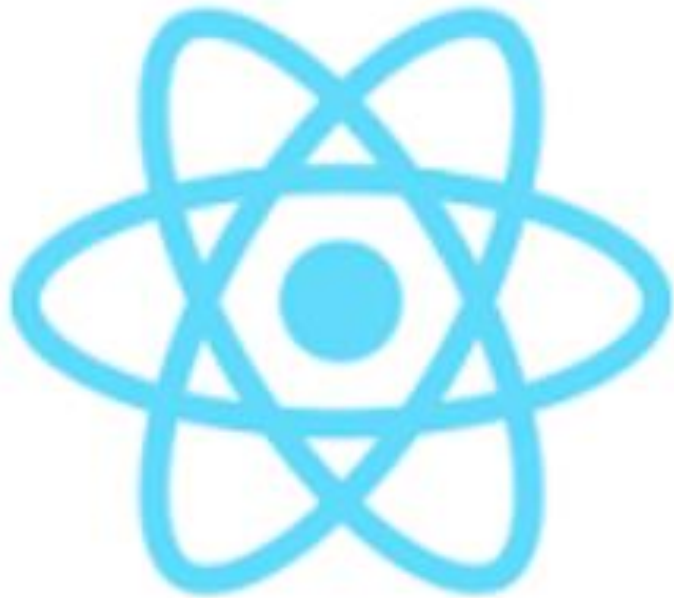


**WD-535**

# **Facebook React with Hooks**

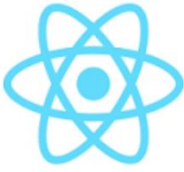


A library for building  
user interfaces



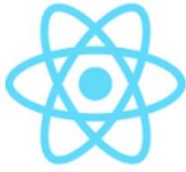
# Objectives

- Overview of current React with Hooks
  - administrate a site with React
  - understand the building blocks
- Overview of Material-UI
  - depend on the best available library to build sites fast



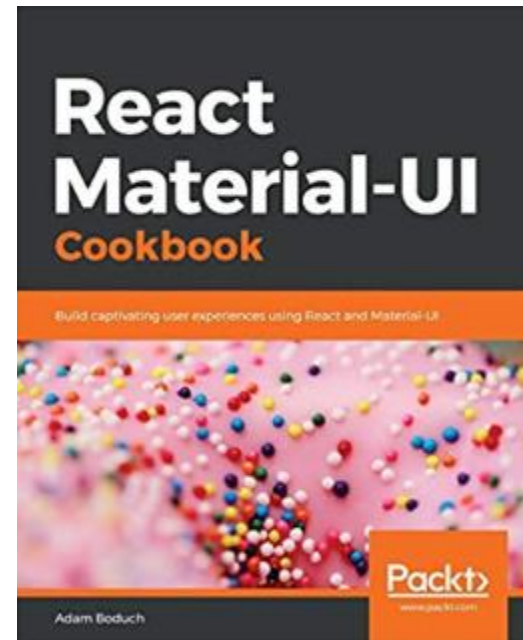
# Prerequisites

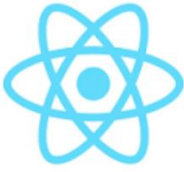
- familiarity with web site administration
- knowledge of HTML and CSS
- knowledge of JavaScript



# Books

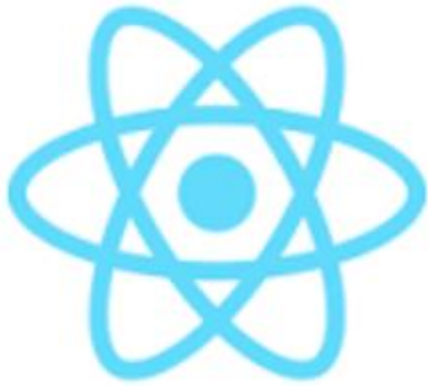
- React Material-UI Cookbook: Build captivating user experiences using React and Material-UI by Adam Boduch
- March 2019



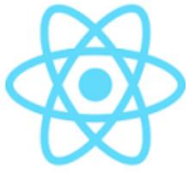


# Exercises

- Completed exercises for the current version will be kept at
- <https://github.com/doughoff/React>

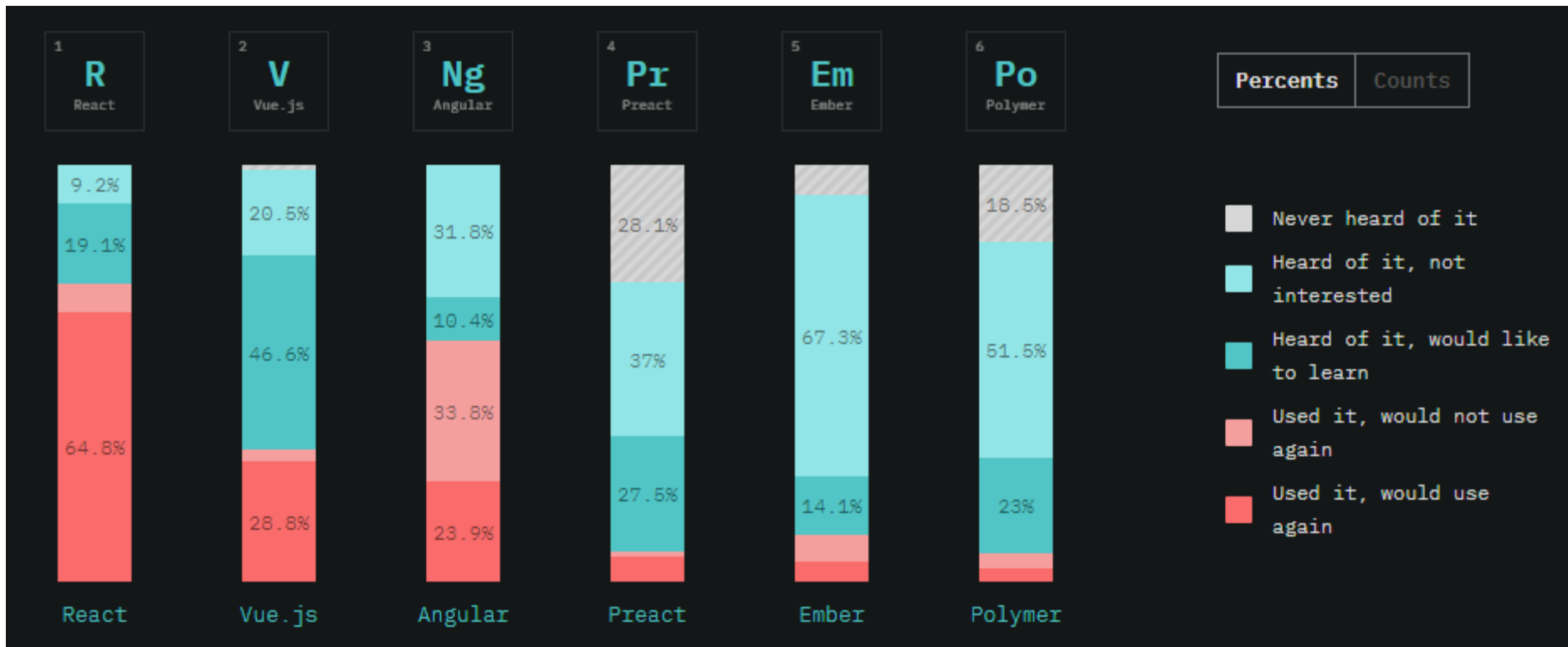


# **React basics**



# Front-end frameworks

- React is the highest rated JS web framework





# How React renders the page

- A (virtual) copy of the DOM is stored in memory
- Updates are very fast (no rendering)
- Rendering workflow
  - **diffing** - comparing snapshot of browser DOM with new virtual DOM to figure out what's changed.
  - **patching** - executing only the necessary DOM operations to make changes
  - **rendering** – changed DOM is redrawn





# Package structure

- DOM API
  - performs rendering on a web page
- Component API
  - JSX: The language used to describe UI structures
  - Data
  - Lifecycle
  - Events



# History

- 16
  - 16.9.0 alpha (April 3, 2019)
  - 16.8.1 (February 6, 2019)
    - added Hooks
  - 16.0.0 (September 26, 2017)
- 15.0.0 (April 7, 2016)
- 0.14.8 (March 29, 2016)
- 0.3.0 (May 29, 2013) – initial release

```
const element = <h2>Running  
{React.version}</h2>;
```





# React v.16 updates

- New architecture using asynchronously rendered chunks of the page (fibers)
  - <https://github.com/acdlite/react-fiber-architecture>
  - <https://reactjs.org/blog/2017/09/26/react-v16.0.html>
- Lifecycle changes
  - <https://reactjs.org/blog/2018/03/27/update-on-async-rendering.html>
- Context API
  - allows access to data from any tree level
  - <https://reactjs.org/docs/context.html>



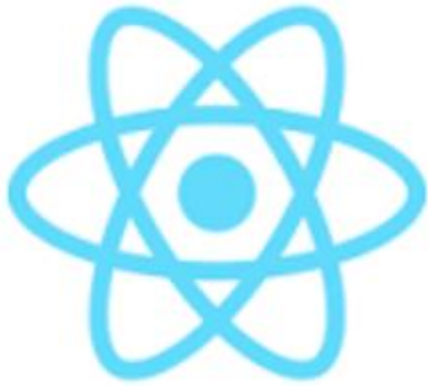
# React v.16 updates

- Fragments
  - non-rendered element to group child elements
  - <https://reactjs.org/docs/fragments.html>
- Portals
  - direct rendering instead of waterfall rendering up to root
  - <https://reactjs.org/docs/portals.html>
- Return types now include strings, not just components and HTML
  - <https://reactjs.org/blog/2017/09/26/react-v16.0.html>



# React v.16 updates

- `componentDidCatch( )`
  - serves as an error boundary by wrapping other components and render other content
  - <https://reactjs.org/docs/error-boundaries.html>
- Server-side rendering
  - <https://hackernoon.com/whats-new-with-server-side-rendering-in-react-16-9b0d78585d67>
  - <https://reactjs.org/docs/react-dom-server.html>

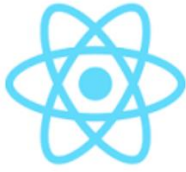


**Install**



# Visual Studio Code

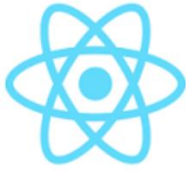
- Builds
  - Stable build
  - Insiders Edition
- Extensions
  - Prettify
  - ESLint
  - vscode-styled-jsx



# Chrome DevTools extension

- optional
- allows you to inspect components in browser
- creates new tab





# External library loading - unpkg

- Quick npm proxy for non-production
  - No JSX allowed without Babel

```
<script crossorigin  
src="https://unpkg.com/react@16/umd/react.produ  
ction.min.js"></script>
```

```
<script crossorigin  
src="https://unpkg.com/react-dom@16/umd/react-  
dom.production.min.js"></script>
```

```
<script crossorigin  
src="https://unpkg.com/babel-  
standalone@6/babel.min.js"></script>
```



