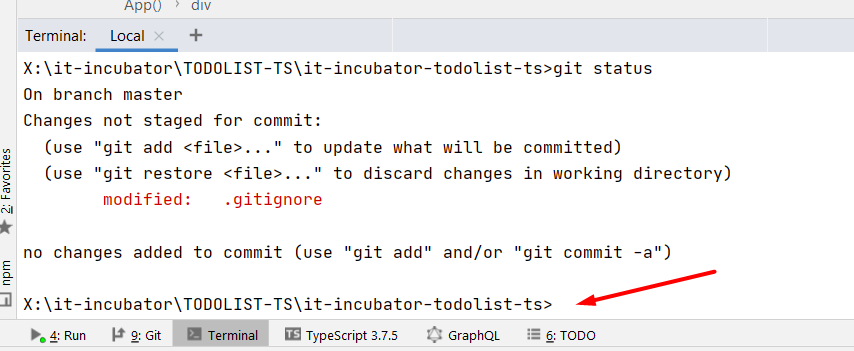
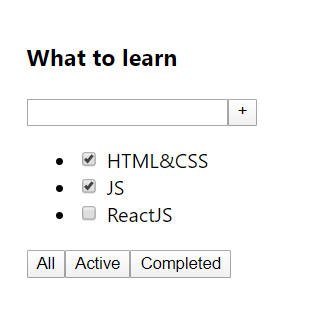
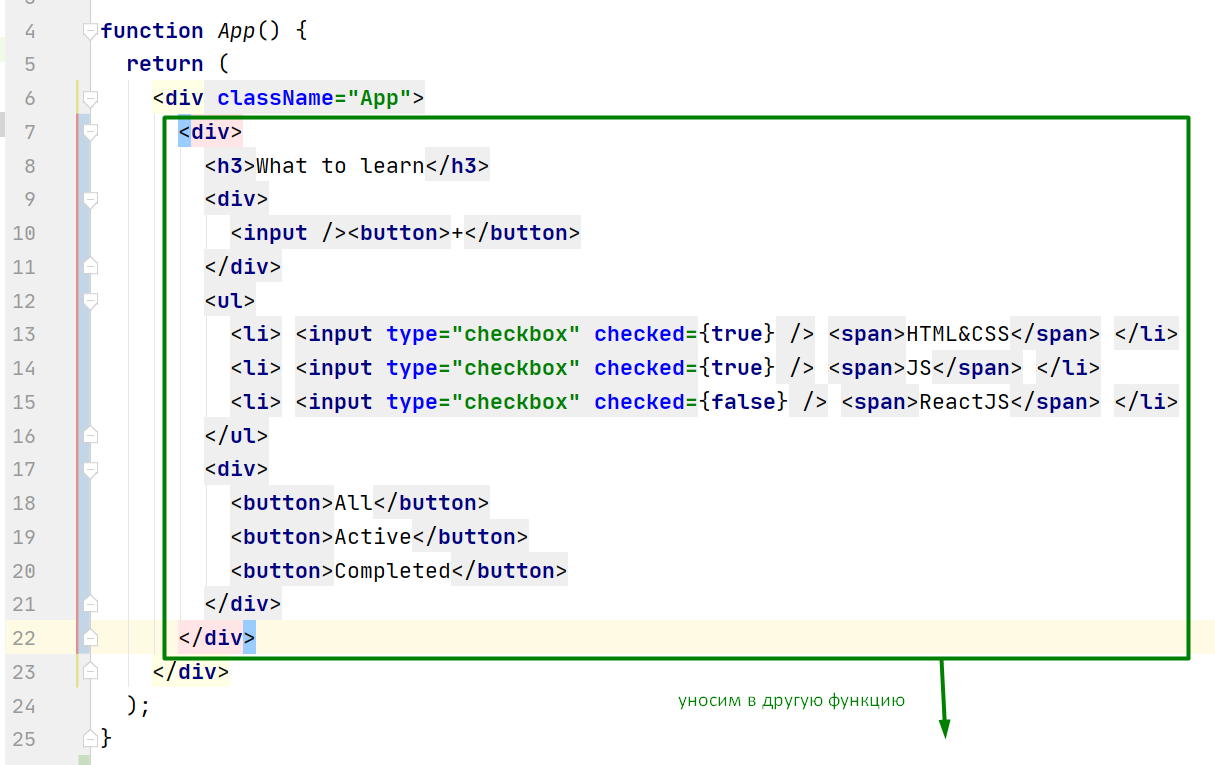
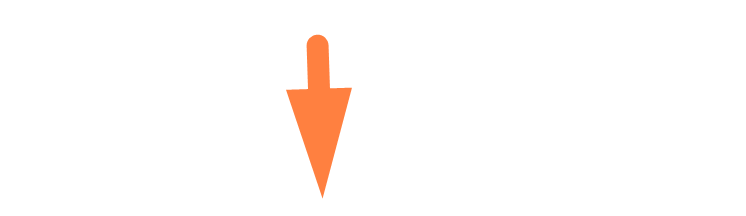
