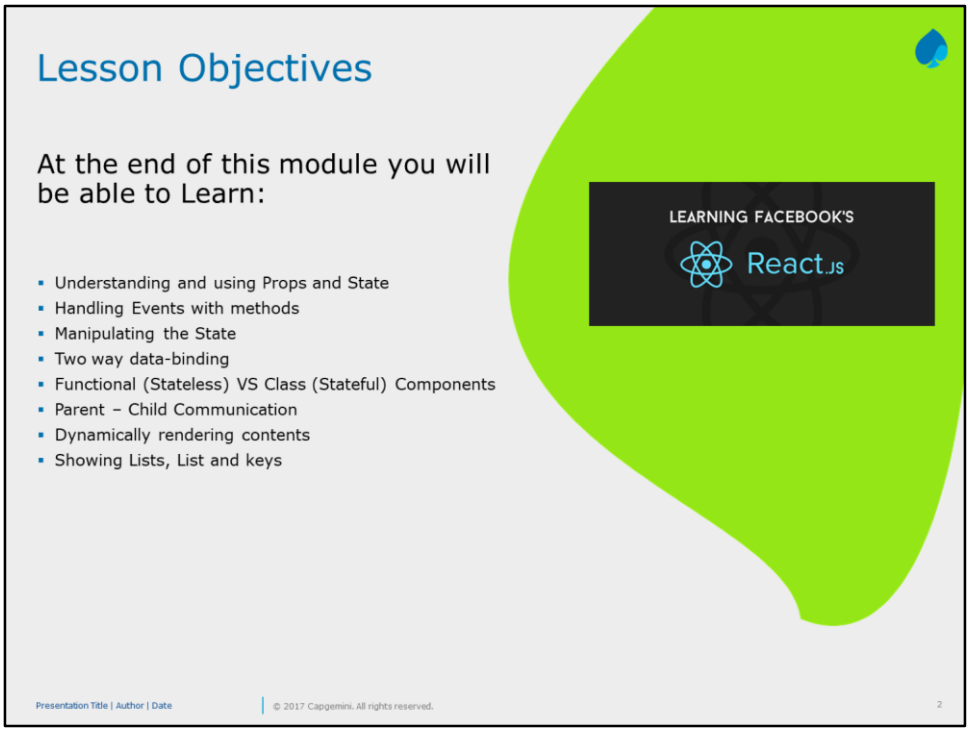
Instructor Notes:

Add instructor notes here.

Instructor Notes:

Add instructor notes here.



Instructor Notes:

Add instructor notes here.

Understanding and using Props and State

- Props:
  - Its shorthand for properties.
  - Using props, components talks to each other ie., they share data between each other.
  - Props are immutable.
  - All Props can be accessible with this.props
- React.PropTypes :
- React.PropTypes are used to run type checking on the props for a component.
- Allows you to control the presence, or types of certain props passed to the child component.
- **Default Props:**

```
App.defaultProps = {   tech: 'React Js'   };
```

Props are immutable. Because these are developed in the concept of pure functions. In pure functions we cannot change the data of parameters. So, also cannot change the data of a prop in ReactJS.

- Props are nothing but properties which are single values or objects containing a set of values that are *passed to React Components*
- Pass custom data to React Component
- this.props contains the props that were defined by the caller of this component,ie., all props can be accessible through this.props

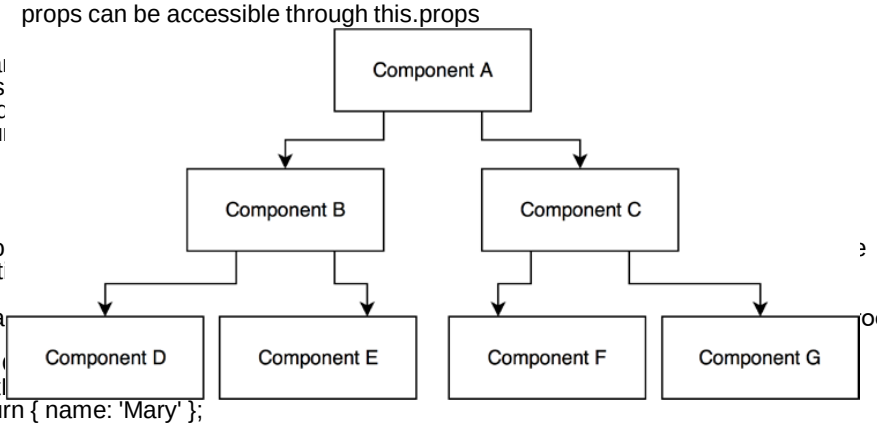
Exa
clas
renc
retu

}

Prio e for a
part

Rea oot'));