# External library loading - cdnjs

- **Unpkg** is hosted by cloudflare also
  - Unpkg has caching and is a better choice

```
<script  
src="https://cdnjs.cloudflare.com/ajax/libs/react/16.8.2/umd/react.production.min.js"></script  
>
```

```
<script  
src="https://cdnjs.cloudflare.com/ajax/libs/react-dom/16.8.2/umd/react-dom.production.min.js">  
</script>
```

```
<script  
src="https://cdnjs.cloudflare.com/ajax/libs/babel-standalone/6.26.0/babel.min.js"></script>
```



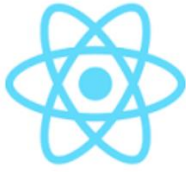
# create-react-app

- Best way to create dev environment
- Docs
  - <https://reactjs.org/blog/2018/10/01/create-react-app-v2.html> (Oct 2018)



# Workflow

- Build - Webpack
  - 'Shakes out' unused imports
  - JSX turned into html, ES6 rewritten to ES5
  - Bundles JavaScript
  - Bundles CSS
  - Rewrites html with new links
  - Outputs /build version
- Server – delivers pages and linked bundles
- Browser
  - Executes JavaScript (React workflow)



# Online editors

- Codepen
  - <https://codepen.io>
  - <https://codepen.io/doughoff>
    - use **React template** for JSX exercises
- StackBlitz – VS Code online!
  - <https://stackblitz.com/>
- JSFiddle
  - <https://jsfiddle.net/boilerplate/react-jsx>
- CodeSandbox
  - <https://codesandbox.io/>



# CodePen setup

- **React template** pen in my account
- Uses Babel pre-processor and unpkg imports
- Uses special import for Material-UI

```
const { } = MaterialUI;
```

```
const element = 'text/html/JSX to render' ;
```

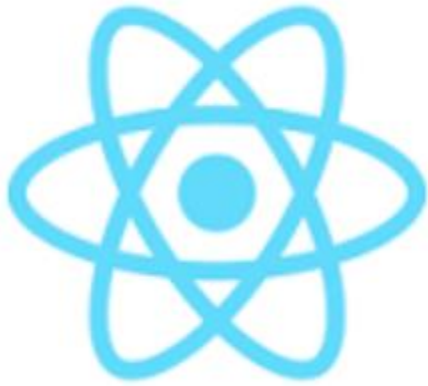
```
ReactDOM.render(element,  
document.querySelector("#root")) ;
```





# Exercises

- 1. Set up development apps
- 2. Set up non-node.js development environment
- 3. Set up node.js development environment.
- 4. Add Material-UI



Pseudo-HTML DSL

**JSX**





# JSX basics

- Not required for React
- Extended JS syntax, not HTML
  - looks like strings embedded in JS without quotes
  - can contain JS expressions
  - turned into HTML by renderer
- Used for any render target
  - React Web
  - React Native
  - React Desktop



# JSX exercises

- Use the **React template** pen
- Clear console.
  - lower left corner – CodePen is OK, use DevTools instead
- Leave
  - first line (Material-UI import)
  - last line (ReactDOM.render...)
- Replace middle with example code and wait.

```
const element = <h1>Hello, world!</h1>;
```





# Rendering

- Use a constant variable to store JSX
- Select DOM element to render to

```
const element = <h1>Hello, world!</h1>
```

```
ReactDOM.render(element,  
  document.getElementById('root')  
);
```





# Rendering

- Alternative forms of triggering a render
  - `querySelector( )` vs. `getElementById( )`
  - JSX passed as an argument

```
const element = <h1>Hello, world!</h1>
ReactDOM.render(element,
document.querySelector("#root"));
// or
ReactDOM.render(<h1>Hello, world!</h1>,
document.querySelector("#root")
);
```





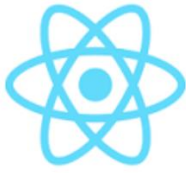
# JSX is like HTML

- Use any html element
  - one root element, must have end tag
  - tag names must be in lower case, Pascal case is for React elements
- Nesting is OK, optionally group lines with parentheses

```
const element = (  
  <div>  
    <h1>Hello!</h1>  
    <h2>We have <b>amazing</b> products!</h2>  
  </div>  
) ;
```



# JSX vs no JSX



```
const element =  
  <h1 className="greeting">  
    Hello, JSX!  
  </h1>  
;
```

```
const element = React.createElement(  
  'h1', {className: 'greeting'},  
  'Hello, JavaScript!'  
);
```





# Fragments

- Eliminates rendering a root element
- Wrap your code to render in a `<Fragment>` element – requires import
  - Removed when rendered
- Easier to use JSX shortcut: `<>` and `</>`

```
const element = <>
  <p>First nested element</p>
  <p>Second nested element</p>
  <p>Third nested element</p>
</>;
```





# Fragment import

- Seen with destructuring import statement
  - `import React, { Component, Fragment } from "react";`
  - **CodePen:** use `<React.Fragment>`

```
const element =  
<React.Fragment>  
  <span>Whoa, </span>  
  <span>yeah.</span>  
</React.Fragment>  
;
```







# Expression containers

- Curly braces evaluate expressions inside JSX

```
const name = 'Jordan Walke';  
const element = <>  
  <h2>{ "literal" }</h2>  
  <h2>{ " "}</h2>  
  <h2>{ 2 + 2 } </h2>  
  <h2>{ "author: " + name } </h2>  
  <h2>{ name.toUpperCase( ) } </h2>  
  <h2>{ new Date().toLocaleDateString() } </h2>  
</>;
```





# Expression containers with variables

- Variables can be evaluated from
  - strings, numbers, arrays, object fields

```
const company = 'Centriq';  
// const company = ["The ", "Factory"];  
const element = <h1>Welcome to {company}</h1>;  
-----  
const time =  
  <span>{new Date().toLocaleTimeString()}</span>  
const element = <h1>Time is now {time}</h1>;
```



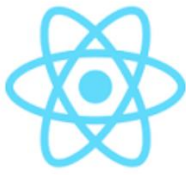


# JSX as return values

- Use any function return values of JSX

```
function formatName(user) {  
  return user.firstName + ' ' + user.lastName;  
}  
  
function getJSXGreeting (user) {  
  return <h1>Howdy, {formatName(user ||  
{lastName: "stranger"}) }!</h1>;  
}  
  
const element = getJSXGreeting(  
  //    {firstName:"Doug", lastName:"Hoff"}  
) ;
```





# JSX - attributes

- Use JS camelCase property naming
  - not HTML attribute naming
  - className instead of class
- String variables don't need to be quoted again
  - literals do

```
const element = <h2 className='red'>I'm red.</h2>
const element = <h2 tabIndex={0}>I'm first.</h2>
const react = {logo:
  'https://cdn4.iconfinder.com/data/icons/logos-
  3/600/React.js_logo-512.png'};
const element = <img src={react.logo}></img>;
```





# Basic component syntax - function

- Element name is function name
- Function returns what is rendered.

```
function Basic() {  
  return 'Basic function component';  
}
```

```
const element = <Basic/>;
```





# Basic component syntax - lambda

- Element name is lambda name
- Lambda returns what is rendered.

```
const Basic = () => {  
  // other statements  
  return 'Basic lambda component';  
}
```

```
const Basic2 = () => 'Basic lambda component';
```

```
const element = <Basic/>;
```





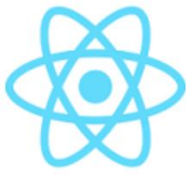
# Basic component syntax - class

- Class extends Component
- Render function returns what is rendered.

```
class Basic extends React.Component {  
  render()  
  {  
    return 'Basic class component';  
  }  
}
```

```
const element = <Basic/>;
```





# Components as JSX

- Can be standard HTML or user-defined JSX type HTML (component)

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

```
const element = <Welcome name="Kitty" />;
```







# JSX injection attacks

- Embedded HTML is escaped to prevent injection attacks

```
const response = {potentiallyMaliciousInput:  
  `javascript:deleteEverything();  
  <h3>ha ha ha ha </h3>`}
```

```
const title =  
response.potentiallyMaliciousInput;
```

```
const element = <h2>{title}</h2>;
```





# Map arrays to elements

- Functions, booleans, and objects do not render
  - not valid React children
- Arrays are joined to create a string

```
const items = ['a', true, 1, ()=> 'f', 1.23,
[1,2,3] ];
const mappedItems = items.map(item =>
  <li>{item}</li>
);
const element = <ul>{mappedItems}</ul>;
```





# Map objects to elements

- Use `Object.keys(someObject)` to get an array of keys
- Use non-dot syntax to access values

```
const object = {a:'abc', b:2 , c:1.23};
```

```
const element = <ul>
  { Object.keys(object).map(key =>
    <li id={key} >
      {key} = {object[key]}
    </li> )}
</ul>;
```





# Element lists need keys

- Lists need unique keys for performance
  - React will warn you with an error
  - key attributes do not render
- Using indexes as keys is not recommended
  - ok with no static list or filtering, reordering, or no ids

```
const object = {a:'def', b:22 , c:4.56};  
const element = <ul>  
  { Object.keys(object).map(key =>  
    <li id={key} key={key}>  
      {object[key]}  
    </li> )} </ul>;
```





# Class components

- Class components
  - use a class structure
  - returns JSX in render( ) function
  - always use proper/Pascal case

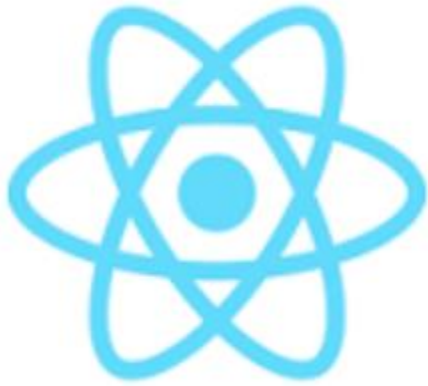
```
const name = "Doug Hoff";  
class Greeting extends React.Component {  
  render() {  
    return <h1>Hello, {name}!</h1>;  
  }  
}  
const element = <Greeting />;
```





# Exercises

- 5. Map arrays to Buttons



Properties from the JSX attributes

# Props



# **{props.<propertyName>}**

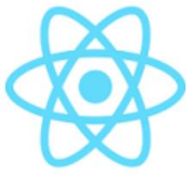
- Pass JSX attributes to component properties in the
  - for functions: first parameter
  - for classes: the class

```
const HelloWorld = (props) =>  
<h2>Hello, {props.nameFirst}!</h2>;
```

```
const element = <HelloWorld  
nameFirst='world' />;
```

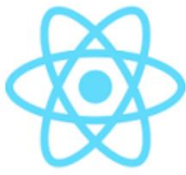






# Props data

- Short for properties object
  - object fields from attributes in an element
  - first argument passed in by React if function
- Props data should only be updated by another component.
  - changing a property value will not re-render JSX
  - changing the whole props will
- Don't rely on data from a prop
  - unless it's a seed or
  - a single source of truth



# Default properties

- Object defined as a field of the component
- Place after a lambda function declaration
  - hoisting only works on simple function declaration

```
const Greetings = (props) =>  
<h3>{props.greeting} {props.firstName}  
{props.lastName}!</h3>;
```

```
Greetings.defaultProps = {firstName:'John',  
lastName:'Smith', greeting: 'Hello,'};
```

```
const element = <Greetings firstName='Alonzo'  
lastName='Church' />
```





