# Занятие 3

1.Добавили функцию, которая добавляет таску.Закинули ее в новый массив

2. setTasks([newTask, ...tasks])  
//...tasks - копия массива (только обькты)  
// ...tasks - копируем обьекты массива из tasks + обьект newTask - все они записываются в переменную tasks

3. Добавили в тудулист онклик, захардкодили текст

4. const [title, setTitle] = useState<string>(' ')

Создали новй юз стайт, добавили он чэндж в инпут, при вводе значений в инпут, данные записываются в тайтл и передаются в addTask => добавляется новая задача.

value={title}

Для записи в локальный стейт(черновик), на будущее

5. из кнопки адд вынесли функцию в конст аддТаск, создали функцию onKeyPressAndTask, котора при нажатии энтер, отправляет текст из инпута в задачу

6. в коллбэк всегда передаем либо литерал, либо имя функции( но не вызов функции)

# Занятие 4

//эти значения дают false - undefined,null,0,'',NaN

//эти значения дают true - ' '(строка с пробелом),{},[] (обычный и пустой массив\обьект)

В переменную таск записывается ссылка на найденный обьект(новый обьект не создается)

const task = tasks.find(t => t.id === taskID)

1.task.isDone записываем новое , пришедшее из пропс,значение в найденный обьект

2. setTasks([...tasks]) – копируем данные со старого массива

[Tasks] - массив с содержимым, ...tasks – просто содержимое. Необходимо использовать именно содержимое

3.При изменении вызывается ивент, который вызывает функцию, в нее передаем два параметра(второй дает тру или фолс). e.currentTarget.checked при клике меняет значение

onChange={e => props.changeTaskStatus(task.id, e.currentTarget.checked)}

4. Если такс.исдан = тру, присваевается класс исдан, если фалс – пустая строка(класса нет)

<li className={task.isDone ? 'is-done':''}

5.

title.trim()//обрезает пробелы вначале и конце троки

# Занятие 5

1. Значение в [] не свойство обьекта

tasks[todoListID] = tasks[todoListID].filter(task => task.id !== taskID)

//находим по ид обьект и с помощью фильтра содаем новый обьект, не включающий удаленную таску

1. нашли тудулист, копировали его, и добавили новую таску. сетТаск ‘перерисовал’ массив(изменения уже произошли)

[...tasks]-копируем массив

{...tasks}-копируем обьект

tasks[todoListID] = [newTask,...tasks[todoListID]]  
setTasks({...tasks})

1. мэп берет каждый элемент массива(их два), и для каждого вызывает тудулист со своим ид\титл\фильтр (значение фильтра определяется вначле (tl.filter))

todoLists.map(tl => {  
 let tasksForTodoList = tasks[tl.id]  
 if (tl.filter === 'active') {  
 tasksForTodoList = tasksForTodoList.filter(t => t.isDone === false)  
 }  
 if (tl.filter === 'completed') {  
 tasksForTodoList = tasksForTodoList.filter(t => t.isDone === true)  
 }  
 return (  
 <TodoList  
 id={tl.id}  
 title={tl.title}  
 tasks={tasksForTodoList}  
 filter={tl.filter}  
 addTask={addTask}  
 removeTasks={removeTask}  
 changeTodoListFilter={changeFilter}  
 changeTaskStatus={changeTaskStatus}  
 />  
  
 )  
})  
}

# Занятие 6

1. AddItemForm – создает какойто название, которое можно использовать как угодно(название такси, юзера итд)
2. Аналогичные записи

title: title = title

title = title

# Занятие 10

1. Функции могут чтото возвращать,могут не возвращать. Могут быть синхронными или ассинхронными.

Ассинхронная функция поток не останавливается, код выполняется дальше

Синхронная – код останавливается, пока не выполнится функция

1. useReducer – первым параметром принимает редьюсер, вторым – стейт

const [todoLists, dispatchTodoLists] = useReducer(todoListReducer,[  
 {id: todoListID\_1, title: 'What to learn', filter: 'all'},  
 {id: todoListID\_2, title: 'What to buy', filter: 'all'},  
])

todoLists – стейт; dispatchTodoLists - редьюсер

По сути useReducer похож на юзСтейт, только стейт меняется с помощью диспатча, а не юзСтейта

<https://build-in-saratov.com/reasons-to-use-usereducer/>

1. useSelector – из всего стейта выбираем нужное нам значение, и присваеваем его переменной

let tasks = useSelector<AppRootStateType,TaskStateType >( state => state.tasks)  
 //вторым параметром в типизации, то, что возвращает

1. useDispatch – получаем функци, которая диспатчит экшн в стор
2. Все это нужно, чтобы не использовать mapStateToProps, mapDispatchToProps как в пути самурая