Introduction to ReactJS

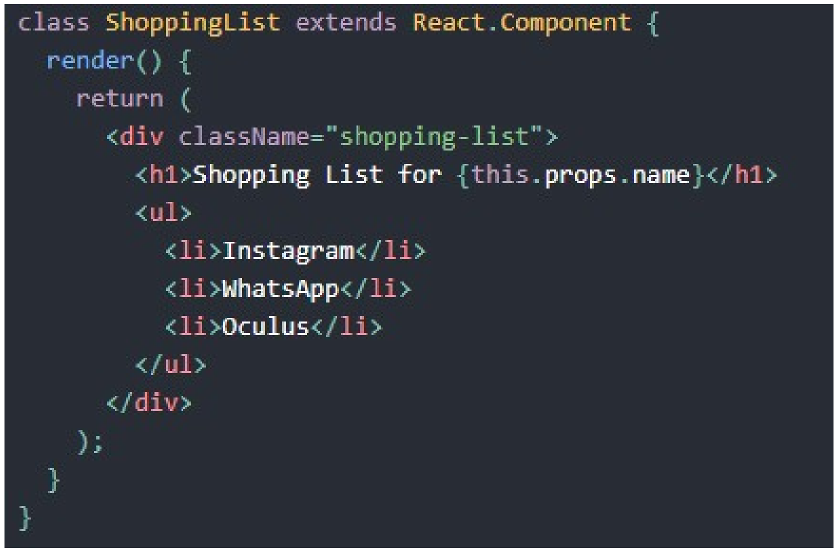
Taken from: https://reactjs.org/tutorial/tutorial.html

ReactJS setup

- Follow instructions for local installation at:
    - https://reactjs.org/tutorial/tutorial.html
- Create React App locally:
    - npm install -g create-react-app
    - npx create-react-app
- Follow instructions for creating a TicTacToe game frame

What is React?

- React is a declarative, efficient, and flexible JavaScript library for building user interfaces.
- It lets you compose complex UIs from small and isolated pieces of code called “components”.



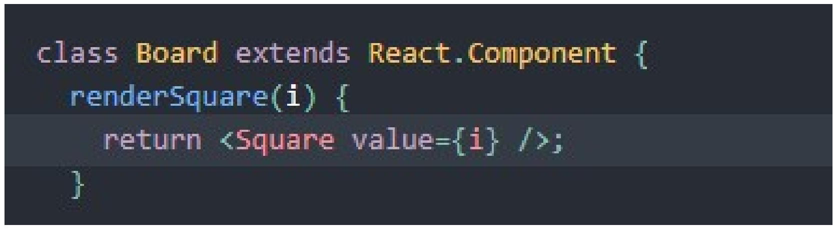
```
class ShoppingList extends React.Component {
  render() {
    return (
      <div className="shopping-list">
        <h1>Shopping List for {this.props.name}</h1>
        <ul>
          <li>Instagram</li>
          <li>WhatsApp</li>
          <li>Oculus</li>
        </ul>
      </div>
    );
  }
}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

React Component Class

- Component tell React what we want to see on the screen
- A component takes in parameters called props
- Render method returns a description of what you want to see on the screen
- Render returns a React element that describes what to render
- React developers use JSX for writing structures to be rendered
- JSX is an XML/HTML-like syntax that allows us to put HTML into JavaScript
- React components can be composed and rendered as required
- We can refer to above shopping list as <ShoppingList />
- Each React component is encapsulated and can operate independently
- This allows building complex UI4s from simple components

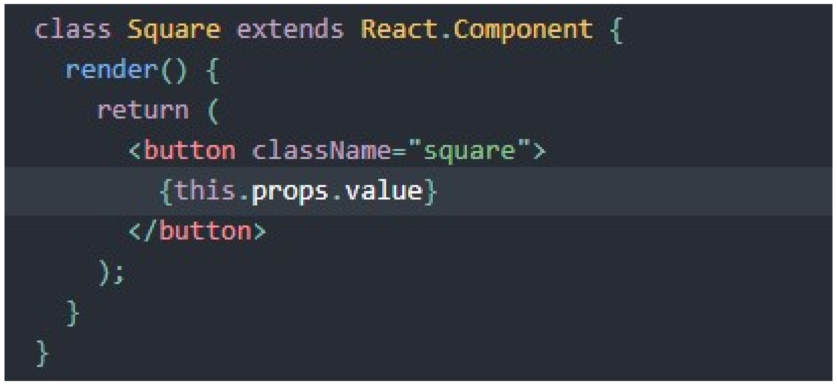
Passing Data Through Props

- Change the renderSquare method to pass a prop called value to the Square:



```
class Board extends React.Component {
  renderSquare(i) {
    return <Square value={i} />;
  }
```

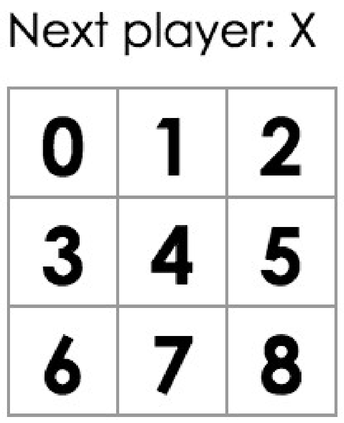
- Change Square’s render method to show the square value:



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

Result of passing a prop

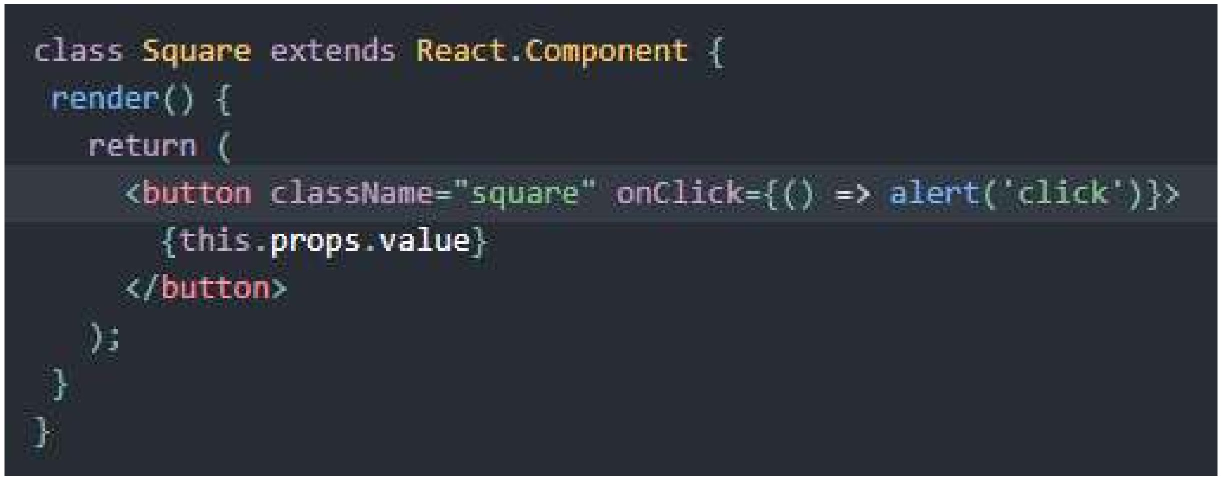
- Refresh the browser to see the number in each square
- We passed a prop from a parent Board component to a Child square component
- Information flows in React apps by passing props from parents to children


Next player: X

0	1	2
3	4	5
6	7	8

Making an interactive Component

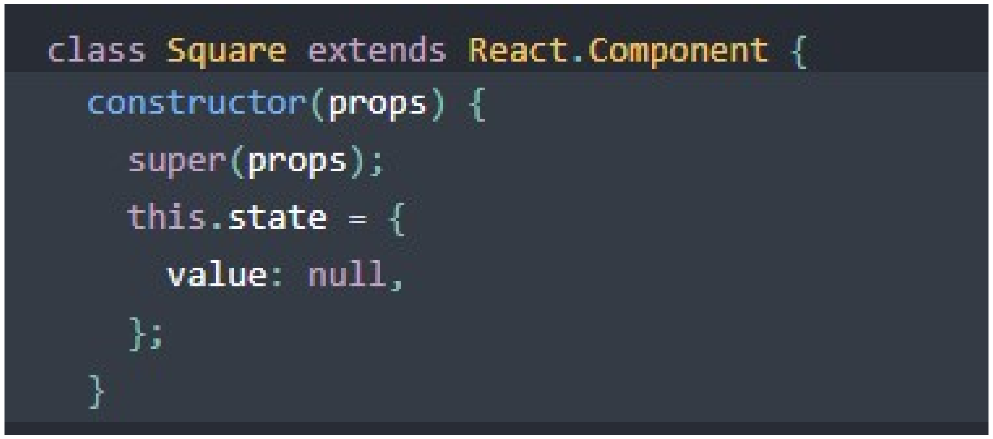
- Getting an alert
- Note the use of arrow function