# Destructuring props

- Replace props parameter with object declaring props field names in any order

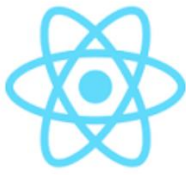
```
const Greetings = ({firstName, lastName, greeting}) =>
```

```
<h2>{greeting} {firstName} {lastName}!</h2>;
```

```
Greetings.defaultProps = {firstName: 'John',  
lastName: 'Smith', greeting: "Thanks,"};
```

```
const element = <Greetings firstName="Alonzo"  
lastName="Church"/>
```





# Destructuring for default props

- Values assigned to object will overwrite
  - default values for function components

```
const Greetings = (props) => {  
  const {greeting='Ola!', firstName='John',  
  lastName='Smith'} = props;  
  return <h2>{greeting} {firstName}  
  {lastName} !</h2>;  
};
```

```
const element = <Greetings firstName="Alonzo"  
lastName="Church"/>
```





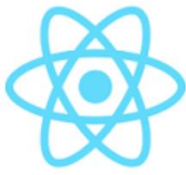
# Destructuring and renaming

- Use a colon after the expected variable name to rename it
- Default values follow the new name

```
const Greetings = (props) => {  
  const {greeting : g = 'Hey!', firstName : fn,  
        lastName :ln} = props;  
  return <h2>{g} {fn} {ln}!</h2>;  
};
```

```
const element = <Greetings firstName="Alonzo"  
lastName="Church"/>
```





# Destructuring nested objects

- Nested object names can also be destructured

```
const alonzo = {firstName: 'Alonzo',  
lastName: 'Church', number: 1234};
```

```
const Greetings = ({person, person: {firstName,  
lastName, number}}) =>
```

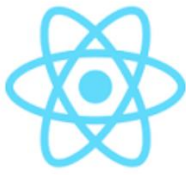
```
<h2>Welcome back {firstName} {lastName}!
```

```
<br/> Member number={number}<br/>  
{person.toString() }
```

```
</h2>;
```

```
const element = <Greetings person={alonzo}/>
```





# Destructuring with ...restProps

- the spread op groups attributes to pass through

```
const Greetings = (props) => {  
  const {greeting='Hi!', firstName='John',  
    lastName='Smith', ...restProps} = props;
```

```
  return <h2 {...restProps}>{greeting}  
    {firstName} {lastName}!</h2>;  
};
```

```
const element = <Greetings firstName="Alonzo"  
  lastName="Church" style={{color:'green',  
  fontSize:'500%'}} id='greeting' title='hey!' />
```





# props.children

- props.children passes the body of the component element back

```
const DataColumn = ({db, field, type,  
children}) =>
```

```
<h2>{field} of {type} {db}: {children} </h2>;
```

```
const author = {company: 'Acme'};
```

```
const element =
```

```
<DataColumn db='Company' field='Name'  
type='Education'>{author.company}</DataColumn>
```







# Counting props.children

- use `React.Children.count` to count children

```
const DataColumn = ({db, field, type, children}) =>
```

```
<span>{field} of {type} {db}: {children} <br/>
```

```
There's {children.length} children here?<br/>
```

```
Really only {React.Children.count(children)} is  
here </span>;
```

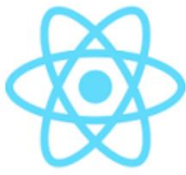
```
const author = {company: 'Acme'};
```

```
const element =
```

```
<DataColumn db='Company' field='Name'
```

```
type='Education'>{author.company}</DataColumn>
```





# Converting props.children to an array

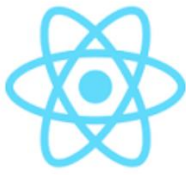
- Sorting children requires an array

```
function OrderByPropsChildren( e1, e2 ) { const a
= e1.props.children, b = e2.props.children;
return (a < b) ? -1 : (( a > b ) ? 1 : 0 );
}
```

```
const SortThese = ({children}) =>
React.Children.toArray(children).sort(OrderByProp
sChildren);
```

```
const element =
<ol><SortThese><li>nectarine</li><li>pear</li><li>
apple</li><li>cherry</li><li>banana</li></SortTh
ese></ol>;
```



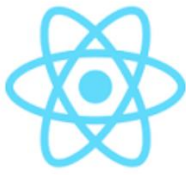


# && - guard op to inline conditionals

- Run function after guard op if first is true
- Or replace function with JSX

```
const errors = ["Bad thing happened."],
unreadMessages=["Hi, Dave.", "Bye!"]
const element =<>
  {errors.length > 0  &&  console.log('There
are errors')}
  {unreadMessages.length > 0 &&
    <h2> You have {unreadMessages.length} unread
messages. </h2>}
</>
```





# && - guard up to inline conditionals

```
const alonzo = {firstName:'Alonzo',  
lastName:'Church', number: 1234};
```

```
const Member = (props)=> <b>Member  
#{props.number}</b>;
```

```
const Greetings = ({person}) => <div>Welcome  
back {person.firstName} {person.lastName}!  
{!!person.number && <><br/><Member  
number={person.number}/></>}  
</div>;
```

```
const element = <Greetings person={alonzo}/>
```





# Prop drilling is bad

- Passing a prop to a component just so it can pass it to its child is a bad practice
  - hides the child
- A delegation for coupled components
  - Usually this is thought of as a convenient practice
  - as in a constructor in an inherited class delegating the data to its superclasses.
- Context API provides a way to pass data through the component tree without having to pass props down manually at every level.



# Prop drilling is bad

```
const Child = ({className}) =>  
<q className={className}>Child controlled by  
parent.</q>;
```

```
const Parent = ({parentClass, childClass}) =>  
<h2 className={parentClass}>  
  <Child className={childClass} />  
</h2>;
```

```
const element = <Parent parentClass={'parent-  
class'} childClass={'child-class'} />;
```





# Use composition to cure prop drilling

```
const Child = ({childClass}) =>  
  <q className={childClass}>  
    Child class is {childClass} </q>;
```

```
const Parent = ({parentClass, children}) =>  
  <h2 className={parentClass}>  
    Parent class is {parentClass}<br/>  
    {children} </h2>;
```

```
const element =  
<Parent parentClass= 'parent-class'>  
  <Child childClass='child-class' />  
</Parent>;
```





# Update by re-rendering

- Use more than one render statement to control rendering

```
function tick() {  
  const element = <h2>It is {new  
Date().toLocaleTimeString()}.</h2>;  
  ReactDOM.render(element, document.  
querySelector("#root"));  
}
```

```
setInterval(tick, 1000);  
const element = <h2>Gimme a sec here...</h2>;
```







# Class components with props

- Class components
  - use `this.props` instead of just `props` since it's an instance field now

```
class Greeting extends React.Component {  
  render() {  
    return <h1>Hello, {this.props.name}!</h1>;  
  }  
}  
  
const element = <Greeting name='world' />;
```





# Typed props

- Add propTypes during development if not using TypeScript

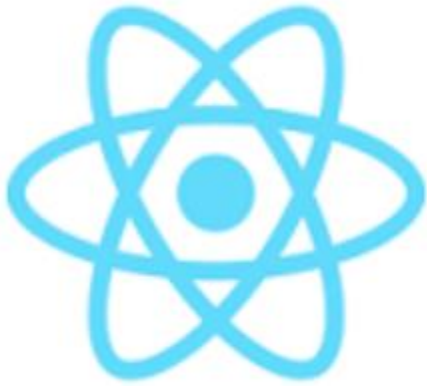
```
import PropTypes from 'prop-types';  
class Greeting extends React.Component {  
  render() { return (  
    <h1>Hello, {this.props.name}</h1>  
  ); }  
}  
Greeting.propTypes = {  
  name: PropTypes.string  
};
```





# Exercises

- 6. Nest a component
- 7. Fragments



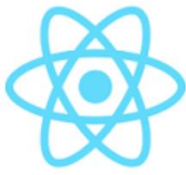
designing without changing state

# Function components



# Component architecture

- Function components are for view content only
  - not business logic
  - not to do database access
  - not a utility function
- Data input can be props or state
  - Forms should use updatable state object in class components or use updatable hooks

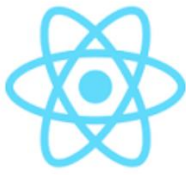


# Function components

- How to design
  - split UI into independent, reusable pieces
  - no prop drilling
  - think about each piece in isolation
- Return JSX

```
function Welcome() {  
  return (  
    <h1>Hello!</h1>  
  );  
}  
const element = <Welcome/>
```





# Functions as lambdas

- Lambdas are nice.

```
const Welcome1 = () => <h1>Hello, lambda  
1!</h1> ;
```

```
const Welcome2 = () => (<h1>Hello, lambda  
2!</h1>) ;
```

```
const Welcome3 = () => {  
  let i = 3;  
  return <h1>Hello, lambda {i}!</h1> ;  
};
```

```
const element = <> <Welcome1 /> <Welcome2 />  
<Welcome3 /> </>;
```





# Functions are better than classes

- Code is more simple, reusable, and modular.
  - Logic and presentation separation
  - Easier to test
- Prevents abuse of the `setState()` API
- Encourages "smart" vs. "dumb" component pattern.
- Allows React to make performance optimizations





# Function components

- Start JSX on same line as return keyword

```
function Squared(props) {  
  let {value: v} = props;  
  return                                // end of function!  
    <button> {v * v}  
    </button> ;  
}  
const element = <Squared value="22222"/>;
```





# Lambdas with props

- Upgrading to lambdas works well
- When using one parameter, the parens are optional... (props) or props

```
const Greetings = (props) =>  
<h2>Hey! {props.firstName} {props.lastName}!  
</h2>;
```

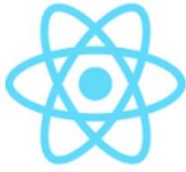
```
const element = <Greetings firstName="Alonzo"  
lastName="Church"/>
```





# Pure functions

- Components should act like pure functions and not change its own input values
  - pure if it renders the same output for the same state and props
- Convert function to a class which extends `React.PureComponent` to leverage on the performance improvements and render optimizations



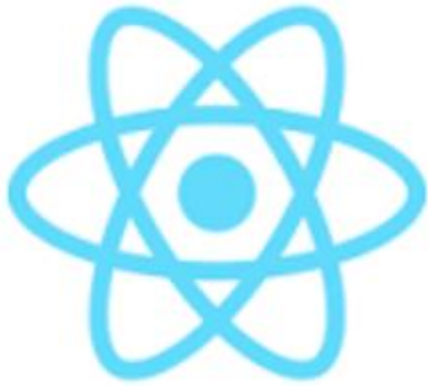
# Exercises

- 8. Function components



## Exercises

- Convert the newspaper-article page to components without any style.