var
getI
return { name: 'Mary' };
},

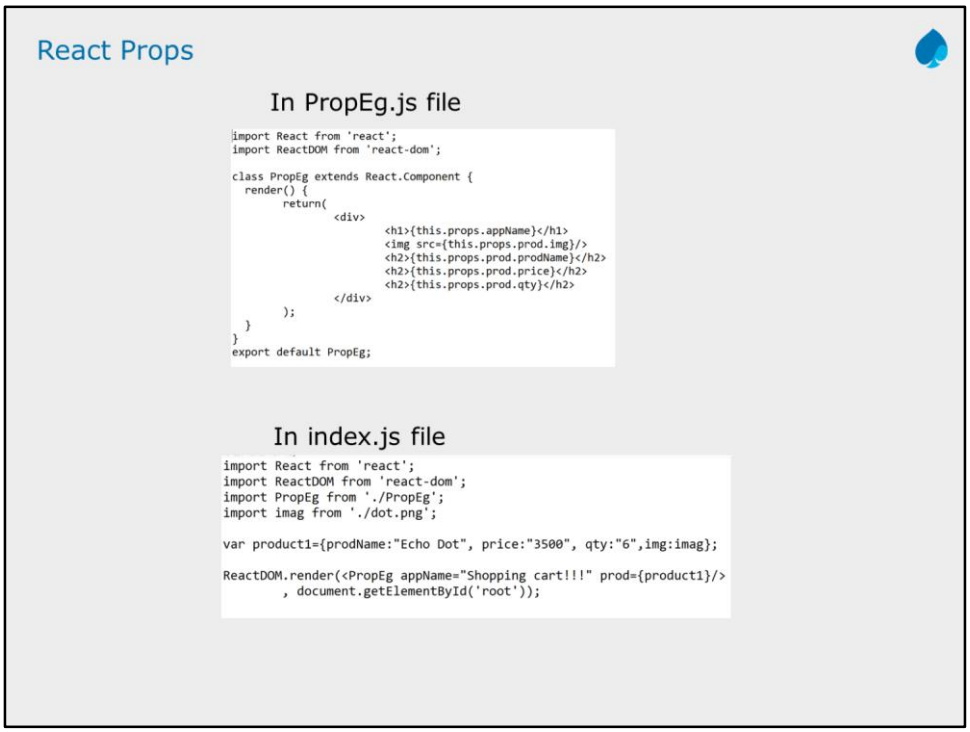


Whether you declare a component as a function or a class, it must never modify its own props.
Such functions are called "pure" because they do not attempt to change their inputs, and always return the same result for the same inputs.

Example:

```
function findDiff(a,b)
{
        return a-b;
}
```

Instructor Notes:



Once you store data in props cannot change the data, its read only info
Can be shared with other components also

Props without using jsx :

React.createElement(***Hello***, { alertNumber : 1 }, ***null***);

Here we also pass ***null***, which in this case indicates "empty" innerHTML. Or more precisely, no children.

React.PropTypes has moved into a different package since React v15.5. Please use theprop-types library instead.

