Dynamic Webpages

SWE 432, Fall 2016

Design and Implementation of Software for the Web

Today's Objectives

- Learn how to write code to interact with HTML & CSS through ES6
- Learn how to manipulate HTML & CSS through jQuery
- Learn how to build pages with the Bootstrap UI framework

MDN reference materials on the DOM:
https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model

jQuery: https://jquery.com/

Dynamic web pages

- Static page
  - Completely described by HTML & CSS
  - May have interactivity (e.g., CSS transforms, hover pseudo-classes)
  - But described in HTML & CSS
- Dynamic page
  - Uses code to generate the page
  - This class: building page elements client-side through ES6

Strict mode

- In order to use ES6 features, need to force browser to use current version of JS
- "use strict";
  - Should be first statement in every script tag.
  - ES6 modules (see next week) are always in strict mode
- Turns mistakes into errors
  - Code that is illegal but tolerated by browser now throws an exception
  - Goal: if a typo creates behavior that is never reasonable, throw an error

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Strict_mode

DOM: Document Object Model

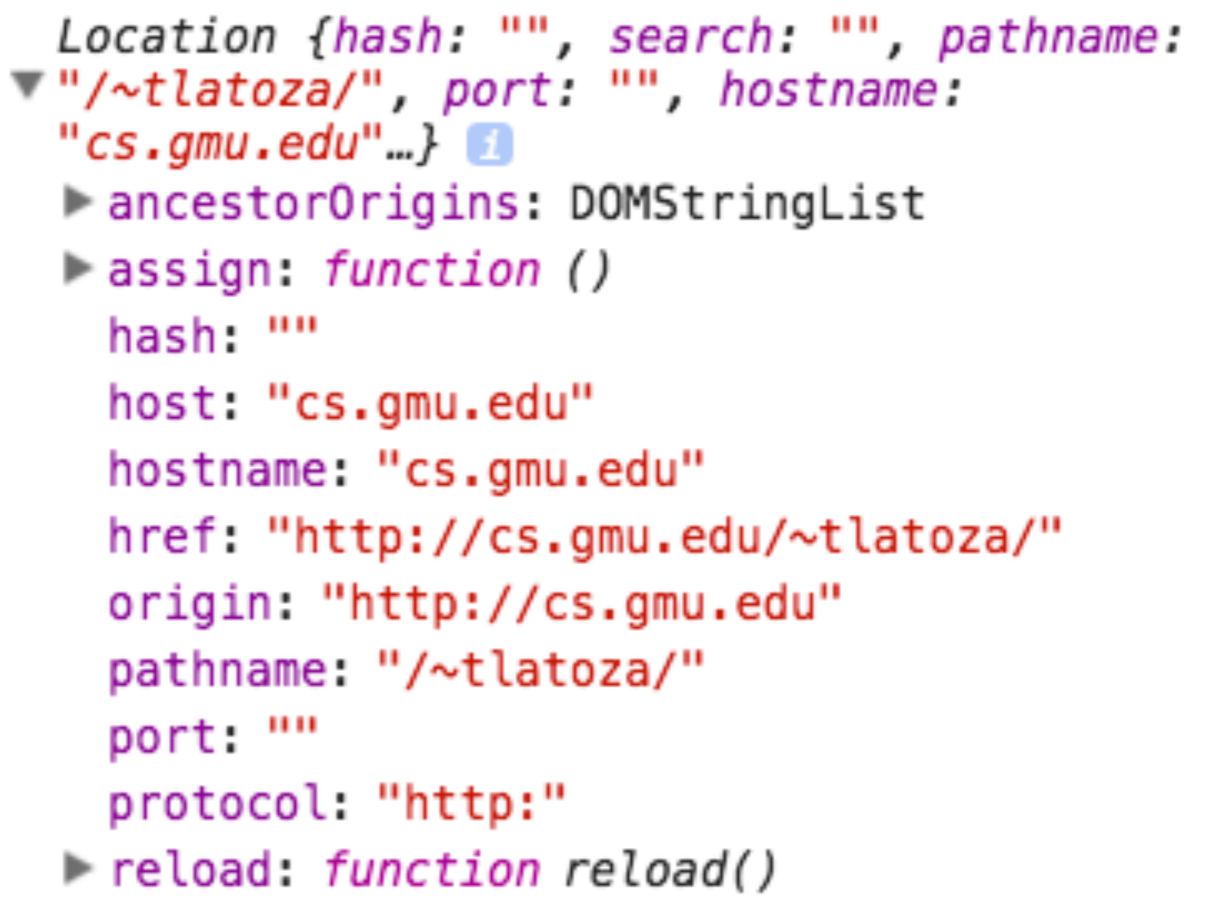
- API for interacting with HTML browser
- Contains objects corresponding to every HTML element
- Contains global objects for using other browser features