CSS, style, themes

**Style**



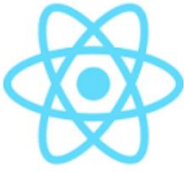
# className

- use the JS property className in JSX
  - translates to the HTML attribute of class

```
.title {color:red}
```

```
-----  
const classes = {title:'title'};  
const element = <p  
  className={classes.title}>Title class</p>;
```



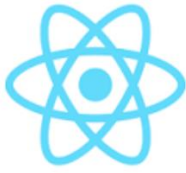


# Style attribute

```
const element= <>
<div style= {{
    margin: 50,
    padding: 10,
    width: 300,
    border: "1px solid black",
    backgroundColor: "black",
    color: "white"
}} >Lorem ipsum.</div>
</>;
```







# Styles in external file

- Create a special style object that will contain all styles.
  - good practice to place the styles in a separate file

```
const styles = { form: { margin: 50, padding: 10,
width: 300, border: "1px solid black",
backgroundColor: "black", color: "white" },
inputGroup: { marginBottom: 10 }, input: {
backgroundColor: "#EFEFFF", marginLeft: 10 }, error:
{ color: "red", margin: 5 } };
```

-----

```
import styles from '../style';
return <><div style={styles.form}>Lorem ipsum</div>
<div style={styles.inputGroup}></>;
```





# CSS-in-JS

- <https://material-ui.com/css-in-js/basics/>
- JSS - <https://cssinjs.org>
  - the core of Material-UI's styling
- Many advantages over LESS, SASS, CSS Modules, etc.
- for large scale CSS

```
npm install @material-ui/styles
```



# Styled components

- <https://material-ui.com/styles/basics/#styled-components-api>
- Not as good as CSS-in-JS overall

```
const Button = styled.a`
display: inline-block; border-radius: 6px;
padding: 0.5rem 0; margin: 0.5rem 1rem;
width: 16rem; background: transparent;
color: white; border: 2px solid white;
${props => props.primary && css`
background: white;
color: palevioletred;
`}
```



# Export withStyles

- In the last line of code in the component file, export the component with the defined styles passed in using withStyles from Material-UI.
- This creates a Higher-order component (HOC) with access to the defined style objects as props.
- The exported component can now be used for composition within other components.

```
export default  
withStyles(styles) (componentName)
```



# makeStyles( )

- v4 styles with makeStyles( ) hook

```
import { makeStyles } from '@material-ui/styles';
```

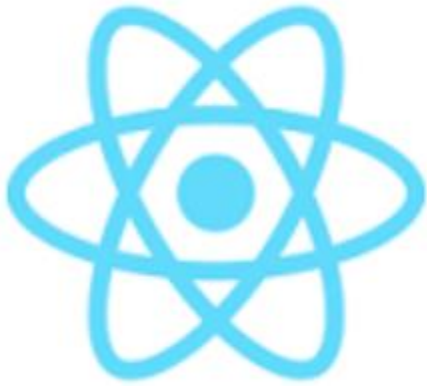
```
const useCustomStyles = makeStyles({  
  root: {  
    border: 0,  borderRadius: 3,  
    boxShadow: '0 3px 5px 2px rgba(255, 105, 135, .3)',  
    color: 'white',  height: 48,  padding: '0 30px', },  
});
```

```
export default function ButtonStyled() {  
  const classes = useCustomStyles();  
  return <button className={classes.root}>Press  
here</button>; }
```



## Exercises

- Add styles to your newspaper-article components.



When you need state

## **Class components with state**



# Properties vs. state

- Both provide data for components
- State
  - mutable during component's lifecycle
  - scoped by component
- Props
  - read-only
  - tend to be coded as function components
- Together?
  - Updated asynchronously so don't use props to update state.





# State initialization – class property

- state is a built-in property, aka field
- Use an object (not an array)

```
class Greeting extends React.Component {  
  state = { one: 'Hello', two: 'whoever' };  
  render() {  
    const { one, two } = this.state;  
    const { name } = this.props;  
    return <h1>{one}, {name || two}!</h1>;  
  }  
}  
  
const element = <Greeting name='React' />;
```





# State initialization - constructor

- Initializing state inside the constructor is an anti-pattern?
  - Unnecessary boilerplate
  - State open to mutations.
- Ok for a starting value though.

```
constructor(props) {  
  super(props) ;  
  this.state = {date: new Date()} ;  
}
```



# State initialization – no constructor

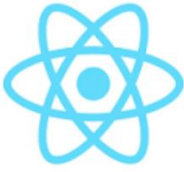
- constructor not necessary
- class property is new
- better choice in general
- update in exercises if you want

```
constructor(props) {  
  super(props) ;  
}  
state = {date: new Date()} ;
```



# Updating state with `setState( )`

- Classes use `setState( )`
  - `this.setState( {itemToUpdate: newValue} )`
- Updater form is recommended.
  - argument is a function not an object
  - <https://reactjs.org/docs/react-component.html#setstate>
- Functions must use Hooks with `useState( )`
- Re-render occurs after update.



# Updating state

- Setting new state by an object is asynchronous
- Multiple changes will not show intermediate updates and go straight to final rendered result.



# State update with setState( object )

```
class Testing extends React.Component {
  state = { counter: 1 };
  constructor(props) {
    super(props);
    setTimeout(() => this.tick(), 1000);
    setTimeout(() => this.tick(), 2000);
  }
  tick() {
    this.setState({ counter: this.state.counter + 1
  });
  }
  render() {
    return <h1>Testing, {this.state.counter}!</h1>;
  }
}
const element = <Testing />;
```





# State update by `setState( f )`

- Updater function, usually a lambda, will make the change
  - recommended by Facebook
- First parameter, `prevState`, is for current state
- Returns an object, in the same structure as the state object, to modify new values
  - not always a complete object



# State update by setState( f )

```
class Testing extends React.Component {  
  state = { counter: 1 };  
  constructor(props) { super(props);  
    setTimeout(() => this.tick(), 1000);  
    setTimeout(() => this.tick(), 2000); }  
  tick() {  
    this.setState( (prevState) => ({ counter:  
prevState.counter + 1 }) );  
  }  
  render() { return <h1>Testing,  
{this.state.counter}</h1>; } }  
const element = <Testing />;
```







# No props copied to state

- Unnecessary
  - use `this.props.color` directly
- Use if you intentionally want to ignore prop updates

```
constructor(props) {  
  super(props) ;  
  // Don't do this!  
  this.state = { color: props.color } ;  
}
```



# State update using props

```
class Testing extends React.Component {
  state = { counter: 1, text : '1' };
  constructor(props) {
    super(props);
    this.timer = setInterval(() => this.tick(), 1000); }
  tick() {
    this.setState((prevState, props) => ({
      counter: ++prevState.counter, text:
prevState.text + props.conj + prevState.counter
    }));
    this.state.counter===3 &&
clearInterval(this.timer); }
  render(){return <h1>Testing,{ ' ' }
{this.state.text}</h1>;}
}
const element = <Testing conj=" anda " />;
```





# State update on condition

```
class Eat extends React.Component {
  state = { items: [{ product: "apples", q: 1 }, { product: "bananas", q:
10 }, { product: "cherries", q: 5 }] };
  constructor(props) {
    super(props); this.timer = setInterval(() => this.tick(), 1000); }

  tick() {
    this.setState((prevState, props) => ({
      items: prevState.items.map(
        item => (item.product === 'bananas'? { ...item, q:item.q - 1} :
item)
      )
    }));
    let b = this.state.items[1].q;
    console.log("eating a banana...", b, 'left');
    (b <= this.props.until) && clearInterval(this.timer); }

  render() { return <><h1>Inventory</h1>
    <ul>{this.state.items.map(
      item => <li key={item.product}>{item.product} = {item.q}
</li>)}</ul></>; } }

const element = <Eat until="5" />;
```





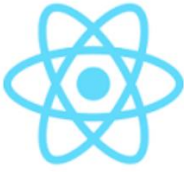
# State management packages

- state management
  - [Redux](#), Hydux, MobX, Apollo Client
  - [Easy Peasy](#) – wrapper for Redux
- Now with Hooks and using the Context API, you don't really need to use it as much
- Solves
  - prop drilling
  - predictable state updates through reducers
- Use Hooks aware packages to be easier



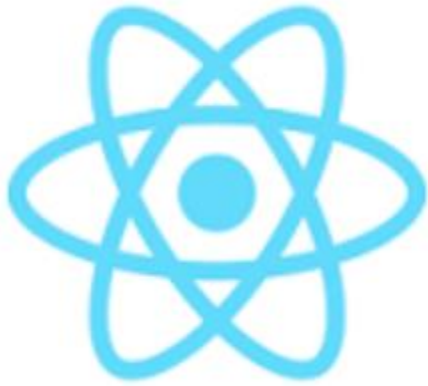
# Context API

- Provides a way to pass data through the component tree without having to pass props down manually at every level.
- Global state
- Powers Redux
  - Easier to use than Redux



# Exercises

- 9. Property data and setState()
- 10. State change



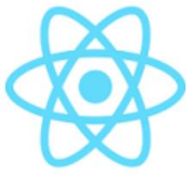
# Lifecycle



# Lifecycle

- Order of calls first time component renders (mounts)
  - constructor()
  - static getDerivedStateFromProps()
  - render()
  - componentDidMount()
- Order of calls on re-rendering
  - static getDerivedStateFromProps()
  - shouldComponentUpdate()
  - render()
  - getSnapshotBeforeUpdate()
  - componentDidUpdate()





# componentDidMount( )

- the component has been rendered once
- used for
  - data fetching
  - measuring before rendering based on size or position
- setState can be called and will re-render before browser updates screen
- removes code from constructor
- componentWillMount is deprecated as of React 16.3



# componentDidMount( )

```
class CountUp extends React.Component {
  state = { count: 0 };
  constructor(props) { super(props);
    console.log("constructor at", this.state.count);
  }
  componentDidMount() {
    this.setState({ count: +this.state.count + 1 });
    console.log("componentDidMount at", this.state.count);
    this.timer = setInterval(() => this.tick(), 1000);
  }
  tick() { this.setState({ count: +this.state.count + 1 }); }
  render() {
    const { count } = this.state;
    if (count >= 5) {
      console.log("render at", count);
      clearInterval(this.timer);
    }
    return <h1>{count} seconds and counting!</h1>;
  }
}

const element = <CountUp />;
```





# Component without output

- Sometimes you need the option not to return any output. Then return null;

```
class CountEven extends React.Component {
  state = { count: 0 };
  componentDidMount() { this.timer = setInterval(() => this.tick(),
1000); }
  tick() { this.setState({ count: +this.state.count + 1 });
    this.props.until <= this.state.count && clearInterval(this.timer); }

  render() {
    const { count } = this.state;
    if ( count % 2 == 0) { return <h1>{count} seconds and
counting!</h1>;};
    return null;
  }
}
const element = <CountEven until='10' />;
```





# Other lifecycle methods

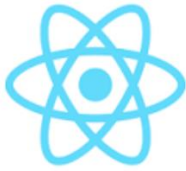
- `componentWillUnmount( )`
  - whenever component is removed
  - good for modal or event listener removal, clean-up
- `componentDidUpdate( )`
  - invoked immediately after updating after the first render
  - network requests after checking for changes to props
    - if (`this.props.userID !== prevProps.userID`)
  - dependent on Boolean returned from `shouldComponentUpdate( )`
    - for optimization, not control – will block for now



# Other lifecycle methods

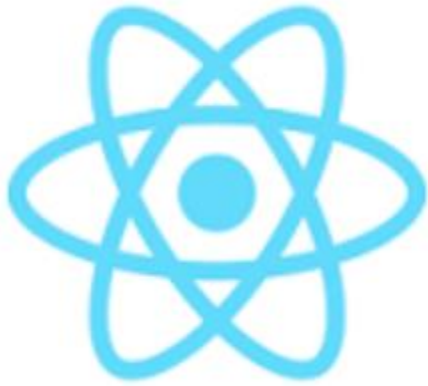
- `static getDerivedStateFromProps( )`
  - should return an object to update the state, or null to update nothing
  - used where the state depends on changes in props over time – rare
- `getSnapshotBeforeUpdate( )`
  - allows capture of information from the DOM
- `getDerivedStateFromError( )` ,  
`componentDidCatch()`
  - used to create error boundaries to recover from exceptions