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

Using state to remember actions

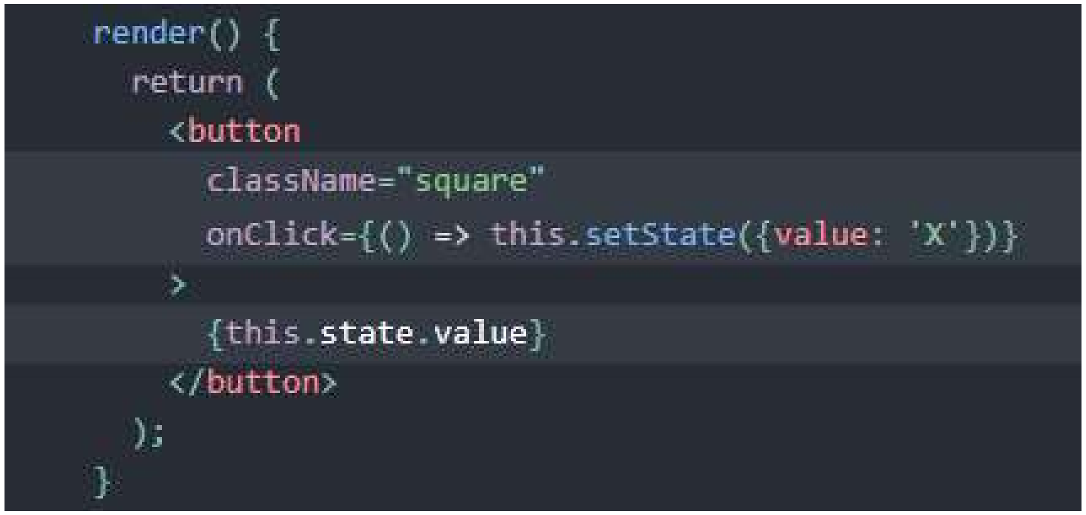
- State of a component is to be initialized in a constructor of a component
- State should be considered as private to a React component
- Add a constructor to the Square class to initialize state
- Following JavaScript guidelines, all React component classes with a constructor should have a super(props) call



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

Changing the states

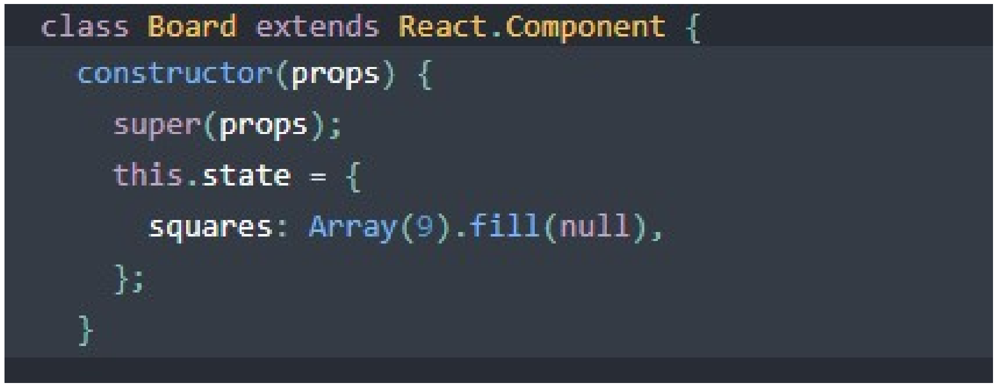
- Use this.setState from an onClickhandler in the render method for changing state of a square
- Calling a setState in a component will automatically update the child components inside it



