CSS & DOM

SWE 432, Fall 2018

Web Application Development

Review: HTML Example

```
<!DOCTYPE html>
<html>
<head>
    <link rel="stylesheet" type="text/css" href="main.css">
    <title>Prof Bell's Webpage</title>
</head>
<body>
<h1>
    Prof Jonathan Bell
</h1>
```

Use <h1>, <h2>, …, <h5> for headings

```
    <h2>Welcome, students!</h2>
        <p>
            <a href="https://www.youtube.com/watch?v=dQw4w9WgXcQ">See how to make
this page</a>
        </p>
    <h2>
        Some funny links
    </h2>
    <p>
        <ul>
            <li><a href="http://www.homestarrunner.com">Homestar Runner</a></li>
            <li><a
href="http://www.wb3w.net/The%20Original%20Hamsterdance.htm">Hamster Dance</a></li>
        </ul>
    </p>
    <h3>
        About Prof Bell
    </h3>
    <p>
        Prof Bell's office is at 4422 Engineering Building. His email address is <a
href="mailto:bellj@gmu.edu">bellj@gmu.edu</a>.
    </p>
    <p>
        Last updated: September 4th, 1999
    </p>
</div>
</body>
</html>
```

https://seecode.run/#-KQgR7vG9Ds7IUJS1kdq

Today

- HW2 Recap
- CSS
- Bootstrap
- DOM
- HW3

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 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 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?

```
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

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

```
#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;

Exercise - What is selected?

1.
```
div.menu-bar ul ul {
  display: none;
}
```

2.
```
div.menu-bar li:hover > ul {
  display: block;
}
```

ul: unordered list
li: list element

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

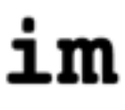
- Can set text color (color) and background color (background-color)
- Several ways to describe color
  - six digit hex code (e.g., #ee3e80)
  - color names: 147 predefined names
  - rgb(red, green, blue): amount of red, green, and blue
  - hsla(hue, saturation, lightness, alpha): alternative scheme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

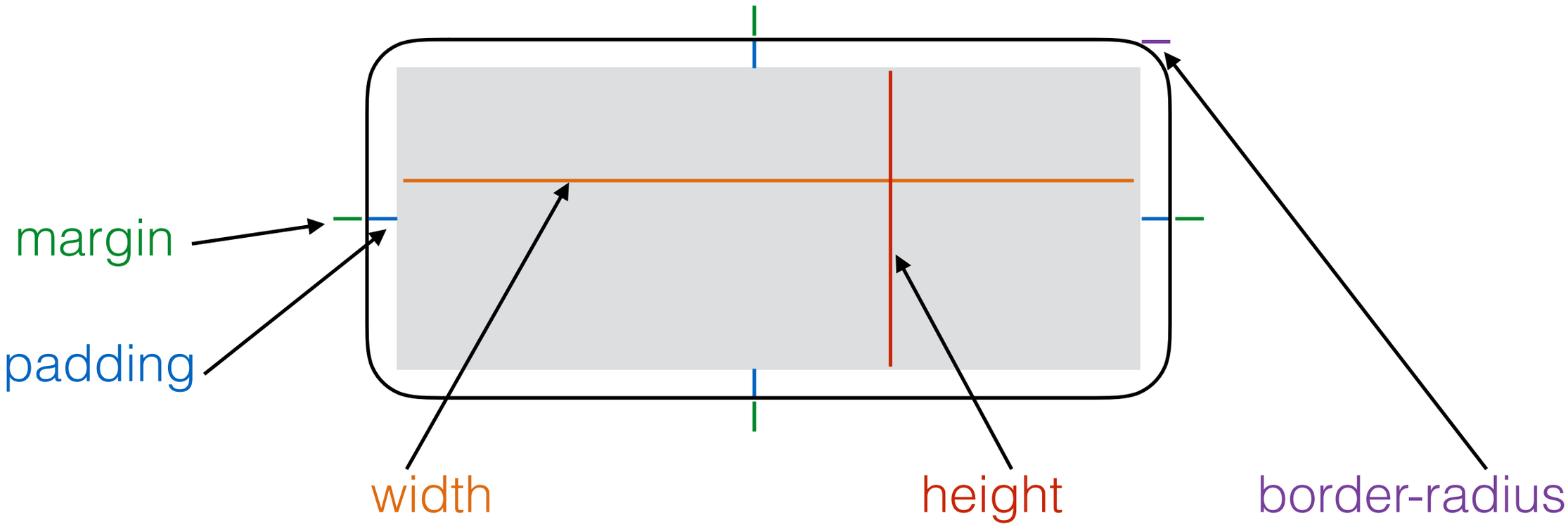
Cursor

```
<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