PropTypes by passing them as an option to createClass():
const Component = React.createClass({ propTypes: { *// propType definitions go here* }, render: function() {} });

Eg:

const Component = React.createClass({ propTypes: { name: React.PropTypes.string }, *// ...* })

After Es6:

Defining propTypes in a class-based component using ES6 syntax is slightly different as it needs to be defined as a class method on the component.
For below code to work use
npm install prop-types –save, inside the project folder

For example:

```
class Component extends React.Component {
                        render() {}
                        }
Component.propTypes = {/* definition goes here*/};
```

Example for ES6:

App.propTypes = { name: PropTypes.string };

This has to be written in index.js

ReactDOM.render(<ParentComponent num="10"/>, document.getElementById('root'));

In ParentComponent.js

```
ParentComponent.defaultProps={
                        num:20
}
```

If num is not there in ParentComponent in above snippet ,then it takes value given by defaultProps, and out will be printing 20

ReactDOM.render(<ParentComponent num="10"/>, document.getElementById('root'));
if it is given then it takes 10 ,as given in above code snippet.

There is a problem in above code snippet, we have passed 10 as string so when it goes to expression in ParentComponent and doesn't add two numbers ,but it concatenates to give output as 510, instead of 15.

Instructor Notes:

Add instructor notes here.

Working with State

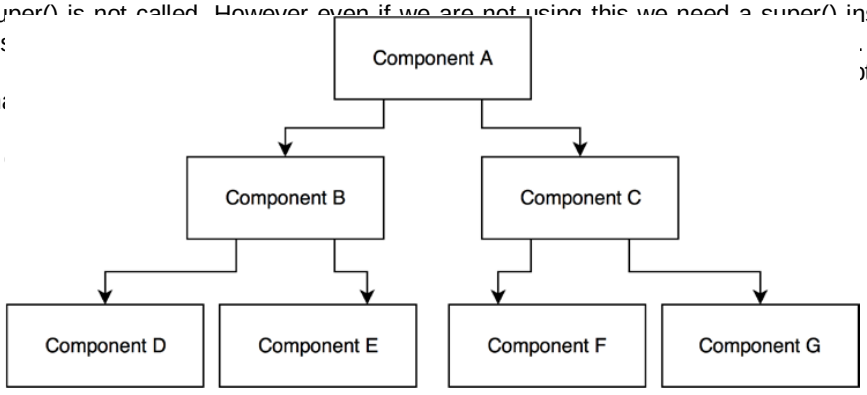
- React components can be made dynamic by adding state to it.
- State is used when a component needs to change independently of its parent.
- React component's initial state can be populated using getInitialState() with an object map of keys to values.
- React component's state can be accessed using this.state
- React component's state can be updated using this.setState() with an object map of keys which can be updated with new values. Keys that are not provided are not affected.
- setState() merges the new state with the old state.
- As a best practice the nested React components should be stateless. They should receive the state data from their parent components via this.props and render that data accordingly.

Whenever setState is called, the virtual DOM re-renders, the diff algorithm runs, and the real DOM is updated with the necessary changes.

React components are composable(Nested components). As a result, we can have a hierarchy of React components. Imagine that we have a parent React component that has two child components, and each of them in turn has another two child components. All the components are stateful and they can manage their own state.

The reason why this cannot be allowed before super() is because this is uninitialized if super() is not called. However even if we are not using this we need a super() inside a cons . Thus, you t have to h

We



It will be difficult to figure out what the last child component in the hierarchy will render if the top component in the hierarchy updates its state. So as a best practice top-level React component are stateful which encapsulate all of the interaction logic, manage the user interface state, and pass that state down the hierarchy to stateless components, using props.

Instructor Notes:

Add instructor notes here.

Demo

State:

react-state-2-2019
react-state-2019

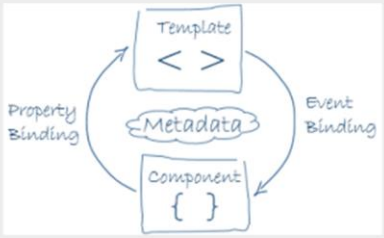
Usage of Function.prototype.bind()

```
this.x = 9;
var module = {
  x: 81,
  getX: function() { return this.x; }
};

module.getX(); // 81

var retrieveX = module.getX;
retrieveX(); // 9, because in this case, "this" refers to the global object

// Create a new function with 'this' bound to module
var boundGetX = retrieveX.bind(module);
boundGetX(); // 81
```

Instructor Notes:

Two way data-binding

- Two Binding is a mechanism for coordinating parts of a template with parts of a component.
- When properties in the model get updated, so does the UI.
- When UI elements get updated, the changes get propagated back to the model.