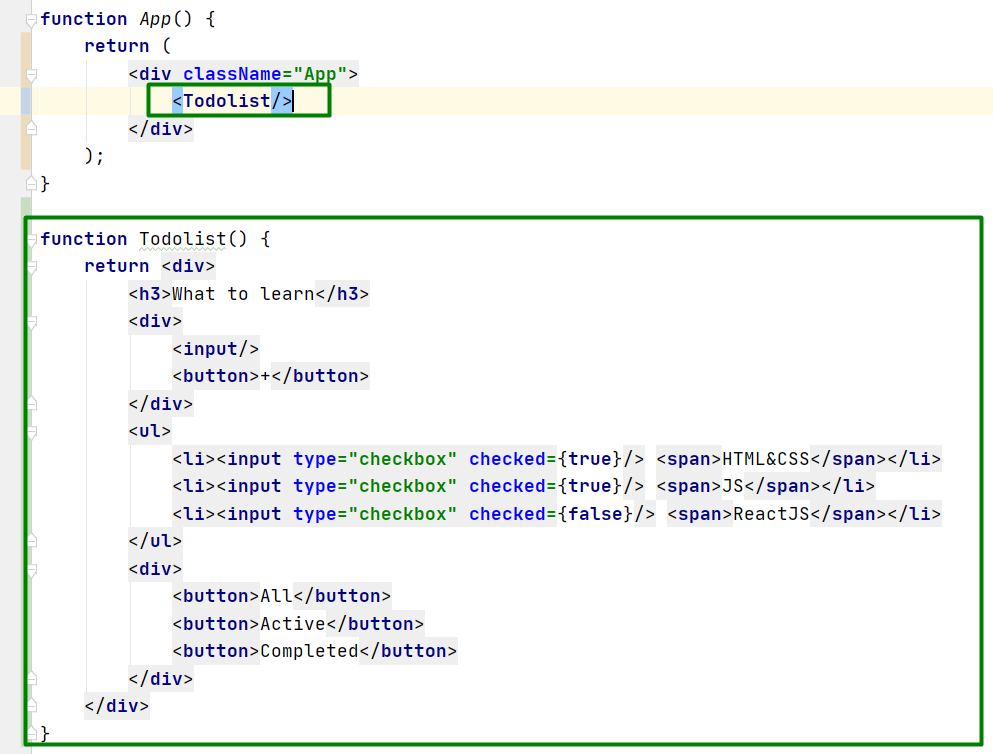
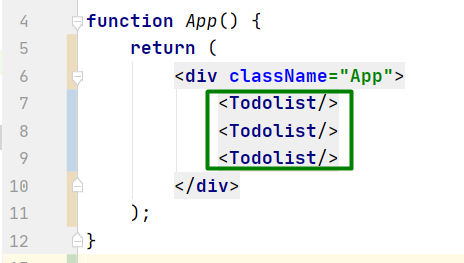
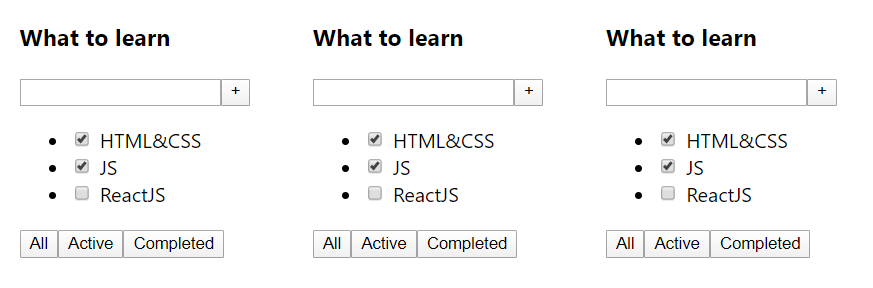
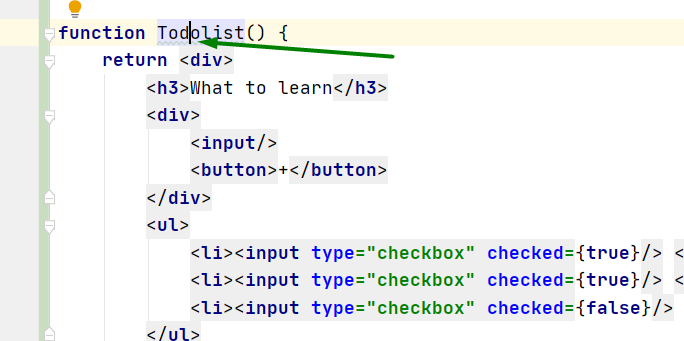