```
render() {
  return (
    <button
      className="square"
      onClick={() => this.setState({value: 'X'})}
    >
      {this.state.value}
    </button>
  );
}
```

Lifting State Up

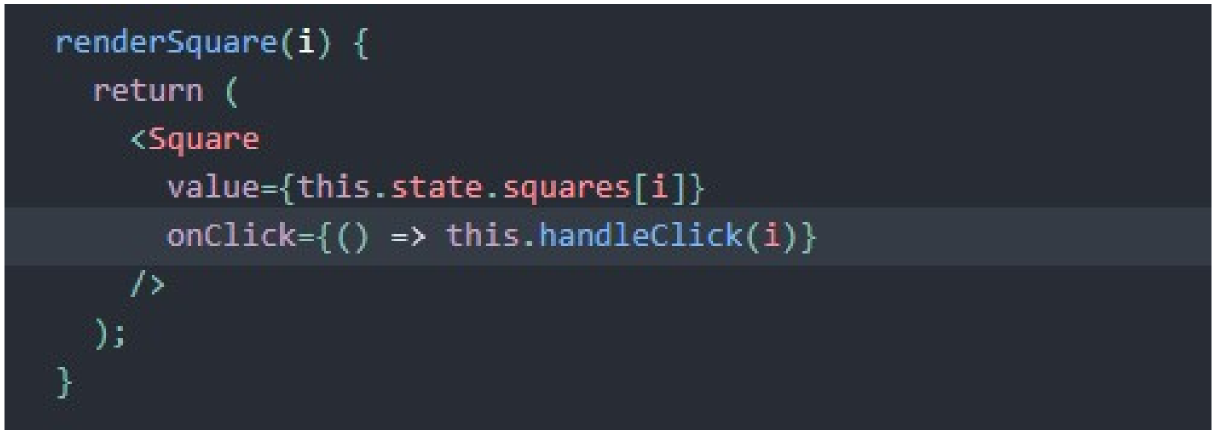
- To determine winner, the value of each of the 9 squares need to be in one location
- Best approach is to store game's state in the parent Board component
- Board tells each square what to display by passing a prop
- Ass a constructor to Board and set initial values with 9 nulls



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

Passing States down to the Squares

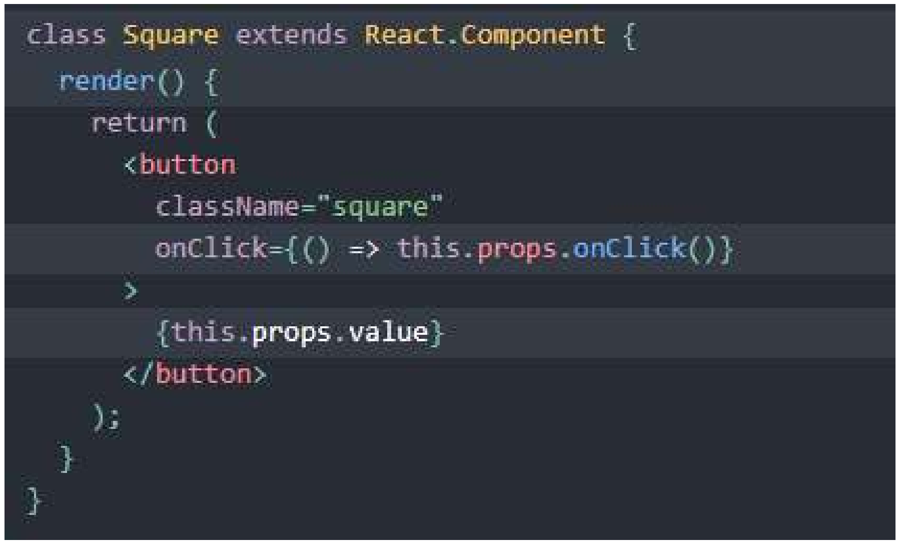
- Modify renderSquare method to read value from the board’s state
- Pass down a function which will get called when a Square is clicked



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

Modify Square Class

- Board class passes down two props to Square: value and onClick
- Square doesn't need to keep track of state now, so we can delete square state
- Also, delete square constructor and change render to the following:

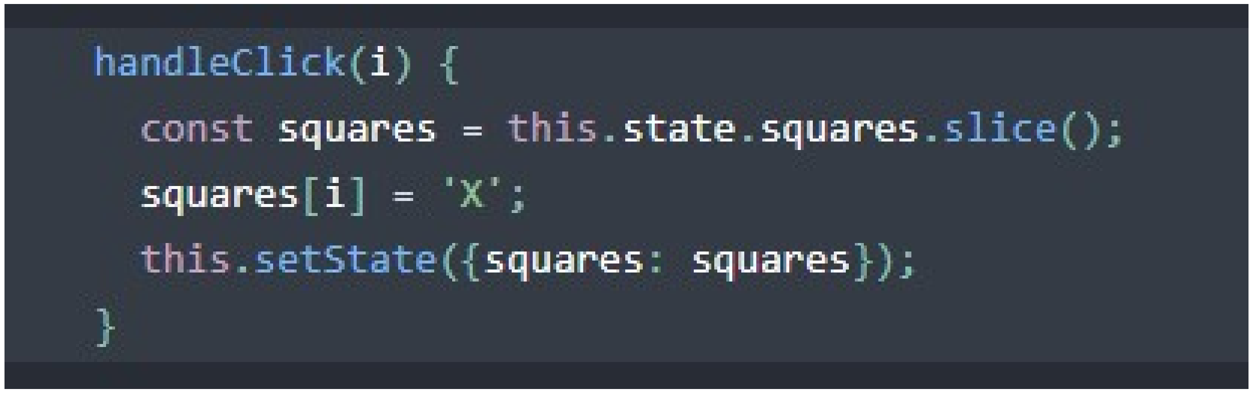