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 location

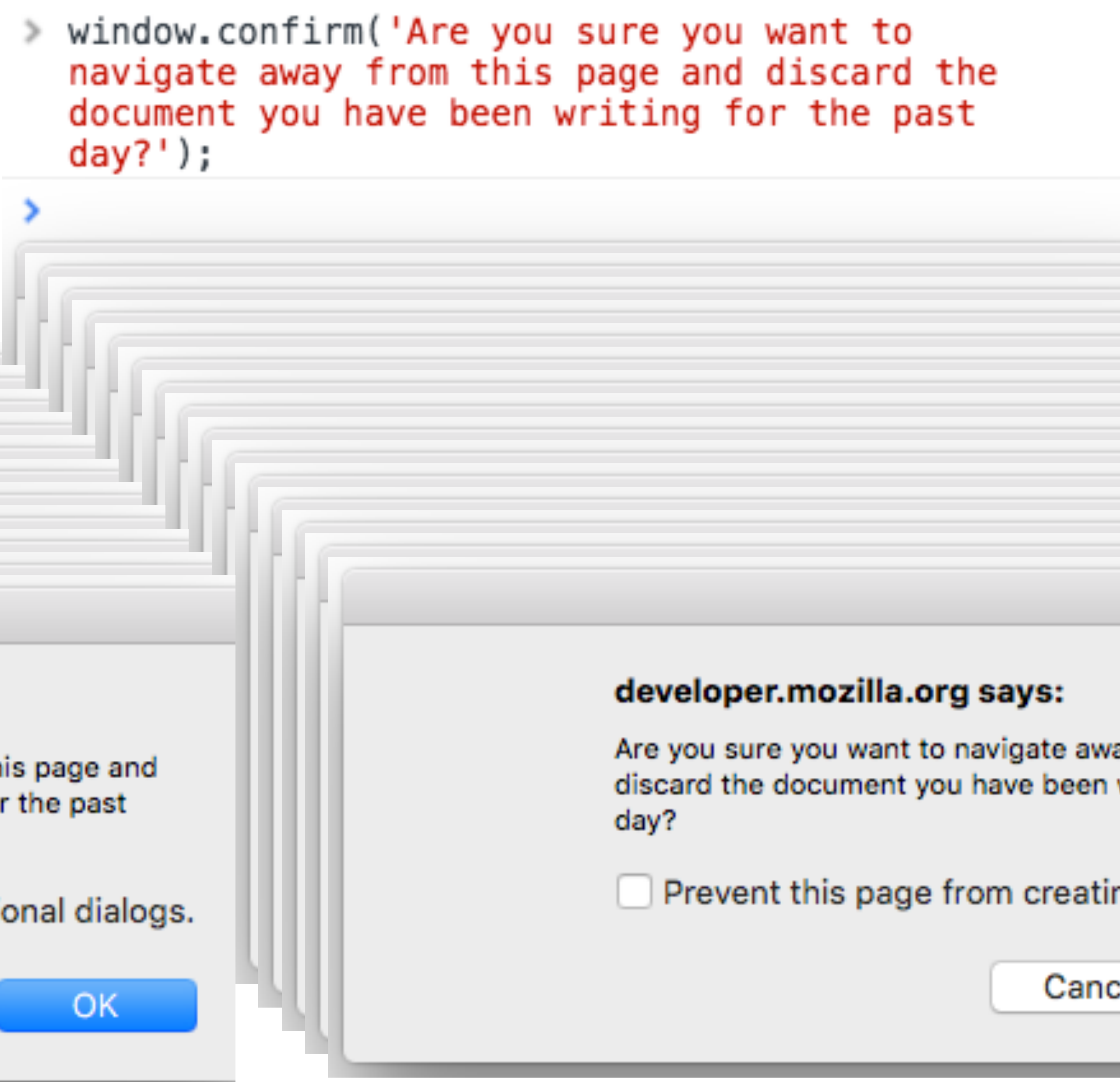
- Some properties
  - location.href - full URL of current location
  - location.protocol - protocol being used
  - location.host - hostname
  - location.port
  - location.pathname
- Can navigate to new page by updating the current location
  - location.href = '[new URL]';