```
.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);
}

<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

Fixed width vs. liquid layouts

- Fixed width
  - Use width="[num]px" to force specific sizes
  - Allows for tightest control of look and feel
  - But can end up with extra whitespace around edge of web page
- Liquid layout
  - Use width="[num]%" to size relative to container sizes
  - Pages expand to fill the entire container size
  - Problems
    - Wide windows may create long lines of text can be difficult to read
    - Very narrow windows may squash words, breaking text onto many lines
  - (Partial) solution
    - Can use min-width, min-height, max-width, max-height to set bounds on sizes

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.

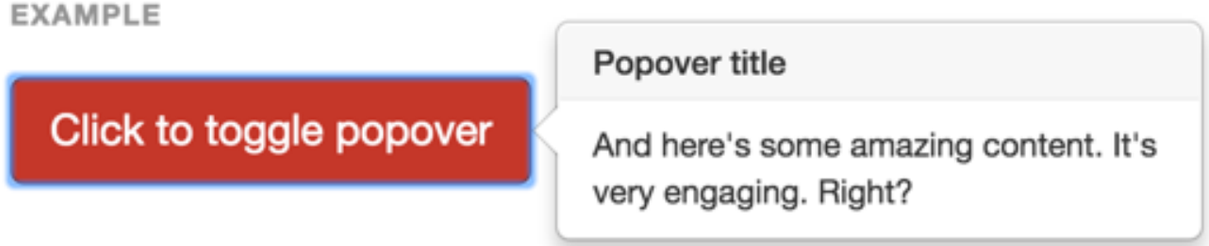
GUI Component Frameworks

- 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

- Hig
  - F
  - E

EXAMPLE




```
<button type="button" class="btn btn-lg btn-danger" data-toggle="popover" title="Popover title"
data-content="And here's some amazing content. It's very engaging. Right?">Click to toggle
popover</button>
```

- 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

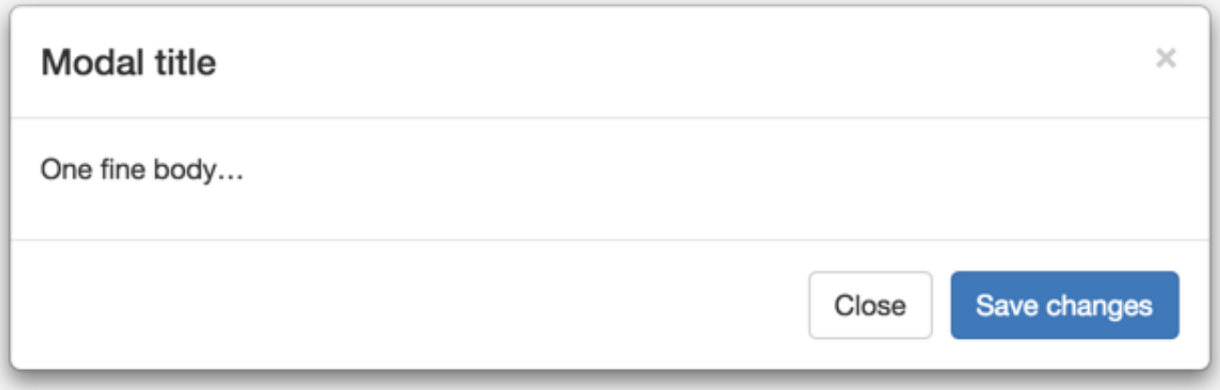
Bootstrap

- 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 Grid Layout

- Offers 12 column grid
  - Build column widths as integer number of columns. Total must add up to exactly 12.
  - Use rows to create horizontal groups of columns.
- Based on space, columns will either appear horizontally, or if not enough space, will be stacked vertically
- Choice between fixed-width (.container) and full-width (.container-fluid)

