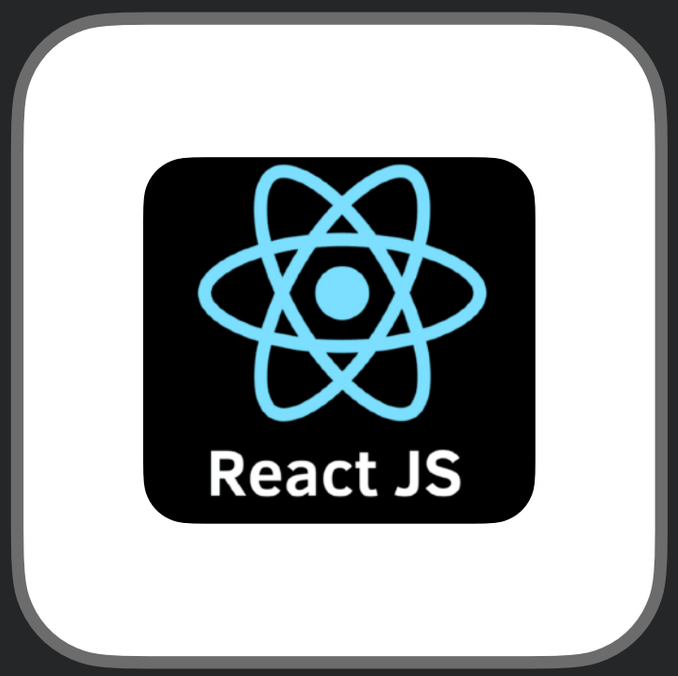
SWE 432 -Web Application Development

Spring 2023

George Mason University

Dr. Kevin Moran

Week 11: CSS, DOM, & React Part III



Administrivia

- *HW Assignment 2* - Grades are posted.
- *HW Assignment 3* - Due by midnight on ***Thursday***, April 13th.
- *Midterm Exam -* Grades out Tomorrow
- ***Mid-Semester Course Feedback Survey:*** Gathering Course Feedback.

Class Overview

- ***Part 1:*** CSS & DOM
  - Styling HTML Elements
- ***Part 2:*** More React Techniques!
  - Quick Lecture
  - Hands-On Session

CSS + DOM

CSS History

- ***1994:*** Cascading HTML style sheets—a proposal
  - Hakon W Lie proposes CSS
  - Working w/ Tim-Berners Lee at CERN
- ***1996:*** CSS1 standard, recommended by W3C
  - Defines basic styling elements like font, color, alignment, margin, padding, etc.
- ***1998:*** CSS2 standard, recommended by W3C
  - Adds positioning schemes, z-index, new font properties
- ***2011:*** CSS3 standards divided into modules, begin adoption
  - Add more powerful selectors, more powerful attributes

https://dev.opera.com/articles/css-twenty-years-hakon/

https://en.wikipedia.org/wiki/Cascading_Style_Sheets#History

CSS Tutorials and Reference

https://developer.mozilla.org/en-US/docs/Web/CSS

CSS: Cascading Style Sheets

- Language for ***styling*** documents

```
p {
    font-family: Arial;}
```

Property — Value

"Select all <p> elements"

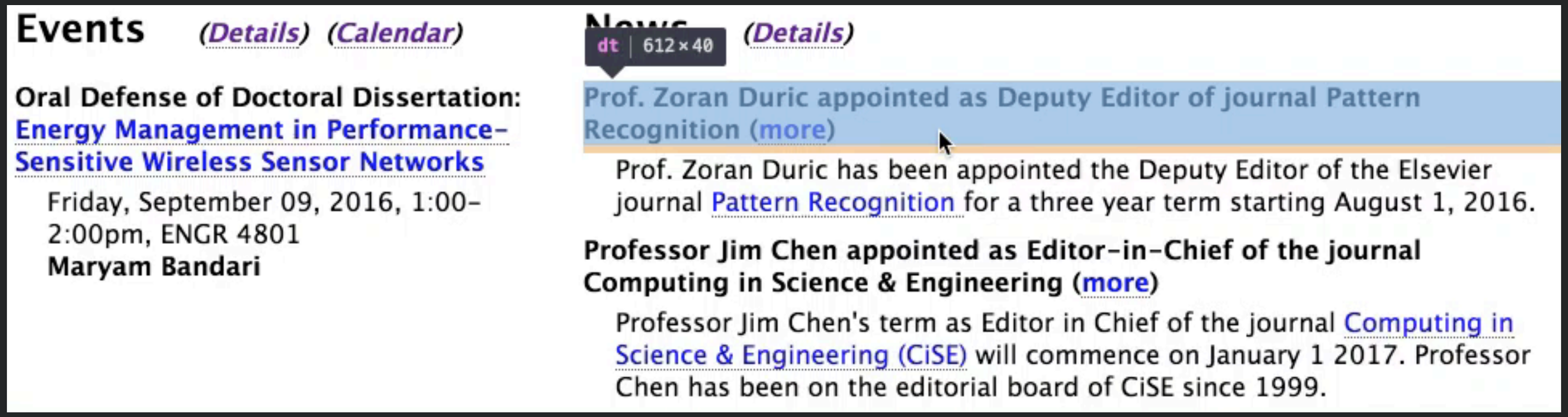
Selector describes a *set* of HTML elements

"Use Arial font family"

Declaration indicates how selected elements should be styled.

- Separates *visual presentation* (CSS) from ***document structure*** (HTML)
  - Enables changes to one or the other.
  - Enables styles to be reused across sets of elements.

CSS Styling



- Invisible box around every element.
- Rules control how sets of boxes and their contents are presented

Example Styles

BOXES	TEXT
Width, height	Typeface
Borders (color, width, style)	Size, color
Position in the browser window	Italics, bold, lowercase

Using CSS

External CSS

```
<!DOCTYPE html>
<html>
<head>
    <link rel="stylesheet" type="text/css" href="main.css">
    <title>Prof Bell's Webpage</title>
</head>
```