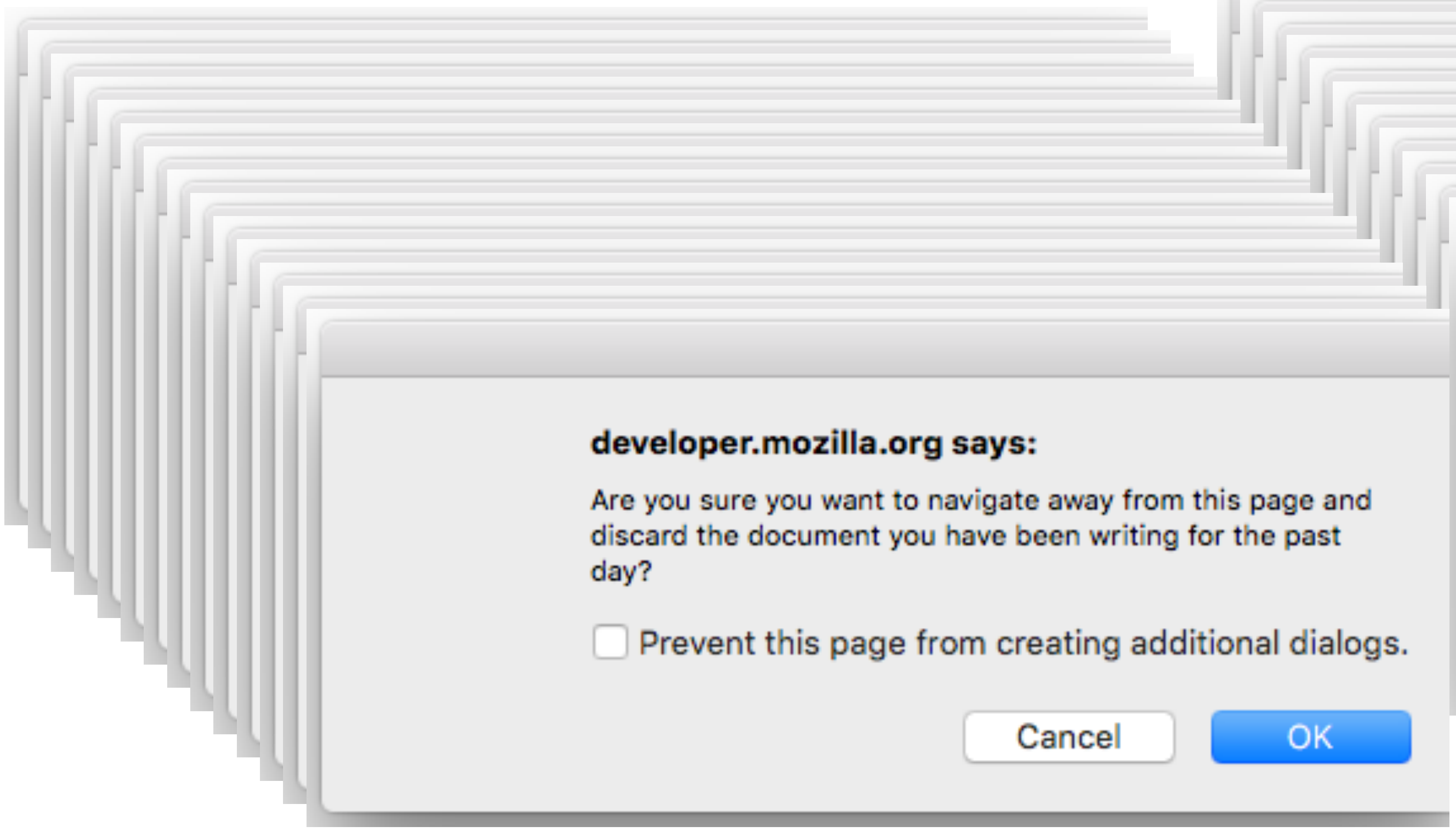
```
Location {hash: "", search: "", pathname:
▼"/~tlatoza/", port: "", hostname:
 "cs.gmu.edu"…}
  ▶ancestorOrigins: DOMStringList
  ▶assign: function ()
   hash: ""
   host: "cs.gmu.edu"
   hostname: "cs.gmu.edu"
   href: "http://cs.gmu.edu/~tlatoza/"
   origin: "http://cs.gmu.edu"
   pathname: "/~tlatoza/"
   port: ""
   protocol: "http:"
  ▶reload: function reload()
```

