Front End Libraries Cheatsheet

With the Front End Libraries Cheatsheet, you'll find out the methods to rapidly style your website using a little something called Bootstrap. You'll also learn how add logic to your CSS styles and extend them with Sass.

Taken together, you’ll have the resources needed to build a shopping cart and powerful Single Page Applications (SPAs) with React and Redux.

——

——

Bootstrap

Bootstrap is a front end framework used to design responsive web pages and applications. It takes a mobile-fist approach to web development, and includes pre-built CSS styles and classes, plus some JavaScript functionality.

In this course, you'll learn how to build responsive websites with Bootstrap, and use its included classes to style buttons, images, forms, navigation, and other common elements.

Basics
The Bootstrap grid system has four classes: xs (for phones - screens less than 768px wide) sm (for tablets - screens equal to or greater than 768px wide) md (for small laptops - screens equal to or greater than 992px wide) lg (for laptops and desktops - screens equal to or greater than 1200px wide).

Links to put into your <head>

- Add Bootstrap into your HTML.

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css" integrity="sha384-BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u" crossorigin=“anonymous"/>

- Add Awesome Fonts’ library of icons

<link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.8.1/css/all.css" integrity="sha384-50oBUHEmvpQ+1lW4y57PTFmhCaXp0ML5d60M1M7uH2+nqUivzIebhndOJK28anvf" crossorigin="anonymous">

——

Make a div responsive to viewport sizing.
<div class=container-fluid>lols and lol.</div>

Make an image responsive to viewport sizing.
<img class=img-responsive>

Separate multiple classes with a space
<h2 class=‘red-text text-center’>hola</h2>

Simply create a perception of depth, or emphasis, in a div.
<div class=well></div>

Bootstrap buttons are much nicer than HTML buttons. Use them.
<button class='btn btn-default’>Like</button>

btn-primary instead of default styles the button, typically as a call to action subset.
btn-info instead of default styles the button, typically as a support subset.

Btn-block to block off an entire line with the button.
<button class='btn btn-block'>Like</button>

col-xs-4 assigns a [by-column width] to a [XS/SM/MD/LG screen] across [n/12 columns]
- *<div class=row> can be used to organise these 'by-column' elements in your code, such that even 6 elements of 6 column width can be in a single 'div-row', even if in 3 'real-life rows'.*
- *Nesting each element inside a div that instead has the column width can keep code cleaner.*
- *This approach can be easily used to create nav bars with text aligned images etc.*

Font Awesome Icons
<button><i class=fas fa-*thumbs-up*>I. Am. An icon. Bzz.</I></button
Access a full list here: https://fontawesome.com/icons?d=gallery

Forms
All textual <input>, <textarea> or <select> elements with the class form-control have 100% width.
You can still however nest an input inside a div to keep it to 100% of a div-level col-xs-n space.

—————————————————————————————————————

jQuery

Query is one of the most widely used JavaScript libraries in the world. In 2006 when it was released, all major browsers handled JavaScript slightly differently. jQuery simplified the process of writing client-side JavaScript, and also ensured that your code worked the same way in all browsers.

In this course, you'll learn how to use jQuery to select, remove, clone, and modify different elements on the page.

Basics
Typically jQuery selects an HTML element with a selector, then does something to that element.
All jQuery functions begin with $, also referred to as bling.

Full information on animate can be found here: https://animate.style/

Links to put in your <head>

- <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>

Load the jQuery library

- Run Javascript once the page has finished loading, to prevent bugs.

```
<script>
 $(document).ready(function() {
 });
</script>
```

- Provide access to an animation library

<link rel=“stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/animate.css/4.1.1/animate.min.css"/>

———————————————————————————————————————

Animating Elements
1) $(‘typeOfElement/class/id’).addClass(‘animated n’) , *OR*
2) class=‘animate__animated animate__n’

You can add multiple classes in the addClass brackets
You can add remove classes in a removeClass brackets

Editing Text Properties
$(typeOFElement/class/id).text(‘string’);

Editing HTML Properties
$(typeOFElement/class/id).css(‘<em>string</em>’);

Editing CSS Properties
$(typeOFElement/class/id).css(‘property/color’, ‘value/blue’);

Editing Other Properties
$(typeOFElement/class/id).prop(‘disabled’, true);

Removing Elements
$(typeOFElement/class/id).remove()

Moving Elements
$(typeOFElement/class/id).appendTo(targetelement); *CUT*
$(typeOFElement/class/id).clone().appendTo(targetelement); *COPY*

Selecting Related Elements
$(typeOFElement/class/id).parent().css(‘background-color', ‘red’)
$(typeOFElement/class/id).children().css(‘color', ‘orange’)

Selecting the nth Element
$(‘.target:nth-child(n)’).addClass(‘animated x’)

Selecting Odd or Even Elements
$(‘.target:odd/even’).addClass(‘animated x’)
Curiously, 0 being the first number inverses odds and evens such that 0 is odd and 1 is even etc.!

———————————————————————————————————————

SASS

Sass, or "Syntactically Awesome StyleSheets", is a language extension of CSS. It adds features that aren't available in basic CSS, which make it easier for you to simplify and maintain the style sheets for your projects.