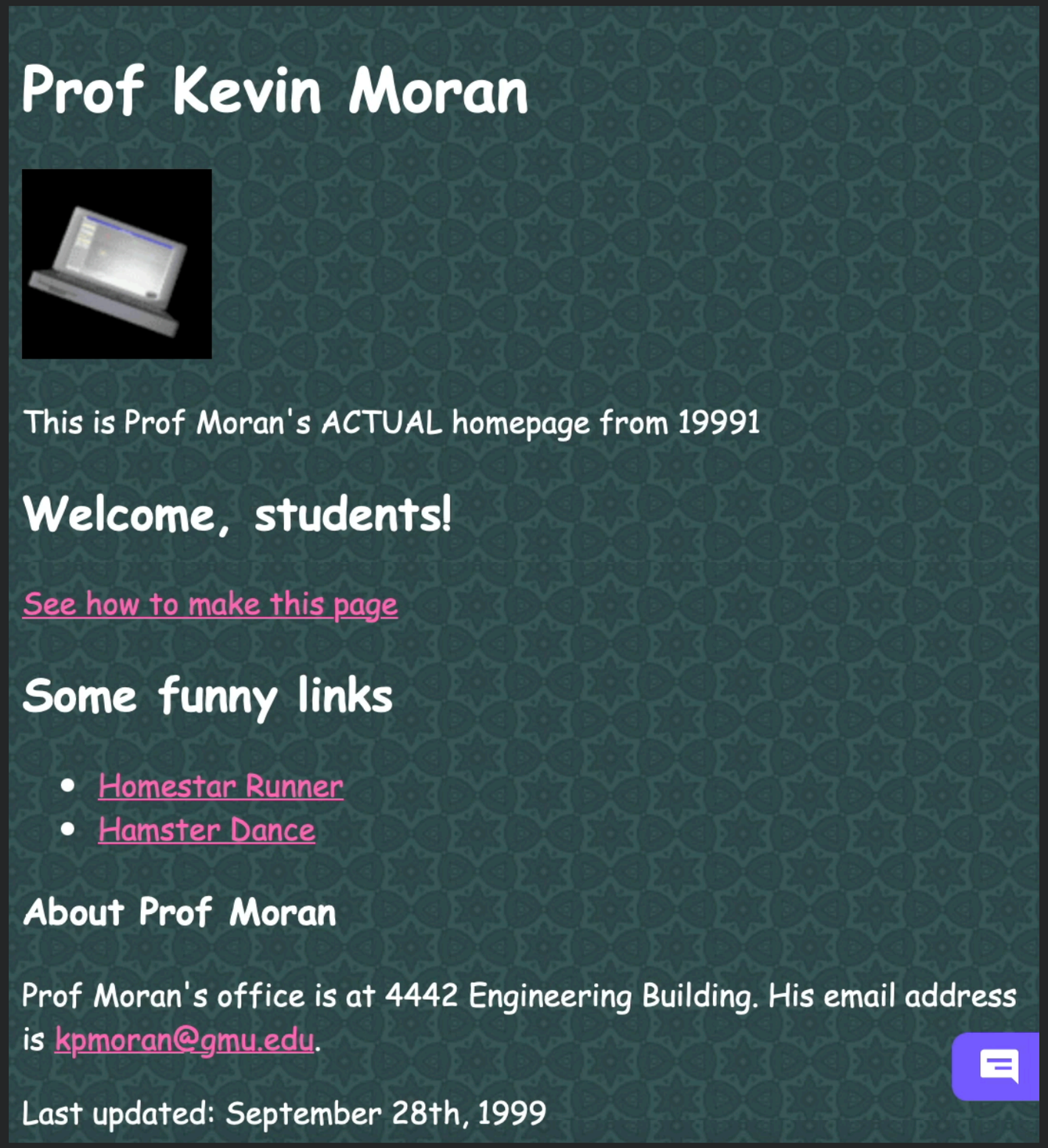
Internal CSS

```
<!DOCTYPE html>
<html>
<head>
    <title>Prof Bell's Webpage</title>
    <style type="text/css">
        body {
            background-image: url("bluerock.jpg");
            font-family: Comic Sans MS, Comic Sans;
            color: #FFFF00;
        }
</style>
```

- External CSS enables stylesheets to be reused across ***multiple*** files
- Can include CSS files
- Can nest CSS files
  - @import url(“file.css”) imports a CSS file in a CSS file

CSS Type Selectors

- What if we wanted more green?



CSS Type Selectors

- What if we wanted more green?

```
h2, h3 {
    color: LightGreen;
}
```

"Select all <h2> and <h3> elements"

Type selector selects one or more element types.

```
* {
    color: LightGreen;
}
```

"Select all elements"

Universal selector selects all elements.

CSS Class Selectors

```
<img src="profilePic.jpg" class="imageLarge" />
```

"Label <img> element with imageLarge class"

```
.imageLarge {
    width: 200px;
    height: 200px;
}
```

"Define class imageLarge."

```
<img src="profilePic.jpg" class="imageLarge transparent" />
```

```
img.large {
    width: 200px;
    height: 200px;
}
```

"Define large class that applies only to <img> elements"

```
.transparent {
    opacity: .50;
}
```

"Define transparent class"

Classes enable the creation of sets of elements that can be styled in the same way.

CSS id Selectors

```html
<div id="exampleElem">
    Some text
</div>
```

```css
#exampleElem {
    font-weight: bold;
}
```

Some text

- ***Advantages***
  - Control presentation of individual elements
- ***Disadvantages***
  - Must write separate rule for *each* element

Additional Selector Types

Selector	Meaning		Example
***Descendant* selector**	Matches all descendants of an element	p a { }	Select <a> elements inside <p> elements
***Child* selector**	Matches a direct child of an element	h1>a { }	Select <a> elements that are directly contained by <h1> elements.
***First child* selector**	Matches the first child of an element	h1:first-child { }	Select the the elements that are the first child of a <h1> element.
***Adjacent* selector**	Matches selector	h1+p { }	Selects the first <p> element after any <h1> element
***Negation* selector**	Selects all elements that are not selected.	body *:not(p)	Select all elements in the body that are not <p> elements.
***Attribute* selector**	Selects all elements that define a specific attribute.	input[invalid]	Select all <input> elements that have the invalid attribute.
Equality attribute selector	Select all elements with a specific attribute value	p[class="invisible"]	Select all <p> elements that have the invisible class.

CSS Selectors

- **Key principles in designing effective styling rules:**
  - Use classes, semantic tags to create sets of elements that share a similar rules
  - Don’t repeat yourself (DRY)
    - Rather than create many identical or similar rules, apply single rule to all similar elements
- Match based on semantic properties, not styling
  - Matching elements based on their pre-existing styling is ***fragile***

Cascading Selectors

- What happens if more than one rule applies?
- Most *specific* rule takes precedence
  - ***p b*** is more specific than ***p***
  - ***#maximizeButton*** is more specific than ***button***
- If otherwise the same, *last* rule wins
- Enables writing generic rules that apply to many elements that are overriden by specific rules applying to a few elements

CSS Inheritance

- When an element is contained inside another element, some styling properties are inherited
  - e.g., font-family, color
- Some properties are not inherited
  - e.g., background-color, border
- Can force many properties to inherit value from parent using the inherit value
  - e.g., padding: inherit;

Pseudo Classes

```css
.invisible {
    display: none;
}

input:invalid {
    border: 2px solid red;
}

input:invalid + div {
    display: block;
}

input:focus + div {
    display: none;
}
```

```html
<label>
    Email: <input type="email" />
    <div class="invisible">Please enter a valid email.</div>
</label>
```

Email:

"Select elements with the invalid attribute."

"Select elements that have focus."

Classes that are automatically attached to elements based on their attributes.

Examples of Pseudo Classes

- :active - elements activated by user. For mouse clicks, occurs between mouse down and mouse up.
- :checked - radio, checkbox, option elements that are checked by user
- :disabled - elements that can’t receive focus
- :empty - elements with no children
- :focus - element that currently has the focus
- :hover - elements that are currently hovered over by mouse
- :invalid - elements that are currently invalid
- :link - link element that has not yet been visited
- :visited - link element that has been visited

Color

- Can set text color (color) and background color (background-color)
- Several ways to describe color
  - six digit hex code (e.g., #ee3e80)
  - color names: 147 predefined names
  - rgb(red, green, blue): amount of red, green, and blue
  - hsla(hue, saturation, lightness, alpha): alternative scheme for describing colors
- Can set opacity (opacity) from 0.0 to 1.0

```
body {
    color: Red;
    background-color: rgb(200, 200, 200); }
h1 {
    background-color: DarkCyan; }
h2 {
    color: #ee3e80; }
p {
    color: hsla(0, 100%, 100%, 0.5); }

div.overlay {
    opacity: 0.5; }
```

Typefaces

Serif

Sans-Serif

Monospace

Cursive

font-family: Georgia, Times, serif;

“Use Georgia if available, otherwise Times, otherwise any serif font”.

font-family enables the typeface to be specified. The typeface must be installed. Lists of fonts enable a browser to select an alternative.