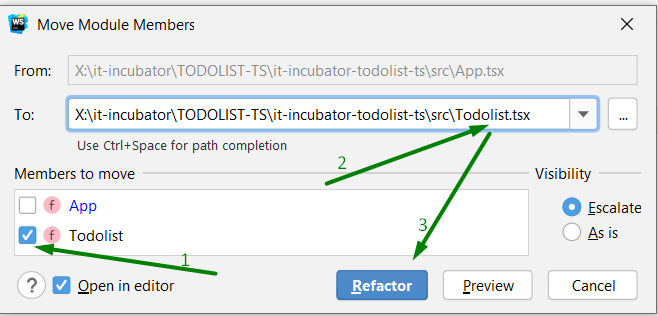
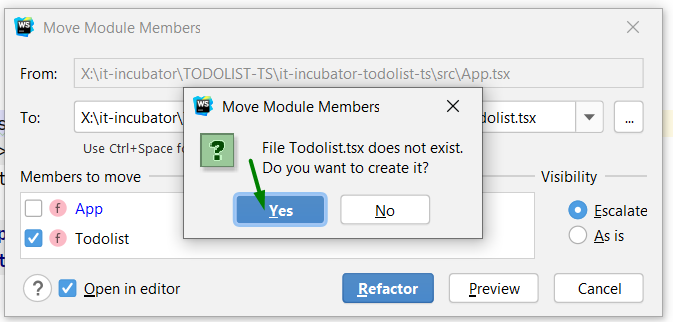
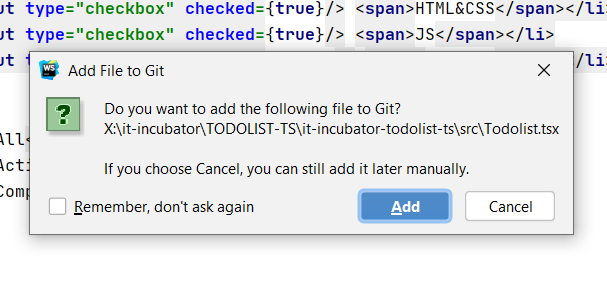
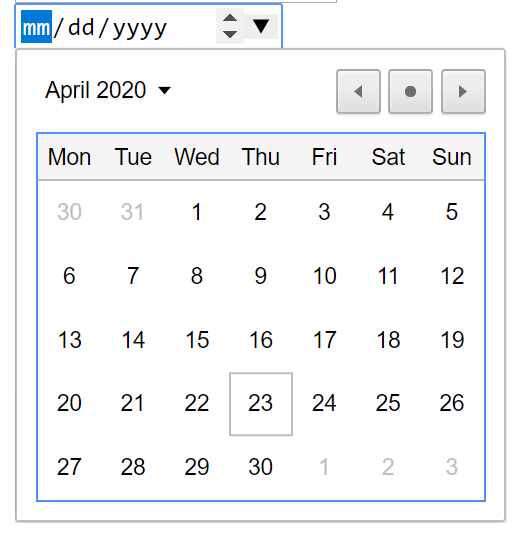
### До занятия:

* Посмотреть и проделать [9 уроков https://www.youtube.com/playlist?list=PLcvhF2Wqh7DNVy1OCUpG3i5lyxyBWhGZ8](https://www.youtube.com/playlist?list=PLcvhF2Wqh7DNVy1OCUpG3i5lyxyBWhGZ8)
* Установить nodeJS, WebStorm, [yarn](https://classic.yarnpkg.com/en/docs/install/#windows-stable)
* Посмотреть и проделать то, что в видео:  
  <https://www.youtube.com/watch?v=G9XMujDWf7U&feature=youtu.be>
* Проделать первый 4 пункта (видео подсказка: <https://youtu.be/a8WaQy7CtwU> либо начало этого видео: <https://youtu.be/4htFhSxXrHI>)
* Важно!!! Без проделанных 4 пунктов (должны увидеть запущенный тудулист-проект) вы не сможете продолжать занятие, преподаватель на первые 4 пункта на решение проблем на уроке не будет тратить время!!! До прихода в инкубатор через общий чат попробуйте решить проблемы, помогая друг другу, если у вас что-то не завелось

### Занятие step 01

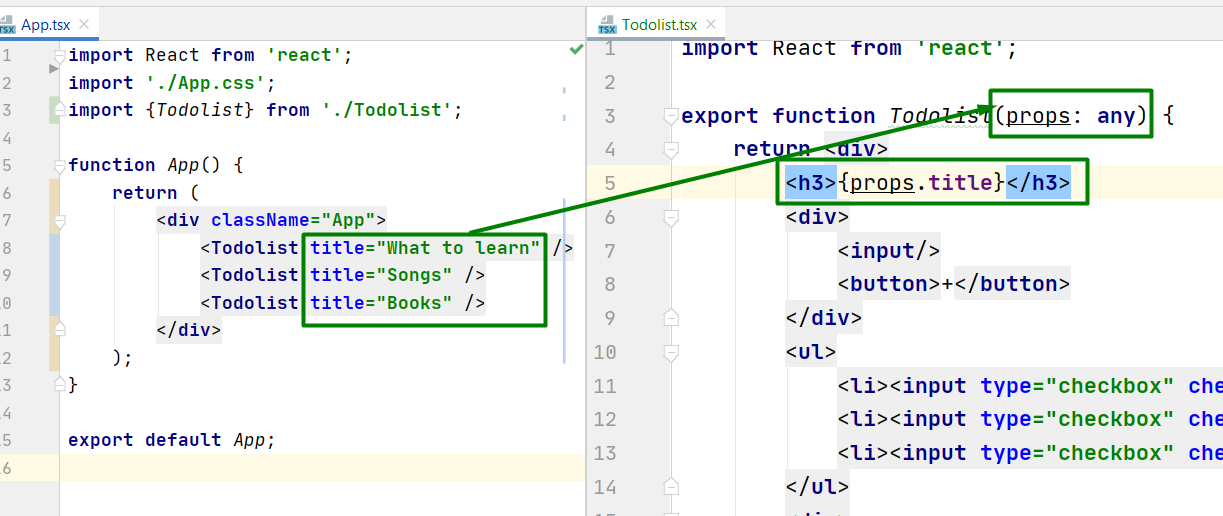
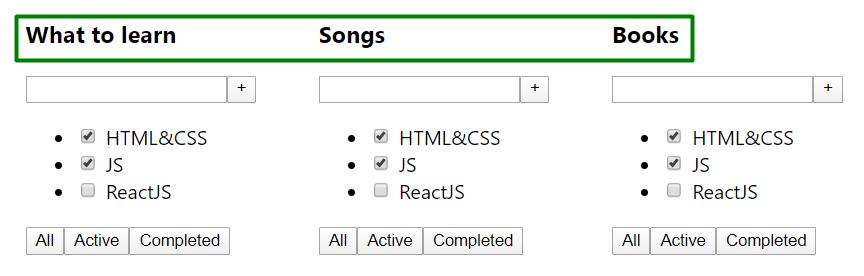
1. [Скачайте архив](https://drive.google.com/open?id=1-Xq90qLIgjCJfpVa7zznYO4T17nkHZr7) и распакуйте его в папку **X:/it-kamasutra** (папка может быть   
   любой, главное не делайте глубокую вложенность)  
     