In this Sass course, you'll learn how to store data in variables, nest CSS, create reusable styles with mixins, add logic and loops to your styles, and more.

Basics
SASS allows for dynamic editing of CSS through its use of variables to designate CSS properties.

In Sass, variables start with a $ followed by the variable name.
E.g.:

```
$main-fonts: Arial, sans-serif;
$headings-color: green;
h1 {
  font-family: $main-fonts;
  color: $headings-color;
}
```

It is possible to nest variables to organise a larger sheet.

```
nav {
  background-color: red;

  ul {
    list-style: none;

    li {
      display: inline-block;
    }
  }
}
```

Partials in Sass are separate files that hold segments of CSS code. These are imported and used in other Sass files. Names for partials start with the underscore (_) character, which tells Sass not to convert it into a CSS file. To bring the code in the partial into another Sass file, use the @import directive. An example:
Your mixins are saved in a partial named "_mixins.scss" and are needed in the "main.scss" file.

```
// In the main.scss file

@import ‘mixins'
```

Note that the underscore and file extension are not needed in the import statement - Sass understands it is a partial.

———————————————————————————————————————

Extending Properties to Like Elements
When making an addition of an element similar but not the same to others, use @extend:

```
.bigger-panel{
  @extend .panel;
  width: 150px;
  font-size: 2em;
}
```

Mixins
A Mixin is used when various rules need to apply to make a single rule functionally useful, such as when a variety of rules are needed to satisfy the requirements of various browsers. An example:

```
@mixin box-shadow($x, $y, $blur, $c){
  -webkit-box-shadow: $x $y $blur $c;
  -moz-box-shadow: $x $y $blur $c;
  -ms-box-shadow: $x $y $blur $c;
  box-shadow: $x $y $blur $c;
}
```

This ensures that box shadows operate uniformly across browsers, as they are updated.

This @mixin is then called by the @include directive:

```
div {
  @include box-shadow(0px, 0px, 4px, #fff);
}
```

If, Else If and Else Statements

The @if directive in Sass is useful to test for a specific case - it works just like the if statement in JavaScript.

```
@mixin make-bold($bool) {
  @if $bool == true {
    font-weight: bold;
  }
}
```

And just like in JavaScript, @else if and @else test for more conditions:

```
@mixin text-effect($val) {
  @if $val == danger {
    color: red;
  }
  @else if $val == alert {
    color: yellow;
  }
  @else if $val == success {
    color: green;
  }
  @else {
    color: black;
  }
}
```

For Loops

The creation of for loops allows for iteratable CSS rules, such as enlarging font or column size.

```
@for $j from 1 to 6 {
    .text-#{$j} {font-size: 15px * $j;}
    }
```

While Loops

While loops allows for conditional looping, for example quickly creating text size presets.

```
$j: 1;
@while $j < 6 {
    .text-#{$j} {font-size: 15px * $j;}
    $j: $j + 1;
}
```

Mapping Items in a List

Sass also offers the @each directive which loops over each item in a list or map. On each iteration, the variable gets assigned to the current value from the list or map.

```
@each $color in blue, red, green {
  .#{$color}-text {color: $color;}
}
```

OR

A map has slightly different syntax. Here's an example:

```
$colors: (color1: blue, color2: red, color3: green);
```

```
@each $key, $color in $colors {
  .#{$color}-text {color: $color;}
}
```
Note that the $key variable is needed to reference the keys in the map.

———————————————————————————————————————

React

React is a popular JavaScript library for building reusable, component-driven user interfaces for web pages or applications. React combines HTML with JavaScript functionality into its own markup language called JSX. React also makes it easy to manage the flow of data throughout the application.

In this course, you'll learn how to create different React components, manage data in the form of state props, use different lifecycle methods like componentDidMount, and much more.

Basics
React is an Open Source view library created and maintained by Facebook. It's a great tool to render the User Interface (UI) of modern web applications.

React uses a syntax extension of JavaScript called JSX that allows you to write HTML directly within JavaScript. You can also write JavaScript directly within JSX. To do this, you simply include the code you want to be treated as JavaScript within curly braces: { 'this is treated as JavaScript code' }.

JSX code must be compiled into JavaScript. The transpiler Babel is a popular tool for this process

In order to transpile JSX must return a single element, often meaning nesting is a necessity.

To put comments inside JSX, you use the syntax {/* */} to wrap around the comment text.

Call: ReactDOM.render(JSX, document.getElementById('root')). This function call is what places your JSX into React's own lightweight representation of the DOM. React then uses snapshots of its own DOM to optimise updating only specific parts of the actual DOM.

