Web Front-End Development
Week 11: React Lists and Data Flow

Lists and Keys

https://reactjs.org/docs/lists-and-keys.html
Review of JavaScript map() function and using it in JSX in a React component.
In React when you render a list of items, React asks you to assign a unique string to the *key* property on each element in a list to differentiate each element from its siblings.
Keys tell React about the identity of each element, so that it can maintain the state across multiple renders.
Keys help React identify which items have changed, are added, or are removed.

- use a string that uniquely identifies a list item
- use IDs from your data as keys
- use the item index as a key as a last resort
- don't use indexes for keys if the items can reorder

If you don't use keys in a list you will receive the following warning:
Warning: Each child in an array or iterator should have a unique "key" prop.

Usually if you're using map() or generating a list, the elements will need keys.
Keys must be unique within a list, among their siblings, but do not need to be unique globally.

Keys are used by React but they don't get passed to your components and there is no way for a component to access its own key. Even though it may look like it is part of props, it CANNOT be referenced with `this.props.key`.
If you need the same value in your component, pass it explicitly as a prop with a different name.

Example:
https://repl.it/@aileenjp/react-favorites-list
Up until now our App component is rendering our Favorite component two times. Let's restructure so it can be set up to render a list of as many favorites as we want.

- Restructure to add a FavoriteList component to handle the list of foods. The App component now just renders FavoriteList.
  - Note that you must import all components you render
  - Also updated App.css
- FavoriteList
  - Instead of having App pass in the names of the food to the component, I created an array of foods in FavoriteList.
    - foods is an array of objects so each object has a key and a name.
    - In the future if I want to take this as input I can move this into state
- The FavoriteList component now returns an unordered list of food
  - We use the map() function to return a <li> element for each item in the foods array
    - The <li> element has a key property using the key value from the array so each item has a unique id.
    - The <Favorite> component is passed the item's name as the prop name.

- The Favorite component is mostly unchanged, only the div for each item's style has moved into FavoriteList

Data Flow
https://reactjs.org/docs/lifting-state-up.html
A component can pass its state to other components as props if needed. But state can only be owned by ONE component and therefore only changed in ONE component.
Along with the ability for a component class to pass down its state as a value to be displayed, it can also pass down the ability to *change* its state through a function.

A *stateful* component is a component which has state. That component defines a function that calls `this.setState()`.
The stateful (parent) component passes that function down to a stateless (child) component as a prop.
A stateless child component receives the parent's state as a prop and can

- access and displays the value through props
- change its parent's state through the function passed through props

A stateful, parent component defines state and a function to change its state.
The stateful, parent component passes the function down an *event handler* to a stateless, child component.

A *stateless* child component then uses that *event handler* to call its parent's function which updates the parent's state.
The child component can define a function that calls the passed-down function by accessing it through props

- The function in the child component can take an *event object* as an argument.
- The child component class uses this new function as an event handler.
- When the event fires, the event handler is called and the parent's state updates.

Parent component:
```
constructor(props) {
  super(props);
  this.state = {name: 'Aileen'};
  this.handleNameChange = this.handleNameChange.bind(this);
}

handleNameChange(newName) {
  this.setState({name: newName});
}

render() {
 return <Child name={this.state.name} onNameChange={this.
handleNameChange} />
}
```

Due to the way that event handlers are bound in JavaScript remember to bind `this.handleNameChange()` to `this` in the constructor method.
`this.methodName = this.methodName.bind(this)` to your constructor function.

Child component:
```
constructor(props) {
  super(props);
  this.handleChange = this.handleChange.bind(this);
}

handleChange() {
  const name = e.target.value;
  this.props.onNameChange(name);
}

<select onChange={this.handleChange}>
```

Here is how the Parent component's state would be set:

1. A user triggers an onChange event by selecting a value in the select list in the child component
2. The onChange event is being listened for on the select element.
3. When the event fires it calls the *event handler* function assigned to it -- `this.handleChange`
4. `handleChange()` automatically gets passed the *event object*. Using the event object (e) you can access the target's value. The target is the element that the event was fired on.
5. That name is then passed to `this.props.onNameChange(name).` `props.onNameChange` was assigned the value of `handleNameChange` in the parent component.
6. Inside of the body of `handleNameChange`, **`this.setState()`** is called and the component's state is changed.

Any time that you call this.setState(), this.setState() AUTOMATICALLY calls .render() as soon as the state has changed.

Think of `this.setState()` as actually being two things: `this.setState()`, immediately followed by `.render()`.

Example:
https://repl.it/@aileenjp/react-favorites-final
We want to be able to show total votes for our list.
Where should that live?
Is it props or state?

- FavoriteList
  - Add a constructor method
  - Add totalVotes as state

    - Create a function called handleVoteChange that updates totalVotes using setState()
        - Bind this method in the constructor
    - Since the votes don't happen in the FavoriteList component we need to pass this function as a prop to the Favorite component so it can get called whenever a vote is made.
        - Named the prop onVoteChange
    - Added the ability to see totalVotes in the last line of return()
- Favorite
    - Every time the user click the button the onClick event fires and the handleClick() method is called
        - Votes is updated (state in Favorite)
        - Call `this.props.onVoteChange();`
            - Calls `handleVoteChange() in` FavoriteList to update totalVotes
        - We could also pass data as an argument if the function needed any

When you pass an event handler as a prop there are two names that you have to choose, both in the component class that defines the event handler and passes it. These two names can be whatever you want. However, there is a naming convention that they often follow. You don't have to follow this convention, but you should understand it when you see it.

1. The name of the event handler itself.
    a. Based the name on the type of event you're listening for, such as "click"
    b. The event handler name should be the word "handle", plus the event type
    c. The event handler would then be named "handleClick".
2. The name of the prop that you will use to pass the event handler. This is the same thing as your attribute name.
    a. The prop name should be the word "on", plus the event type
    b. If you are listening for a "click" event, then the prop name would be "onClick"

The name `onClick` does *not* create an event listener when used on `<Greeting />`, it's just an arbitrary attribute name since `<Greeting />` is not an HTML-like JSX element, it's a component.

Names like `onClick` only create event listeners if they're used on HTML-like JSX elements. Otherwise, they're just ordinary prop names.

Lifting State Up

https://reactjs.org/docs/lifting-state-up.html
State is usually first added to whatever component needs it for rendering.
So what happens if more than one component needs to access or change that data?
If other components need that data then you have to think through where it should live and how other components will use that data. State lives and is updated in only ONE component – making it the "single source of truth" for the data.
To determine where state should live for each piece of state in your application:

- Identify every component that renders something based on that state.

- Find a common owner component (a single component above all the components that need the state in the hierarchy).
- Either the common owner or another component higher up in the hierarchy should own the state.
- If you can’t find a component where it makes sense to own the state, create a new component simply for holding the state and add it somewhere in the hierarchy above the common owner component.
- Pass state down to other components as props.

In React, sharing state is accomplished by moving it up to the closest common ancestor of the components that need it. This is called “lifting state up”.

- When you want to aggregate data from multiple children or to have two child components communicate with each other, move the state upwards so that it lives in the parent component. The parent can then pass the state back down to the children via props, so that the child components are always in sync with each other and with the parent.
- State lives in one component and only that component can change it. Don’t try to sync state between different components, let the data flow top-down.
- If something can be derived from either props or state, it probably shouldn’t be in the state, just calculate it in render().