   ***И ни в коем случае*** *не используйте для этих целей* ***Рабочий стол*** *:)*
2. Откройте **папку-проект** в вашей IDE (**ПРОЕКТОМ** является та папка, внутри которой лежит файл **package.json**) (проектом с точки зрения **NODE JS**)
3. Откройте **terminal** в вашей **ide** (убедитесь, что путь там установлен к нужной папке-проекту)  
     
     
     
      
   и пропишите команду   
    ***yarn***Что произойдёт? NPM (node js пакетный менеджер) залезет в файл **package.json**, являющийся “сердцем” проекта, посмотрит, от каких модулей (пакетов) зависит ваш проект (он же модуль, пакет) и скачает\установит всё необходимое
4. Давайте запустим проект командой: ***yarn start***Должны увидеть с вами “Список дел” (todo list).  
     
   
5. Вся движуха начинается в файле **index.ts** (берём и рисуем тег **<App />**)  
   Что такое тег **App**? Откуда он взялся? Мы его импортировали из другого файла:  
     
     
     
     
   И что же такое тег **App**?
6. Открываем с вами **App.ts** и изучаем:  
   **  
     
   App** (внимание, с большой буквы) - это просто функция (стрелочная или созданная через слово function - не важно). И эта функция ДОЛЖНА нам вернуть JSX-разметку (визуально JSX - это HTML внутри JS)  
     
   И вот если у нас есть такая функция с именем с большой буквы, возвращающая JSX, то у нас в системе появляется новый тег, соответствующий названию функции.  
     
   Компоненты (свои теги) нужны, чтобы скомпоновать вместе какие-то куски, создав новые строительные блоки.
7. Давайте вынесем из **App** часть разметки в новый компонент **Todolist.** В этом же файле создайте новую функцию:  
     
     
   ****  
   
8. Отрисуйте 3 Todolist-а:  
     
     
      
   Результат:  
     
   
9. Давайте унесем функцию-компоненту Todolist в отдельный файл.  
   Помещаем текстовый курсор (кликом по названию) на название функции Todolist:  
     
     
     
   Жмём F6, говорим какую функцию в какой файл хотим унести и жмём Refactor:  
     
     
   Подтверждаем, что мы хотим создать несуществующий файл (Yes):  
     
     
     
