Lists in React

We've rendered a few primitive variables in JSX and now we'll render a list of items.

We'll cover the following

- Exercises:

We'll experiment with sample data at first, then we'll apply that to fetch data from a remote API. First, let's define the array as a variable. As before, we can define a variable outside or inside the component. The following defines it outside:

```
import React from 'react';

const list = [
  {
    title: 'React',
    url: 'https://reactjs.org/',
    author: 'Jordan Walke',
    num_comments: 3,
    points: 4,
    objectID: 0,
  },
  {
    title: 'Redux',
    url: 'https://redux.js.org/',
    author: 'Dan Abramov, Andrew Clark',
    num_comments: 2,
    points: 5,
    objectID: 1,
  },
];

function App() { ... }

export default App;
```

src/App.js

I used a `...` here as a placeholder, to keep my code snippet small (without App component implementation details) and focused on the essential parts (the `list` variable outside of the App component). I will use the `...` throughout the rest of this learning experience as a placeholder for code blocks that I have established in previous exercises. If you get lost, you can always verify your code by running it live on the Educative SPA widget below.

Each item in the list has a title, a url, an author, an identifier (`objectID`), points – which indicate the popularity of an item – and a count of comments. Next, we'll render the list within our JSX dynamically:

```
function App() {
  return (
    <div>
      <h1>My Hacker Stories</h1>

      <label htmlFor="search">Search: </label>
      <input id="search" type="text" />
      <hr />
      {/* render the list here */}
    </div>
  );
}
```

src/App.js

You can use the built-in JavaScript map method for arrays to iterate over each item of the list and return a new version of each:

```
const numbers = [1, 4, 9, 16];

const newNumbers = numbers.map(function(number) {
  return number * 2;
});

console.log(newNumbers);
// [2, 8, 18, 32]
```

We won't map from one JavaScript data type to another in our case. Instead, we return a JSX fragment that renders each item of the list:

```
.App {
  text-align: center;
}

.App-logo {
  animation: App-logo-spin infinite 20s linear;
  height: 80px;
}

.App-header {
  background-color: #222;
  height: 150px;
  padding: 20px;
```

```
  color: white;
}

.App-title {
  font-size: 1.5em;
}

.App-intro {
  font-size: large;
}

@keyframes App-logo-spin {
  from { transform: rotate(0deg); }
  to { transform: rotate(360deg); }
}
```

Actually, one of my first React “Aha” moments was using barebones JavaScript to map a list of JavaScript objects to HTML elements without any other HTML templating syntax. It’s just JavaScript in HTML.

React will display each item now, but you can still improve your code so React handles advanced dynamic lists more gracefully. By assigning a key attribute to each list item’s element, React can identify modified items if the list changes (e.g. re-ordering). Fortunately, our list items come with an identifier:

```
.App {
  text-align: center;
}

.App-logo {
  animation: App-logo-spin infinite 20s linear;
  height: 80px;
}

.App-header {
  background-color: #222;
  height: 150px;
  padding: 20px;
  color: white;
}

.App-title {
  font-size: 1.5em;
}

.App-intro {
  font-size: large;
}

@keyframes App-logo-spin {
  from { transform: rotate(0deg); }
  to { transform: rotate(360deg); }
}
```

We avoid using the index of the item in the array to make sure the key attribute is

a stable identifier. If the list changes its order, for example, React will not be able to identify the items properly:

```
// don't do this
{list.map(function(item, index) {
  return (
    <div key={index}> // do not pass the list index, because if the list changes
                          //order, React will not be able to indetify properly

    ...
    </div>
  );
})}
```

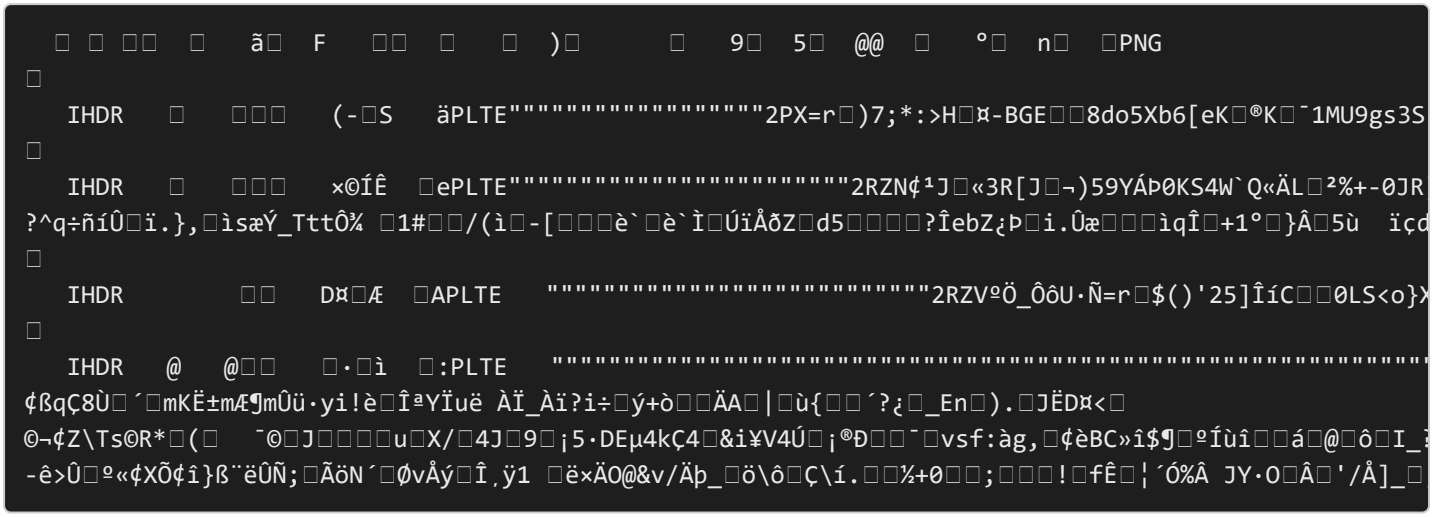
src/App.js

So far, only the title is displayed for each item. Let's experiment with displaying more of the item's properties:

```
.App {
  text-align: center;
}

.App-logo {
  animation: App-logo-spin infinite 20s linear;
  height: 80px;
}

.App-header {
  background-color: #222;
  height: 150px;
  padding: 20px;
  color: white;
}

.App-title {
  font-size: 1.5em;
}

.App-intro {
  font-size: large;
}

@keyframes App-logo-spin {
  from { transform: rotate(0deg); }
  to { transform: rotate(360deg); }
}
```

The map function is inlined concisely in your JSX. Within the map function, we have access to each item and its properties. The `url` property of each item is used as dynamic `href` attribute for the anchor tag. Not only can JavaScript in JSX be used to display items, but also to assign HTML attributes dynamically.

Let's see the rendered list's elements in the browser.



Exercises:

- Confirm the changes from the last section.
- Read more about why React's key attribute is needed ([0], [1], [2]). Don't worry if you don't understand the implementation yet, just focus on what problem it causes for dynamic lists.
- Recap the standard built-in array methods – especially *map*, *filter*, and *reduce* – which are available in native JavaScript.
- What happens if you return `null` instead of the JSX?
- Extend the list with some more items to make the example more realistic.
- Practice using different JavaScript expressions in JSX.