:10](https://egghead.io/lessons/react-use-the-key-prop-when-rendering-a-list-with-react-b95caad8) The way React updates the DOM is it has a reference to the JSX elements that you gave it the last time it rendered this app component. It compares those React elements with the new React elements that you just returned at this time. Then it updates the DOM accordingly.  [3:24](https://egghead.io/lessons/react-use-the-key-prop-when-rendering-a-list-with-react-b95caad8) When you're giving it an array of React elements, unless React has some sort of identifier to know which element is which, it doesn't know whether you removed an element or maybe you added three and removed four or maybe you changed the order and just removed the first one.  [3:39](https://egghead.io/lessons/react-use-the-key-prop-when-rendering-a-list-with-react-b95caad8) It doesn't have any insight into what it is that you did to this array of React elements between the last time it rendered and this time. Any time you're rendering an array of React elements, you need to give it a key so that it can determine whether elements were removed, added, or modified.  [3:57](https://egghead.io/lessons/react-use-the-key-prop-when-rendering-a-list-with-react-b95caad8) That's why providing a unique key for each one of these items is going to fix this problem. Now, if we click remove on the grape, all of the inputs and their labels are correct.  [4:08](https://egghead.io/lessons/react-use-the-key-prop-when-rendering-a-list-with-react-b95caad8) You'll notice also that if you try to provide some-key where it's the same key for each one of these, you're going to get a warning there because it encountered two children with the same key. Keys need to be unique so that components maintain their identity across updates. Non-unique keys may cause children to be duplicated and/or omitted -- the behavior is unsupported and could change in a future version.  [4:32](https://egghead.io/lessons/react-use-the-key-prop-when-rendering-a-list-with-react-b95caad8) Hardcoding a specific key that's duplicated across these elements could lead to some very unexpected behavior, and we still experience that bug that we had before. The key that you provide for each element of this array of React elements needs to be unique to the item that you're rendering. Typically, that's going to be some sort of id, as in our case.  [4:56](https://egghead.io/lessons/react-use-the-key-prop-when-rendering-a-list-with-react-b95caad8) Another mistake that I see people make sometimes is they try to use the index as the key. While you get rid of the warning, you do not get rid of the bugs. That's because as React is comparing the previous version with the new version, what you're saying is the element that was at index four is actually now at index three, but React doesn't know that.  [5:17](https://egghead.io/lessons/react-use-the-key-prop-when-rendering-a-list-with-react-b95caad8) It maintains the state for this input to be the same as the one that was at index three the last time rather than being the one that was in index four last time. It's important that you keep the key as something that's unique to the item that this element in the array is representing.  [5:36](https://egghead.io/lessons/react-use-the-key-prop-when-rendering-a-list-with-react-b95caad8) Here's another little demo that I have at the bottom of this file. I'm just going to uncomment that. We'll save this. We'll get a refresh. In this example, we have those items. Every 1,000 milliseconds, we have this interval that shuffles those items and triggers a re-render with that shuffled version of the items. We randomize the items every time.  [5:57](https://egghead.io/lessons/react-use-the-key-prop-when-rendering-a-list-with-react-b95caad8) We have three versions of this, one that renders those items as inputs without a key, another that renders those items with inputs with the key of index, and then the last that renders those items with an appropriate key. You'll notice that they're all updating correctly, meaning that they're all jumping around the screen as they should, but the focus is not updating correctly.  [6:22](https://egghead.io/lessons/react-use-the-key-prop-when-rendering-a-list-with-react-b95caad8) I'm focused on apple right now. When that one moves, my focus doesn't go around with apple. Also, if I try to highlight one of these, then my highlight goes away as well. You'll notice that the With Key as Index suffers from the exact same problem. Even though it's not getting the warning in the console, you're still not fixing this bug.  [6:42](https://egghead.io/lessons/react-use-the-key-prop-when-rendering-a-list-with-react-b95caad8) Only when you have a key in there will your focus travel around with the input that it's associated with, because React is able to determine where to move the focus as your component updates.  [6:55](https://egghead.io/lessons/react-use-the-key-prop-when-rendering-a-list-with-react-b95caad8) In review, it's common in React to take an array and map that array to an array of elements and render that directly in your JSX. When you do that, it's important that you add a key to the root React element of each element in the array so that React can track changes over time and make sure that it preserves the state of each element in the array and its descendants. |
| **26. Подъем и размещение React State** | **25. Lifting and colocating React State** |
| Обычный вопрос от новичков в React - как разделить состояние между двумя родственными компонентами. Ответ состоит в том, чтобы [поднять состояние,](https://reactjs.org/docs/lifting-state-up.html) что в основном сводится к поиску самого низкого общего родителя, общего для двух компонентов, и размещению там управления состоянием, а затем передаче состояния и механизма обновления этого состояния компонентам, которые в нем нуждаются.  Как сообщество, мы неплохо справляемся с этим, и со временем это становится естественным. Одна вещь, которую мы обычно не запоминаем, - это вернуть состояние вниз (или [совместить состояние](https://kentcdodds.com/blog/state-colocation-will-make-your-react-app-faster) ). В этом уроке мы узнаем, как поднять состояние вверх и вернуть состояние вниз. | A common question from React beginners is how to share state between two sibling components. The answer is to [Lift the state](https://reactjs.org/docs/lifting-state-up.html) which basically amounts to finding the lowest common parent shared between the two components and placing the state management there, and then passing the state and a mechanism for updating that state down into the components that need it.  As a community we’re pretty good at doing this and it becomes natural over time. One thing that we typically have trouble remembering to do is to push state back down (or [colocate state](https://kentcdodds.com/blog/state-colocation-will-make-your-react-app-faster)). In this lesson we’ll learn how to lift state up and push state back down |