   Файл создаться и функция перенесётся туда.  
   При создании нового файла WebStorm предложит добавить его в git. Жмите **Add**:  
     
   
10. Все тудулисты называются сейчас **What to learn.** А хочется, чтобы можно было, отрисовывая каждый тудулист, передавать ему внутрь какие-то данные:  
      
      
      
    Это похоже на то, как мы пользуемся обычными тегами, передавая им на вход какие-то данные:  
      
    <**input type="date"** />

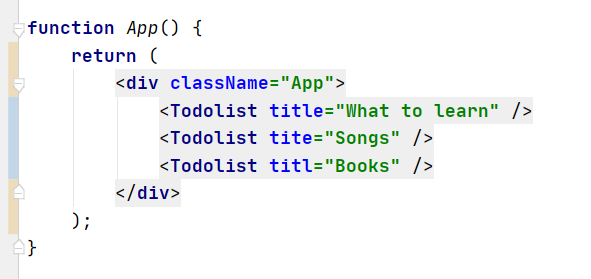
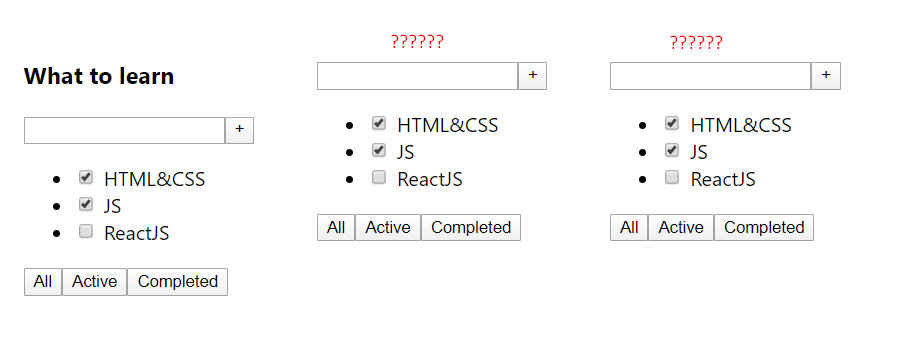
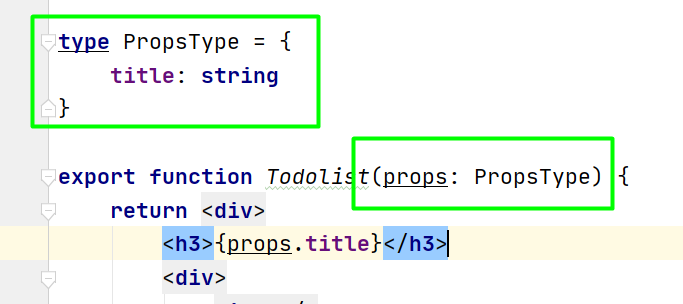
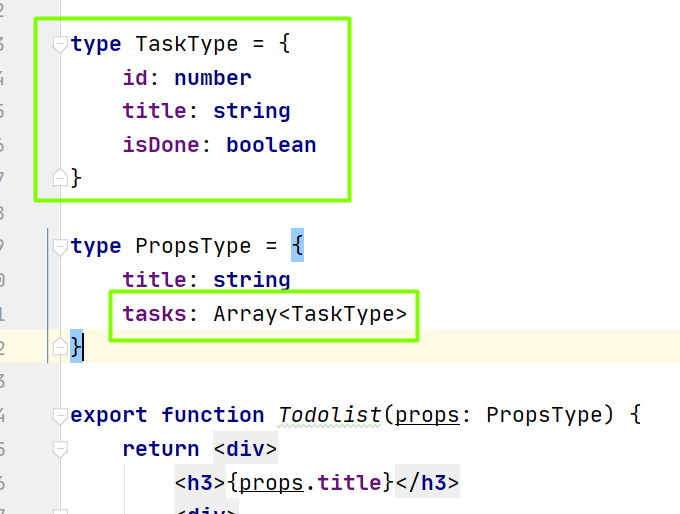
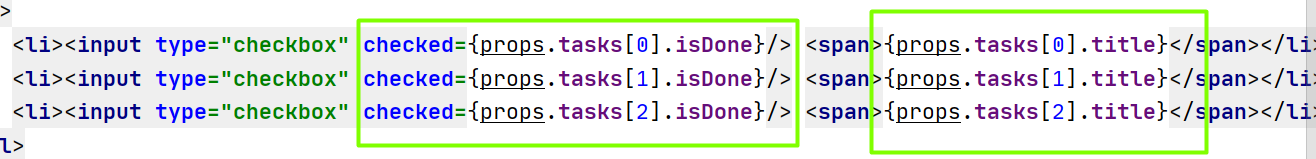
даст нам такую отрисовку:  