———————————————————————————————————————

Create a Component
Everything is a component in React. You create a component in two ways. A 1)function or 2)class:
```
const DemoComponent = function() {
  return (
    <div className='customClass' />
  );
};

class Kitten extends React.Component {
constructor(props) {
super(props);}
render() {
Return (<h1>lolatronmk7</h1>);}}
```

Render JSX from React to the DOM *(Using the React API ReactDOM)*
```
const JSX = (
  <div>
    <h1>Hello World</h1>
    <p>Lets render this to the DOM</p>
  </div>
);
// Change code below this line
```

ReactDOM.render(ComponenttoRender, document.getElementById(‘target-div’)) *OR For React Components:*
ReactDOM.render(<ReactComponent />, document.getElementById(‘target-div’))

Correct Tagging in JSX
All tags must close and any JSX tag can be made self-closing, hence:
<div></div> *OR* <div /> - *it is important to note that no content can be added to the 2nd instance.*

Rendering Multiple Components

Return (	
<App>	*Parent open*
<Nav />	*Child open closed*
<Dashboard />	*Child open closed*
<Footer />	*Child open closed*
</App>	*Parent closed*
)	

Defining Element Class in JSX
<div className=‘divClass’></div>

Summary of What We’ve Gone Over So Far:

```
class Fruits extends React.Component {
  constructor(props) {
    super(props);
  }
  render() {
    return (
      <div>
        <h2>Fruits:</h2>
 <NonCitrus />
    <Citrus />
     </div>
    );
  }
};

class TypesOfFood extends React.Component {
  constructor(props) {
     super(props);
  }
  render() {
    return (
      <div>
        <h1>Types of Food:</h1>
       <Fruits />
       <Vegetables />
      </div>
    );
  }
};
ReactDOM.render(<ReactComponent />, document.getElementById('target-div'))
```

Passing Props from Parent to Child

```
const List = (props) => {
 return <p>{props.tasks.join(', ')}</p>
};

class ToDo extends React.Component {
  constructor(props) {
    super(props);
  }
  render() {
    return (
      <div>
        <h1>To Do Lists</h1>
        <h2>Today</h2>
       <List tasks={['walk', 'sit', 'run']}/>
        <h2>Tomorrow</h2>
        <List tasks={['walk', 'sit', 'run']}/>
     </div>
    );
  }
};
```

OR, for ES6 Components
`{this.props.tempPassword}`

Setting Default Props

```
const ShoppingCart = (props) => {
  return (
    <div>
      <h1>Shopping Cart Component</h1>
    </div>
  )
};
ShoppingCart.defaultProps = {items: 0}
```

Checking PropTypes for Typechecking Purposes

```
const Items = (props) => {
  return <h1>Current Quantity of Items in Cart: {props.quantity}</h1>
};

Items.propTypes = {quantity: PropTypes.number.isRequired}

Items.defaultProps = {
  quantity: 0
};

class ShoppingCart extends React.Component {
  constructor(props) {
    super(props);
  }
  render() {
    return <Items />
  }
};
```

Creating States
State consists of any data your application needs to know about, that can change over time. If a component is stateful, it will always have access to the data in state in its render() method. You can access the data with this.state. If you want to access a state value within the return of the render method, you have to enclose the value in curly braces.

You can write states into the render pre-return without curly brackets. This also allows assigning variables to any potential computations involving them in the render, for access in the return.

Note that you must create a class component by extending React.Component in order to create state like this:

```
class StatefulComponent extends React.Component {
  constructor(props) {
    super(props);
this.state = {
  name: 'Monty'
}}
  render() {const name =  this.state.name;
return
      (<div><h1>{this.state.name *OR* name}</h1></div>);
}};
```

Changing States
Takes place within the component.
this.setState({key: value})

```
class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      name: 'Initial State'
    };
    this.handleClick = this.handleClick.bind(this);
  }
  handleClick() {
 this.setState({
    name: 'React Rocks!'
  });
 }
  render() {
    return (
      <div>
        <button onClick={this.handleClick}>Click Me</button>
        <h1>{this.state.name}</h1>
      </div>
    );
  }
};
```

On Click Functionality
```
class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      text: "Hello"
    };
 this.handleClick = this.handleClick.bind(this)
 }
  handleClick() {
    this.setState({
      text: "You clicked!"
    });
  }
  render() {
    return (
      <div>
       <button onClick = {this.handleClick}>Click Me</button>
       <h1>{this.state.text}</h1>
      </div>
    );
  }
};
```

Lifecycle Hooks
React components have several special methods that provide opportunities to perform actions at specific points in the lifecycle of a component. These are called lifecycle methods, or lifecycle hooks, and allow you to catch components at certain points in time. This can be before they are rendered, before they update, before they receive props, before they unmount, and so on. Here is a list of some of the main lifecycle methods:

- componentWillMount()
- componentDidMount()
- shouldComponentUpdate()
- componentDidUpdate()
- componentWillUnmount().

Inline Style
```
<div style={{fontSize: 72, color: ‘red’}}> lols </div>
```

Extended Notes
A stateless functional component is any function you write which accepts props and returns JSX. A stateless component, on the other hand, is a class that extends React.Component, but does not use internal state. Finally, a stateful component is a class component that does maintain its own internal state. You may see stateful components referred to simply as components or React components.

A common pattern is to try to minimize statefulness and to create stateless functional components wherever possible. This helps contain your state management to a specific area of your application. In turn, this improves development and maintenance of your app by making it easier to follow how changes to state affect its behavior.

———————————————————————————————————————