JS Array Functions

Not really next-gen JavaScript, but also important: JavaScript array functions like map() , filter() , reduce() etc.

You'll see me use them quite a bit since a lot of React concepts rely on working with arrays (in immutable ways).

The following page gives a good overview over the various methods you can use on the array prototype - feel free to click through them and refresh your knowledge as required:

[**https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/Array**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array)

Particularly important in this course are:

map() => https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/Array/map

find() => https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/Array/find

findIndex() => https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/Array/findIndex

filter() => https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/Array/filter

reduce() => https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/Array/Reduce?v=b

concat() => https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/Array/concat?v=b

slice() => https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/Array/slice

splice() => https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/Array/splice

Module Resources

You may want to compare your code to mine (e.g. to find + fix errors).

Finished Code on CodeSandbox: **https://codesandbox.io/s/javascript-refresher-j56djd**

You also find the finished code snapshot for this module here in this Github repository: **https://github.com/academind/react-complete-guide-code/tree/02-js-refresher**

Usage instructions can be found on the page that link is leading to.

You also find section slides (if available) in that Github repository.

**Impo : All modules are present in this link**

https://github.com/academind/react-complete-guide-course-resources

**A Closer Look: Components & File Extensions**

* At this point, you've built a first custom component and you, of course, also worked with the App component.
* For the moment, both components are stored in the App.jsx file (this will change later though).
* .jsx is a file extension that's not supported by the browser! It's working because you're working in a React project that supports this special extension. Because this extension "tells" the underlying build process (which is running behind the scenes when the development server is running) that a file contains JSX code (which is also not supported by browsers).
* It's important to understand that it's really just that build process that cares about this extension.
* And therefore, you'll also find React projects that don't use .jsx but instead just .js as a file extension. And in those .js files, you'll also find JSX code. Because it simply depends on the underlying build process which extension is expected when using this JSX syntax in a file.
* Since it doesn't work in the browser either way, there is no hard rule regarding this. Instead, you'll find projects that require .jsx (like the project setup we use in this course) and you'll find projects that also support .js (with JSX code inside).
* I'm emphasizing this here so that you're not confused if you encounter React projects that don't use .jsx files.
* In addition, you'll also find projects that require the file extension as part of file imports (e.g., import App from './App.jsx') and you'll find other projects that don't require this (i.e., there, you could just use import App from './App').
* This, again, has nothing to do with the browser or "standard JavaScript" - instead it simply depends on the requirements of the code build process that's part of the project setup you chose.

**More Prop Syntaxes :** Beyond the various ways of setting and extracting props about which you learned in the previous lecture, there are even more ways of dealing with props.

But no worries, you'll see all these different features & syntaxes in action throughout the course!

**Passing a Single Prop Object**

If you got data that's already organized as a JavaScript object, you can pass that object as a single prop value instead of splitting it across multiple props.

I.e., instead of

<CoreConcept

title={CORE\_CONCEPTS[0].title}

description={CORE\_CONCEPTS[0].description}

image={CORE\_CONCEPTS[0].image} />

or

<CoreConcept

{...CORE\_CONCEPTS[0]} />

you could also pass a single concept (or any name of your choice) prop to the CoreConcept component:

<CoreConcept

concept={CORE\_CONCEPTS[0]} />

In the CoreConcept component, you would then get that one single prop:

export default function CoreConcept({ concept }) {

// Use concept.title, concept.description etc.

// Or destructure the concept object: const { title, description, image } = concept;

}

It is entirely up to you which syntax & approach you prefer.

Grouping Received Props Into a Single Object

You can also pass multiple props to a component and then, in the component function, group them into a single object via JavaScript's "Rest Property" syntax.

I.e., if a component is used like this:

<CoreConcept

title={CORE\_CONCEPTS[0].title}

description={CORE\_CONCEPTS[0].description}

image={CORE\_CONCEPTS[0].image} />

You could group the received props into a single object like this:

export default function CoreConcept({ ...concept }) {

// ...concept groups multiple values into a single object

// Use concept.title, concept.description etc.

// Or destructure the concept object: const { title, description, image } = concept;

}

If that syntax is a bit confusing - worry not! You'll also see concrete examples for this syntax (and for why you might want to use it in certain situations) throughout the course!

**Default Prop Values**

Sometimes, you'll build components that may receive an optional prop. For example, a custom Button component may receive a type prop.

So the Button component should be usable either with a type being set:

<Button type="submit" caption="My Button" />

Or without it:

<Button caption="My Button" />

To make this component work, you might want to set a default value for the type prop - in case it's not passed.

This can easily be achieved since JavaScript supports default values when using object destructuring:

export default function Button({ caption, type = "submit" }) {

// caption has no default value, type has a default value of "submit"

}