React Setup (using traditional way)

Components: *Heart of React*

index.html

```
<html>
<body>
<h1>Demo: Hello</h1>

<div id="root">
<!-- component will go in this div -->  </div>

<script src="https://unpkg.com/react/umd/react.development.js"> </script>

<script src="https://unpkg.com/react-dom/umd/react-dom.development.js"> </script>

<script src="https://unpkg.com/babel-standalone"> </script> //for JSX support

<script src="index.js" type="text/jsx"> </script> //it is text/jsx

</body>
</html>
```

A **component** is a React class with a ***render*** method:

demo/hello/index.js

```
class Hello extends React.Component {
  render() {
    return <p>Hi Everyone!</p>;
  }
}
```

We add our component to HTML with ***ReactDOM.render*** :

*demo/hello/index.**js***

```
ReactDOM.render(<Hello />, document.getElementById("root"));
```

```
// Note – Component name first letter must be capital to render properly.
```

// In JSX, lower-case tag names are considered to be HTML tags. However, lower-case tag names with a dot (property accessor) aren't.

JSX: JSX is like HTML embedded in JavaScript

```
class Hello extends React.Component {
  render() {
    return <p>Hi Everyone!</p>;
  }
}

ReactDOM.render(<Hello />, document.getElementById("root"));
```

JavaScript:

```
if (score > 100) {
  return <b>You win!</b>
}
```

You can also "re-embed" JavaScript in JSX:

```
if (score > 100) {
  return <b>You win, { playerName }</b>
}
```

Using JSX

- JSX isn't legal JavaScript
    - It has to be "transpiled" to JavaScript
- You can do this with Babel

JSX Rules

JSX is more strict than HTML — elements must either:

- Have an explicit closing tag: `<b> ... </b>`
- Be explicitly self-closed: `<input name="msg" />`
- Cannot leave `/` or will get syntax error
- Note ***div*** wrapper — JSX often renders a *single top-level element.*

App

It's conventional for the top-level component to be named ***App***.

This renders the other components:

App.js

```
class App extends React.Component {
  render() {
    return (
      <div>
        <h1>Greetings!</h1>
        <Hello/>
        <Goodbye/>
      </div>
    );
  }
}
```

- This way, readers of code know where to start
- This is usually the only thing rendered in ***index.js***

Order of Script Tags

demo/hello-2/index.html

```
<script src=
  "http://unpkg.com/react/umd/react.development.js"></script>
<script src=
  "http://unpkg.com/react-dom/umd/react-dom.development.js">
</script>

<script src="http://unpkg.com/babel-standalone"></script>

<script src="Hello.js" type="text/jsx"></script>
<script src="index.js" type="text/jsx"></script>
```

Make sure any components you need in a file are loaded by a previous ***script*** tag.

Properties: aka props

- A useful component is a reusable one.
- This often means making it configurable or customizable.

Hello.js

```
class Hello extends React.Component {
  render() {
    return <p>Hi Everyone!</p>;
  }
}
```

It would be better if we could *configure* our greeting.
Our greeting will be *Hi* _____ *from* _____.
Let's make two "properties":
to
Who we are greeting
from
Who our greeting is from

Demo: Hello-2

demo/hello-2/index.js

```
ReactDOM.render(
<Hello to="me" from="you" />, document.getElementById("root")
);
```

Set properties on element; get using ***this.props.propName*.**

demo/hello-2/Hello.js

```
class Hello extends React.Component {
  render() {
    return (
      <div>
        <p>Secret Message: </p>
        <p>Hi {this.props.to} from {this.props.from}</p>
      </div>
    );
  }
}
```

Properties Requirements

- Properties are for *configuring* your component
- Properties are immutable
- Properties can be strings:

```
<User name="Jane" title="CEO" />
```

- For other types, embed JS expression using the curly braces:

```
<User name="Jane" salary={ 100000 }
    hobbies={ ["bridge", "reading", "tea"] } />
```

Using Properties

- Get to properties inside class with ***this.props.propertyName***
- Properties are immutable — cannot change!

Conditionals in JSX:

The ***render()*** method can return either:

- a **single valid** DOM object (`return <div>...</div>`)
- an array of DOM objects (but don't do this yet!)
- ***null*** (***undefined*** is not ok!)

You can put whatever logic you want in your ***render()*** method for this:

```
class Lottery extends React.Component {
  render() {
    if (this.props.winner)
      return <b>You win</b>;
    else
      return <b>You lose</b>;
  }
}
```

Ternary

It's very common in ***render()*** to use ternary operators:

```
class Lottery extends React.Component {
  render() {
    return (
      <b>You { this.props.winner ? "win" : "lose" } </b>
    )
  }
}
```

Demo: Slots!

demo/slots/Machine.js

```
class Machine extends React.Component {
  render() {
    const { s1, s2, s3 } = this.props;
    const winner = (s1 === s2) && (s2 === s3);

    return (
      <div className="Machine">
        <b>{s1}</b> <b>{s2}</b> <b>{s3}</b>
        <p>You {winner ? "win!" : "lose!"}</p>
      </div>
    );
  }
}
```

demo/slots/index.js

```
ReactDOM.render(
  <Machine s1="🍇" s2="🍇" s3="🍇" />,
  document.getElementById("root")
);
```

Looping in JSX:

It's common to use ***array.map(fn)*** to output loops in JSX:

```
class Messages extends React.Component {
  render() {
    const msgs = [
      {id: 1, text: "Greetings!"},
      {id: 2, text: "Goodbye!"},
    ];

    return (
      <ul>
        { msgs.map(m => <li>{m.text}</li>) }
      </ul>
    );
  }
}
```

Demo: Friends!

demo/friends/Friend.js

```
class Friend extends React.Component {
  render() {
    const { name, hobbies } = this.props;
    return (
      <div>
        <h1>{name}</h1>
        <ul>
          {hobbies.map(h => <li>{h}</li>)}
        </ul>
      </div>
    );
  }
}
```

demo/friends/index.js

```
ReactDOM.render(
  <div>
    <Friend name="Jessica" hobbies={["Tea", "Frisbee"]} />
    <Friend name="Jake" hobbies={["Chess", "Cats"]} />
  </div>,
  document.getElementById("root")
);
```

Default Props:

Components can specify default values for missing props

Demo: Hello-3

demo/hello-3/Hello.js

```
class Hello extends React.Component {
  static defaultProps = {
    from: "Joel",
  };

  render() {
    return <p>Hi {this.props.to} from {this.props.from}</p>;
  }
}
```

Set properties on element; get using ***this.props.propName***.

demo/hello-3/index.js

```
ReactDOM.render(
  <div>
    <Hello to="Students" from="Elie" />
    <Hello to="World" />
  </div>,
  document.getElementById("root")
);
```

THE END

Create react app (new way to setup)