# Lifecycle methods



```
class Lifecycle extends React.Component {
  constructor(props) { super(props); console.log("constructor");
    this.state = { a: 1 };
  }
  static getDerivedStateFromProps() { console.log("getDerivedStateFromProps()");
    return null;
  }
  shouldComponentUpdate() { console.log("shouldComponentUpdate()");
    return true;
  }
  render() { console.log("render()"); return <h2>Lifecycle</h2>;
  }
  componentDidMount() { console.log("componentDidMount()");
    console.log("----- end 1st render.");
    setTimeout(
      () => this.setState(prevState => ({ a: prevState.a + Math.random() })), 1000 );
  }
  getSnapshotBeforeUpdate(prevProps, prevState) {
    console.log("getSnapshotBeforeUpdate()");
    return this.state.a - prevState.a;
  }
  componentDidUpdate(prevProps, prevState, snapshot){
    console.log('componentDidUpdate() ');
    console.log('snapshot =', snapshot)
  }
}
const element = <Lifecycle />;
```





**Material-UI**

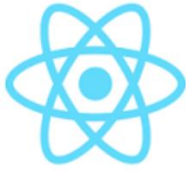


# Material-UI

- <https://material-ui.com/>
  - v4 (Hooks support)
- `npm install @material-ui/core`
  - `@next` was for pre-release versions
- Google's 2014 design system implementation in React
- uses `react-jss`, a CSS-in-JS library
  - by Oleg Isonen
- only loads styles for what it uses
  - Bootstrap loads ALL its CSS



`https://unpkg.com/@material-ui/core/umd/material-ui.production.min.js`



# Adding MUI to CodePen

- JS Gear icon (settings)
  - JavaScript / Add External Scripts/Pens
  - Add package like below
- Browse all packages at <https://unpkg.com/@material-ui/core/>





# Importing components in projects

- Without { } it has two meanings
  - import Nav.js
  - import Nav/index.js

```
import Nav from './Nav'
```





# Named imports for project

- Webpack setup allows for tree shaking
  - Tree shaking - import { a class } from @module

```
import { Paper, Typography, TextField } from  
'@material-ui/core/';
```





# CodePen MUI imports

- Import to CodePen is different than a project
- Import is different from CodePen than previous style as of Feb 2019
  - **old**: `const { Button } = window['material-ui'];`

```
const { Button } = MaterialUI;  
const { MuiThemeProvider, createMuiTheme } =  
MaterialUI;  
const { colors: {pink, indigo, red} } =  
MaterialUI;
```





# Roboto font import

- npm install typeface-roboto --save
  - import 'typeface-roboto'
- CodePen – HTML settings - add to stuff for <head>, or **CSS**

```
<link rel="stylesheet"
href="https://fonts.googleapis.com/css?family=Roboto:300,400,500">

@import
url("https://fonts.googleapis.com/css?family=Roboto:300,400,400i,700");

* {font-family: Roboto, sans-serif}
```



# Material icons – import to project

- 900+ icons, best from web font

```
<link rel="stylesheet"
href="https://fonts.googleapis.com/icon?family=
Material+Icons">
```

or

```
@import
url("https://fonts.googleapis.com/icon?family=M
aterial+Icons" );
```

-----

```
<i class="material-icons">face</i>
```



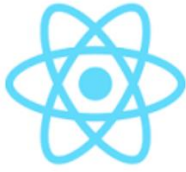
# Material icons - classes

```
.material-icons.md-18 { font-size: 18px; }  
.material-icons.md-24 { font-size: 24px; }  
.material-icons.md-36 { font-size: 36px; }  
.material-icons.md-48 { font-size: 48px; }
```

```
.material-icons.md-dark { color: rgba(0, 0, 0, 0.54); }  
.material-icons.md-dark.md-inactive { color: rgba(0, 0, 0, 0.26); }  
.material-icons.md-light { color: rgba(255, 255, 255, 1); }  
.material-icons.md-light.md-inactive { color: rgba(255, 255, 255, 0.3); }
```

```
const element = <i className="material-icons md-48 md-dark">local_cafe</i>;
```





# Material Icons

```
const element = (<>
  <i className="material-icons md-18">face</i> <br />
  <i className="material-icons md-24">face</i> <br />
  <i className="material-icons md-36">face</i> <br />
  <i className="material-icons md-48">face</i> <br />
  <i className="material-icons md-48 md-dark">face</i>
<br />
  <i className="material-icons md-48 md-dark md-
inactive">face</i><br />
  <i style={{ backgroundColor: "black" }} className =
"material-icons md-48 md-light">face</i><br />
  <i style={{ backgroundColor: "black" }} className =
"material-icons md-48 md-light md-inactive">face</i>
<br />
</> );
```







# CSS reset

- CSSBaseline will
  - remove margin
  - apply default MD background color
  - applies font anti-aliasing

```
import CssBaseline from '@material-  
ui/core/CssBaseline';  
import CssBaseline from @material-ui/core;  
-----  
const { CssBaseline } = MaterialUI; // CodePen  
const element = ( <CssBaseline /> );
```

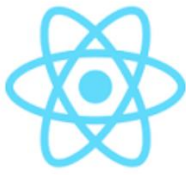




# Viewport

- CodePen – HTML settings - add to stuff for `<head>`

```
<meta  
  name="viewport"  
  content="minimum-scale=1, initial-scale=1,  
  width=device-width, shrink-to-fit=no"  
>
```



# Style attributes

- use object to map CSS rules in JSX eval block
- JavaScript properties don't need to be quoted
- Sometimes, px is not needed but is a good practice
  - margin: '0 5 0 10' doesn't work

```
style={{ 'margin': '10px' }}
```

```
style={{ margin: '10px' }}
```

```
style={{ margin: 10 }}
```

```
style={{color: 'white', backgroundColor:  
'green' }}
```



# Typography

- <https://material-ui.com/api/typography/>
- (theme) color - 'default', 'error', 'inherit', 'primary', 'secondary', 'textPrimary', 'textSecondary'

```
const {Typography} = MaterialUI; // and so on...
```

```
const element = <Typography variant='h1'  
color='textPrimary'>
```

```
  Home Page
```

```
</Typography>
```





# Typography

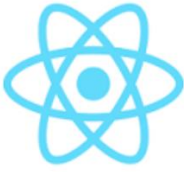
- variant – style/component for root, default <p>
  - 'h1', 'h2', 'h3', 'h4', 'h5', 'h6', 'subtitle1', 'subtitle2', **'body1'**, 'body2', 'caption', 'button', 'overline', 'srOnly', 'inherit'
- align - 'inherit', 'left', 'center', 'right', 'justify'

```
const element = <Typography variant='body2'
color='textPrimary' align='center'
gutterBottom>
```

```
  Lorem ipsum dolor sit amet
```

```
</Typography>
```





# Typography

- **booleans** – gutterBottom, noWrap, paragraph
- **component** – any other element that should be what is used for the root
  - for maintaining style to stylesheet rules
  - For switching to inline element.



# Button

- variant - 'text', 'outlined', '**contained**'
- color – 'default', 'inherit', 'primary', 'secondary'
- booleans - disabled, fullWidth
- size - 'small', 'medium', 'large'
- HTML - href, type

```
const element = <Button variant="contained"  
color="primary" fullWidth={true} >Hello  
World</Button>;
```





# Paper

- component – the element of the root node
  - a base style
- elevation – shadow depth, 0 – 24
- square – rounded corners are default (false)
  - hardly noticeable

```
const element = <Paper component='blockquote'
elevation={6} square={true}>
  <Typography variant='h2' gutterBottom >Paper
Project</Typography>
  <Button variant='outlined' style={{margin:
"-10px 0 5px 8px" }}>Enter</Button>
</Paper>;
```







# List

- Used for a container of ListItems
- component – base node
- subheader – subheader node
- booleans – dense, disablePadding (vert.)

```
const element = <List  
style={{maxWidth: '20rem',  
backgroundColor: 'hsl(15, 80%, 80%)' }}>  
</List>;
```





# ListItem

- alignItems - 'flex-start', 'center'
- component – base style (li or div based on button)
- booleans – autofocus, dense, button, disabled, disableGutters, divider, selected

```
const element = <><List style={{maxWidth: '20rem',  
backgroundColor: 'hsl(15, 80%, 80%)' }}>  
  <ListItem>Not clickable</ListItem>  
  <ListItem button>Clickable button.</ListItem>  
</List>  
<List component="nav">  
  <ListItem button>New list in a nav</ListItem>  
</List></>;
```





# ListItemText

- primary, secondary – text to show
- inset – boolean, indent, use when no icon
- other booleans – disableTypography
- primaryTypographyProps, secondaryTypographyProps – style objects

```
const element = <List style={{maxWidth:'20rem',
backgroundColor:'coral'}}>
  <ListItem button>
    <ListItemText primary = 'Primary text' secondary
= {new Date().toLocaleString()}
secondaryTypographyProps = {{color:'primary'}}
inset/>
  </ListItem> </List>;
```





# ListItemIcon

```
const element = <List style={{ maxWidth: "20rem",  
backgroundColor: "coral" }}>  
<ListItem button>  
  <ListItemIcon>  
    <i className="material-icons md-  
48">account_circle</i>  
  </ListItemIcon>  
  <ListItemText primary="Primary person" />  
</ListItem>  
<ListItem button>  
  <ListItemText secondary="Next person" />  
  <ListItemIcon>  
    <i className="material-icons md-  
48">account_box</i>  
  </ListItemIcon>  
</ListItem></List>;
```



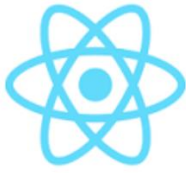


# ListItem as a link

```
const element = <>
<Typography variant="h6">Cars</Typography>
<List dense style={{ maxWidth: "20rem",
backgroundColor: "coral" }}>
  <ListItem style={{ maxHeight: "3rem" }} button
    component="a"
    href='https://www.audiusa.com/models/audi-e-tron'>
    <ListItemIcon style={{ color: 'white' }}><i
      className="material-icons md-48">directions_car</i>
    </ListItemIcon>
    <ListItemText primary="Audi e-tron" />
  </ListItem>
  <ListItem style={{ maxHeight: "3rem" }} button
    component="a" href='https://www.hyundaiusa.com/kona-
    electric/index.aspx'>
    <ListItemIcon><i className="material-icons md-
    48">directions_car</i></ListItemIcon>
    <ListItemText primary="Hyundai Kona" />
  </ListItem> </List> </>;
```