Styling text

```
h2 {
      text-transform: uppercase;
      text-decoration: underline;
      letter-spacing: 0.2em;
      text-align: center;
      line-height: 2em;
      vertical-align: middle;
      text-shadow: 1px 1px 0 #666666;

}
```

THIS TEXT IS IMPORTANT

- text-transform: uppercase, lowercase, capitalize
- text-decoration: none, underline, overline, line-through, blink
- letter-spacing: space between letters (kerning)
- text-align: left, right, center, justify
- line-height: total of font height and empty space between lines
- vertical-align: top, middle, bottom, …
- text-shadow: [x offset][y offset][blur offset][color]

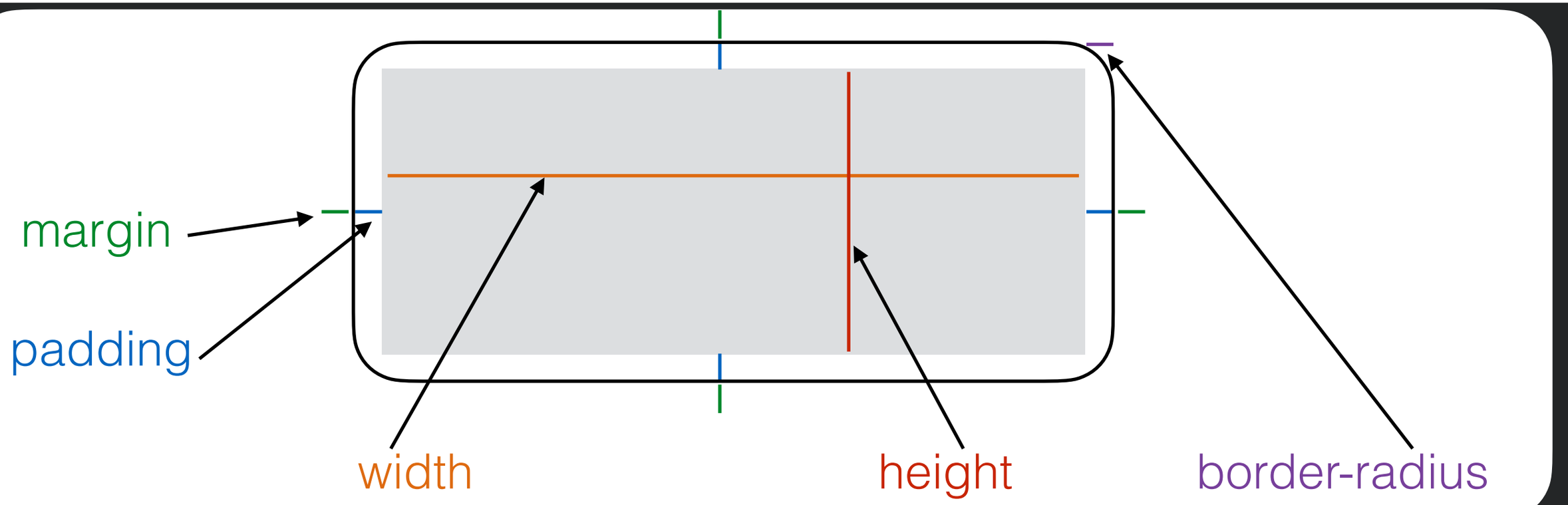
Cursor

```
<a class="movableItem">Walt Whitman</a>


a.movableItem {
    cursor: move;
}
```

- Can change the default cursor with cursor attribute
  - auto, crosshair, pointer, move, text, wait, help, url(“cursor.gif”)
- Should *only* do this if action being taken clearly matches cursor type

CSS "Box" Model



- Boxes, by default, are sized ***just*** large enough to fit their contents.
- Can specify sizes using `px` or `%`
  - % values are relative to the container dimensions
- margin: 10px 5px 10px 5px; (clockwise order - [top] [right] [bottom] [left])
- border: 3px dotted #0088dd; ([width] [style] [color])
  - style may be: solid, dotted,dashed, double, groove, ridge, inset, outset, hidden / none

Centering Content

```css
.centered {
    width: 300px;
    margin: 10px auto 10px auto;
    border: 2px solid #0088dd;
}
```

This box is centered in its container.

- How do you center an element inside a container?
- Step 1: Must first ensure that element is *narrower* than container.
  - By default, element will expand to fill entire container.
  - So must usually explicitly set width for element.
- Step 2: Use *auto* value for left and right to create equal gaps

Visibility and layout

- Can force elements to be inline or block element.
  - display: inline
  - display: block
- Can cause element to not be laid out or take up any space
  - display: none
  - *Very* useful for content that is dynamically added and removed.
- Can cause boxes to be invisible, but still take up space
  - visibility: hidden;

```
<ul>
    <li>Home</li>
    <li>Products</li>
    <li class="coming-soon">Services</li>
    <li>About</li>
    <li>Contact</li>
</ul>
```

```
li {
    display: inline;
    margin-right: 10px; }
li.coming-soon {
    display: none; }
```

Home Products About Contact

```
li {
    display: inline;
    margin-right: 10px; }
li.coming-soon {
    visibility: hidden; }
```

Home Products About Contact

Positioning Schemes

Normal flow (default)

Lorem Ipsum

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.

Block level elements appear on a new line. Even if there is space, boxes will not appear next to each other.

Relative positioning

Lorem Ipsum

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Ut enim ad minim veniam, quis nostrud exercitation u nisi ut aliquip ex ea commodo consequat.

Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.

```
p.example {
    position:relative;
    top: 10px;
    left: 100px;
}
```

Element shifted from normal flow. Position of other elements is ***not*** affected.

Absolute positioning

eiusmod tempor incididunt ut labore et dolore magna **Lorem Ipsum**

Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.

```
h3 {
    position: absolute;
    background-color: LightGray;
    left: 350px;
    width: 250px;
}
```

Element taken out of normal flow and does not affect position of other elements. Moves as user scrolls.

Fixed positioning

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusm**Lorem Ipsum**re magna aliqua.

Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.

```
h3 {
    position: fixed;
    background-color: LightGray;
    left: 40px;
    width: 250px;
}
```

Element taken out of normal flow and does not affect position of other elements. Fixed in window position as user scrolls.

Floating elements

Lorem Ipsum Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.

```
h3 {
    float: left;
    background-color: LightGray;
    left: 40px;
    width: 250px;
}
```

Element taken out of normal flow and position to far left or right of container. Element becomes block element that others flow around.