Working with popups

- alert, confirm, prompt
  - Create *modal* popups



```
> window.confirm('Are you sure you want to
  navigate away from this page and discard the
  document you have been writing for the past
  day?');
>
```



developer.mozilla.org says:

Are you sure you want to navigate away from this page and discard the document you have been writing for the past day?

☐ Prevent this page from creating additional dialogs.

developer.mozilla.org says:

Are you sure you want to navigate awa
discard the document you have been
day?

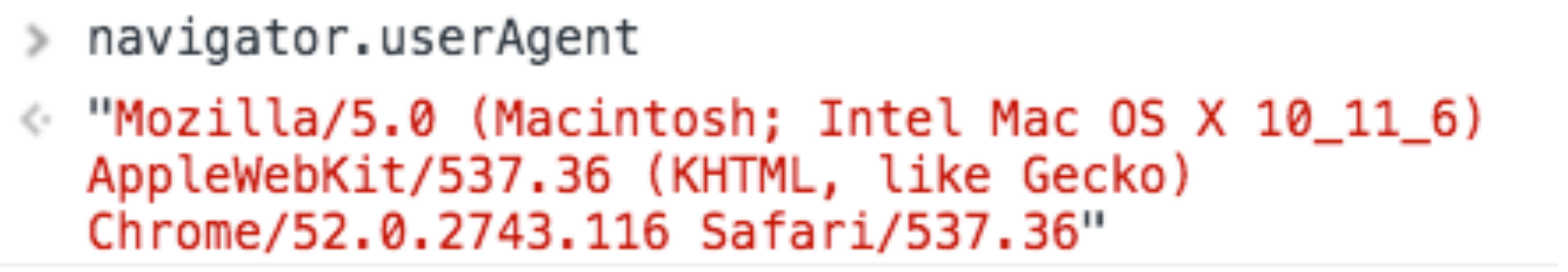
☐ Prevent this page from creatir

Working with windows

- open(URL, name, options) - open new browser window
- close() - close browser window
- print() - print a web page
- blur(), focus() - bring window to foreground or background
- moveBy(dx, dy), moveTo(x, y) - move browser position on screen
- resizeBy(dx, dy), resizeTo(x, y) - resize browser window
- scrollBy(dx, dy), scrollTo(x, y) - scroll browser window

Working with navigator

- Properties
  - appName - browser name
  - appVersion - browser version
  - language - user’s language
  - platform - user’s OS
  - userAgent