| Кент С. Доддс: [0:00] Здесь у нас есть простой компонент приложения, в котором мы управляем некоторым состоянием имени, чтобы мы могли передать его нашему компоненту имени и передать его нашему компоненту отображения, чтобы мы могли визуализировать, «Hey name. You are great».  [0:12] У нас также есть FavoriteAnimal, которое управляет своим собственным состоянием и отображает ввод с помощью значения и обработчика onChange.  [0:21] Допустим, этот компонент Display, который является родственником нашего компонента FavoriteAnimal, на самом деле должен знать, что это за животное, потому что вместо «You are great» мы хотим сказать: «Your favorite animal is..."а потом животное прямо здесь.  [0:36] Мы примем это как prop, но как мы собираемся получить доступ к этому animal, если это состояние находится в нашем компоненте FavoriteAnimal? Эти два являются родственными компонентами, поэтому компонент FavoriteAnimal не может передать animal Display компоненту.  [0:54] Давайте продолжим. Нам нужно будет принять animal, и оно откуда-то придет. Мы пока не знаем, где именно.  [1:00] Судя по всему, нам нужно будет сделать с нашим состоянием animal то же самое, что мы делаем с нашим состоянием name. В частности, нам нужно поднять состояние от компонента FavoriteAnimal до компонента App, который является наименее общим родителем между этими двумя компонентами.  [1:16] Давайте сделаем это. Я просто возьму это и перенесу сюда. Теперь FavoriteAnimal не имеет доступа к Animal или setAnimal. Собираюсь принять animal и onAnimalChange. Мы сделаем то же самое, что и для нашего компонента имени.  [1:37] OnAnimalChange будет передан в свойство onChange onAnimalChange. animal проходит ко входу. Затем мы переходим к FavoriteAnimal, и нам нужно передать animal и onAnimalChange.  [1:54] Вставим то, что было раньше. Теперь у нашего дисплея есть доступ к животному. Если мы сохраним это, все должно работать, кроме этого с ошибкой. Давайте займемся животным. Вот и все. Отлично. Теперь мы можем сказать, что любимое животное Мулан и Мулан - дракон, но особенно Mushu.  [2:16] То, что мы здесь сделали, называется состоянием подъема. Как правило, вы хотите, чтобы ваше состояние было как можно ближе к коду, который использует это состояние, поэтому у нас есть состояние в этом компоненте FavoriteAnimal. Когда у нас был вариант использования для одноуровневого компонента, чтобы иметь доступ к этому состоянию, мы должны подняться до этого состояния до наименее общего родителя, которым был этот App компонент.  [2:36] Мы поместили это прямо здесь, а затем мы передали это состояние механизму обновления состояния, который является нашей функцией setAnimal в качестве свойств для компонентов, которые в этом нуждаются.  [2:46] Здесь для нашего FavoriteAnimal мы передаем свойство animal и свойство onAnimalChange, поэтому у нас есть механизм для обновления этого состояния. Тогда здесь для дисплея все, что ему нужно, это животное, так что это все, что мы прошли.  [2:59] Как только вы научитесь это делать, это станет вашей второй натурой. Одна вещь, в которой мы не так хороши, - это отталкивать состояние назад или совмещать состояние. Предположим, что наш вариант использования Display был изменен, и нам больше не нужно передавать животное вниз. Мы собираемся сказать: «Вы молодцы!» очередной раз. Этот animal prop нам больше не нужен.  [3:18] Часто люди просто оставляют это прямо здесь, не внося никаких других изменений, но мы должны помнить, что нам больше не нужно принимать этот animal prop на том Дисплее, где мы ее визуализируем. Поскольку состояние животного используется только одним компонентом, это означает, что мы можем переместить это состояние обратно в этот компонент, чтобы разместить его вместе.  [3:38] Давайте вернем его сюда. Нам больше не нужно принимать ни один из этих props, и мы можем спуститься сюда и взять его. Удалите оба этих props. Затем мы вставим этот обработчик события onChange обратно в наш вход FavoriteAnimal onChange.  [3:55] Благодаря этому мы разместили наше состояние в одном месте, что упростило поддержку нашего приложения в долгосрочной перспективе. Подводя итоги, мы изучили, как поднять состояние, а затем вернуть его вниз с помощью совместного размещения состояний. Наш компонент приложения здесь отображает некоторое состояние, которое должно использоваться несколькими элементами.  [4:13] Наше приложение поддерживает состояние имени. У нас также есть FavoriteAnimal, которое поддерживает собственное состояние, но затем нашему компоненту Display необходимо было иметь доступ к этому состоянию, поэтому мы подняли состояние животного до наименее распространенного родителя, которым был наш компонент приложения.  [4:29] Затем мы передали это состояние и механизм обновления этого состояния компонентам, которым это нужно. Это была часть состояния подъема, а затем мы сделали обратное, чтобы вернуть состояние вниз для совместного размещения, переместив состояние обратно туда, где оно было до удаления состояния из компонента, который больше не нуждается в нем, и реквизит, который больше не нужен.  [4:51] Это улучшает как производительность, так и удобство сопровождения управления состоянием нашего приложения | Kent C. Dodds: [0:00] Here, we have a simple app component where we are managing some name state so we can pass that to our name component, and pass it to our display component, so that we can render out, "Hey name. You are great."  [0:12] We also have a FavoriteAnimal right here that's managing its own state and rendering the input with the value and onChange handler.  [0:21] Let's say that this Display component, which is a sibling to our FavoriteAnimal component actually needs to know what the animal is, because instead of, "You are great," we want to say, "Your favorite animal is..." and then animal right here.  [0:36] We'll accept that as a prop, but how are we going to get access to that animal, if that state is living in our FavoriteAnimal component? These two are sibling components, so the FavoriteAnimal component can't pass the animal to the Display component.  [0:54] Let's keep on going. We're going to need to accept an animal, and that's going to come from somewhere. We're not sure where yet.  [1:00] From the looks of things, we're going to need to do the same thing to our animal state that we're doing with our name state. That is specifically we need to lift the state from the FavoriteAnimal component to the App component which is the least common parent between these two components.  [1:16] Let's do that. I'm just going to grab this and I'll move that down to right here. Now the FavoriteAnimal doesn't have access to Animal or setAnimal. I'm going to accept animal and onAnimalChange. We'll do the same thing that we did for our name component.  [1:37] OnAnimalChange will be passed to the onChange prop onAnimalChange. The animal gets passed along to the input. Then we come down here to the FavoriteAnimal and we need to pass the animal and onAnimalChange.  [1:54] We'll paste in what we had before. Now, our Display has access to the animal. If we save that then everything should work except this is misspelled. Let's do animal. There we go. Great. Now we can say Mulan and Mulan's favorite animal is a dragon but specifically Mushu.  [2:16] What we did here is what's called lifting state. Typically, you want to have your state as close to the code that's using that state so that's why we have the state in this FavoriteAnimal component. When we had a use case for a sibling component to have access to that state, we have to lift to that state to the least common parent which was this App component.  [2:36] We put it right there and then we passed that state in to the mechanism for updating state which is our setAnimal function as props to the components that need it.  [2:46] Here for our FavoriteAnimal, we're passing the animal prop and onAnimalChange prop so we have a mechanism for updating that state. Then here for the Display, all it needs is the animal so that's all that we passed.  [2:59] Once you learn how to do this, it becomes second nature. One thing we're not quite as good at is pushing state back down or co-locating state. Let's say that our Display use case gets changed and we no longer need to pass the animal down. We're going to say, "You are great!" again. We no longer need this animal prop.  [3:18] Often, people will actually just leave it right there without making any other changes but we need to remember that we no longer need to accept this animal prop on this Display where we're rendering it. Because the animal state is only being used by a single component, that means we can move that state back into that component to get it co-located.  [3:38] Let's move it back up here. We no longer need to accept either one of these props and we can come down here, grab that. Remove both of these props. Then we'll paste that onChange event handler back into our FavoriteAnimal input onChange prop.  [3:55] With that, we've co-located our state making it easier to maintain our application in the long term. In review, what we did here was we explored how to lift a state and then push it back down with state co-location. Our App component here is rendering out some state which does need to be used by multiple elements.  [4:13] Our App is maintaining the name state. We also have this FavoriteAnimal which is maintaining its own state but then our Display component needed to have access to that state so we lifted the animal state up to the least common parent which was our App component.  [4:29] Then we passed that state and a mechanism for updating that state down to the components that needed those things. That was the lifting state part and then we did the reverse to push the state back down for state co-location by moving the state back to where it was before removing the state from the component that doesn't need it anymore, and removing the props that are no longer necessary.  [4:51] This enhances both the performance and the state management maintainability of our application |
| 26. Делайте HTTP-запросы с помощью React | 26. Make HTTP Requests with React |
| Наиболее полезные приложения React включают взаимодействие с сервером для загрузки и сохранения данных. Для этого в Интернете мы используем HTTP-запросы со [встроенным в браузер fetch API](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API)  (или вы можете использовать другую библиотеку с открытым исходным кодом, построенную на основе этого API). Подобные HTTP-запросы по своей природе асинхронны, и они также являются побочными эффектами, поэтому нам нужно управлять не только запуском запроса, но и тем, что мы должны показывать пользователю, пока запрос находится «в полете».  В этом уроке мы будем использовать общедоступный сервер GraphQL, который обслуживает данные покемонов для загрузки информации для данного имени покемона. Мы узнаем, как получить эти данные внутри **React.useEffect**callback и отобразить результаты, когда запрос завершится. | Most useful React applications involve interacting with a server to load and persist data. To do this on the web, we use HTTP requests with [the browser’s built-in fetch API](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API) (or you may use some other open source library that’s built on top of this API). HTTP requests like this are inherently asynchronous in nature and they’re also side-effects so we’ll need to manage not only starting the request, but also what we should show the user while the request is “in flight.”  In this lesson we’ll use a public GraphQL server that serves up pokemon data to load information for a given pokemon name. We’ll learn how to fetch that data inside a **React.useEffect** callback and display the results when the request completes. |
| Инструктор: [0:00] Здесь у нас есть приложение, которое управляет состоянием имени покемона, а затем мы выводим форму и вводим данные. Каждый раз, когда мы отправляем нашу форму, мы собираемся обновить имя покемона все равно, что пользователь ввел во входные данные.  [0:13] Когда это будет повторно отрисовано, мы собираемся отобразить информацию о покемоне с этим именем покемона, а затем информация о покемоне должна отобразить нашу информацию на основе этого покемона. Если мы введем Пикачу, мы должны запросить информацию о Пикачу.  [0:31] Нам нужно получить эту информацию, загрузив ее с сервера. У нас есть вспомогательная функция под названием fetch Pokémon, которая создает Pokémon GraphQL query. Затем мы используем Window.fetch для получения этого общедоступного API, в котором есть информация о покемонах.  [0:47] Мы публикуем сообщение. У нас есть правильные заголовки для приема JSON, затем мы сериализуем и указываем, что наше тело является строковой версией нашего запроса JSON. Переменные для нашего запроса - это имя покемона.  [1:03] Затем, когда этот promise разрешится, мы собираемся проанализировать ответ как JSON, и когда это будет сделано, мы получим этот объект ответа и уберем покемона с данных в ответе. Мы можем вызвать эту функцию, чтобы получить информацию об этом покемоне, которую мы получаем в качестве prop, но наш метод рендеринга здесь должен быть синхронным.  [1:27] Наша функция fetchPokémon асинхронная. Выполнение подобного HTTP-запроса - это побочный эффект, поэтому мы собираемся использовать the React useEffect hook. Мы скажем React.useEffect, и первое, что я хочу здесь сказать, это если нет pokémonName, они еще не ввели имя покемона или отправили пустое имя, тогда мы просто вернемся. Нам не нужно запрашивать об этом.  [1:52] В противном случае мы вызовем этот fetchPokémon с именем pokémonName, а затем в нашем успешном обработчике (handler) для этого promise мы получим Pokémon data и сможем установить какое-то состояние. Давайте продолжим и управляем некоторым состоянием для этого.  [2:07] Мы будем использовать состояние для Pokémon, назовем setPokémon и инициализируем его значением null. Давайте переименуем это из Pokémon в pokémonData. Затем мы можем вызвать setPokémon с помощью pokémonData.  [2:25] Здесь мы можем сказать, что если покемонов нет, то мы вернем просто ... чтобы указать на перезагрузку. В противном случае мы знаем, что у нас есть pokémonData, и поэтому можем отобразить их прямо здесь. Давайте сделаем это с помощью JSON. Я вставлю это в предварительную запись, чтобы мы получили правильный интервал. Скажем, JSON преобразовывает Pokémon, null, два, чтобы получить хороший интервал для JSON.  [2:51] Мы это сохраним. У нас есть ... прямо здесь. Поскольку покемонов нет, мы можем войти в Пикачу, отправить это, и мы получим информацию о Пикачу.  [3:01] Итак, чтобы сделать что-либо асинхронное, это побочный эффект, который должен произойти внутри useEffect callback. Для нашей информации о покемонах мы принимаем это имя pokémonName и в нашем callback, если не указано имя pokémonName, мы просто вернемся. В противном случае мы получим данные pokémonData с этим именем pokémonName.  [3:19] Затем, когда мы получим эти данные, мы обновим наше собственное состояние, чтобы иметь эти данные, которые будут запускать повторную визуализацию информации о покемонах. Мы можем вернуть строковую версию этого pokémonData. Мы могли бы применить здесь оптимизацию, чтобы убедиться, что этот useEffect запускается только тогда, когда мы этого хотим, поместив сюда pokémonName.  [3:39] Таким образом, этот useEffect будет запускаться повторно только при изменении имени покемона. Мы также могли бы добавить здесь оператор if, если не указано имя pokémonName, тогда мы могли бы сказать return, отправить Pokémon. После этого мы отправим покемона, а затем сможем ввести что-то вроде Пикачу. Отправьте это, и мы загрузим информацию о Пикачу. | Instructor: [0:00] Here we have an app that's managing a Pokémon name state, and then we're rendering out a form, and we're rendering an input. Every time we submit our form, we're going to update the Pokémon name to whatever the user typed in the input.  [0:13] When this is re-rendered, we're going to render the Pokémon info with that Pokémon name, and then the Pokémon info should render our information based on that Pokémon. If we type in Pikachu, then we should request in formation for Pikachu.  [0:31] We need to get that information by fetching it from a server. We have this helper function here called fetch Pokémon that creates a Pokémon GraphQL query. Then we're using Window.fetch to fetch this public API that has Pokémon information.  [0:47] We're making a post. We We have our proper headers to accept JSON then we serialize and we specify our body is a JSON stringified version of our query. The variables for our query which is the name of the Pokémon.  [1:03] Then when that promise resolves we're going to take the response parse it as JSON and when that's done, we'll get that response object and pluck the Pokémon off of the data in the response. We can call this function to get the information for this Pokémon that we get as a prop but our render method here has to be synchronous.  [1:27] Our fetchPokémon function is asynchronous. Making HTTP request like this is a side effect so we're going to use the React useEffect hook. We'll say React.useEffect and the first thing that I want to say here is if there's no pokémonName, they haven't entered a Pokémon name yet or they submitted an empty one then we'll simply return. We don't need to make a request for that.  [1:52] Otherwise, we'll call that fetchPokémon with the pokémonName and then in our success handler for this promise, we'll get the Pokémon data and we can set some state. Let's go ahead and manage some state for this.  [2:07] We'll have use state for Pokémon and we'll call setPokémon, and we'll initialize this to null. Down here, let's rename this from Pokémon to pokémonData. Then we can call setPokémon with the pokémonData.  [2:25] Down here we can say if no Pokémon then we'll return just a ... to indicate the reloading. Otherwise we know we have the pokémonData and so, we can render it right here. Let's JSON stringify this. I'll put that in a pre so we get the right spacing. Say, JSON stringify Pokémon, null, two to get nice spacing for JSON.  [2:51] We'll save this. We have a ... right here. Because there's no Pokémon then we can enter in Pikachu, submit that, and we get Pikachu's information.  [3:01] In review, to do anything asynchronous, that is a side effect that needs to happen inside a useEffect callback. For our Pokémon info, we accept that pokémonName and in our callback, if there's no pokémonName specified, then we'll simply return. Otherwise, we'll fetch the pokémonData with that pokémonName.  [3:19] Then when we get that data back, we'll update our own state to have that data that will trigger re-rendering of Pokémon info. We can return the stringified version of that pokémonData. We could apply an optimization here to make sure that this useEffect only runs when we want it to by putting pokémonName in here.  [3:39] That way, this useEffect will only rerun when the pokémonName changes. We could also add an if statement here for if there's no pokémonName supplied, then we could say return, submit a Pokémon. With that we get submit a Pokémon then we can type in something like Pikachu. Submit that and we'll get Pikachu's information loaded up. |
| Обработка ошибок HTTP с помощью React | Handle HTTP Errors with React |
| К сожалению, иногда серверный запрос не выполняется, и нам нужно показать пользователю полезное сообщение об ошибке. В этом уроке мы обработаем отклонение обещания, чтобы мы могли собрать эту информацию об ошибке, а также узнаем, как лучше всего отображать управление состоянием нашего запроса, чтобы у нас был детерминированный метод рендеринга, чтобы всегда показывать пользователю правильная информация, основанная на текущем состоянии нашего компонента React.  Распространенная ошибка, которую допускают люди, - создать переменную состояния с именем **isLoading**и установить для нее значение **true**или **false**. Вместо этого мы будем использовать **status**переменную , которая может быть установлена на **idle**, **pending**, **resolved**, или **rejected**. Вы можете узнать больше о том, почему это важно, из [Stop using isLoading booleans](https://kentcdodds.com/blog/stop-using-isloading-booleans) . | Unfortunately, sometimes a server request fails and we need to display a helpful error message to the user. In this lesson we’ll handle a promise rejection so we can collect that error information, and we’ll also learn how we can best display manage the state of our request so we have a deterministic render method to ensure we always show the user the proper information based on the current state of our React component.  A common mistake people make is to create a state variable called **isLoading** and set that to **true** or **false**. Instead, we’ll be using a **status** variable which can be set to **idle**, **pending**, **resolved**, or **rejected**. You can learn more about why this is important from [Stop using isLoading booleans](https://kentcdodds.com/blog/stop-using-isloading-booleans). |
| Инструктор: [0:00] А что будет, если возникнет какая-то ошибка сервера или, возможно, мы сделали запрос неправильно? Давайте посмотрим здесь.  [0:06] Давайте перейдем к нашему запросу. Сделаем опечатку. Мы скажем «Nam» вместо «Name». Мы это сохраним.  [0:12] Когда я набираю здесь, я могу сказать «Mew», а затем отправить. Как пользователь, я просто хочу увидеть это "...". Это вообще бесполезно. Давайте посмотрим, что здесь происходит в наших инструментах разработчика. Я пойду и обновлю.  [0:26] Мы снова наберем Mew. Очистим нашу вкладку сети. Затем нажимаем "Отправить". Мы получим сетевую ошибку, указывающую на то, что мы не можем запросить имя поля для типа Pokémon. Предлагается вопрос: «Did you mean name?»  [0:41] Конкретная ошибка не имеет значения. Нам просто нужно показать пользователю что-то более полезное, чем оставлять его в состоянии загрузки навсегда.  [0:49] Давайте вернемся сюда, прежде чем исправим наш код. Давайте добавим некоторое состояние для состояния ошибки. Мы скажем setError. Мы инициализируем это значение null. Тогда мы скажем, если есть ошибка, и вернемся: «О, нет ...»  [1:06] В реальном приложении, возможно, вы были бы более полезны, чем это. Давайте добавим здесь обработчик ошибок в качестве второго аргумента к нашему вызову then. Это будут наши данные об ошибках.  [1:17] Мы скажем setError с данными об ошибке. Тогда мы могли бы это представить. Мы снова введем имя покемона. Мы видим: «О, нет». Теперь нам нужно попробовать еще раз.  [1:29] Проблема, с которой мы столкнемся сейчас, заключается в том, что если пользователь попытается снова и сервер будет успешным, то способ, которым мы структурируем наш код, не будет работать с этим. Что я собираюсь сделать, так это добавить здесь новое состояние для статуса.  [1:42] У нас будет setStatus. Мы инициализируем этот статус как бездействующий (to idle), что означает, что сейчас информация о покемонах не делает ничего полезного.  [1:51] Тогда мы можем сказать, что статус простоя (idle status ) - это когда мы хотим отобразить submitAPokemon. Если статус неактивен (is idle), мы скажем «submitAPokemon». Когда мы начинаем получать нового покемона, мы можем сказать: "setStatus pending."  [2:10] Затем мы можем получить представление о том, что статус ожидания (is pending), затем мы хотим вернуть «...», чтобы указать пользователю, что мы ожидаем рассмотрения (we're pending). Когда у нас есть успешный запрос, мы можем сказать: «setStatus to resolved».  [2:26] Если мы решим проблему, мы хотим вернуть данные о покемонах. Если статус разрешен, мы вернем данные о покемонах в предварительном теге сюда.  [2:38] Затем, если возникнет ошибка, мы скажем: «setStatus отклонен». Здесь внизу мы скажем: «Статус отклонен». Затем мы отобразим «О, нет».  [2:50] Теперь наш метод рендеринга очень предсказуем. Мы всегда знаем, когда что будет рендерить наш компонент. Если я попытаюсь ввести Mew еще раз, мы увидим: «Сначала отправьте покемона».  [3:01] Мы увидим «...», пока он не будет принят. Тогда мы увидим «О, нет». когда он был отклонен.  [3:06] Мы исправим нашу опечатку прямо здесь. Мы это сохраним. Теперь мы видим «Отправить покемона», потому что мы бездельничали (we're idle). Мы введем Мяу. Нажмите "Отправить".  [3:15] Мы получаем это "..." Затем мы видим детали Мяу. Если мы попробуем другого Покемона, то мы увидим «...» Мы увидим детали этого Покемона.  [3:25] Давайте продолжим и рассмотрим. Мы добавили некоторую обработку ошибок, создав некоторое управление состояниями ошибок. Мы добавили обработчик ошибок в нашу цепочку обещаний. Если мы получим некоторые данные об ошибках, мы собираемся установить эти данные об ошибках, чтобы мы могли отображать что-то полезное для пользователя, указывающее на то, что возникла проблема.  [3:42] Чтобы избежать некоторых ошибок состояния, мы добавили состояние состояния, чтобы мы могли начать с ожидания. Когда мы начнем получать покемона, мы можем установить его в режим ожидания. Когда мы получаем покемона, мы можем установить его как разрешенный, или, если есть сбой в получении покемона, мы устанавливаем его как отклоненный. Это помогает нам избежать ошибок. | Instructor: [0:00] Now what would happen if there was some sort of server error or maybe we made the request incorrectly? Let's take a look here.  [0:06] Let's go down to our query. We'll make a typo. We'll say, "Nam" instead of "Name." We'll save that.  [0:12] When I type in here, I can say, "Mew" and then submit. As a user, I'm just going to see this "...". That's not going to be useful at all. Let's see what's going on here in our developer tools. I'll go ahead and refresh.  [0:26] We'll type Mew again. Let's clear out our network tab. Then we hit submit. We're going to get a network error indicating that we cannot query field nam on type Pokémon. It gives a suggestion for, "Did you mean name?"  [0:41] The specific error is beside the point. We just need to show the user something a little bit more useful than leaving them in a loading state forever.  [0:49] Let's come back up here before we fix our code. Let's add some state for the error state. We'll say setError. We'll initialize that to null. Then we'll say if there's an error, then we'll return, "Oh, no..."  [1:06] In a real application, maybe you'd be a little bit more helpful than that. Let's add an error handler here as a second argument to our then call. This will be our error data.  [1:17] We'll say setError with the error data. Then we could submit that. We'll type the Pokémon name again. We see, "Oh, no." Now we need to try again.  [1:29] The problem that we'll face now is that if the user does try again and the server is successful, the way that we have our code structured is not going to work with that. What I'm going to do is I'm going to add a new state here for status.  [1:42] We'll have setStatus. We'll initialize that status to idle, meaning right now the Pokemon info is not doing anything useful.  [1:51] Then we can say that the idle status is when we want to render submitAPokemon. If the status is idle, then we'll say, "submitAPokemon." When we start fetching a new Pokémon, then we can say, "setStatus pending."  [2:10] Then we can have this represent if the status is pending, then we want to return a "..." to indicate to the user that we're pending. When we have a successful request, then we can say, "setStatus to resolved."  [2:26] If we're resolved, then we want to return the Pokémon data. If the status is resolved, then we'll return the Pokémon data in a pre tag here.  [2:38] Then if there's an error, we'll say, "setStatus rejected." Down here, we'll say, "Status is rejected." Then we'll render, "Oh, no."  [2:50] Now our render method is very predictable. We always know when our component is going to render what. If I try to type in Mew again, we're going to see, "Submit a Pokémon first."  [3:01] We'll see "..." while it's pending. Then we'll see "Oh, no." when it's been rejected.  [3:06] We'll fix our typo right here. We'll save that. Now we see "Submit a Pokémon" because we're idle. We'll type in Mew. Hit submit.  [3:15] We get that "..." Then we see Mew's details. If we try another Pokémon, then we'll see a "..." We'll see that Pokémon's details.  [3:25] Let's go ahead and review. We added some error handling by creating some error state management. We added an error handler to our promise chain. If we get some error data, then we're going to set that error data so that we can render something useful to the user indicating that there's been a problem.  [3:42] To avoid some state bugs, we added a status state so that we could start out with idle. When we start fetching the Pokémon, we can set it to pending. When we get the Pokémon, we can set it to resolved, or if there's a failure in getting the Pokémon, then we set it to rejected. That helps us to avoid bugs. |
| Установите и используйте React DevTools | Install and use React DevTools |
| Идти одному опасно. Вот возьми это: [React DevTools](https://chrome.google.com/webstore/detail/react-developer-tools/fmkadmapgofadopljbjfkapdkoienihi?hl=en) . Расширение React Developer Tools чрезвычайно полезно при разработке и отладке приложений React. В этом уроке мы установим его и поэкспериментируем с некоторыми функциями. Определенно найдите время, чтобы поиграться с DevTools самостоятельно. Вы будете их часто использовать. | It’s dangerous to go alone. Here take this: [React DevTools](https://chrome.google.com/webstore/detail/react-developer-tools/fmkadmapgofadopljbjfkapdkoienihi?hl=en). The React Developer Tools extension is extremely useful in developing and debugging React applications. In this lesson we’ll get it installed and play around with some of the features. Definitely take some time to play around with the DevTools on your own. You’ll be using these a lot. |
| Инструктор: [0:00] Здесь я могу ввести имя покемона, например Пикачу. Я подчиняюсь. У меня есть данные о Пикачу. Если я сделаю здесь опечатку, то ничего не верну. Мне может быть полезно понять, что происходит за кулисами в состоянии компонентов, которые я визуализирую на этой странице, чтобы я мог отлаживать проблемы, возникающие в моих компонентах.  [0:22] Я собираюсь использовать Google React DevTools. В результате появится Интернет-магазин Chrome для React DevTools. Также есть React DevTools, поддерживаемые для других браузеров. Я использую Chrome, поэтому собираюсь добавить его в Chrome. Щелкните «Добавить расширение». Затем в Chrome были добавлены инструменты разработчика React. Я могу использовать это, нажав на этот значок.  [0:47] Когда я вернусь в свое приложение, я обновлюсь здесь. Мы увидим, что значок расширения Chrome для React DevTools стал красным. Это указывает нам на то, что React находится на этой странице, и мы используем разрабатываемую сборку React.  [1:01] Когда мы используем производственную сборку React, это будет выглядеть немного иначе. Мы можем убедиться в этом, перейдя на twitter.com, потому что Twitter построен с использованием React. Здесь на этой странице используется производственная сборка React. Если мы закроем это, и теперь я открою свои DevTools, я могу пойти и проверить здесь, или я могу перейти сюда, в Дополнительные инструменты и Инструменты разработчика.  [1:26] Здесь откроются Инструменты разработчика. Если я щелкну здесь и посмотрю на компоненты, то это покажет мне компоненты, которые управляют этой страницей. Я вижу, что у нас есть компонент приложения и компонент информации о покемонах.  [1:42] Под компонентом приложения я вижу все ловушки, которые используются для работы этого компонента приложения. У нас есть некоторое состояние, которое является пустой строкой. Когда я ввожу здесь и нажимаю «Отправить», теперь мы видим, что это состояние - это то, что я ввел. Мы даже можем изменить это значение здесь. Мы можем нажать «Отменить», если хотим отменить это изменение, или нажать «Ввод», чтобы зафиксировать это значение.  [2:05] Теперь это значение подтверждено. Вы увидите, что мы получаем значение null, потому что по имени значения нет покемонов. Если мы изменим это на Mew и нажмем Enter, то мы получим информацию Mew. На стороне информации о покемонах мы видим, что имя покемона поступает через реквизиты. Фактически мы можем изменить эту prop на Mewtwo. Поменяйте там эту prop.  [2:28] Затем мы также можем увидеть hooks, которые используются компонентом информации о покемонах. У нас состояние разрешено. У нас также есть еще несколько состояний для самой информации о покемонах и еще несколько состояний. Это для нашего состояния ошибки. Мы видим, что этот хук тоже имеет эффект.  [2:45] У нас есть много вариантов. Мы можем нажать на это. Это покажет мне исходный код для этого конкретного компонента. Если мы вернемся к нашим компонентам здесь и нажмем на эту кнопку, то информация об этом компоненте будет записана в консоль.  [2:59] Мы можем щелкнуть по нему, чтобы развернуть. Мы увидим реквизиты, хуки и корневой узел DOM, который визуализируется этим компонентом. Мы вернемся сюда снова. Если хотите, можете перетащить его, так как мы собираемся его много использовать. Я собираюсь переместить это туда.  [3:14] У нас также есть эта кнопка для проверки соответствующего элемента DOM. Это приведет нас прямо к нашей вкладке элементов здесь с выделенным элементом. Тогда это для саспенса. Если вы используете какие-либо компоненты React Suspense, вы можете протестировать их здесь, приостановив выбранный компонент. У нас также есть несколько вариантов темы и способа ее отображения.  [3:37] Кроме того, у нас есть несколько вариантов просмотра для вкладки компонентов, где мы можем отфильтровать различные типы компонентов или даже отфильтровать их по имени. Если бы мы хотели избавиться от этого приложения, мы могли бы ввести его. Теперь наше приложение больше не отображается. Я хочу это показать, поэтому сниму. Теперь отображается наше приложение.  [3:59] У нас также есть вкладка профилировщика, которая поможет нам отлаживать проблемы с производительностью в нашем приложении. Мы можем включить запись во время рендеринга каждого компонента. Мы также можем скрыть коммиты ниже определенного порога. Если фиксация выполняется действительно быстро - возможно, для обновления DOM требуется всего пять миллисекунд - тогда мы можем включить это.  [4:16] Теперь я могу перейти к своему профайлеру. Я тоже перетащу сюда. Мы можем нажать эту кнопку, чтобы начать профилирование там, где мы находимся, или мы можем нажать эту кнопку, чтобы перезагрузить и начать профилирование при загрузке страницы. Мы нажмем на это. Затем мы можем нажать кнопку «Стоп». Мы увидим, что нет данных, соответствующих текущим критериям фильтра.  [4:35] Давайте продолжим, и мы снимем галочку с этой кнопки. Теперь мы видим, что произошла фиксация. Это произошло очень быстро. Эта фиксация была сделана для первоначального рендеринга нашего компонента. У нас есть наше приложение и информация о покемонах. В нем говорится, что это было визуализировано, потому что это было впервые.  [4:51] Теперь давайте продолжим и сделаем еще одну запись. Мы скажем Мяу. Мы отправим это и нажмем «Стоп». Здесь мы увидим пару коммитов. Для этого он был повторно отрисован, потому что изменились хуки и изменились реквизиты.  [5:05] Для этого он был повторно отрисован, потому что изменились хуки. В этом коммите он перерисовался, потому что перехватчики снова изменились, поэтому данные вернулись из запроса Pokémon. Затем это было повторно отрисовано, потому что мы обновили наше состояние.  [5:19] React DevTools предоставляет нам эти компоненты и эту вкладку профилировщика в наших инструментах разработчика, чтобы мы могли лучше понять, что делают наши компоненты во время их разработки. Затем у нас также есть этот индикатор, который сообщает нам, когда мы используем сборку разработки React по сравнению с производственной сборкой, чтобы мы могли убедиться, что наше производственное приложение использует производственную сборку.  [5:40] И последний совет от профессионалов, который я хочу вам показать: если вы откроете консоль, когда у вас открыта вкладка компонентов - вы можете сделать это, нажав клавишу выхода - тогда наша консоль будет здесь. Вы можете ввести $r. Это даст вам hooks, реквизиты и тип компонента, который вы выбрали в данный момент.  [5:58] Мы можем нажать на приложение. Мы можем снова сделать $r. Теперь мы увидим the hooks, the props, and the type выбранного нами компонента. На самом деле это очень похоже на вкладку элементов, где вы можете сделать $.  [6:09] Это даст вам узел DOM, который вы в данный момент выбрали на вкладке элементов. На вкладке компонентов у вас есть $r. Это даст вам информацию о компоненте, который вы выбрали в данный момент.  [6:22] Я настоятельно рекомендую вам установить React DevTools и поэкспериментировать с ними, чтобы вы познакомились с их возможностями и повысили продуктивность разработки приложений React. | Instructor: [0:00] Here, I can fill out a Pokémon name like Pikachu. I submit. I get some Pikachu data. If I make a typo here, then I'm not going to get anything back. It might be useful for me to get an understanding of what's going on behind the scenes in the state of the components that I'm rendering to this page so that I can debug the problems that are happening in my components.  [0:22] What I'm going to do is I'm going to Google React DevTools. That'll come up with the Chrome Web Store for React DevTools. There's also React DevTools supported for other browsers. I'm using Chrome, so I'm going to add this to Chrome. Click on "Add Extension." Then React Developer Tools has been added to Chrome. I can use that by clicking on that icon.  [0:47] When I come back to my app, I'll go ahead and refresh here. We'll see that Chrome extension icon for React DevTools has turned red. That's indicating to us that React is on this page and we're using the development build of React.  [1:01] When we use the production build of React, then this will look a little differently. We can see that by going to twitter.com because Twitter is built using React. Here, this page is using the production build of React. If we close this and now I open up my DevTools, I can go inspect here, or I can go here to More Tools and Developer Tools.  [1:26] That'll open up the Developer Tools here. If I click here and look at components, then this is going to show me the components that are driving this page. I can see that we have an app component and we have a Pokémon info component.  [1:42] Under the app component, I can see all of the hooks that are being used for this app component to work. We have some state here that is an empty string. As I type in here and click submit, now we see that that state is what I typed. We can even change that value here. We can click undo if we want to undo that change, or we can hit enter to commit that value.  [2:05] Now that value has been committed. You will see that we get null is printed because there is no Pokémon by the name of value. If we change this to Mew and hit enter, then we're going to retrieve Mew's information. On the Pokémon info side, we can see the Pokémon name is coming in via props. We can actually change this prop to Mewtwo. Change that prop there.  [2:28] Then we can also see the hooks that are being used by the Pokémon info component. We have state is resolved. We also have some more state for the Pokémon information itself and some more state. This is for our error state. We can see this hook also has an effect.  [2:45] There are a lot of options that we have here. We can click on this. That will show me the source code for this particular component. If we go back to our components here and click on this button, then that will log this component's information to the console.  [2:59] We can click that to expand it. We'll see the props, the hooks, and the root DOM node that's being rendered by this component. We'll come back here again. If you want to, you can actually drag this over, since we're going to be using this a lot. I'm going to move that over there.  [3:14] We also have this button to inspect the matching DOM element. That'll take us straight to our elements tab here with that element highlighted. Then this is for Suspense. If you're using any React Suspense components, you can test those out here by suspending the selected component. We also have some options here for the theme and the way that this is displayed.  [3:37] Then we also have some view options for our components tab, where we can filter out different types of components or ever filter them out by their name. If we wanted to get rid of that app, then we could type that in. Now our app is no longer being shown. I do want to show that, so I'll remove that. Now our app is being shown.  [3:59] We also have a profiler tab that will help us debug performance issues in our app. We can enable record while each component is rendering. We can also hide commits below a certain threshold. If a commit is really fast -- maybe it only takes five milliseconds to update the DOM -- then we can enable that.  [4:16] Now I can go to my profiler. I'll drag that over here as well. We can hit this button to start profiling where we are, or we can hit this button to reload and start profiling when the page loads. We'll click on that. Then we can click stop. We'll see that there's no data matching the current filter criteria.  [4:35] Let's go ahead and we'll uncheck this button. Now we can see that there was a commit. It just happened really fast. That commit was for the initial render of our component. We have our app and our Pokémon info. It says that this was rendered because it's the first time that it was rendered.  [4:51] Now let's go ahead and do another recording. We'll say Mew. We'll submit that and then hit stop. We'll see a couple of commits here. For this one, it re-rendered because hooks changed and props changed.  [5:05] For this one, it re-rendered because hooks changed. On this commit, it re-rendered because hooks changed again, so the data came back from the Pokémon request. Then this re-rendered because we updated our state.  [5:19] The React DevTools gives us this components and this profiler tab in our Developer Tools to give us a lot of insight into what our components are doing while we're developing them. Then we also have this indicator up here to tell us when we're using the development build of React versus the production build so we can make sure that our production application uses the production build.  [5:40] One last pro tip I want to show you is if you open up the console while you have the components tab open -- you can do that by hitting the escape key -- then we have our console here. You can type $r. That will give you the hooks, the props, and the type for the component that you have currently selected.  [5:58] We can click on the app. We can do $r again. Now we'll see the hooks, the props, and the type for that component that we have selected. This is actually very similar to the elements tab, where you can do a $.  [6:09] That will give you the DOM node that you have currently selected in the elements tab. In the components tab, you have a $r. That will give you information on the component that you have currently selected.  [6:22] I strongly advise that you install the React DevTools and play around with them so you become familiar with their capabilities and make you more productive developing React applications. |
| Создайте и разверните приложение React с помощью Codesandbox, GitHub и Netlify | Build and deploy a React Application with Codesandbox, GitHub, and Netlify |
| Я создал и развернул несколько приложений, не загружая и не редактируя ни одной строчки кода на свой локальный компьютер, и сделал это с помощью [Codesandbox](https://codesandbox.io/) , [GitHub](https://github.com/) и [Netlify](https://www.netlify.com/) ( [подробнее](https://kentcdodds.com/blog/building-production-apps-100-in-the-browser) ). В этом уроке я покажу вам некоторые из этих инструментов, и мне не терпится увидеть, что вы делаете с этими удивительными инструментами и своими новыми знаниями о React. | I’ve built and deployed several applications without downloading or editing a single line of code to my local computer, and I did it with [Codesandbox](https://codesandbox.io/), [GitHub](https://github.com/), and [Netlify](https://www.netlify.com/) ([learn more](https://kentcdodds.com/blog/building-production-apps-100-in-the-browser)). In this lesson I’ll show you around some of these tools, and I can’t wait to see what you do with these amazing tools and your new found knowledge of React. |
| Кент С. Доддс: [0:00] После того, как вы выяснили, как React работает локально в index.html, вы, вероятно, захотите создать реальное приложение, и лучше всего начать с codeandbox.io.  [0:11] CodeSandbox великолепен. Это полноценная IDE и инструмент для создания прототипов для быстрой веб-разработки, встроенный прямо в браузер. Это похоже на типичную среду разработки, которую можно ожидать от локальной разработки.  [0:26] Давайте создадим Sandbox (песочницу). Я собираюсь создать здесь Sandbox React из шаблона React. Здесь справа наш браузер. Слева у нас есть Sandbox. Я могу сказать: «Здравствуйте!» и он просто обновляется автоматически по мере продвижения.  [0:44] В CodeSandbox можно делать массу вещей. Вы можете добавить зависимости от npm, вы можете добавить внешние ресурсы, вы даже можете изменить свой шрифт. Если мы нажмем Command и запятую, вы можете открыть здесь свои настройки и изменить их так же, как если бы вы использовали VS Code локально. Это действительно потрясающе.  [1:05] Здесь вы можете создать свое приложение целиком. Вы можете иметь несколько файлов и использовать операторы и модули импорта. Вы даже можете писать здесь тесты.  [1:12] Мы могли бы создать новый каталог с именем **tests** , а затем в этом каталоге у вас мог бы быть файл test.js. Затем мы щелкаем здесь вкладку «Тест» и запускаем все тесты, которые есть здесь. Можно сказать «тест» («это работает»). Затем наши тесты будут запущены.  [1:33] Еще одна вещь, которую вы можете сделать с этим, - вы можете подключиться к этому с помощью GitHub. Я собираюсь войти в GitHub прямо здесь. Я авторизую CodeSandbox, а затем мы войдем прямо здесь, чтобы авторизовать CodeSandbox для создания репозиториев. Затем я могу создать новый репозиторий example -codeandbox. Мы нажмем «Создать репозиторий», и CodeSandbox создаст репозиторий на GitHub на основе того, что у меня есть прямо здесь.  [2:03] Теперь я могу перейти к информации здесь, и я вижу, что есть ссылка. Я могу нажать на это. Это открывается в новом репозитории GitHub, в котором есть весь код, который у меня был для этого CodeSandbox.  [2:15] Чтобы сделать еще один шаг вперед, я могу перейти к этому варианту развертывания и развернуть его с помощью Now или netlify. Давайте продолжим и попробуем развернуть это с помощью netlify. Мы нажмем «Развернуть», и тогда я получу сайт песочницы. Теперь netlify создает мой CodeSandbox. Я могу посмотреть логи здесь. Это займет минуту или две, поэтому я ускорю это для вас, чтобы вам не пришлось ждать.  [2:45] Похоже, он развернут. Я могу просмотреть все эти журналы и посмотреть, что здесь сделано, а затем нажать «Посетить». Это откроет сайт в другой вкладке. Он доступен во всемирной паутине, поэтому вы можете отправить этот URL-адрес кому угодно, и они смогут увидеть потрясающую работу, созданную вами в CodeSandbox.  [3:05] Вы также можете заявить права на этот сайт, чтобы связать его со своей учетной записью Netlify и даже подключить его к имеющемуся у вас репозиторию GitHub, поэтому каждый раз, когда вы вносите изменения в репозиторий GitHub, вы можете получить свой сайт redeployed. В Netlify вы даже можете создать собственный домен и, таким образом, вы можете создать целое приложение, используя только CodeSandbox, Netlify и GitHub.  [3:27] О, и когда вы вносите изменения, например, если мы говорим «Привет!», CodeSandbox автоматически разветвляет вашу песочницу, и вы можете фиксировать изменения в репозитории GitHub, к которому прикреплен этот CodeSandbox. Мы могли бы сказать «изменить заголовок», мы могли бы зафиксировать это напрямую или даже открыть PR.  [3:49] Давайте продолжим и сделаем коммит, он создаст коммит и поместит его в наше репо. Теперь мы видим, что заголовок изменения был последней фиксацией. Если мы подключим Netlify к нашему репозиторию GitHub, Netlify также автоматически deploy это изменение.  [4:05] Вы можете работать полностью в CodeSandbox, или, если вы предпочитаете работать локально, вы также можете экспортировать это в zip, разархивировать и работать с ним локально, или вы можете клонировать этот репозиторий и работать с ним локально также.  [4:19] Есть целый мир вещей, которые вы можете делать с React, когда эти инструменты у вас под рукой. Я с нетерпением жду возможности увидеть классные вещи, которые вы создаете, чтобы сделать мир лучше, с вашими новообретенными знаниями React. | Kent C. Dodds: [0:00] Once you figured out how React works locally in an index.html, you probably want to build an actual application, and the best next place to start is by going to codesandbox.io.  [0:11] CodeSandbox is amazing. It's an entire IDE and prototyping tool for rapid web development built right into the browser. It's just like a typical development environment that you'd expect from a local development.  [0:26] Let's go ahead and create a Sandbox. I'm going to create a React Sandbox here from the React template. Here on the right we have our browser. On the left we have our Sandbox. I can say, "Hello You!" and it just gets updated automatically as we go.  [0:44] There are a ton of things that you can do in CodeSandbox. You can add dependencies from npm, you can add external resources, you can even change your typeface. If we hit Command and a comma, you can open up your settings here and change the settings just like you would if you were using VS Code locally. It's really amazing.  [1:05] You can build your whole app out here. You can have multiple files and use import statements and modules. You can even write tests in here.  [1:12] We could create a new directory called **tests**, and then in that directory, you could have a test.js file. Then, we click on the Test tab, here, and we get to run all the tests that we have in here. We can say test('this works'). Then, our tests will be run.  [1:33] Another thing you can do with this is you can connect to this with GitHub. I'm going to go ahead and sign in with GitHub, right here. I'll authorize CodeSandbox and then we'll sign in right here to authorize CodeSandbox to create repos. Then, I can create a new repository, example-codesandbox. We'll click Create Repository and CodeSandbox will create a repository on GitHub based on what I have right here.  [2:03] With that, now, I can go to the information here and I see that there's a link. I can click on that. That opens up in this new GitHub repo that has all of the code that I had for that CodeSandbox.  [2:15] To take it a step even further, I can go to this deployment option and I can deploy this with Now or netlify. Let's go ahead and try deploying this with netlify. We'll click Deploy, and then I get a Sandbox Site. Now, netlify is building my CodeSandbox. I can look at the logs here. It's going to take a minute or two to complete, so I'll speed that up for you, so you don't have to wait.  [2:45] It looks like it's been deployed. I can scroll through all these logs and take a look at what it's done here and then I can click Visit. That will pop open the site in another tab. This is available on the World Wide Web, so you can send this URL to anyone and they can see the amazing work that you created within CodeSandbox.  [3:05] You can also claim this site so you can make it associated to your Netlify account and even have it wired up to the GitHub repo that you have, so anytime you push changes to this GitHub repo, you can get your site redeployed. Within Netlify, you can even make a custom domain and so you could create an entire application using just CodeSandbox, Netlify and GitHub.  [3:27] Oh, and as you make changes, for example if we say, "Hello There!", then CodeSandbox will automatically fork your sandbox and you can commit changes to the GitHub repo that this CodeSandbox is attached to. We could say change title, we could commit it directly or even open a PR.  [3:49] Let's go ahead and commit, it'll create the commit and put it in our repo here. Now we see change title was the latest commit. If we wire up Netlify to our GitHub repo, then Netlify will automatically deploy that change as well.  [4:05] You can work entirely in CodeSandbox, or, if you'd rather work locally, then you can also export this to a zip, unzip that and work through it locally or you can clone this repository and work through it locally as well.  [4:19] There's an entire world of things that you can do with React once you have these tools at your fingertips. I look forward to seeing the cool things that you build to make the world a better place with your newfound knowledge of React. |
| Руководство для начинающих по React Outro | A Beginners Guide to React Outro |
| ент С. Доддс: [0:00] Вы хорошо провели время? Я хорошо провел время.  [0:02] Я надеюсь, что это дало вам основу, которая вам нужна, если вы только начинаете работать с React, чтобы вы могли опираться на это и получать потрясающие впечатления от React. Если вы какое-то время использовали React и пришли сюда только для того, чтобы укрепить свое понимание основ, я надеюсь, что это помогло и вам.  [0:18] Мне нравится React, потому что я считаю, что он может сделать мир лучше, делая нас более продуктивными в создании приложений, которые мы создаем для наших пользователей. Я надеюсь, что вы воспользуетесь возможностью получить потрясающий опыт работы с этой замечательной библиотекой, которая у нас есть, которую мы любим, и которую мы называем React. | Kent C. Dodds: [0:00] Did you have a good time? I had a good time.  [0:02] I hope this gave you the foundation that you need if you're just getting started with React, so that you can build upon this and make amazing experiences with React. If you've been using React for a while and you just came here to solidify your understanding of the fundamentals, I hope this did that for you, too.  [0:18] I'm excited about React because I think that it can make the world a better place by making us more productive in building the applications that we're building for our users. I hope that you take the opportunity to build some amazing experiences with this amazing library that we have, that we love, that we call React. |
|  |  |