- Angular supports two way binding
- React Js supports one way binding only (uni-directional flow).
- React is more performant than angular.

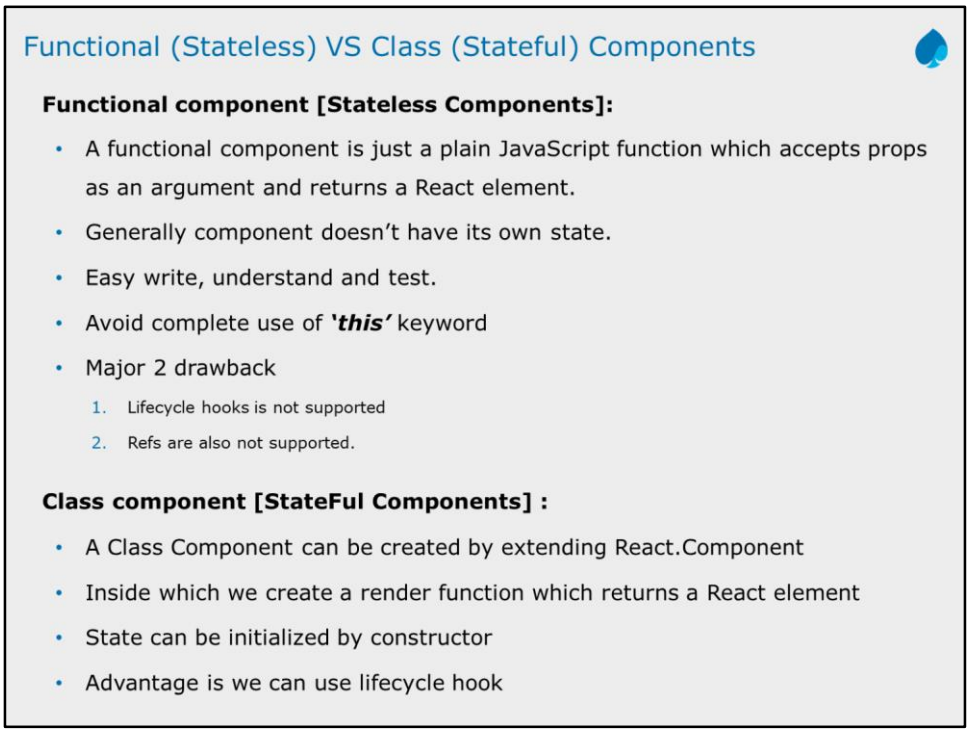
A unidirectional data flow means that when designing a React app you often nest child components within higher-order parent components. The parent component(s) will have a container for the state of your app.

render() { return <input value={this.state.value} onChange={this.handleChange} /> } handleChange(e) { this.setState({value: e.target.value}); }
The value of the <input /> is controlled *entirely* by the render function.

The only way to update this value is from the component itself, which is done by attaching an onChange event to the <input /> which sets this.state.value to with the React component method setState

The <input />does not have direct access to the components state, and so it cannot make changes

Instructor Notes:



A Functional component is just a plain JavaScript function, you cannot use setState() in your component

Functional component are much **easier to read and test** because they are plain JavaScript functions without state or lifecycle-hooks

We can use Arrow functions for defining component

Example:

```
({ name }) => (<div>Hello, {name}!</div>);
```

Instructor Notes:

Differences between functional and class-Components

1. Main difference is Syntax
2. A functional component is just a plain JavaScript function which accepts props as an argument and returns a React element. But class component requires you to extend from React.Component and create a render function which returns a React element.