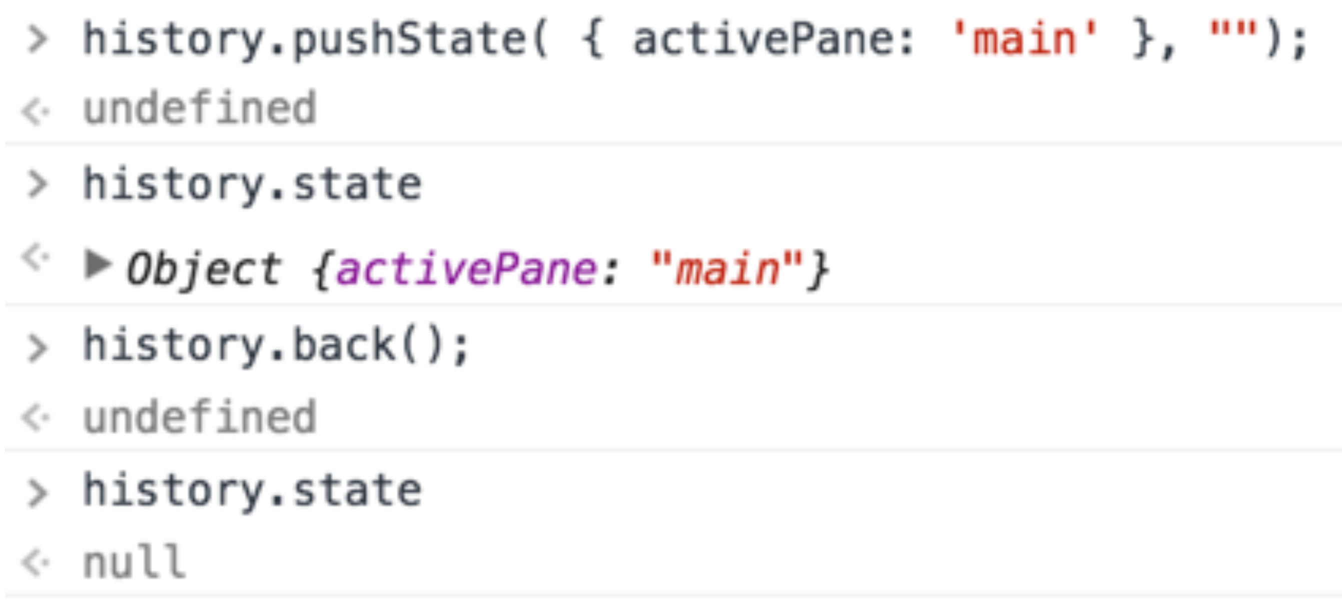


```
> navigator.userAgent
< "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6)
  AppleWebKit/537.36 (KHTML, like Gecko)
  Chrome/52.0.2743.116 Safari/537.36"
```

- Information about user’s computer and their browser
- Can use to customize content based on browser, if executing on mobile device, language, etc.

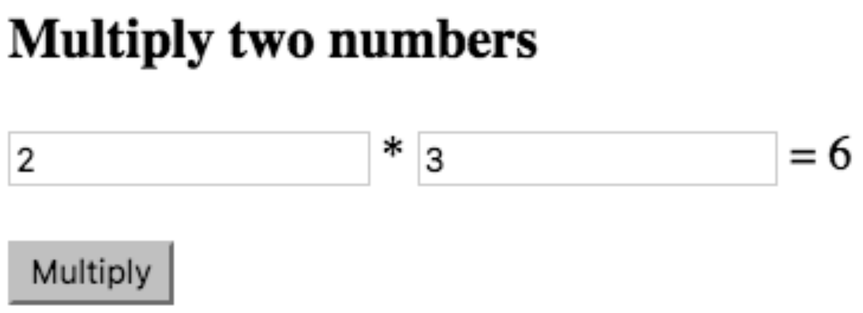
Traveling through history

- 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 object using history.pushState() and history.state



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

DOM Manipulation



```html
<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>
```

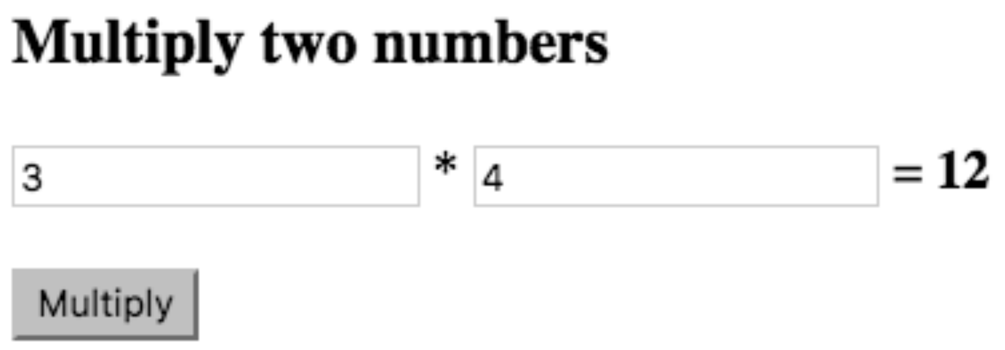
```javascript
document.getElementById('compute')
        .addEventListener("click", multiply);
function multiply()
{
    var x = document.getElementById('num1').value;
    var y = document.getElementById('num2').value;
    var productElem = document.getElementById('product');
    productElem.innerHTML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