Stacking elements

```
h3 {
    position: absolute;
    background: LightGray;
    opacity: 0.6;
    z-index: 10;
}
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.

- Elements taken out of normal flow may be stacked on top of each other
- Can set order with z-index property
  - Higher numbers appear in front
- Can set opacity of element, making occluded elements partially visible

Transform - examples

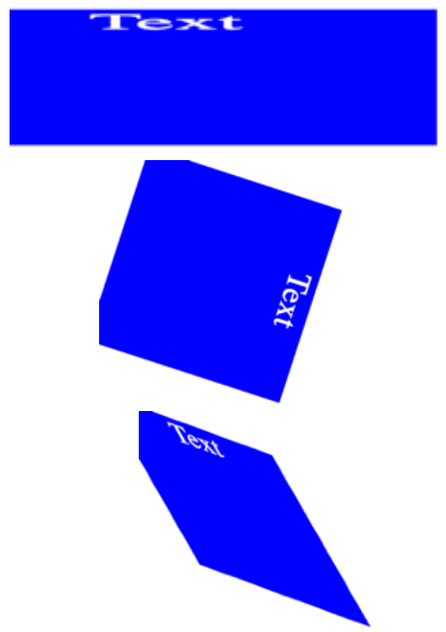
```
.box {
    width: 100px;
    height: 100px;
    color: White;
    text-align: center;
    background-color: #0000FF;
}


.transform1 {
    transform: translate(12px, 50%);
}

.transform2 {
    transform: scale(2, 0.5);
}

.transform3 {
    transform: rotate(0.3turn);
}

.transform4 {
   transform: skew(30deg, 20deg);
}
<div class="box">Text</div>
```



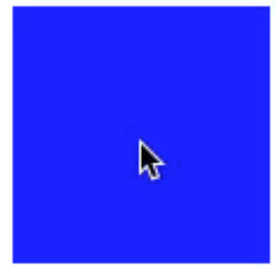
- Can modify coordinate space of element to rotate, skew, distort

Transitions

```css
.box {
    width: 100px;
    height: 100px;
    background-color: #0000FF;
    transition: width 2s, height 2s, background-color 2s, transform 2s;
}

.box:hover {
    background-color: #FFCCCC;
    width: 200px;
    height: 200px;
    transform: rotate(180deg);
}
```

```html
<div class="box"></div>
```



- transition: [property time], …, [property time]
  - When new class is applied, specifies the time it will take for each property to change
  - Can use *all* to select all changed properties

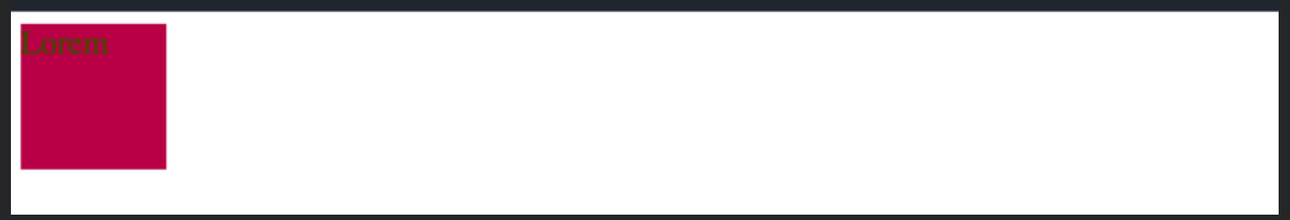
Transition: Example

https://jsfiddle.net/vs2qo9r1/

```css
.parent {
    width: 250px;
    height: 125px;
}

.box {
    width: 100px;
    height: 100px;
    background-color: red;
    font-size: 20px;
    left: 0px;
    top: 0px;
    position: absolute;
     -webkit-transition-property: width height background-color font-size left top color;
    -webkit-transition-duration: 2s;
    -webkit-transition-delay: 1s;
    -webkit-transition-timing-function: linear;
    transition-property: width height background-color font-size left top color;
    transition-duration: 2s;
    transition-delay: 1s;
    transition-timing-function: linear;
}

.box1{
    width: 50px;
    height: 50px;
    background-color: blue;
    color: yellow;
    font-size: 18px;
    left: 150px;
    top: 25px;
    position: absolute;
    -webkit-transition-property: width height background-color font-size left top color;
    -webkit-transition-duration: 2s;
    -webkit-transition-delay: 1s;
    -webkit-transition-timing-function: linear;
    transition-property: width height background-color font-size left top color;
    transition-duration: 2s;
    transition-delay: 1s;
    transition-timing-function: linear;
}
```

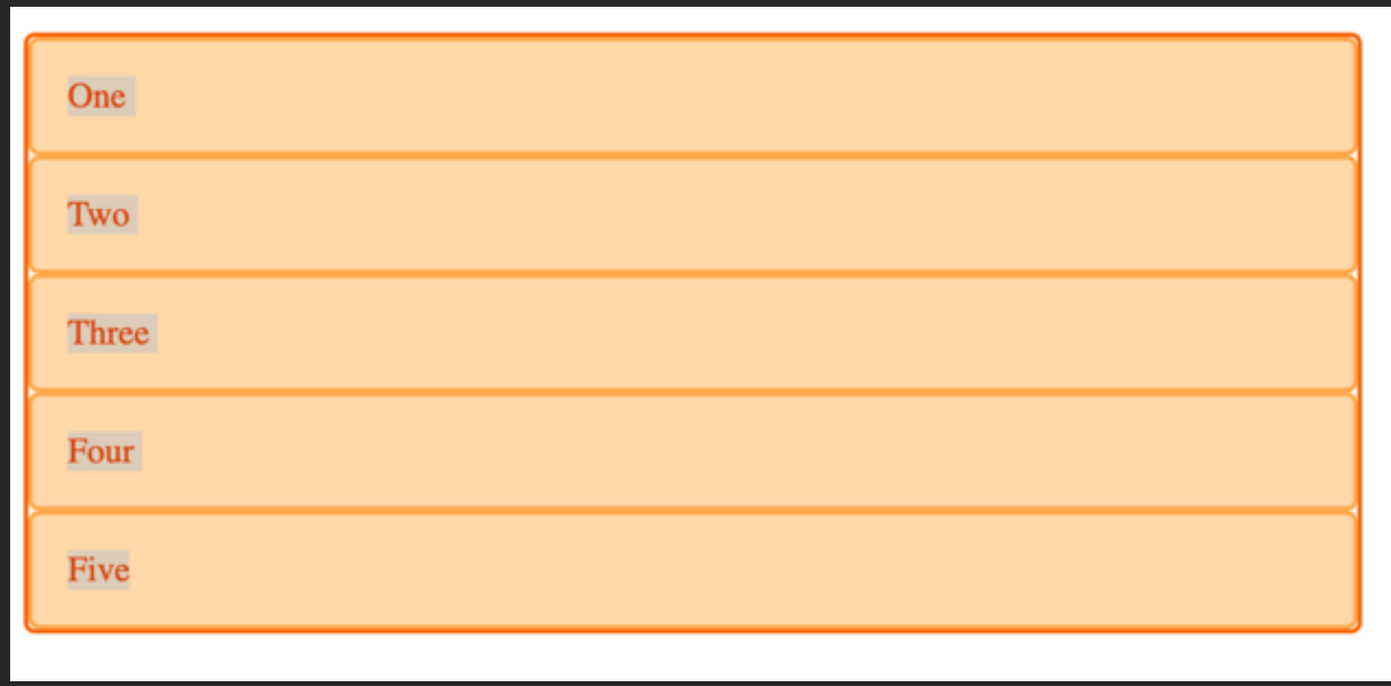


Grid layout

- Create using display: grid or display: inline-grid

```
<div class="wrapper">
  <div>One</div>
  <div>Two</div>
  <div>Three</div>
  <div>Four</div>
  <div>Five</div>
</div>