```
class Square extends React.Component {
  render() {
    return (
      <button
        className="square"
        onClick={() => this.props.onClick()}
      >
        {this.props.value}
      </button>
    );
  }
}
```

What will happen when a Square is clicked?

What will happen when a Square is clicked?

- We have not defined the handleClick() method yet, so our code crashes
- Add handleClick to the Board class



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

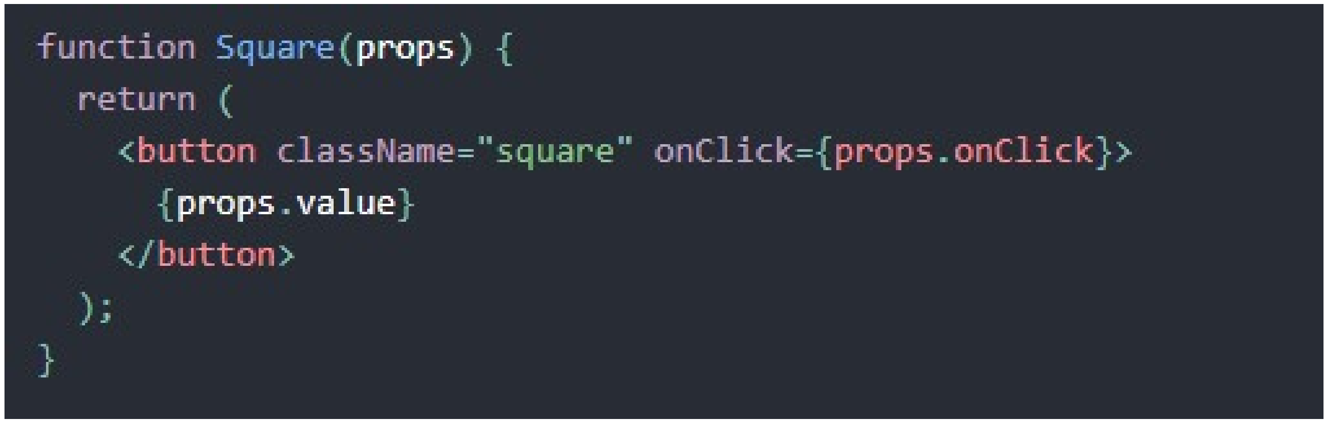
- The Square component receive values from Board component
- Square components are now controlled components

Mutability vs. Immutability

- We used the .slice() operator to create a copy of the squares array instead of modifying the existing array
- Mutation refers to changing data directly, other approach is replacing the data with a new copy
- Immutability allows us to implement ‘time travel’ – useful for undo and redo operations
- Detecting changes in the immutable objects is easier

Function Components

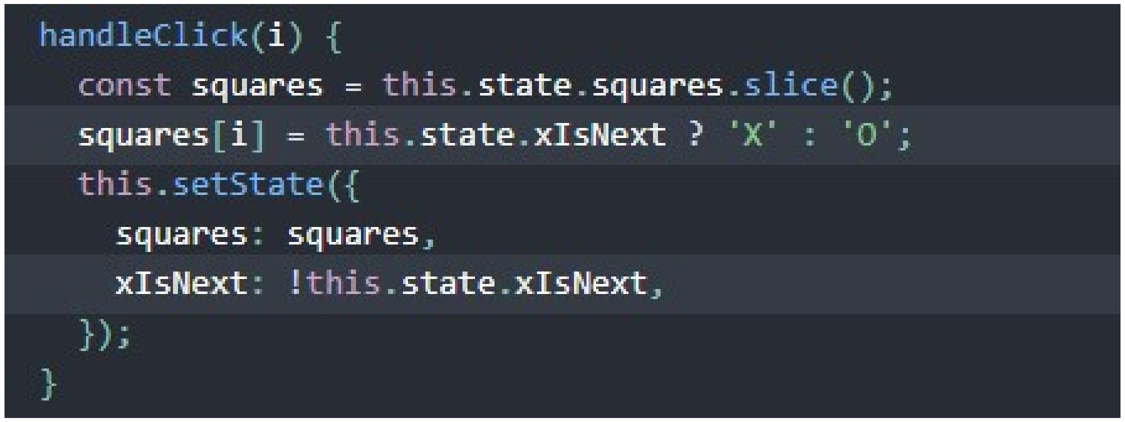
- React classes which contain only render method and don’t have own state could be converted to function components
- Function takes props as input and returns what should be rendered
- Replace the Square class with a function