```
<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

Multiply two numbers

```
This is a lot of typing…  Is there a shorter
way to do all this?
```

```html
<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>
```

```javascript
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>’;
}
```

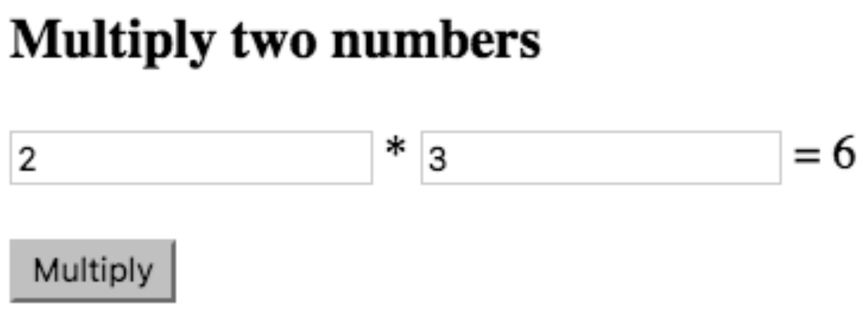
“Get the current value of the num1 element”

“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 w/ jQuery

```html
<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>
```



```javascript
$('#compute').click(function() {
    var output = $('#num1').val() * $('#num2').val();
    $('#product').html('<b>' + output + '</b>');
});
```

"Call the $ function"

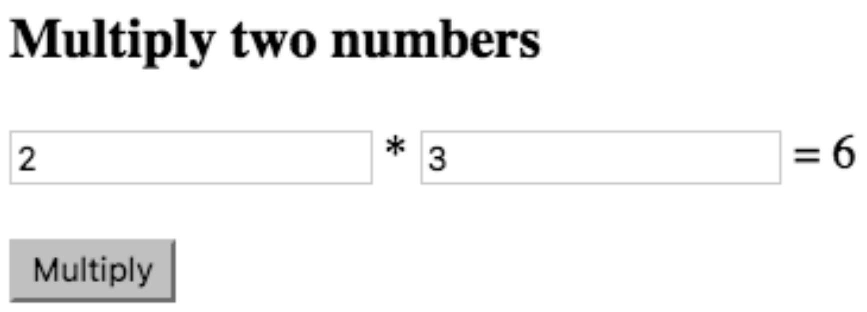
To make code as concise as possible, jQuery defines a jQuery function and a $ alias that can be used interchangeably. $ is an identifier, not a keyword.

"Select all elements with an id of compute."

Any valid CSS selector may be used (including nesting simple selectors). Queries may return zero, one, or more than one element. Some functions may be applied to *all* matching elements. Others will select only the first element.

DOM Manipulation w/ jQuery

```
<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>
```



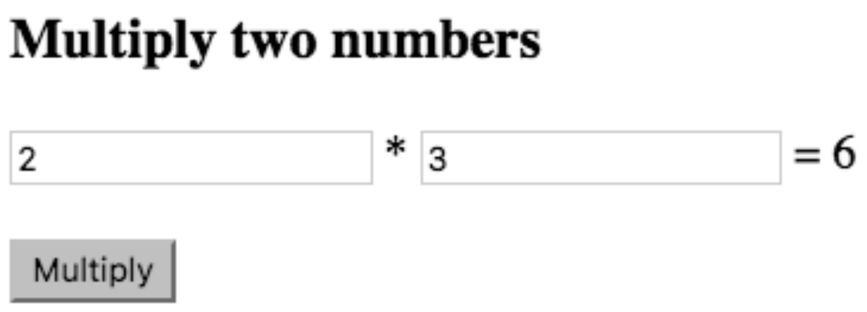
```
$('#compute').click(function() {
     var output = $('#num1').val() * $('#num2').val();
     $('#product').html('<b>' + output + '</b>');
});
```

“Bind an event handler to the click event”

“Call this function whenever the event occurs.”

