ReactJS Notes

Reference: reactjs.org/tutorial/tutorial.html

What is React?

React is a declarative, efficient, and flexible JS library to build user interfaces with.
How? Through small pieces of code called “components”

React.component subclasses:

```
class ShoppingList extends React.Component {
  render() {
    return (
      <div className="shopping-list">
        <h1>Shopping List for {this.props.name}</h1>
        <ul>
          <li>Instagram</li>
          <li>WhatsApp</li>
          <li>Oculus</li>
        </ul>
      </div>
    );
  }
}

// Example usage: <ShoppingList name="Mark" />
```

JSX syntax is another popular syntax for writing structures:

React.createElement("div", { className: "shopping-list"}, React.createElement("h1", null, "Shopping List for ", props.name), React.createElement("ul", null, React.createElement("li", null, "Instagram"), React.createElement("li", null, "WhatsApp"), React.createElement("li", null, "Oculus")));

Syntax structure:
React.createElement(type, [props], [...children])
Where type is a tag name, i.e. ‘div’ / class / react fragment, props are properties, and children are internal nodes

^ShoppingList class is a **React component class / React component type**
Component takes *params*, aka **props**, returns hierarchy of views to display by render method.
Render method: returns a descriptions of what you want to see on the screen.
Render returns react element, a lightweight description of what to render.
NOTE: you can render custom react components, i.e. <ShoppingLIst />!

TUTORIAL- creating tic-tac-toe:

- The Square component renders a single `<button>`
- the Board renders 9 squares.
- The Game component renders a board with placeholder values

Saving state:

```
class Square extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      value: null,
    };
  }
```

Create a constructor with state;

*NOTE: All React component classes that have a constructor should start with “super(props)”, where super is the parent class constructor

Development: testing in Chrome/ Firefox has react devTools extension, which can show component tree.

*State private to a component that defines it. Thus, to setup a “controlled component”, have the controlling component(left) hold the properties constructor and event handlers, while letting the controlled component (right) refence the properties.

```
class Board extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      squares: Array(9).fill(null),
    };
  }

  handleClick(i) {
    const squares = this.state.squares.slice();
    squares[i] = 'X';
    this.setState({squares: squares});
  }

  renderSquare(i) {
    return (
      <Square
        value={this.state.squares[i]}
        onClick={() => this.handleClick(i)}
      />
    );
  }

  render() {
    const status = 'Next player: X';

    return (
      <div>
        <div className="status">{status}</div>
        <div className="board-row">
          {this.renderSquare(0)}
```

```
class Square extends React.Component {
  render() {
    return (
      <button
        className="square"
        onClick={() => this.props.onClick()}
      >
        {this.props.value}
      </button>
    );
  }
}
```

Our original class, square, had kept track of individual statuses in the cells. To be able to compare cells, we needed a way to store all cells on the board for easy comparison, so we “lifted state up”, creating a parent called board.

MUTABILITY

You can change data by mutating it or replace the data w/ a new copy that has been updated

- .slice() method: creates a copy of array instead of modifying existing array

Why make a copy?

- Complex features easier to implement, i.e. game history, and “jump back” to previous moves
- Detecting changes for immutable object easier (functionality: comparing object w/ previous copies of itself)
- Component re-rendering support

Function components- components that only contain a render method. Square class can become a function!

(+): easier to write than classes

Feature in newer JS: Able to compare array elements.

```
const lines = [
  [0, 1, 2],
  [3, 4, 5],
  [6, 7, 8],
  [0, 3, 6],
  [1, 4, 7],
  [2, 5, 8],
  [0, 4, 8],
  [2, 4, 6],
];
for (let i = 0; i < lines.length; i++) {
  const [a, b, c] = lines[i];
```

CONSTRUCTING A HISTORY

Lift state up from board.

Picking a key:

For rendering a list, react needs to determine what has changed. Easy way to determine changes is by using keys.

Ex) using alexa, ben, and Claudia as keys to do swapping

Imagine transitioning from

```
<li>Alexa: 7 tasks left</li>
<li>Ben: 5 tasks left</li>
```

to

```
<li>Ben: 9 tasks left</li>
<li>Claudia: 8 tasks left</li>
<li>Alexa: 5 tasks left</li>
```