```
function Square(props) {
  return (
    <button className="square" onClick={props.onClick}>
      {props.value}
    </button>
  );
}
```

Adding Logic to Take Turns

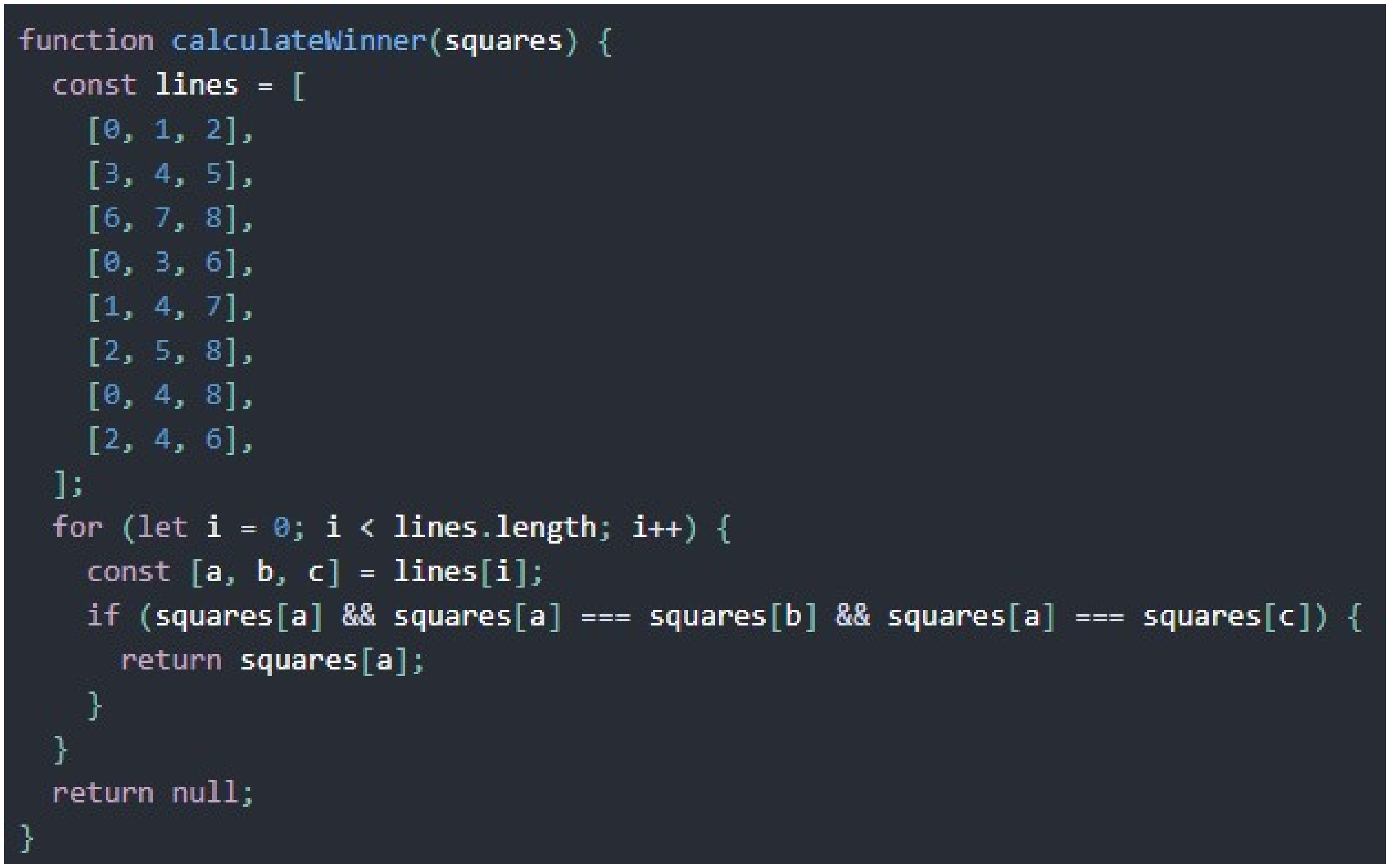
- First move is always X
  - Add xIsNext: true, to the Board state in it's constructor
- Change handleClick function to change the value of squares and xIsNext depending on the turn