Functional Component

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

Class Component

```
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

Main difference between functional and class components will be seen clearly through the transpiled code.

Instructor Notes:

As functional component transpiled by Babel

```
function Welcome(props) {
  return _react2.default.createElement(
    'h1',
    null,
    'Hello, ',
    props.name
  );
}
```

The exact same component written as class component transpiled with Babel

```
var Welcome = function (_React$Component) {
  _inherits(Welcome, _React$Component);

  function Welcome() {
    _classCallCheck(this, Welcome);

    return _possibleConstructorReturn(this, (Welcome.__proto__ || Object.getPrototypeOf(Welcome)).apply(this, arguments));
  }

  _createClass(Welcome, [{
    key: 'render',
    value: function render() {
      return _react2.default.createElement(
        'h1',
        null,
        'Hello, ',
        this.props.name
      );
    }
  }]);

  return Welcome;
}(_react2.default.Component);
```

Instructor Notes:

Differences between Functional and Class Components

Since functional component is just a plain JavaScript function, you cannot use setState() in your component. That's the reason why they also get called functional stateless components.

we cannot use in functional components are lifecycle hooks.

When Should we Use Functional component is :

1. Functional component are much **easier to read and test** because they are plain JavaScript functions without state or lifecycle-hooks
2. It will get easier to separate container and presentational components because you need to think more about your component's state if you don't have access to setState() in your component
3. Less Code when using this.

Instructor Notes:

Add instructor notes here.

Demo

Demo of

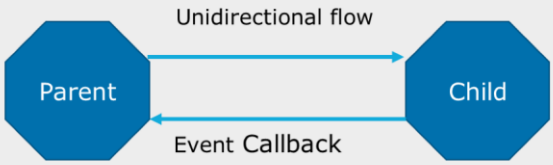
react-functional-class-component2019

Add the notes here.

Instructor Notes:

Parent – Child Communication

- In React greatest advantage is using 'uni-directional flow'
- Data is passed from Parent to child components
- A child component can never send a prop back to the parent component that is called unidirectional data flow.
- If we can pass the data from child to parent. You can use callback and state in your application.



Passing props down the component tree - Props drilling

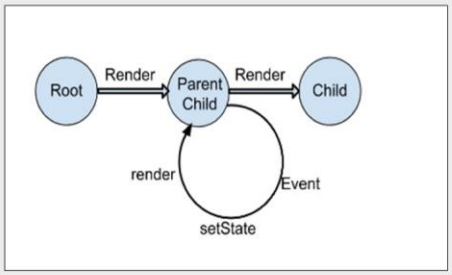
Context provides a way to share values like this between components without having to explicitly pass a prop through every level of the tree.

Instructor Notes:

Add instructor notes here.

Unidirectional data flow

- In React, application data flow is unidirectional via the state and props objects, as opposed to the two-way binding of libraries like Angular.
- In a multi component hierarchy, a common parent component will manage the state and pass it down the chain via props.
- State should be updated using the setState method to ensure that a UI to update and the resulting values should be passed down to child components using attributes that are accessible in said children via this.props



In two way data binding the view is updated when the state changes, and vice versa. For example, when you change a model in AngularJS the view automatically reflects the changes. Also an input field in the view can alter model. While this works for many apps, it can lead to cascading updates and changing one model may trigger more updates. As the state can be mutated(alter) by both controller and view, sometimes the data flow becomes unpredictable.

React doesn't encourage bi-directional binding to make sure you are following a clean data flow architecture. The major benefit of this approach is that data flows throughout your app in a single direction and you have better control over it.

By keeping the data flow unidirectional you keep a single source of truth. Views are just the functions of the application state. Change in the state will change the view. This is way more predictable and gives a clear idea about how different components react to state change.

Instructor Notes:

Dynamically rendering contents

Conditional rendering in React works the same way conditions work in JavaScript. Use JavaScript operators like if or the conditional operator to create elements representing the current state, and let React update the UI to match them.

```
function Product(props) {
  const available= props.available;
  if (available){
    return <SubProduct />;
  }
  return <NoProduct />;
}

ReactDOM.render(
   <Product available={false} />,
  document.getElementById('root')
);
```

We can also make use of switch cases, if else, ternay operator etc,.

Instructor Notes:

Showing Lists and Keys

Lists:

Assume that we have a huge set of numbers, say some 2000 numbers and we have to find the square of number and get it on the screen. Old way is to make use is using normal for loop.

So we have a Array.map() method function to create a new array that has the same number of elements, and where each element is the result of calling the function you provide.