# Avatar



- alt, component, imgProps, sizes, src, srcSet

```
const element = <> <Typography variant="h6">Cars</Typography>
<List dense style={{ maxWidth: "20rem" }}>
  <ListItem button>
    <Avatar style={{marginRight: '.5rem'}}><i className="material-
icons md-12">directions_car</i></Avatar>
    <ListItemText primary="Audi e-tron" /></ListItem>
    <ListItem button>
      <Avatar style={{fontSize: '.8rem', marginRight: '.5rem',
color: 'red'}}>HK</Avatar>
      <ListItemText primary="Hyundai Kona" /></ListItem>
      <ListItem button>
        <Avatar
src='https://d3g9pb5nvr3u7.cloudfront.net/sites/539a28913f3c0
fd71ed4e43c/-1406957656/256.png'
style={{marginRight: '.5rem'}}></Avatar>
        <ListItemText primary="Tesla Model 3" />
      </ListItem></List></>;
```





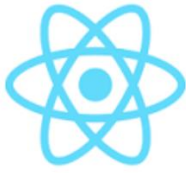
# Divider

- variant – fullWidth, inset, middle
- component – the base element
- booleans – light, absolute

```
const element = <> <Typography variant="h4">Cars</Typography>
  <Divider />
  <List style={{ maxWidth: "20rem" }}>
    <ListItem button ><ListItemText inset primary="Audi e-
tron" /></ListItem>
    <Divider variant='inset' />
    <ListItem button><ListItemText primary="Hyundai Kona"
/></ListItem>
    <Divider variant='middle' />
    <ListItem button><ListItemText primary="Tesla Model 3"
/></ListItem>
  <Divider /> </List></>;
```



# Card, CardMedia, CardContent, CardActions



```
const element= <><Card style={{ maxWidth:"20rem" }}>
  <CardMedia style={{ height: 0, padding: '40%
80%' }} image="https://airplantstore.com/wp-
content/uploads/2018/10/IMG_2968-e1539744704333.jpg"
title="shells"/>
  <CardContent>
    <Typography gutterBottom variant="headline"
component="h2">Unicorn Shells</Typography>
    <Typography component="p">Marine snails having a
prominent spine on the lip of the shell</Typography>
  </CardContent>
  <CardActions><Button size="small"
color="primary">Go to Topic</Button></CardActions>
</Card></>;
```







# CardHeader, IconButton

- nodes - action, avatar, subheader, title
- props –subheaderTypographyProps, titleTypographyProps
- booleans - disableTypography

```
<Card style={{ maxWidth: "20rem" }}>
  <CardHeader
    avatar={<Avatar>SL</Avatar>}
    action={<IconButton><i className="material-
icons md-12">beach_access</i></IconButton>}
    title="Sea Life"
    subheader="September 14, 2016"/>
  <CardMedia...
```

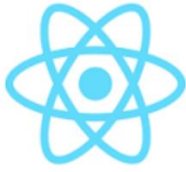




# Icon, SvgIcon, Font Awesome

```
const element= <>
<Icon color={'primary'}
style={{fontSize: '128px'}}>star</Icon>
<Icon color={'secondary'} className='material-
icons md-48'>arrow_back</Icon>
<SvgIcon>
  <path d="M20 12l-1.41-1.41L13 16.17V4h-
2v12.17l-5.58-5.59L4 12l8 8-8z" />
</SvgIcon>
<i className='far fa-hand-point-left fa-5x' />
</>;
```





# Page layout templates

- Templates to use
  - <https://material-ui.com/getting-started/page-layout-examples/>
  - <https://themes.material-ui.com/>
  - <https://themeforest.net/tags/material%20ui>



# Grid

- based on Flexbox
- two types – container, item
- set widths in %
- RWD breakpoints – xs, sm, md, lg, xl

```
<Grid container style={{border: '2px solid red', padding: 5}} >
```

```
  <Grid item style={{border: '1px solid gray'}}>small component</Grid>
```

```
  <Grid item style={{border: '1px solid gray', padding: 20}}>another small component  </Grid>
</Grid>
```

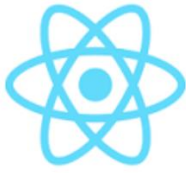


# Data for examples

```
const imageURL =  
'https://images.unsplash.com/photo-1561266436-  
05386f8c5a98?ixlib=rb-1.2.1';  
const data = [  
  {img: imageURL, title: 'Image 1',  author:  
    'author 1', cols: '1'},  
  {img: imageURL, title: 'Image 2',  author:  
    'author 2', cols: '2'},  
  {img: imageURL, title: 'Image 3',  author:  
    'author 3', cols: '2'},  
  {img: imageURL, title: 'Image 4',  author:  
    'author 4', cols: '1' }  
];
```



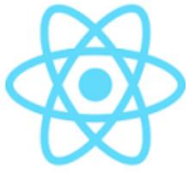
# GridList, GridListTile, GridListTileBar, ListSubheader



```
const element= <GridList cellHeight={250} cols={3}
style={{maxWidth: '900px'}}>
  <GridListTile key="Subheader" cols={3} style={{
height: 'auto' }}>
    <ListSubheader component="div"><Typography
variant='h3'>Photos</Typography></ListSubheader>
  </GridListTile>
  {data.map(tile => (
    <GridListTile key={tile.author} cols={tile.cols ||
1}>
      <img src={tile.img} alt={tile.title} />
      <GridListTileBar
        title={tile.title} subtitle={<span>by:
{tile.author}</span>}
        actionIcon={ <IconButton title={`info about
${tile.title}`}><Avatar>Img</Avatar></IconButton> } />
      </GridListTile>))) }
</GridList>;
```



# Table, TableBody, TableCell, TableHead, TableRow,



- Complex but feature-rich

- <https://material-ui.com/components/tables/>

```
const element = <Paper><Table>
  <TableHead><TableRow>
    <TableCell component=''>Title</TableCell>
    <TableCell>Author</TableCell> <TableCell
    align="right">Columns</TableCell>
  </TableRow></TableHead>
  <TableBody>{data.map(row => (
    <TableRow key={row.author}>
      <TableCell>{row.title}</TableCell>
      <TableCell>{row.author}</TableCell>
      <TableCell align="right">{row.cols}</TableCell>
    </TableRow>)) }
  </TableBody>
</Table></Paper>;
```





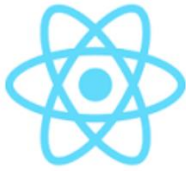
# TextField

- defaultValue, helperText, label, name, value
- variant – standard, outlines, filled
- margin – dense, none, normal
- booleans – autofocus, disabled, error, fullWidth, multiline (rows, rowsMax), required, select

```
const element= <> <TextField label="Brand"
placeholder="Brand" name="brand"/><br/>
<TextField label="Notes" placeholder="Notes"
name="model" multiline />
</>;
```







# TextField

- For html attributes not implemented use  
inputProps

```
<TextField label="Notes" placeholder="Notes"
name="model" helperText='write anything you
like' multiline
  inputProps={{maxLength: '5'}}
/>
```



# Date and time pickers

```
const today = new Date();  
const tomorrow = new  
Date(today.getTime()+1000*60*60*24);  
  
const element= <> <Paper style={{padding:10,  
margin:10}}>  
<form noValidate>  
<TextField id="date" label="Due date" type="date"  
defaultValue={tomorrow.toISOString().slice(0,10)}  
InputLabelProps={{shrink: true, }}/>  
</form>  
</Paper> </>;
```





# Material-UI themes

- This object contains
  - spacing, fontFamily, palette, zIndex
  - keys for customizing each component (AppBar, avatar...)
- Default is the lightBaseTheme
  - darkBaseTheme

```
import getMuiTheme from 'material-  
ui/styles/getMuiTheme;  
  
import darkBaseTheme from 'material-  
ui/styles/baseThemes/darkBaseTheme';  
  
const muiTheme = getMuiTheme(darkBaseTheme);
```



# Material-UI themes

```
const { Button } = MaterialUI;
const { MuiThemeProvider, createMuiTheme } = MaterialUI;
const { colors: {pink, indigo, red} } = MaterialUI;

const theme = createMuiTheme({
  palette: {
    primary: { light: "#757de8", main: "#3f51b5", dark:
"#002984", contrastText: "#fff" },
    secondary: { light: "#ff79b0", main: "#ff4081", dark:
"#c60055", contrastText: "#000" },
    openTitle: indigo[400], protectedTitle: pink[400],
    type: "light"
  } });

const element = (
  <MuiThemeProvider theme={theme}>
    <Button variant="contained" color="primary">Hello
World</Button>
  </MuiThemeProvider> );
```





# Themes

- Free and paid themes
- <https://themes.material-ui.com/>



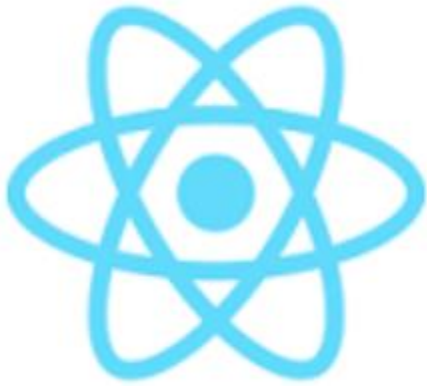
Material Dashboard  
Admin & Dashboard

FREE



## Exercises

- Add Material-UI components to your article.



Lifecycle management

**Events**



# Task splitting

- State management
  - Events update state
  - Determine changed data
- State rendering
  - State updates UI (JSX)
  - Adapt JSX to new data





# Events

- HTML used all lowercase
  - `onclick='doSomething()'`
- JSX uses camelCase
  - `onClick = {doSomething}`
- no returning false;
  - use `preventDefault( )`



# Event handling

- Write a component event handler
- Assign it to an event attribute of the element it listens to.
  - `onClick={this.handleClick}`
- Bind. Class methods are not bound by default.
  - Solutions
    - Use bind in
      - constructor
      - non-class function
    - \*Use experimental syntax with lambda



# Binding in constructor

- Allows access to state and other vars

```
constructor(props) {  
    super(props) ;  
    this.onClick = this.onClick.bind(this) ;  
}
```



# Binding by lambda

- Class props syntax in Create React App

```
const handleClick = () =>
{console.log('clicked');}
```

```
const element = <Button variant='contained'
color='secondary' onClick={handleClick}
>Button</Button>;
```





# Synthetic events

- React's way of wrapping the event object in the handler

```
const handleClick = (e) => {  
  console.info("Click");  
  console.dir(e.currentTarget);  
  document.querySelector('#b').innerHTML =  
  'clicked';  
}
```

```
const element = <Button id='b'  
variant='contained' color='secondary'  
onClick={handleClick} >Button</Button>;
```



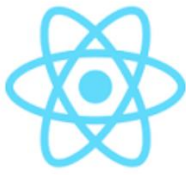


# Inline event handler

- Instead of binding to a named function, you can inline the function definition with a lambda

```
const element = <Button
  onClick={e => {
    console.info("Synthetic event:", e);
  }}
  variant="contained"
  color="primary"
>Log something</Button>;
```





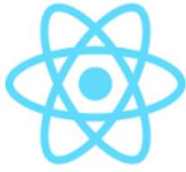
# Sending arguments to event handlers

- Change the onClick binding to a lambda function
- Add as many parameters as you need to capture the info. We create a new function here.

```
const element = <Button variant="contained"
color="primary" onClick={ (e) =>
this.handleClick(e, 'abc', 123) }>Log 3
things</Button>;
```

```
handleClick = (e, arg1, arg2) =>
{console.log(e, arg1, arg2);};
```

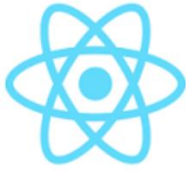




# Caching / memoization

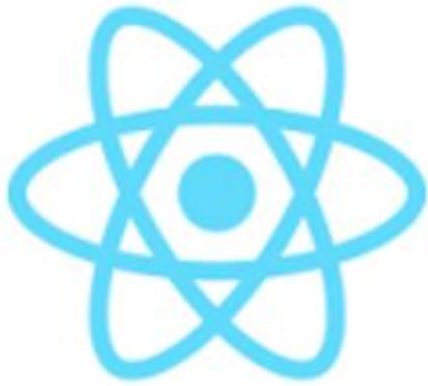
- Caching your responses so you don't calculate them again is memoization
- [https://medium.com/@Charles\\_Stover/cache-your-react-event-listeners-to-improve-performance-14f635a62e15](https://medium.com/@Charles_Stover/cache-your-react-event-listeners-to-improve-performance-14f635a62e15)





# Exercises

- 11. Event handling



## **Event-driven MUI**



# AppBar, Toolbar

- AppBar stacks content
- Toolbar inlines them

```
const element = <AppBar position="static">
  <Typography variant="h5" >
    My First Nav Bar
  </Typography>
  <Toolbar>
    <Button variant="outlined"
color="inherit">Button 1</Button>
    <Button variant="outlined"
color="inherit">Button 2</Button>
  </Toolbar>
</AppBar>;
```

My First Nav Bar

BUTTON 1

BUTTON 2





# Tabs

- Example: <https://codesandbox.io/s/qlq1j47l2w>
- But, it's better to use the router to allow paths to be put on the history.