```
handleClick(i) {
  const squares = this.state.squares.slice();
  squares[i] = this.state.xIsNext ? 'X' : 'O';
  this.setState({
    squares: squares,
    xIsNext: !this.state.xIsNext,
  });
}
```

- Change status text in Board's render to display player with next turn
  - const status = 'Next player: ' + (this.state.xIsNext ? 'X' : 'O');

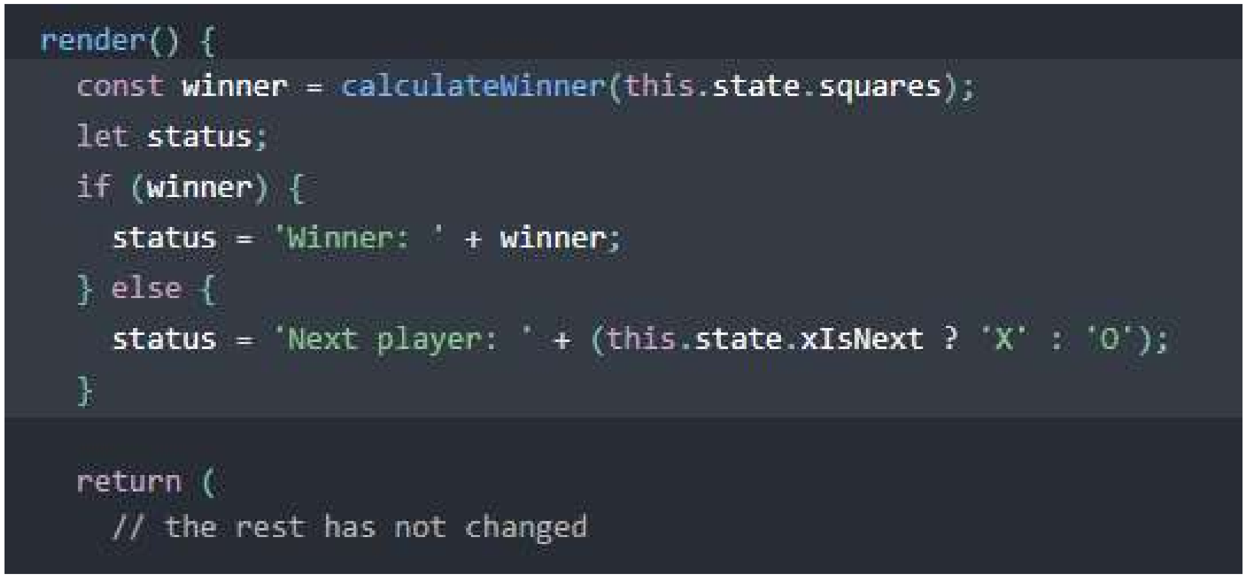
Calculating a Winner – Helper Function



