HOW TO DISPLAY A LIST OF ITEMS IN REACT?

The following [function component](https://www.robinwieruch.de/react-function-component/) shows how to render a list of items (JS primitives). It would work the same with a list of numbers instead of strings. Since we can use JavaScript in JSX by using curly braces, we can use the [built-in JavaScript array map method](https://www.robinwieruch.de/javascript-map-array/) to iterate over our list items; and to map them from JavaScript primitive to HTML elements. Each element receives a mandatory [key prop](https://www.robinwieruch.de/react-list-key/):

const SimpleList = () => (

<ul>

{['a', 'b', 'c'].map(function(item) {

return <li *key*={item}>{item}</li>;

})}

</ul>

);

We didn't define the list but merely inlined it. In the case of declaring the list as variable, it would look like the following:

const list = ['a', 'b', 'c'];

const SimpleList = () => (

<ul>

{list.map(function(item) {

return <li *key*={item}>{item}</li>;

})}

</ul>

);

We can also use JavaScript arrow function to make the inline function for the map more lightweight:

const list = ['a', 'b', 'c'];

const SimpleList = () => (

<ul>

{list.map(item => {

return <li *key*={item}>{item}</li>;

})}

</ul>

);

Since we are not doing anything in the function's block body, we can also refactor it to a concise body and omit the return statement and the curly braces for the function body:

const list = ['a', 'b', 'c'];

const SimpleList = () => (

<ul>

{list.map(item => (

<li *key*={item}>{item}</li>

))}

</ul>

);

If we would use the List as child component in another component, we could pass the list as [props](https://www.robinwieruch.de/react-pass-props-to-component/) to it:

const mylist = ['a', 'b', 'c'];

const App = () => (

<SimpleList *list*={mylist} />

);

const SimpleList = ({ list }) => (

<ul>

{list.map(item => (

<li *key*={item}>{item}</li>

))}

</ul>

);

That's a simple list component example for React. We only have a list of JavaScript primitives, like strings or integers, map over it, and render a HTML listitem for each item in our array.

HOW TO DISPLAY A LIST OF OBJECTS IN REACT?

It doesn't work any different for complex objects in JavaScript arrays. You iterate over the list with the map method again and output for each list item your HTML elements. Only the value argument given in the map function is an object and no primitive anymore. Hence you can access the object in your JSX to output the different properties of it:

import React from 'react';

const list = [

{

id: 'a',

firstname: 'Robin',

lastname: 'Wieruch',

year: 1988,

},

{

id: 'b',

firstname: 'Dave',

lastname: 'Davidds',

year: 1990,

},

];

const ComplexList = () => (

<ul>

{list.map(item => (

<li *key*={item.id}>

<div>{item.id}</div>

<div>{item.firstname}</div>

<div>{item.lastname}</div>

<div>{item.year}</div>

</li>

))}

</ul>

);

export default ComplexList;

The implementation isn't much different here. We map a list of objects over to a list of HTML elements. If you would want a list table in favor of a bullet list, you would have to use table, th, tr, td elements instead of ul, li list elements.

HOW TO DISPLAY NESTED LISTS IN REACT?

If you run into 2 dimensional lists, you can still use the same implementation techniques from before. First, map over your first array and then map over your second array within. For the sake of simplification, we use two times the same list in our nested list:

import React from 'react';

const list = [

{

id: 'a',

firstname: 'Robin',

lastname: 'Wieruch',

year: 1988,

},

{

id: 'b',

firstname: 'Dave',

lastname: 'Davidds',

year: 1990,

},

];

const nestedLists = [list, list];

const NestedList = () => (

<ul>

{nestedLists.map((nestedList, index) => (

<ul *key*={index}>

<h4>List {index + 1}</h4>

{nestedList.map(item => (

<li *key*={item.id}>

<div>{item.id}</div>

<div>{item.firstname}</div>

<div>{item.lastname}</div>

<div>{item.year}</div>

</li>

))}

</ul>

))}

</ul>

);

export default NestedList;

However, nesting multiple maps into each other becomes quickly a burden for the maintainability of the application. That's why it's great to extract smaller components from your large list component (e.g. NestedList, List, Item components).