<**input placeholder="date of birth"**/>

даст нам тот самый серенький текст-заглушку:  
  


<**input type="button" value="text inside button"**/>

отрисует нам кнопку с текстом внутри:  
  
  
  
Входные параметры, которые принимает компонент, называются props (пропсы). Пропсы - это объект, свойствами которого будут атрибуты, которые мы передаём.  
  
**props -** это параметр нашей функции, это объект, который приходит в нашу функцию-компоненту. Компонента может использовать эти данные для отображения. Внутри JSX-разметки нужно в фигурных скобках обратиться к объекту и через точку прочитать её св-во **title** (потому что именно title передали в атрибутах):  
  
  
  
  
Результат - 3 разные заголовка:  


1. props - это объект, который приходит в нашу компоненту. Этот объект должен иметь определённую структуру. Попробуйте опечататься при передачи **title**, напишите что-то вроде этого:  
     
     
    Результат:   
   Такое стало возможным, потому что сказали, что наш **props: any** (props: что\_угодно). То есть не задали чётко тип, который будет отслеживаться TypeScript-ом
2. Создадим такой тип:  
     
     
     
   Обратите внимание: теперь в App-компоненте компилятор TS ругается, подсказывая нам, что мы ошиблись:  
     
     
   Исправьте “опечатки” и всё заработает как прежде
3. Давайте будем передавать Тудулисту внутрь через пропсы задачи, которые ему нужно отображать, а то список один и тот же для всех задач!  
     
   Что такое задача, она же task? Каждая задача - это объект. Таких задач у нас много, поэтому помещаем их в массив (создадим 2 массива для 2-ух Todolist-ов, отрисовку третьего тудулиста удалим):  
     
     
     
   Передадим по массиву тасок в каждый Todolist. Но Todolist не ждёт такой массив. Добавим типизацию для массива с тасками. Так как Таска - это объект. Создадим тип для этого объекта. А потому укажем его для массива:  
     
     
     
   Результат: ошибка пропала, видим тудулисты как и раньше. Осталось нашему тудулисту научиться реагировать на приходящие извне таски и ренедрить (рисовать) именно эти таски
4. Обращаемся к **props**, потом к **tasks,** потом по индексу к конкретной таске:  
     
   

**ВЫВОДЫ**

Компонент(-а) - функция, возвращающая JSX-разметку. Название функции должно быть с Большой буквы

Функция может принимать props (пропсы) - объект, который компонента может использовать внутри себя для отрисовки конкретных переданных в неё данных

Видео по данному занятию (ОБЯЗАТЕЛЬНО попытаться разобрать и понять глубоко):  
<https://youtu.be/4htFhSxXrHI>  
  
То, что будем делать на следующем занятии (ОБЯЗАТЕЛЬНО посмотреть перед приходом на следующее занятие, но НЕ ДЕЛАТЬ) Если нет времени, позырить на x2 скорости, так вам будет понятнее, что происходит на занятии:  
<https://youtu.be/HSLVvWuS6s0>