React State and Arrays

New Attempt

Due No Due Date **Points** 1 **Submitting** a website url

FORK (https://github.com/learn-co-curriculum/react-hooks-state-arrays/fork) (https://github.com/learn-co-curriculum/react-hooks-state-arrays) (https://github.com/learn-co-curriculum/react-hooks-state-arrays/issues/new)

Learning Goals

- Understand how to work with arrays in state

Working With Arrays

Adding Elements To Arrays In State

When we need to represent a list of data in our UI, it's often a good idea to have the data for that list stored in an array! To give an example, let's build out a component that does the following:

- Shows a button to generate a new spicy food
- When the button is clicked, adds the newly generated food to a list

The starter code for this component is in `SpicyFoodList.js` . Before we walk through the solution, see if you can get this working by:

- using a **state variable** to hold an **array** of spicy foods;
- using that array to display each spicy food as an `<li>` ; and
- **adding a new spicy food to the array** when the button is clicked.

One important rule to keep in mind when working with objects and arrays in state:

React will only update state if a new object/array is passed to `setState` . That means it's important to keep in mind which array methods *mutate* arrays, and which can be used to make *copies* of arrays (**hint**: the spread operator is your friend here).

Ok, give it a shot! Then scroll down to see a solution.

...

...

...

...

...

Help

...

...

...

First, let's update our component to return some JSX elements based on this array in state. We can use `.map` on our array to generate an array of `<li>` elements from our array of foods, and display them in the `<ul>` :

```
const foodList = foods.map((food) => (
  <li key={food.id}>
    {food.name} | Heat: {food.heatLevel} | Cuisine: {food.cuisine}
  </li>
));

return (
  <div>
    <button onClick={handleAddFood}>Add New Food</button>
    <ul>{foodList}</ul>
  </div>
);
```

Now that our foods are displaying, time for the moment of truth: can we update state and get new foods to display dynamically?

```
function handleAddFood() {
  const newFood = getNewSpicyFood();
  const newFoodArray = [...foods, newFood];
  setFoods(newFoodArray);
}
```

This step is crucial, so let's break it down!

```
const newFoodArray = [...foods, newFood];
```

Here, we're using the spread operator (`...`) to make a *copy* of our `foods` array, and insert it into a *new* array. We're also adding the newly generated food returned by the `getNewSpicyFood` function at the end of the array.

Remember, whenever we are updating state, it's important that we always pass a new object/array to `setState` . That's why we're using the spread operator here to make a copy of the array, instead of `.push` , which will mutate the original array.

Again, to repeat! React will *only* re-render our component when we set state with a *new* value; so we need to create a new **copy** of our original array to pass to the setter function, rather than mutating the original array directly and passing a reference to the original array.

Help

After setting state, our component should automatically re-render with the new list of foods.

Removing Elements From Arrays In State

Let's add another feature. When a user clicks on a food, that food should be *removed* from the list.

First, we'll need to add a click handler to the `<li>` elements, and pass in the id of the food we're trying to remove:

```
const foodList = foods.map((food) => (
  <li key={food.id} onClick={() => handleLiClick(food.id)}>
    {food.name} | Heat: {food.heatLevel} | Cuisine: {food.cuisine}
  </li>
));
```

Next, in the `handleLiClick` function, we need to figure out a way to update our array in state so it no longer includes the food.

There are a few approaches you could take here, so try to find a solution on your own before peeking at the answer! Remember, we want to find a way to remove the food by *creating a new array that has all the original elements, except the one we want removed*.

...

...

...

...

One common approach to this problem of creating a new array that doesn't include a specific element is using the `.filter` method. Here's how we can do it:

```
function handleLiClick(id) {
  const newFoodArray = foods.filter((food) => food.id !== id);
  setFoods(newFoodArray);
}
```

Calling `.filter` will return a *new array* based on which elements match our criteria in the callback function. So if we write our callback function in `.filter` to look for all foods *except* the number we're trying to remove, we'll get back a new, shortened list of foods:

```
[1, 2, 3].filter((number) => number !== 3);
// => [1,2]
```

Setting state with this updated list of foods will re-render our component, causing the food to be removed from the list.

Updating Elements in Arrays in State

Help

Here's a tough one! We've seen how to add and remove elements from arrays, but what about updating them?

Let's update our click feature so that when a user clicks on a food, that food's heat level is incremented by 1.

In the `handleLiClick` function, we need to figure out a way to update our array in state and increment the heat level *only* for the food that was clicked.

Once again, there are a few approaches you could take here, so try to find a solution on your own before peeking at the answer! Remember, we want to find a way to update the heat level by *creating a new array that has all the elements from the original array, with one element updated*.