.wrapper {
  display: grid;
}
```



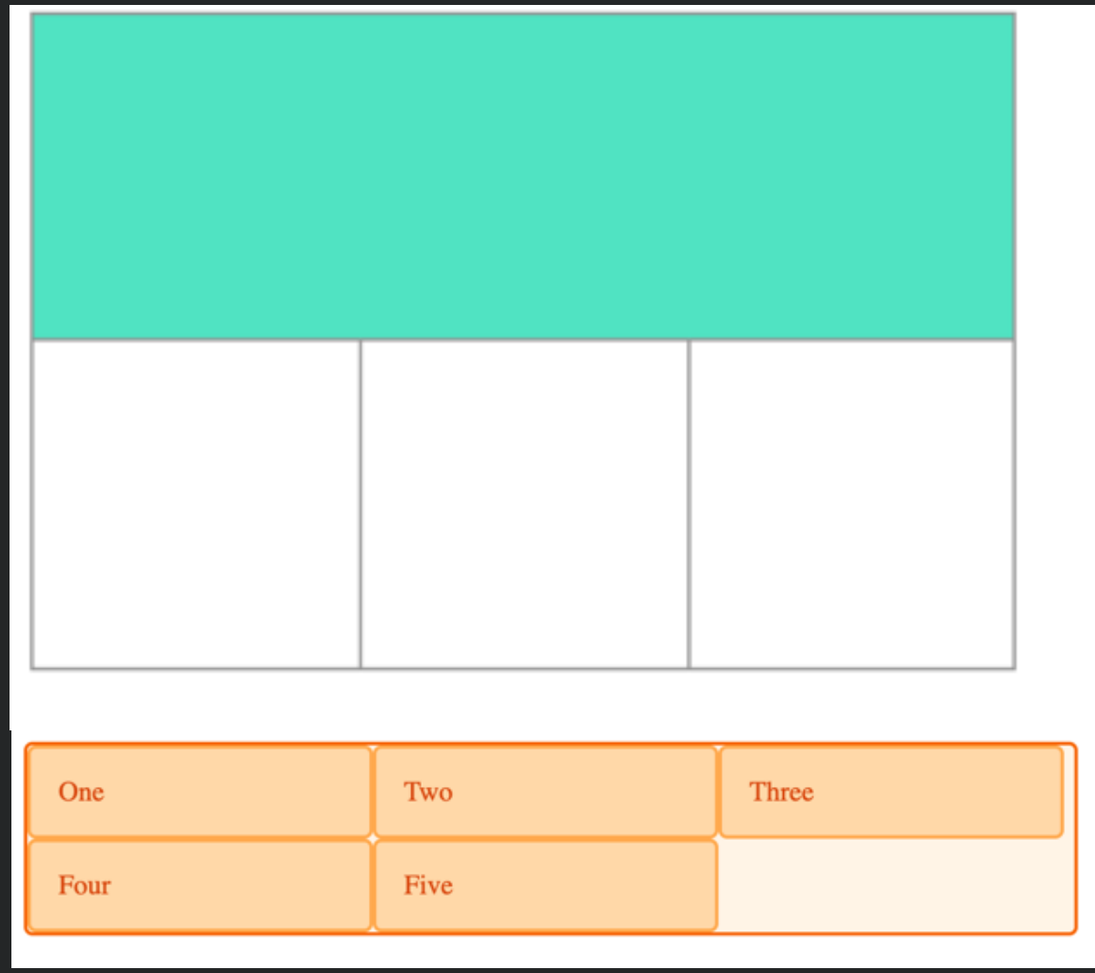
https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Grid_Layout/Basic_Concepts_of_Grid_Layout

Grid tracks

- Define rows and columns on grid with the ***grid-template-columns*** and ***grid-template-rows*** properties.
- Define grid tracks.
- A grid track is the space between any two lines on the grid.

```
<div class="wrapper">
  <div>One</div>
  <div>Two</div>
  <div>Three</div>
  <div>Four</div>
  <div>Five</div>
</div>

.wrapper {
  display: grid;
  grid-template-columns: 200px 200px 200px;
}
```



Liquid layouts

- `fr` represents a fraction of available space fo grid container.
- Can mix absolute and flexible, where flexible occupies any remaining space after flexible is subtracted

```
<div class="wrapper">
  <div>One</div>
  <div>Two</div>
  <div>Three</div>
  <div>Four</div>
  <div>Five</div>
</div>


.wrapper {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
}
```

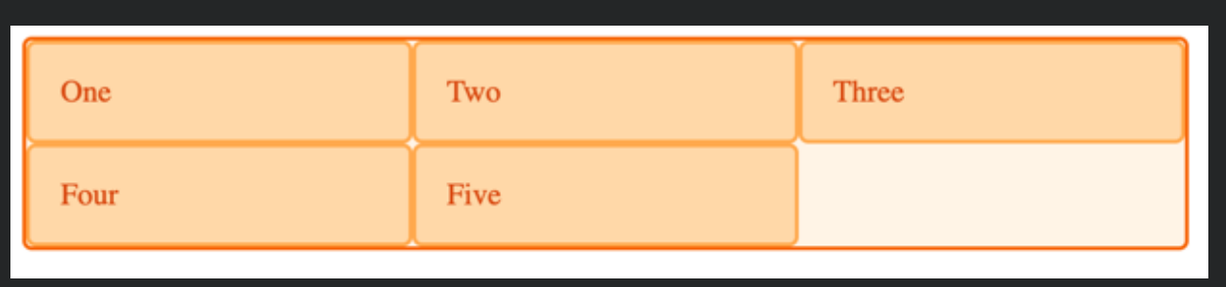

```
.wrapper {
  display: grid;
  grid-template-columns: 500px 1fr 2fr;
}
```

Liquid layouts

- `fr` represents a fraction of available space fo grid container.
- Can mix absolute and flexible, where flexible occupies any remaining space after flexible is subtracted

```
<div class="wrapper">
  <div>One</div>
  <div>Two</div>
  <div>Three</div>
  <div>Four</div>
  <div>Five</div>
</div>


.wrapper {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr;
}
```

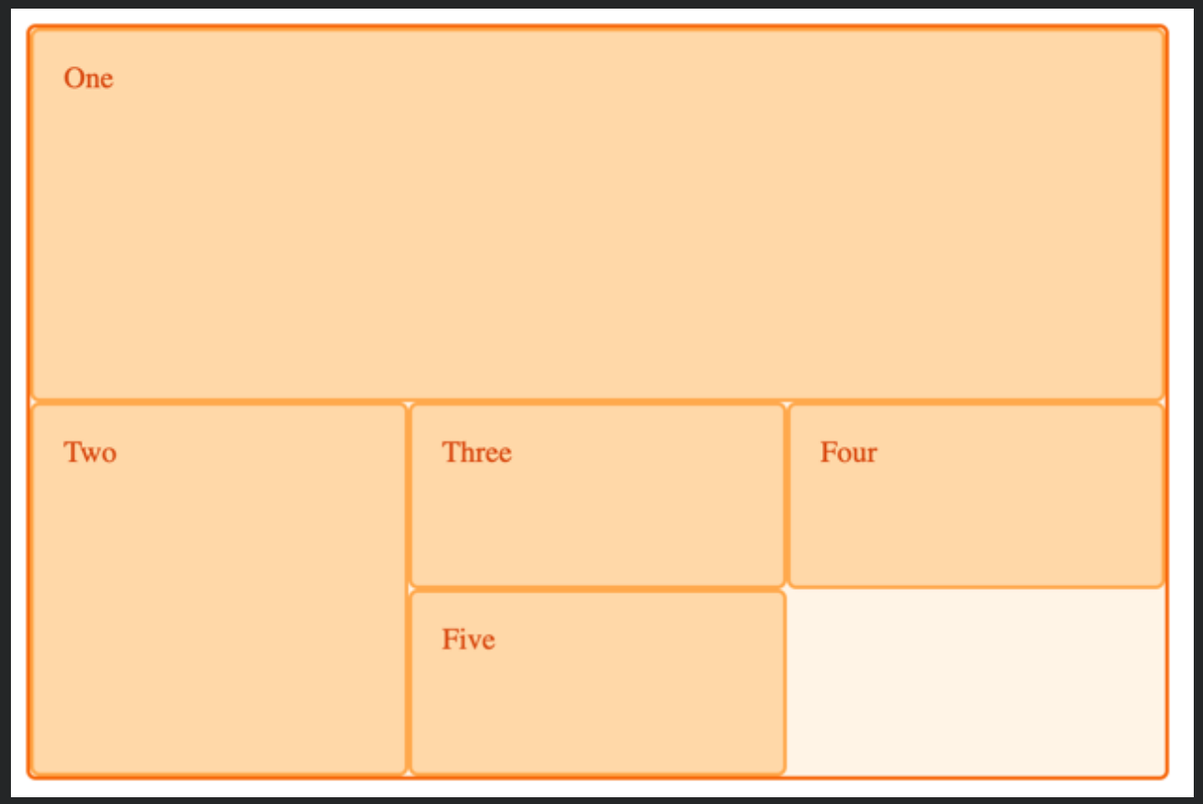