```
function calculateWinner(squares) {
  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];
    if (squares[a] && squares[a] === squares[b] && squares[a] === squares[c]) {
      return squares[a];
    }
  }
  return null;
}
```

Announcing a Winner

- Call calculateWinner(squares) in the Board’s render function
- If a player has won, display text such as “Winner: X” or “Winner: O”

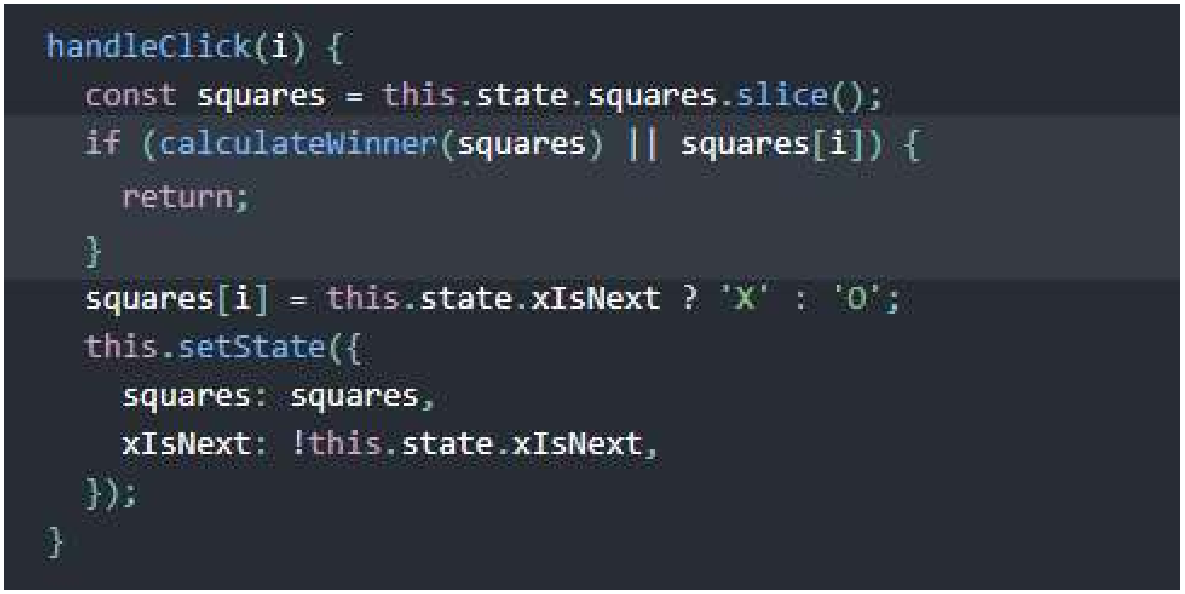


```
render() {
  const winner = calculateWinner(this.state.squares);
  let status;
  if (winner) {
    status = 'Winner: ' + winner;
  } else {
    status = 'Next player: ' + (this.state.xIsNext ? 'X' : 'O');
  }

  return (
    // the rest has not changed
```

Ignore Clicks if Game is Finished

- Change the Board’s handleClick function to return early by ignoring a click if someone has won the game or if a Square is already filled



```
handleClick(i) {
  const squares = this.state.squares.slice();
  if (calculateWinner(squares) || squares[i]) {
    return;
  }
  squares[i] = this.state.xIsNext ? 'X' : 'O';
  this.setState({
    squares: squares,
    xIsNext: !this.state.xIsNext,
  });
}
```

Adding Time Travel

- Homework