DOM Manipulation w/ jQuery

```
<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>
```



```
$('#compute').click(function() {
    var output = $('#num1').val() * $('#num2').val();
    $('#product').html('<b>' + output + '</b>');
});
```

“Get the value of num1”

“Set the innerHTML”

JQuery

- jQuery.   jquery.com
    - Initially released in 2006
    - Provides abstractions & wrapper functions for DOM manipulation & AJAX
        - Defacto language for interacting with DOM
        - Used by >65% of 10 million most trafficked sites
- Can include locally or include through a content distribution network (CDN)

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

Working with JQuery elements

- $(‘#elem’) and document.getElementById are not equivalent
  - First creates jQuery object that includes jQuery wrapper functions in addition to DOM elements
- Can use each to iterate over elements

  -
    ```
    $( "li" ).each(function() {
        $( this ).addClass( "foo" );
    });
    ```
- For functions that apply to multiple elements, can do directly
  - `$( "li" ).addClass( "bar" );`

DOM Manipulation Pattern

- Wait for some event
  - click(), hover(), focus(), submit(), keypress(), …
- Do some computation
  - Read data from event, controls, and/or previous application state
  - Update application state based on what happened
- Update the DOM
  - Edit HTML elements or CSS attributes

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

Read data from DOM

- Most events have parameters

```
$( "#target" ).keyup(function( event ) {
    console.log('key ' + event.which + ' up.')
});
```

- Can read data from HTML attributes & properties
  - $(elem).attr("checked")
  - $(elem).prop("selectedIndex")

Update the DOM

- html(), append(), prepend()
  - Can replace existing content or keep it
- addClass(), removeClass()
  - Apply / remove classes to update styling
- hide(), show()
  - Shortcut for toggling { display: none; }
- attr()
  - Set an attribute

Demo

- Display to the user if the browser is currently connected to the network.

Loading pages

- What is the output of the following?