REACT LIST COMPONENTS

In order to keep your React list components tidy, you can extract them to standalone component that only care about their concerns. For instance, the List component makes sure to map over the array to render a list of ListItem components for each item as child component:

const list = [

{

id: 'a',

firstname: 'Robin',

lastname: 'Wieruch',

year: 1988,

},

{

id: 'b',

firstname: 'Dave',

lastname: 'Davidds',

year: 1990,

},

];

const App = () => <List *list*={list} />;

const List = ({ list }) => (

<ul>

{list.map(item => (

<ListItem *key*={item.id} *item*={item} />

))}

</ul>

);

const ListItem = ({ item }) => (

<li>

<div>{item.id}</div>

<div>{item.firstname}</div>

<div>{item.lastname}</div>

<div>{item.year}</div>

</li>

);

The List component offers an [API](https://www.robinwieruch.de/what-is-an-api-javascript/) to the outside: This way the App component can pass the array as list props to the List component. One little trick for [conditional rendering](https://www.robinwieruch.de/conditional-rendering-react/): If you don't know whether the incoming list is null or undefined, default to an empty list yourself:

const List = ({ list }) => (

<ul>

{(list || []).map(item => (

<ListItem *key*={item.id} *item*={item} />

))}

</ul>

);

The List and ListItem component are used so often in React applications, that they can be taken as list template or boilerplate, because often you just copy and paste the same implementation for a simple list over to your code. So keeping this structure for a list component in mind doesn't harm.

REACT LIST: UPDATE ITEMS

So far, we have only seen list items that are not changed, because they are just declared a variable or passed down as props. But what about a list managed as state? It's possible to add, update, and remove items to/in/from the list. For all following examples, we will start with the same boilerplate list component:

import React from 'react';

const initialList = [];

const List = () => {

const [list, setList] = React.useState(initialList);

return (

<div>

<ul>

{list.map(item => (

<li *key*={item}>{item}</li>

))}

</ul>

</div>

);

};

export default List;

Let's dive into the different examples to update our list items with [React Hooks](https://www.robinwieruch.de/react-hooks/). All the following patterns are the foundation for sophisticated [state management in React](https://www.robinwieruch.de/react-state/).

React List: Add Item

The following List component shows a stateful managed list where it's possible to add an item to the list with a [controlled form element](https://www.robinwieruch.de/react-controlled-components/):

import React from 'react';

const initialList = [

'Learn React',

'Learn Firebase',

'Learn GraphQL',

];

const ListWithAddItem = () => {

const [value, setValue] = React.useState('');

const [list, setList] = React.useState(initialList);

const handleChange = event => {

setValue(event.target.value);

};

const handleSubmit = event => {

if (value) {

setList(list.concat(value));

}

setValue('');

event.preventDefault();

};

return (

<div>

<ul>

{list.map(item => (

<li *key*={item}>{item}</li>

))}

</ul>

<form *onSubmit*={handleSubmit}>

<input *type*="text" *value*={value} *onChange*={handleChange} />

<button *type*="submit">Add Item</button>

</form>

</div>

);

};

export default ListWithAddItem;

By using the submit button to initiate the creation of the item, the handler makes sure to add the item to the stateful list. Also the [native browser behavior is prevented](https://www.robinwieruch.de/react-preventdefault/) by using the click event; otherwise the browser would refresh after the submit event.

React List: Update Item

The following List component shows a stateful managed list where it's possible to update an item in the list with a input element:

import React from 'react';

const initialList = [

{ id: 'a', name: 'Learn React', complete: false },

{ id: 'b', name: 'Learn Firebase', complete: false },

{ id: 'c', name: 'Learn GraphQL', complete: false },

];

const ListWithUpdateItem = () => {

const [list, setList] = React.useState(initialList);

const handleChangeCheckbox = id => {

setList(

list.map(item => {

if (item.id === id) {

return { ...item, complete: !item.complete };

} else {

return item;

}

})

);

};

return (

<ul>

{list.map(item => (

<li *key*={item.id}>

<label>

<input

*type*="checkbox"

*checked*={item.complete}

*onChange*={() => handleChangeCheckbox(item.id)}

/>

{item.name}

</label>

</li>

))}

</ul>

);

};

export default ListWithUpdateItem;

By using the checkbox element to initiate the update of the item, the handler makes sure to toggle the boolean flag of the item in the stateful list.

React List: Remove Item

The following List component shows a stateful managed list where it's possible to remove an item from the list with a button element:

import React from 'react';

const initialList = [

{ id: 'a', name: 'Learn React' },

{ id: 'b', name: 'Learn Firebase' },

{ id: 'c', name: 'Learn GraphQL' },

];

const ListWithRemoveItem = () => {

const [list, setList] = React.useState(initialList);

const handleClick = id => {

setList(list.filter(item => item.id !== id));

};

return (

<ul>

{list.map(item => (

<li *key*={item.id}>

<label>{item.name}</label>

<button *type*="button" *onClick*={() => handleClick(item.id)}>

Remove

</button>

</li>

))}

</ul>

);

};

export default ListWithRemoveItem;

By using the delete button to initiate the removal of the item, the handler makes sure to remove the item from the stateful list. React makes sure to update the list after the delete when using the function to set the state.