...

...

...

...

One approach we can take to *updating* items in arrays by creating a new array involves using the `.map` method. Calling `.map` will return a new array with the same length as our original array (which is what we want), with some transformations applied to the elements in the array.

Here's an example of using `.map` to update *one element* of an array:

```
[1, 2, 3].map((number) => {
  if (number === 3) {
    // if the number is the one we're looking for, increment it
    return number + 100;
  } else {
    // otherwise, return the original number
    return number;
  }
});
// => [1,2,103]
```

So to use that technique to solve our problem, here's how our click event handler would look:

```
function handleLiClick(id) {
  const newFoodArray = foods.map((food) => {
    if (food.id === id) {
      return {
        ...food,
        heatLevel: food.heatLevel + 1,
      };
    } else {
      return food;
    }
```

Help

```
    });
    setFoods(newFoodArray);
  }
```

To break down the steps:

- `.map` will iterate through the array and return a new array
- Whatever value is returned by the callback function that we pass to `.map` will be added to this new array
- If the ID of the food we're iterating over matches the ID of the food we're updating, return a new food object with the heat level incremented by 1
- Otherwise, return the original food object

Array Cheat Sheet

Here's a quick reference of some common techniques for manipulating arrays in state. Keep this in mind, because working with arrays will be important as a React developer!

- **Add**: use the spread operator (`[...]`)
- **Remove**: use `.filter`
- **Update**: use `.map`

Working With Multiple State Variables

Sometimes, a component needs multiple state variables to represent multiple UI states. To give an example, let's add a feature to our `SpicyFoodList` component that lets the user filter the list to only show foods from a certain cuisine.

Here's the JSX you'll need for this feature:

```
<select name="filter">
  <option value="All">All</option>
  <option value="American">American</option>
  <option value="Sichuan">Sichuan</option>
  <option value="Thai">Thai</option>
  <option value="Mexican">Mexican</option>
</select>
```

Try building out this feature on your own, then we'll go through the solution. Think about what new *state variable* you'll need to add, and how to use that variable to determine which foods are being displayed!

...

...

...

...

Let's start by talking through what new state we'll need to add. We need some way of keeping track of which option the user selected from the `<select>` tag. We'll also need to use that data to *filter* the list of foods to determine which ones to display.

Let's set up our initial state to be a string of "All" to match the first `<option>` in our dropdown:

```
const [filterBy, setFilterBy] = useState("All");
```

With this state variable in place, we can update the `<select>` element to set the `filterBy` variable when its value is changed, like so:

```
function handleFilterChange(event) {
  setFilterBy(event.target.value);
}

return (
  <select name="filter" onChange={handleFilterChange}>
    <option value="All">All</option>
    <option value="American">American</option>
    <option value="Sichuan">Sichuan</option>
    <option value="Thai">Thai</option>
    <option value="Mexican">Mexican</option>
  </select>
);
```

Next, let's figure out how this filter value can be used to update what foods are displayed. We will need to use *both* of our state variables together to solve this problem! Here's how we can use the filter value to update which items are displayed:

```
const [foods, setFoods] = useState(spicyFoods);
const [filterBy, setFilterBy] = useState("All");

const foodsToDisplay = foods.filter((food) => {
  if (filterBy === "All") {
    return true;
  } else {
    return food.cuisine === filterBy;
  }
});
```

> Remember, `.filter` returns a new array that is a shortened version of the elements in the original array. It expects a callback that will return either true or false. For all elements of the original array where the callback returns true, add those elements to the new array. For all elements that return false, don't add them to the new array.

This will give us a new variable, `foodsToDisplay`, that will be an array of:

Help

- All foods from `foods` array if `filterBy` is set to "All"
- Only foods that match the cuisine in `filterBy` if `filterBy` is not set to "All"

Now, we just need to use `foodsToDisplay` instead of `foods` when we're generating the `<li>` elements:

```
const foodList = foodsToDisplay.map((food) => (
  <li key={food.id} onClick={() => handleLiClick(food.id)}>
    {food.name} | Heat: {food.heatLevel} | Cuisine: {food.cuisine}
  </li>
));
```

Having both of these variables in state and knowing how to use them in conjunction with each other gives us a lot of power in React! All we need to worry about is using our programming tools — working with *data*; manipulating *arrays* — and React can take care of all the hard work of updating the DOM correctly.

Conclusion

When working with arrays in state, it's important to find ways to set state by passing in a new array rather than mutating the original array. That means using array methods like `map` and `filter`, or the spread operator, to create copies of arrays before setting state.

Resources

- **React State Arrays (https://www.robinwieruch.de/react-state-array-add-update-remove)**

Help