```
.wrapper {
  display: grid;
  grid-template-columns: 500px 1fr 2fr;
}
```

Positioning items

- Can explicitly place elements inside grid into grid areas

```
<div class="wrapper">
  <div class="box1">One</div>
  <div class="box2">Two</div>
  <div class="box3">Three</div>
  <div class="box4">Four</div>
  <div class="box5">Five</div>
</div>
.wrapper {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  grid-auto-rows: 100px;
}
.box1 {
  grid-column-start: 1;
  grid-column-end: 4;
  grid-row-start: 1;
  grid-row-end: 3;
}
.box2 {
  grid-column-start: 1;
  grid-row-start: 3;
  grid-row-end: 5;
}
```



Gaps

- Can set gaps between columns and rows

```html
<div class="wrapper">
  <div>One</div>
  <div>Two</div>
  <div>Three</div>
  <div>Four</div>
  <div>Five</div>
</div>
```

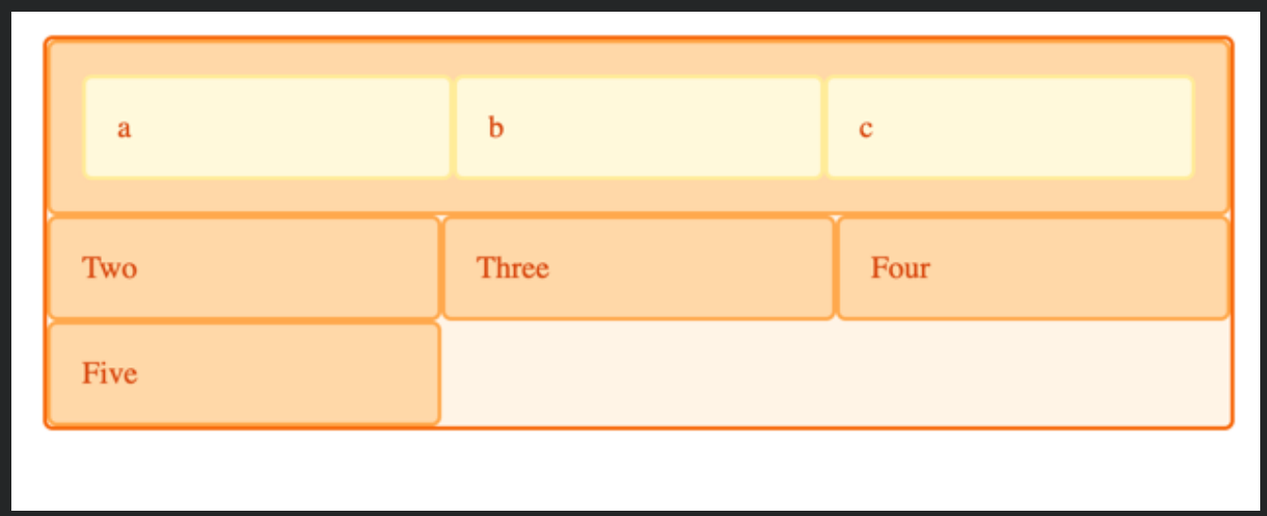
```css
.wrapper {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  column-gap: 10px;
  row-gap: 1em;
}
```

Nesting

- Can nest grids, which behave just like top-level

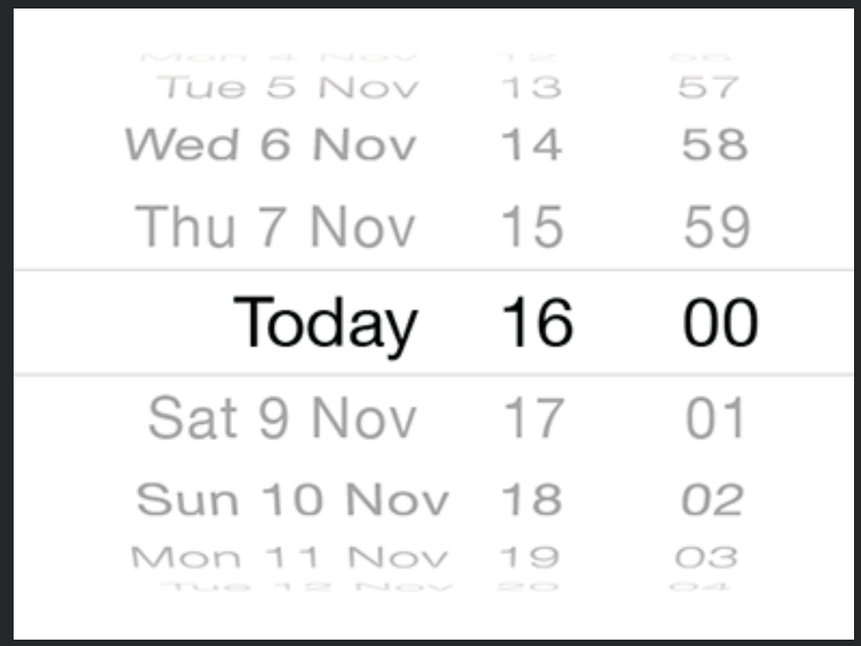
```
<div class="wrapper">
  <div class="box box1">
    <div class="nested">a</div>
    <div class="nested">b</div>
    <div class="nested">c</div>
  </div>
  <div class="box box2">Two</div>
  <div class="box box3">Three</div>
  <div class="box box4">Four</div>
  <div class="box box5">Five</div>
</div>


.box1 {
  grid-column-start: 1;
  grid-column-end: 4;
  grid-row-start: 1;
  grid-row-end: 3;
  display: grid;
  grid-template-columns: repeat(3, 1fr);
}
```



Designing for mobile devices

- Different devices have different aspect ratios.
  - Important to test for different device sizes.
  - May sometimes build alternative layouts for different device sizes.
- Using specialized controls important.
  - Enables mobile browsers to use custom device-specific widgets that may be much easier to use.



CSS Best Practices

- When possible, use CSS to declaratively describe behavior rather than code
  - Easier to read, can be optimized more effectively by browser
- Don’t repeat yourself (DRY)
  - Rather than duplicating rules, create selectors to style all related elements with single rule
- CSS should be readable
  - Use organization, indentation, meaningful identifiers, etc.

SWE 432 - Web Application Development

George Mason University

Instructor:
Dr. Kevin Moran

Teaching Assistant:
Oyindamola Oluyemo

Class will start in:

10:00

Review of Previous React Concepts

Review: Handling Events

```
class Toggle extends React.Component {
  constructor(props) {
    super(props);
    this.state = {isToggleOn: true};