ITEM ONE    ITEM TWO    **ITEM THREE**    ITEM FOUR    ITEM FIVE    ITEM SIX

Item Three



# Modal

- A lower level component used by the dialog, drawer, menu, and popover.
  - You probably don't need to use this.



# Snackbar

- Requires
  - state fields of **open**, **message**
  - two events of a trigger and **handleClose**

```
class Sb extends React.Component {  
  state={open: true, message:'a message for 5  
secs'};  
  handleClose = (e) => {  
    this.setState({ open: false });  
  };  
  render() {return <Snackbar style = {{width: 300,  
color: 'green'}} open={this.state.open}  
onClose={this.handleClose} autoHideDuration={5000}  
message={this.state.message} />; }  
}  
const element = <Sb/> ;
```



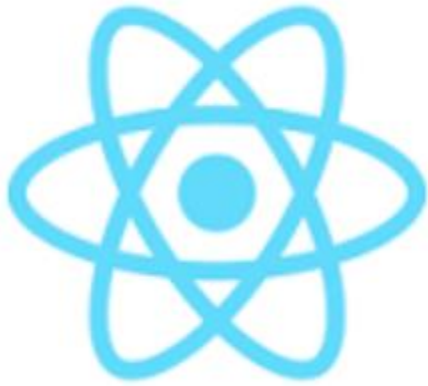
# Dialog



```
function DialogTest() {
  const [open, setOpen] = React.useState(false);
  const handleClose = () => {setOpen(false);};
  const handleClickOpen = () => {setOpen(true); };
  return (<>
    <Button variant="outlined" color="primary" onClick={handleClickOpen}>Open alert
dialog</Button>
    <Dialog open={open} onClose={handleClose}>
      <DialogTitle id="alert-dialog-title">
        {"Use Google's location service?"}
      </DialogTitle>
      <DialogContent>
        <DialogContentText id="alert-dialog-description">
Let Google help apps determine location. This means sending anonymous location data to
Google, even when no apps are running.
        </DialogContentText>
      </DialogContent>
      <DialogActions>
        <Button onClick={handleClose} color="primary"> Disagree</Button>
        <Button onClick={handleClose} color="primary" autoFocus>
Agree</Button>
      </DialogActions> </Dialog> </>);}

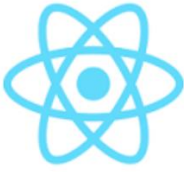
const element = <DialogTest />;
```





# **Component patterns**





# Component patterns

- Container
- Presentational
- Higher order components (HOC's)
- Render callback



# Pattern - Container

- Fetches data then renders its corresponding sub-component (e.g. a presentational one)

```
class Greeting extends React.Component {  
  constructor() {  
    super(); this.state = {name: "",};  
  }  
  componentDidMount() { // AJAX happens  
    this.setState(() => {  
      return { name: "William",}; });  
    }  
  render() { return (  
    <div> <GreetingCard name={this.state.name} />  
    </div> ); }  
}
```



# Pattern - Presentational

- utilize props, render, and context (*stateless API's*)
- receive data and callbacks from props only, which can be provided by its container or parent component

```
const GreetingCard = (props) => {  
  return (  
    <div>  
      <h1>Hello! {props.name}</h1>  
    </div>  
  ) }
```



# Pattern – HOCs

- Higher-order components
- functions that take a component and return a new component, enhancing the original in some way
- use to inject functionality from the module into your components
- obtain metrics about a wrapped component that you can then inject as props

```
const EnhancedComponent =  
hoc(OriginalComponent) ;
```



# Pattern – HOCs

- should be prefixed with **with** or **get**
- with HOCs are expected to inject functionality
- get HOCs are expected to inject data into the original component.
- apply HOCs to components at export

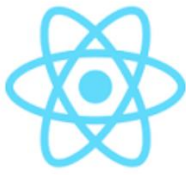
```
class OriginalComponent extends React.Component
{
  ...
}
export default
withFunctions(OriginalComponent) ;
```



# Pattern – HOCs

- Building a class around original, no mutation

```
export function withFunctions(OriginalComponent) {  
  return class extends React.Component {  
    // make some enhancements  
    ...  
    render() {  
      //return original component with more props  
      return <OriginalComponent {...this.props} />  
    }  
  }  
}
```



# HOCs for function components

- return an enhanced function
- use with `useState()` and `useEffect()` for state

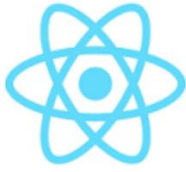
```
import React, { useState } from 'react';  
function withCountState(Wrapped) {  
  return function (...props) {  
    const [count, setCount] = useState(0);  
  
    props['count'] = count;  
    props['setCount'] = setCount;  
    return <Wrapped {...props} />;  
  }  
}
```



# Pattern - render callbacks

- aka Render props
- used to share or reuse component logic
- reducing namespace collision and better illustrate where exactly the logic is coming from than HOCs
- Heavy handed when doing a small modification to component – use "overrides" pattern





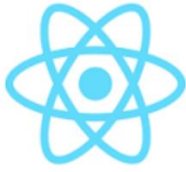
# Pattern - render callbacks

- a technique for sharing code between components using a prop whose value is a function
- <https://medium.freecodecamp.org/how-to-develop-your-react-superpowers-with-the-render-props-pattern-b74e68c6d053>

# Customized components – "overrides"



- Allow for reusability by defining a single prop for developers to change anything.
- <https://medium.com/@dschnr/better-reusable-react-components-with-the-overrides-pattern-9eca2339f646>



# Compound components

- delegates rendering control to the component consumer
- manages its own internal state
- [https://medium.com/@Dane\\_s/react-js-compound-components-a6e54b5c9992](https://medium.com/@Dane_s/react-js-compound-components-a6e54b5c9992) –



# PureComponent

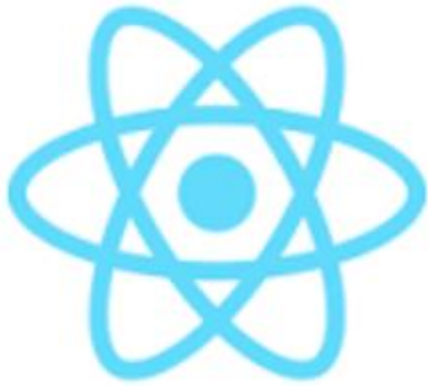
- Performance replacement for Component
- React re-renders when a component gets a new prop – even when they haven't changed
- PureComponent prevents this re-render



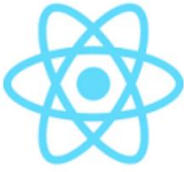
# Namespacing

- To help bring in multiple classes at one time, bundle classes under a parent class
- You also could export the Child1 and Child2 if you wanted direct access to them.

```
class Parent { }  
class Child1 { }  
class Child2 { }  
Parent.C1 = Child1;  
Parent.C2 = Child2;
```

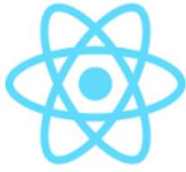


**Forms**



# State is required

- Some form elements do not update without state
  - textarea
  - checkbox
  - radio buttons



# Using Redux?

- linking form elements to your global Redux state is unnecessary
  - unless you interrupt with a different UI to save and restore – bad UI





# onChange prop

- for responsive textarea, checkbox and radio components
- JSX also uses defaultValue and defaultChecked props to handle the values



# defaultValue

- The text field attribute **value** usually holds a default value.
- JSX makes it read-only when used.
- Use `defaultValue='some default value'` instead.
- The textarea can use the attribute value.



# defaultValue

- The DOM value in the element is overridden by the jsx value attribute
- This allows you to control initial state but allow updates

```
<input type="checkbox" defaultChecked = '?'>
```

```
<input type="radio" defaultChecked = '?'>
```

```
<select defaultValue = '?'>
```

```
<textarea defaultValue = '?'>
```



# Controlled components

- Components that get and set their value through the state object.
  - A single source of truth
  - The authority

```
<input type="text" value={this.state.value}  
onChange={this.handleChange} />
```



# Controlled components

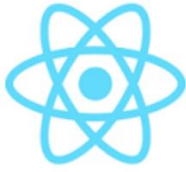
- Managing the form data through the state object
  - value property is where the element saves data
  - you usually ask the element for its value
- Now use the `this.state` object
- Uncontrolled components don't require as much event handling but controlled are recommended.



# Select values

- Use a value attribute only for the select parent
- The children options will be matched and selected.

```
<select value={this.state.value}  
  onChange={this.handleChange}>  
  <option value="grape">Grape</option>  
  <option value="lime">Lime</option>
```



# Object schema validation & parsing

- Yup – front end browser based
- Joi – server-side JS
- data security
- readability



# Form values

- Set initial value in constructor
- Update value in event handler
- Show value from state in value property

```
// constructor
this.state = {value: ''};
// event handler
this.setState({value: event.target.value});
// jsx
<input type="text" value={this.state.value}
onChange={this.handleChange} />
```





# textarea

- No difference from input/text
- HTML puts value as body

```
<textarea value={this.state.value}  
onChange={this.handleChange} />
```



# select

- HTML uses a selected attribute on the option
- Grouped options in a select parent
- React uses the value attribute again.