```
<div>
 {props.items.map((item, index) => ( <Item key={index} item={item} /> )}
</div>
```

Keys :

Keys help React identify which items have changed, are added, or are removed.

Keys should be given to the elements inside the array to give the elements a stable identity.

Rules for keys :

unique : Every item in the list must have a unique key

Permanent: An item's key must not change between re-renders, unless that item is different

Remember React uses a virtual DOM, and it only redraws the components that changed since the last render.

'Key' helps to identify items in the list

function displayOutput(props) {()= []; for(let i = 0; i < props.items.length; i++) { array.push(<Item key={i} item={props.items[i]} />); }
t return ({array}); }

What are some exceptions where it is safe to use index as key?
If your list is static and will not change.
The list will never be re-ordered.
The list will not be filtered (adding/removing items from the list).
There are no ids for the items in the list.
If all these exceptions qualify, then you can use an index as a key.

Summary

Lists are performant heavy and need to be used carefully.
Make sure every item in the list has a unique key.
It is prefered to not use indexes as a key unless you know for sure that the list is a static list (no additions/re-ordering/removal to the list).
Never use unstable keys like *Math.random()* to generate a key.
React will run into performance degradation and unexpected behaviour if unstable keys are used.

Instructor Notes:

Add instructor notes here.

Demo

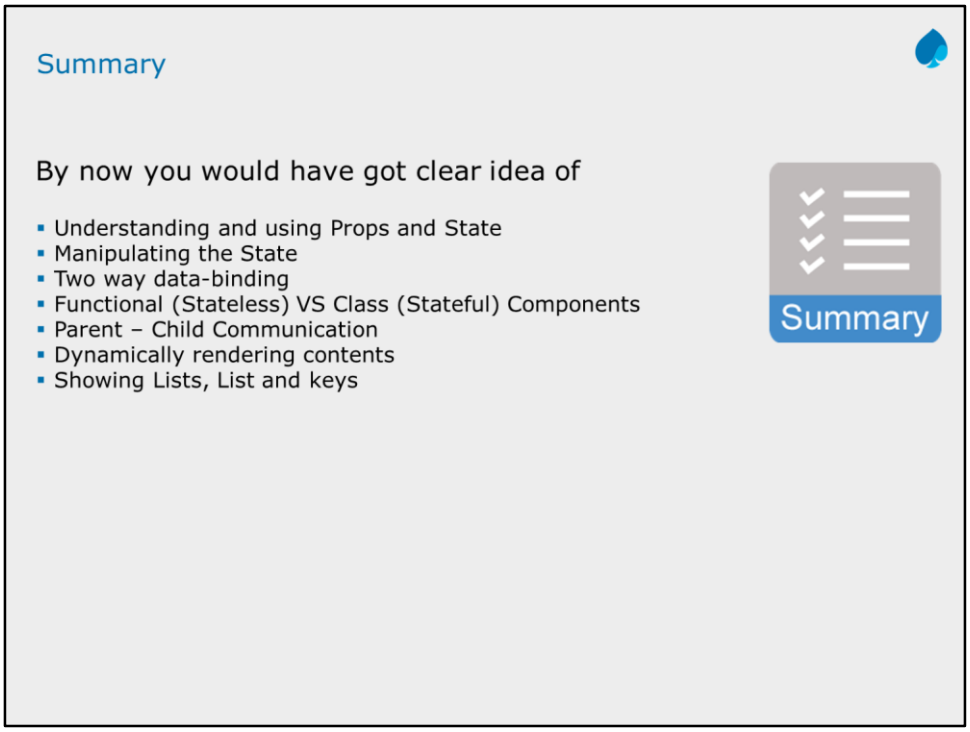
Demo of

react-list_keys_2019
react-list-keys-2-2019

Add the notes here.

Instructor Notes:

Add instructor notes here.



Add the notes here.

Instructor Notes:

Review Questions

- Question 1:
- _____ help React identify which items have changed, are added, or are removed
  - A) List
  - B) index
  - C) keys
  - D) list items
- Question 2:
- Can we use setState method in functional Compoennts?
- A) True
- B) False