    // This binding is necessary to make `this` work in the callback
    this.handleClick = this.handleClick.bind(this);
  }

  handleClick() {
    this.setState(prevState => ({ isToggleOn: !prevState.isToggleOn }));
  }

  render() {
    return (
      <button onClick={this.handleClick}>
        {this.state.isToggleOn ? 'ON' : 'OFF'}
      </button>
    );
  }
}

ReactDOM.render(
  <Toggle />, document.getElementById('root')
);
```

https://reactjs.org/docs/handling-events.html

Review: Component Lifecycle

```
class Timer extends React.Component {
  constructor(props) {
    super(props);
    this.state = { seconds: 0 };
  }

  tick() {
    this.setState(prevState => ({
      seconds: prevState.seconds + 1
    }));
  }

  componentDidMount() {
    this.interval = setInterval(() => this.tick(), 1000);
  }

  componentWillUnmount() {
    clearInterval(this.interval);
  }

  render() {
    return (
      <div>
        Seconds: {this.state.seconds}
      </div>
    );
  }
}

ReactDOM.render(<Timer />, mountNode);
```

ReactDOM.render(...)
[component created]
constructor(...)
render()
componentDidMount()

tick()
render()
...

[component rendered
again by parent]
componentWillUnmount()
[component created]
...

Review: Controlled Components

```
class EssayForm extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      value: 'Please write an essay about your favorite DOM element.'
    };

    this.handleChange = this.handleChange.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
  }

  handleChange(event) {
    this.setState({value: event.target.value});
  }

  handleSubmit(event) {
    alert('An essay was submitted: ' + this.state.value);
    event.preventDefault();
  }

  render() {
    return (
      <form onSubmit={this.handleSubmit}>
        <label>
          Name:
          <textarea value={this.state.value} onChange={this.handleChange} />
        </label>
        <input type="submit" value="Submit" />
      </form>
    );
  }
```

Functional Components + Hooks

But what if we want state + clean functional components??

```
import React, { Component } from 'react';

class Counter extends Component {
  constructor(props) {
    super(props);

    this.state = {
      count: 0,
    };
  }

  render() {
    return (
      <div>
        <p>You clicked {this.state.count} times</p>
        <button
          onClick={() =>
            this.setState({ count: this.state.count + 1 })
          }
        >
          Click me
        </button>
      </div>
    );
  }
}

export default Counter;
```

Functional Components + Hooks

Now we can have both with functional components + hooks!

```jsx
import React from 'react';

// how to use the state hook in a React function component
function Counter() {
  const [count, setCount] = React.useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}

export default Counter;
```

Review: Controlled Components

- Single source of truth
- Whenever a control changes its value
  - React is notified
  - State is updated
- Whenever state is updated
  - If necessary, render function executes and generates control with new value

Review: Reconciliation

```
<Card>
  <p>Paragraph 1</p>
  <p>Paragraph 2</p>
</Card>
```

```
<Card>
  <p>Paragraph 2</p>
</Card>
```

- Process by which React updates the DOM with each new render pass
- Occurs based on order of components
  - Second child of Card is destroyed.
  - First child of Card has text mutated.

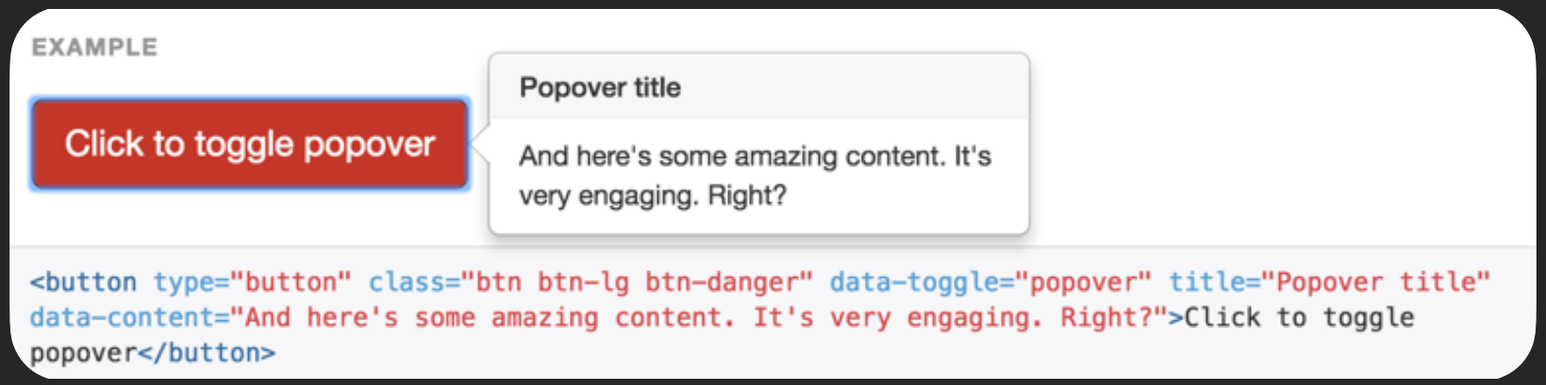
https://reactjs.org/docs/reconciliation.html

More React Programming

GUI Component Frameworks

- Can build arbitrarily complex UIs from the primitives we've seen
  - menus, nav bars, multiple views, movable panes, …
- But *lots* of work
  - Lots of functionality / behavior / styling to build from scratch
  - Browsers are not always consistent (*especially* before HTML5, CSS3)
  - Responsive layouts add complexity
- Solution: GUI component frameworks

GUI Component Frameworks



- Higher-level abstractions for GUI components
  - Rather than building a nav
  - Exposes new options, events, properties
- Integrated component
  - Associate HTML elements with components using CSS classes
  - Framework dynamically updates HTML as necessary through JS
  - Offers higher-level abstractions for interacting with components

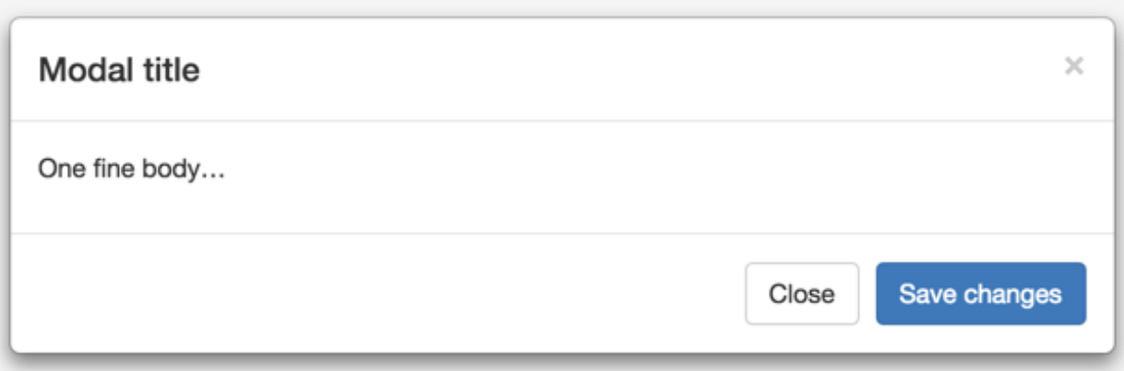
Bootstrap

- Popular GUI component framework
  - http://getbootstrap.com/
- Originally built and released by developers at Twitter in 2011
- Open source
- Offers baseline CSS styling & library of GUI components

Examples

Single toggle

```
<button type="button" class="btn btn-primary" data-toggle="button" aria-pressed="false"
autocomplete="off">
  Single toggle
</button>
```




```
<div class="modal fade" tabindex="-1" role="dialog">
  <div class="modal-dialog" role="document">
    <div class="modal-content">
      <div class="modal-header">
        <button type="button" class="close" data-dismiss="modal" aria-label="Close"><span aria-
hidden="true">&times;</span></button>
        <h4 class="modal-title">Modal title</h4>
      </div>
      <div class="modal-body">
        <p>One fine body&hellip;</p>
      </div>
      <div class="modal-footer">
        <button type="button" class="btn btn-default" data-dismiss="modal">Close</button>
        <button type="button" class="btn btn-primary">Save changes</button>
      </div>
    </div><!-- /.modal-content -->
  </div><!-- /.modal-dialog -->
</div><!-- /.modal -->
```

Bootstrap & React

- We'll use the react-bootstrap NPM module - Bootstrap for React!
- https://react-bootstrap.github.io

EXAMPLE

Holy guacamole! Best check yo self, you're not looking too good.