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

```
  <option...
```

```
  <option...
```

```
  <option...
```

```
  <option...
```



# Name the fields

- Using names for fields allows for detection in the event handler
- Checkboxes require a different value

```
// jsx
```

```
<input name = 'thisData'
```

```
// handler
```

```
const name = event.target.name;
```

```
const value = target.type === 'checkbox' ?
```

```
target.checked : target.value;
```

```
this.setState({ [name]: value });
```



# Validation

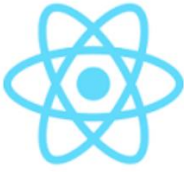
- <https://itnext.io/form-validation-with-react-hooks-ab0dbba23b9f>
- Yup - <https://medium.com/@rossbulat/introduction-to-yup-object-validation-in-react-9863af93dc0e>



# Redux-Form?

- Form state is inherently ephemeral and local, so tracking it in Redux is unnecessary
- Redux-Form calls your entire top-level Redux reducer multiple times **ON EVERY SINGLE KEYSTROKE**.
  - This is fine for small apps, but as your Redux app grows, input latency will continue to increase if you use Redux-Form.
- Redux-Form is 22.5 kB minified gzipped
  - Formik is 12.7kB

# Formik

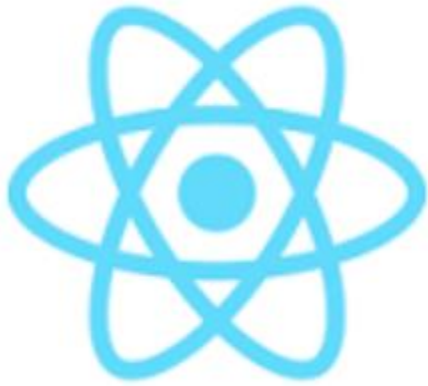


- <https://jaredpalmer.com/formik/>



# Exercises

- 12. Forms
- 13. Kitchen sink form



# Hooks and lifecycles







# Hook basics

- Allows the use of state in a functional component
  - much cleaner code
- No rewrites to include or exclude state, just adding / deleting lines.
- Customizable and reusable across components unlike class component state
- No breaking changes



# useState( )

- replaces state
  - not merging like in setState() in a class

```
import { useState } from React;
const MyVar = ( ) => {
  const [var, setVar] = useState('initial value')
  return (
    <h2>Var: { var }</h2>
    <button onClick={() => setVar(var + 1)} > +1
  </button>
  );
};
```



# useEffect( )

- replaces many life cycle methods
  - componentDidMount, componentDidUpdate
  - effect run after every render
  - effect run after changes flushed to DOM including first render
  - effect run when watched variables change
- start with set up statements
- return a lambda function for clean up
- No need to memoize lifecycle methods with useEffect



# useEffect( )

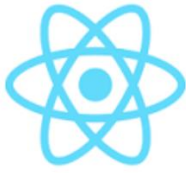
- Closure at the time of running effect
- Each render
  - has its own state
  - has its own props
  - has its own event handlers
- Lifecycle calls use a mutated state to show most recent value – not expected



# useEffect( )

```
const MyVar = () => {  
  const [ var, setVar ] = useState(0);  
  useEffect(() => {  
    getVar().then((count) => {  
      setVar(count);  
    })  
  }, []);  
  ...  
};
```

# useContext( )

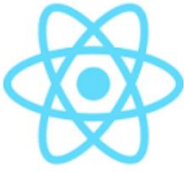


- <https://medium.com/digio-australia/using-the-react-usecontext-hook-9f55461c4eae>



# useReducer( )

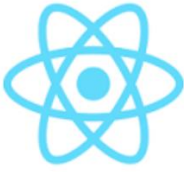
- <https://medium.com/free-code-camp/why-you-should-choose-usestate-instead-of-usereducer-ffc80057f815>
- <https://css-tricks.com/getting-to-know-the-usereducer-react-hook/>
- <https://medium.freecodecamp.org/hooks-how-to-use-reacts-usereducer-2fe8f486b963>



# Other hooks

- useCallback
- useMemo
- useRef
- useImperativeHandle
- useEffect
- useDebugValue





# Custom hooks

- when you need to share state without a parent component
- start function name with use...



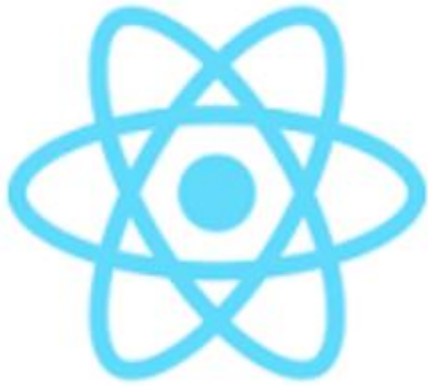
# ESLint plug-in

- `npm install eslint-plugin-react-hooks`
- Enforces rules:
  - Calling Hooks from React function components.
  - Calling Hooks from custom Hooks



# Exercises

- 14. Hooks – useState( )
- 15. Hooks – useEffect( )



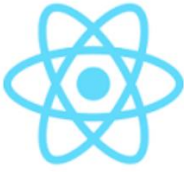
**Design**



# Single-Responsibility Principle

- A module should do one thing, and it should do it well.
- Even the smallest components should be placed in a separate file.

# Atomic design



- Brad Frost
  - <http://bradfrost.com/blog/post/atomic-web-design/>
  - 2013
- Functional decomposition



# Singel

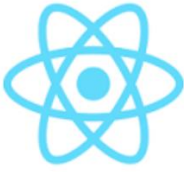
- Rules
  - Render only one element
  - Never break the app
  - Render all HTML attributes passed as props
  - Always merge the styles passed as props
  - Add all the event handlers passed as props
- Suggestions
  - Avoid adding custom props
  - Receive the underlying HTML element as a prop



# Dependency inversion

- Three main use cases
  - Putting Business Logic in a Component
    - Bad idea in React – presentation only
    - Separation of concerns
  - Putting API-Calling logic in a Component
  - Putting other Components in a Component





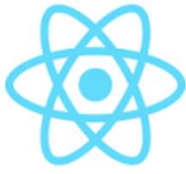
# Pure functions

- Given the same input, always return the same output, and
- Produce no side-effects



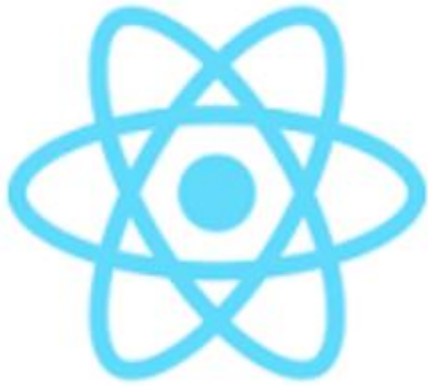
# After basic concepts

- <https://medium.freecodecamp.org/these-are-the-concepts-you-should-know-in-react-js-after-you-learn-the-basics-ee1d2f4b8030>

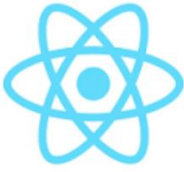


# React Suspense

- React Suspense for Data Fetching - `React.lazy`
- React Suspense for Code Splitting - `React.Suspense`



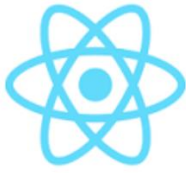
**Related packages and resources**



# React Styleguidist

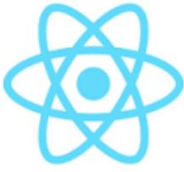
- UI dev environments ● Trial **New** Nov 2018
  - <https://react-styleguidist.js.org/>

# Apollo



- [Apollo](#) ● Trial May 2018
- Since it was first introduced in the Radar, we've seen a steady adoption of [GraphQL](#), particularly as a remote interface for a [Backend for Frontend \(BFF\)](#). As they gain more experience, our teams have reached consensus on Apollo, a GraphQL client, as the preferred way to access GraphQL data from a [React](#) application. Although the [Apollo](#) project also provides a server framework and a GraphQL gateway, the Apollo client simplifies the problem of binding UI components to data served by any GraphQL backend. Notably, Apollo is used by Amazon AWS in their recent launch of the new [AWS AppSync service](#).

# Axios



- a lightweight HTTP client



# React Redux

- [Redux](#) ● Adopt Mar 2017
- <https://react-redux.js.org/>
- 7.x uses Hooks
- With the increasing complexity of single-page JavaScript applications, we have seen a more pressing need to make client-side state management predictable. [Redux](#), with its [three principles](#) of restrictions for updating state, has proven to be invaluable in a number of projects we have implemented. [Getting Started with Redux](#) and [idiomatic Redux](#) tutorials are a good starting point for new and experienced users. Its minimal library design has spawned a rich set of tools, and we encourage you to check out the [redux-ecosystem-links](#) project for examples, middleware and utility libraries. We also particularly like the testability story: Dispatching actions, state transitions and rendering can be unit-tested separately from one another and with minimal amounts of mocking.





# Jest

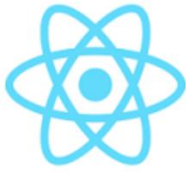
- [Jest](#) ● Trial Nov 2017
- Our teams are delighted with the results of using [Jest](#) for front-end testing. It provides a 'zero-configuration' experience and has out-of-the-box features such as mocking and code coverage. You can apply this testing framework not only to [React](#) applications, but also to other JavaScript frameworks. One of Jest's often hyped features is UI snapshot testing. Snapshot testing would be a good addition to the upper layer of the [test pyramid](#), but remember, unit testing is still the solid foundation.



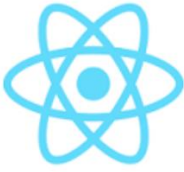
# Enzyme

- [Enzyme](#) ● Adopt May 2018
- [Enzyme](#) has become the defacto standard for unit testing [React](#) UI components. Unlike many other snapshot-based testing utilities, Enzyme enables you to test without doing on-device rendering, which results in faster and more granular testing. This is a contributing factor in our ability to massively reduce the amount of functional testing we find we have to do in React applications. In many of our projects it's used within a unit testing framework such as [Jest](#).

# Bit



- <https://bitsrc.io/>
- Organize, share, discover components from any project in your codebase
- Install when you need them.



# Component libraries

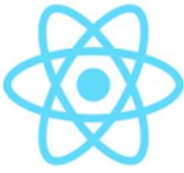
- <https://blog.bitsrc.io/11-react-component-libraries-you-should-know-178eb1dd6aa4>
- Material design - <https://material-ui.com/>

# Joi

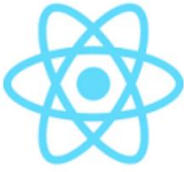


- Object schema description language and validator for JavaScript objects
- <https://github.com/hapijs/joi>

# Animation

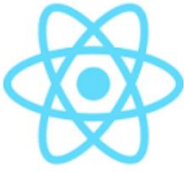


- <https://github.com/chenglou/react-motion>



# Data table – ag-Grid

- <https://react-grid.ag-grid.com>
- <https://www.ag-grid.com/react-getting-started/>



# Testing

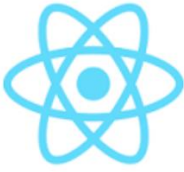
- <https://github.com/testing-library/react-testing-library>
- <https://github.com/ericelliott/riteway>





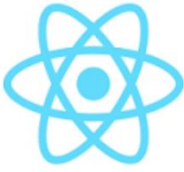
# Helpful add-ons

- Recharts
  - <https://github.com/recharts/recharts>
  - D3 facade
- React Helmet
  - <https://github.com/nfl/react-helmet>
  - manage <head> children per component per page



# Static web sites (JAMstacks)

- Gatsby
  - <https://www.gatsbyjs.org/>



# Blogs

- <https://www.robinwieruch.de>
- <https://overreacted.io/> - Dan Abramov