http://getbootstrap.com/css/

Example: Stacked-to horizontal

.col-md-1	.col-md-1	.col-md-1	.col-md-1	.col-md-1	.col-md-1	.col-md-1	.col-md-1	.col-md-1	.col-md-1	.col-md-1	.col-md-1
.col-md-8								.col-md-4			
.col-md-4				.col-md-4				.col-md-4			
.col-md-6						.col-md-6					

```
<div class="row">
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
</div>
<div class="row">
  <div class="col-md-8">.col-md-8</div>
  <div class="col-md-4">.col-md-4</div>
</div>
<div class="row">
  <div class="col-md-4">.col-md-4</div>
  <div class="col-md-4">.col-md-4</div>
  <div class="col-md-4">.col-md-4</div>
</div>
<div class="row">
  <div class="col-md-6">.col-md-6</div>
  <div class="col-md-6">.col-md-6</div>
</div>
```

.col-md-1

.col-md-1

.col-md-1

.col-md-1

.col-md-1

.col-md-1

.col-md-1

.col-md-1

.col-md-1

.col-md-1

.col-md-8

.col-md-4

.col-md-4

.col-md-4

.col-md-4

.col-md-6

.col-md-6

http://getbootstrap.com/css/

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>;
```

Frontend JavaScript

- Static page
  - Completely described by HTML & CSS
- Dynamic page
  - Adds interactivity, updating HTML based on user interactions
  - Adding JS to frontend:

```
<script>
  console.log("Hello, world!");
</script>
```

  - We try to avoid doing this because:
    - Hard to organize
    - Different browsers support different things

DOM: Document Object Model

- API for interacting with HTML browser
- Contains objects corresponding to every HTML element
- Contains global objects for using other browser features

Reference and tutorials

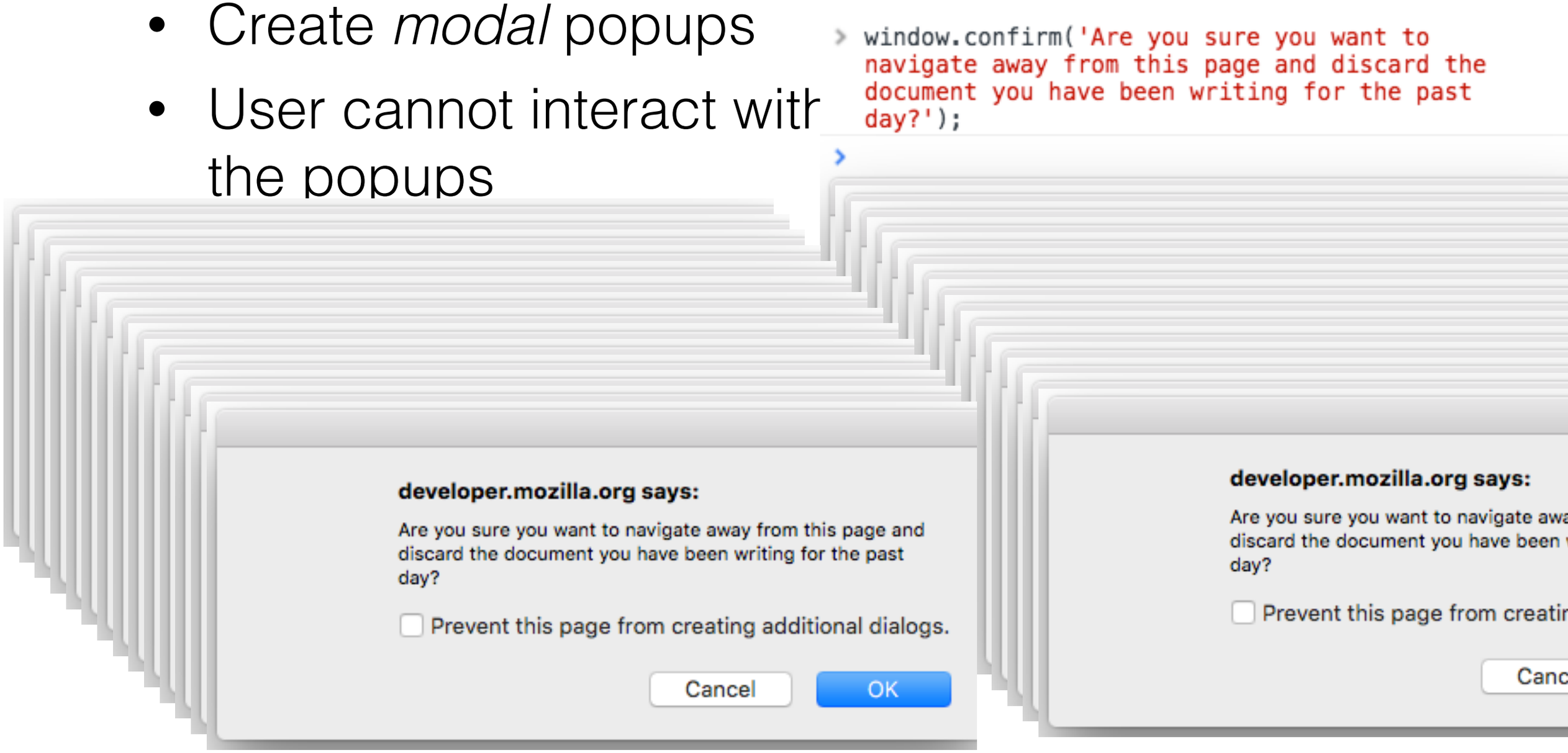
https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model

Global DOM objects

- window - the browser window
  - Has properties for following objects (e.g., window.document)
  - Or can refer to them directly (e.g., document)
- document - the current web page
- history - the list of pages the user has visited previously
- location - URL of current web page
- navigator - web browser being used
- screen - the area occupied by the browser & page

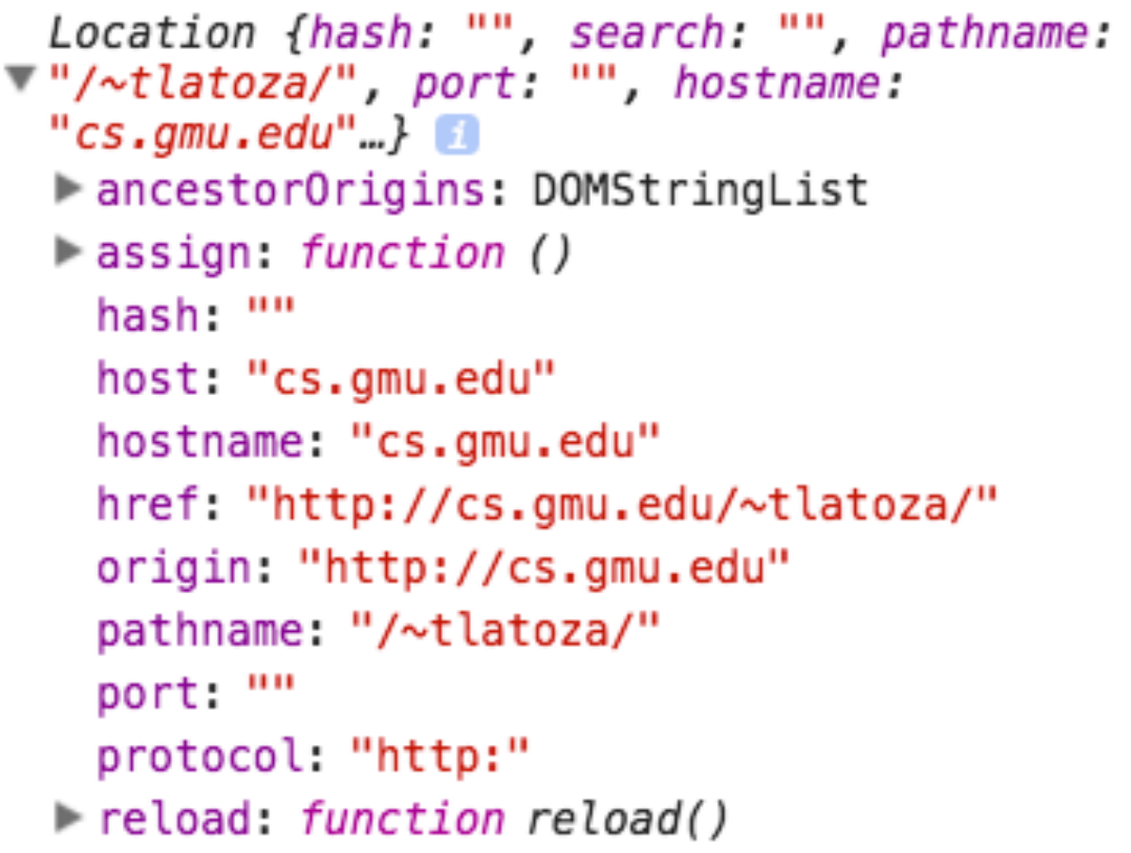
Working with popups

- alert, confirm, prompt
  - Create *modal* popups
  - User cannot interact with the popups



Working with location

- Some properties
  - location.href - full URL of c
  - location.protocol - protocc
  - location.host - hostname
  - location.port
  - location.pathname
- Can navigate to new page b location
  - location.href = '[new URL]';



Traveling through history

- history.back(), history.forward(), history.go(delta)
- What if you have an SPA & user navigates through different views?
  - Want to be able to jump between different views *within* a single URL
- Solution: manipulate history state
  - Add entries to history stack describing past views
  - Store and retrieve objec
    history.pushState() and