```
<Alert bsStyle="warning">
  <strong>Holy guacamole!</strong> Best check yo self, you're not looking too
  good.
</Alert>;
```

Bootstrap & React

- We'll use the react-bootstrap NPM module - Bootstrap for React!
- https://react-bootstrap.github.io

EXAMPLE

Holy guacamole! Best check yo self, you're not looking too good.

```
<Alert bsStyle="warning">
  <strong>Holy guacamole!</strong> Best check yo self, you're not looking too
  good.
</Alert>;
```

Conditional Rendering

- Based on state or props of component, render something

```
function UserGreeting(props) {
  return <h1>Welcome back!</h1>;
}

function GuestGreeting(props) {
  return <h1>Please sign up.</h1>;
}

function Greeting(props) {
  const isLoggedIn = props.isLoggedIn;
  if (isLoggedIn) {
    return <UserGreeting />;
  }
  return <GuestGreeting />;
}
```

Example: https://replit.com/@kmoran/swe-432-cond-render

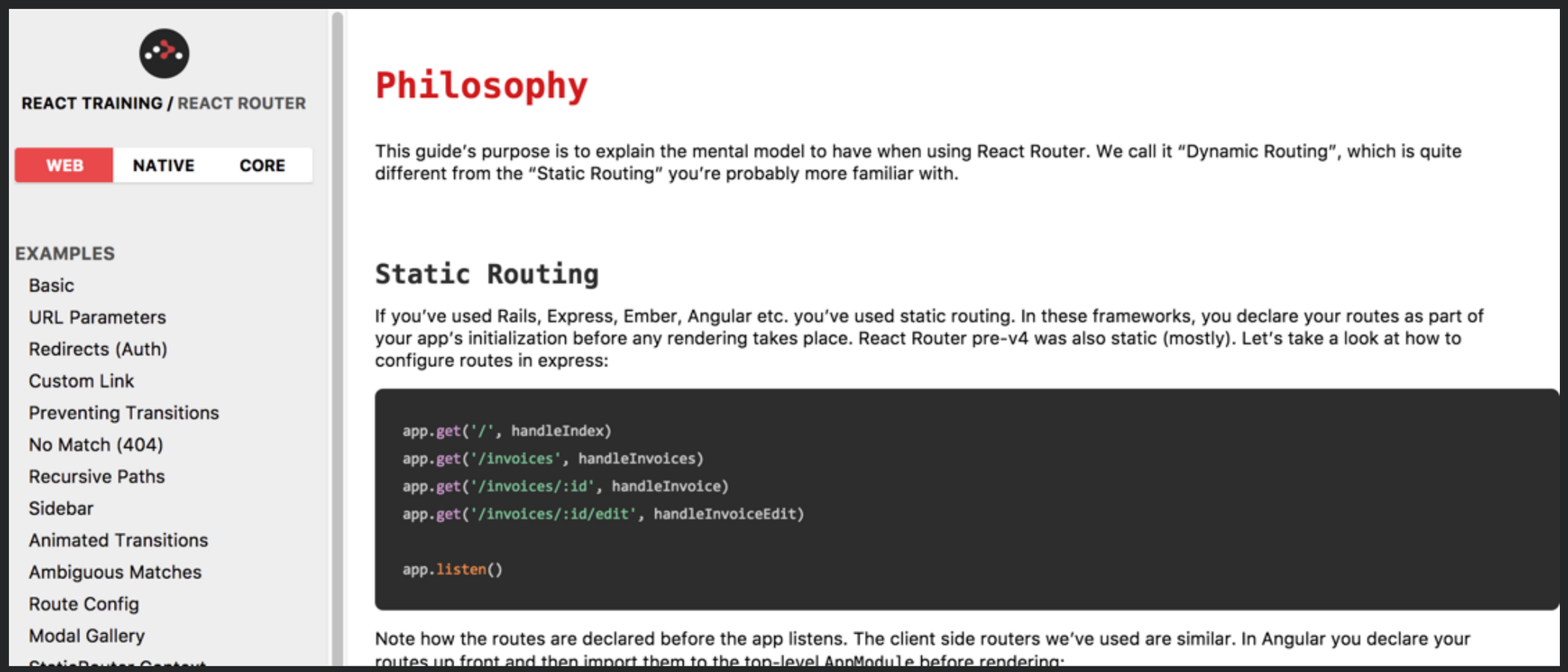
Front End Routing

- Using state to represent views is great
- But....
  - Does not offer unique URL for each view
  - Breaks the back / forward buttons
  - Makes it harder to deep link to specific views

- Would be great to simply render a component based on the current URL
  - => front end routing

React-Router

```
npm install react-router-dom
```



REACT TRAINING / REACT ROUTER

WEB | NATIVE | CORE

EXAMPLES

- Basic
- URL Parameters
- Redirects (Auth)
- Custom Link
- Preventing Transitions
- No Match (404)
- Recursive Paths
- Sidebar
- Animated Transitions
- Ambiguous Matches
- Route Config
- Modal Gallery

Philosophy

This guide's purpose is to explain the mental model to have when using React Router. We call it "Dynamic Routing", which is quite different from the "Static Routing" you're probably more familiar with.

Static Routing

If you've used Rails, Express, Ember, Angular etc. you've used static routing. In these frameworks, you declare your routes as part of your app's initialization before any rendering takes place. React Router pre-v4 was also static (mostly). Let's take a look at how to configure routes in express:

```
app.get('/', handleIndex)
app.get('/invoices', handleInvoices)
app.get('/invoices/:id', handleInvoice)
app.get('/invoices/:id/edit', handleInvoiceEdit)

app.listen()
```

Note how the routes are declared before the app listens. The client side routers we've used are similar. In Angular you declare your routes up front and then import them to the top-level AppModule before rendering:

https://reacttraining.com/react-router/web/guides/philosophy

Router example

- https://codesandbox.io/s/react-router-basic-forked-fite7q?file=/example.js

Functional React & CSS Exercises

- *React Exercise:*
  - *Instructions:* Implement conditional rendering so that the Message is displayed when a user presses the button.
- *CSS Exercise:*
  - *Instructions:*
    - Center a component inside it's container
    - Use a display grid to create layout with multiple rows and columns
    - Override one of the Bootstrap selectors
  - https://replit.com/@kmoran/swe-432-react-example#src/App.jsx

Please Turn-in Your In-Class Activity for a Quiz Grade

https://go.gmu.edu/react-activity

Please provide the URL of your Replit Repo

Acknowledgements

- Slides Adapted from Dr. Thomas LaToza's SWE-432 course