```
> history.pushState( { activePane: 'main' }, "");
<· undefined
> history.state
<· ▶Object {activePane: "main"}
> history.back();
<· undefined
> history.state
<· null
```

DOM Manipulation

- We can also manipulate the DOM directly
- For this class, we will *not* focus on doing this, but will use React instead
- This is how React works though - it manipulates the DOM

DOM Manipulation

Multiply two numbers

2 * 3 = 6

Multiply

- value

```
<h3>Multiply two numbers</h3>
<div>
    <input id="num1" type="number" /> *
    <input id="num2" type="number" /> =
    <span id="product"></span>
    <br/><br/>
    <button id="compute">Multiply</button>
</div>
```

```
document.getElementById('compute')
        .addEventListener("click", multiply);
function multiply()
{
    var x = document.getElementById('num1').value;
    var y = document.getElementById('num2').value;
    var productElem = document.getElementById('product');
    productElem.innerHTML = x * y;
}
```

"Get compute element"

"When compute is clicked, call multiply"

May choose any event that the compute element produces. May pass the name of a function or define an anonymous function inline.

DOM Manipulation

Multiply two numbers

3 * 4 = 12

Multiply

```
<h3>Multiply two numbers</h3>
<div>
    <input id="num1" type="number" /> *
    <input id="num2" type="number" /> =
    <span id="product"></span>
    <br/><br/>
    <button id="compute">Multiply</button>
</div>
```

```
document.getElementById('compute')
        .addEventListener("click", multiply);
function multiply()
{
    var x = document.getElementById('num1').value;
    var y = document.getElementById('num2').value;
    var productElem = document.getElementById('product');
    productElem.innerHTML = '<b>' + x * y + '</b>';
}
```

"Get the current value of the num1 element"

"Set the HTML between the tags of productElem to the value of x * y"

Manipulates the DOM by programmatically updating the value of the HTML content. DOM offers accessors for updating all of the DOM state.

DOM Manipulation Pattern

- Wait for some event
  - click, hover, focus, keypress, …
- Do some computation
  - Read data from event, controls, and/or previous application state
  - Update application state based on what happened
- Update the DOM
  - Generate HTML based on new application state
- Also: JQuery

Examples of events

- Form element events
  - change, focus, blur
- Network events
  - online, offline
- View events
  - resize, scroll
- Clipboard events
  - cut, copy, paste
- Keyboard events
  - keydown, keypress, keypup
- Mouse events
  - mouseenter, mouseleave, mousemove, mousedown, mouseup, click, dblclick, select

List of events: https://www.w3.org/TR/DOM-Level-3-Events/

DOM Manipulation Example

https://jsfiddle.net/Lbnhs8aa/1/

React vs DOM manipulation

- React will help us a lot when:
  - State changes (who wants to keep track of where the state is on the page?)
  - State needs to appear in multiple places (and be synchronized)
  - Page contains lots of data not shown on the page (and you need to swap out what's shown often)

Loading pages

- What is the output of the following?

```
<script>
    document.getElementById('elem').innerHTML
= 'New content';
</script>
```

```
<div id="elem">Original content</div>
```

Answer: cannot set property innerHTML of undefined
Solution: Put your script in after the rest of the page is loaded
Or, perhaps better solution: don't do DOM manipulation

HW3 Discussion

https://www.jonbell.net/swe-432-fall-2018-web-programming/homework-3/