```
<script>
    $('#elem').html('Updated content');
</script>

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

document ready

- Code in script tags will run in the order in which it is contained in the page
- Solution: wait for an event that signals that all of the HTML on the page has been loaded
- Different from all of the *resources* has been loaded
    - Resources (e.g., images) might still be loading. That’s usually ok, as this might take longer and not usually relevant.
    - document.load event signals that everything has been loaded

```
<script>
    $( document ).ready(function() {
        $('#elem').html('Updated content');
    });
</script>

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

Associating controls with model

- Image creating a list of divs containing content and a close button
    - When user clicks close button, how do you know which div to delete?

Associating controls with model

- onclick=“handlerFunction(this)”
  - This object will be the element that got the event

- Data-* attributes can store arbitrary data with elements
  - `<div id="elem" data-index=3></div>`
  - `console.log($('#elem').data('index'));`
  - Name of data attribute may be anything
    - But should *always* be lower case or may cause unexpected parsing

- May use one or both depending on situation & other information available

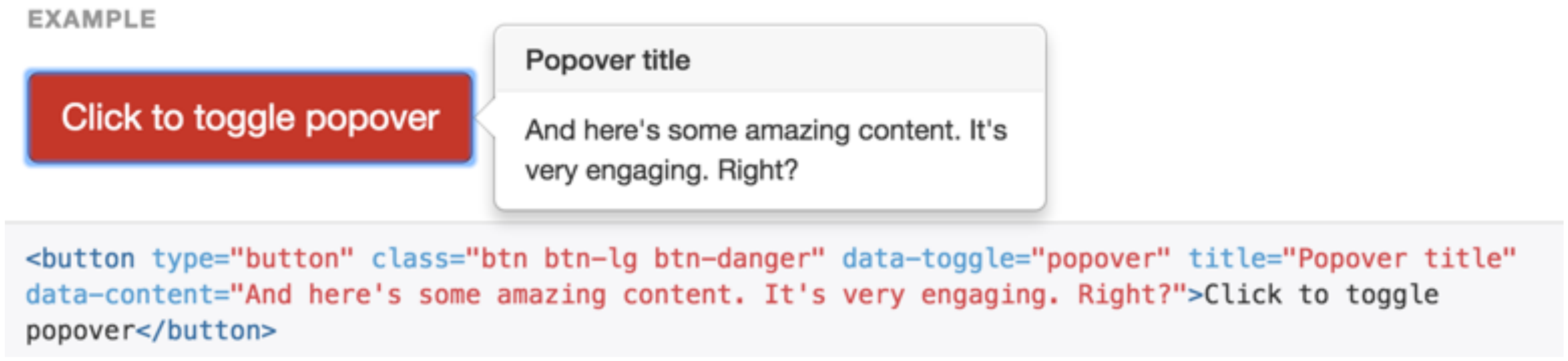
Demo

- *Really* simple todo app

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



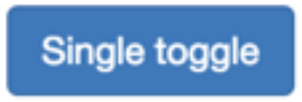
```
<button type="button" class="btn btn-lg btn-danger" data-toggle="popover" title="Popover title"
data-content="And here's some amazing content. It's very engaging. Right?">Click to toggle
popover</button>
```

- 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

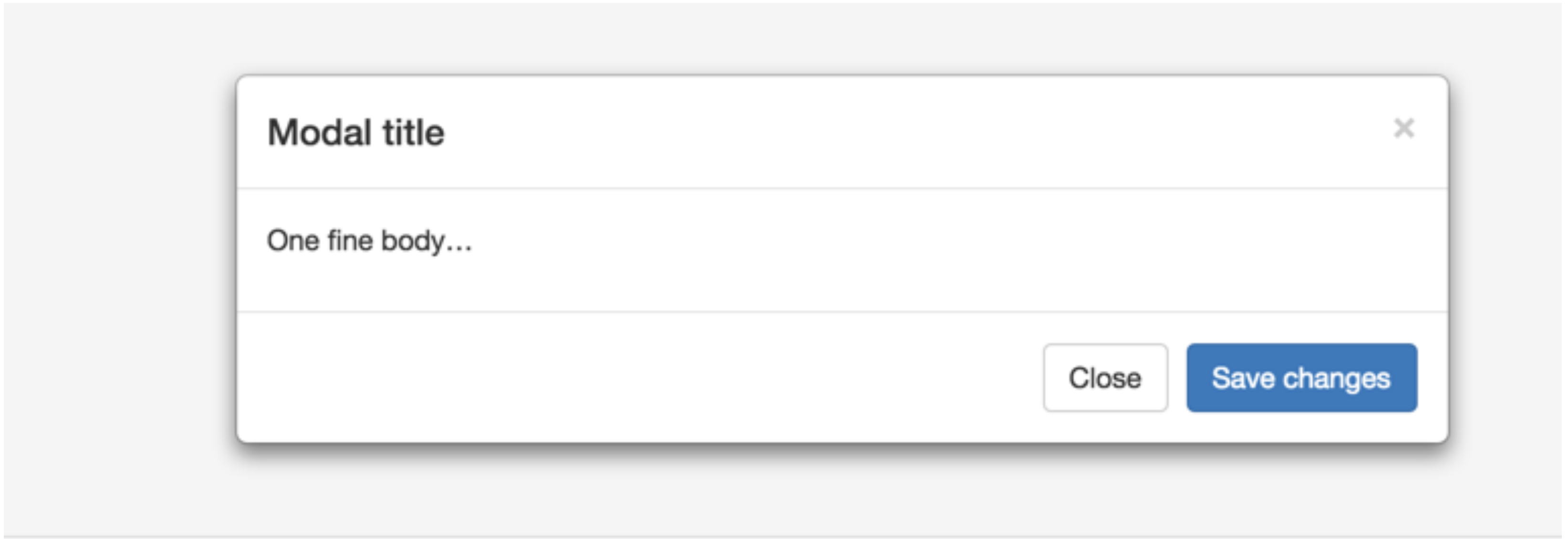
Bootstrap

- 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



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



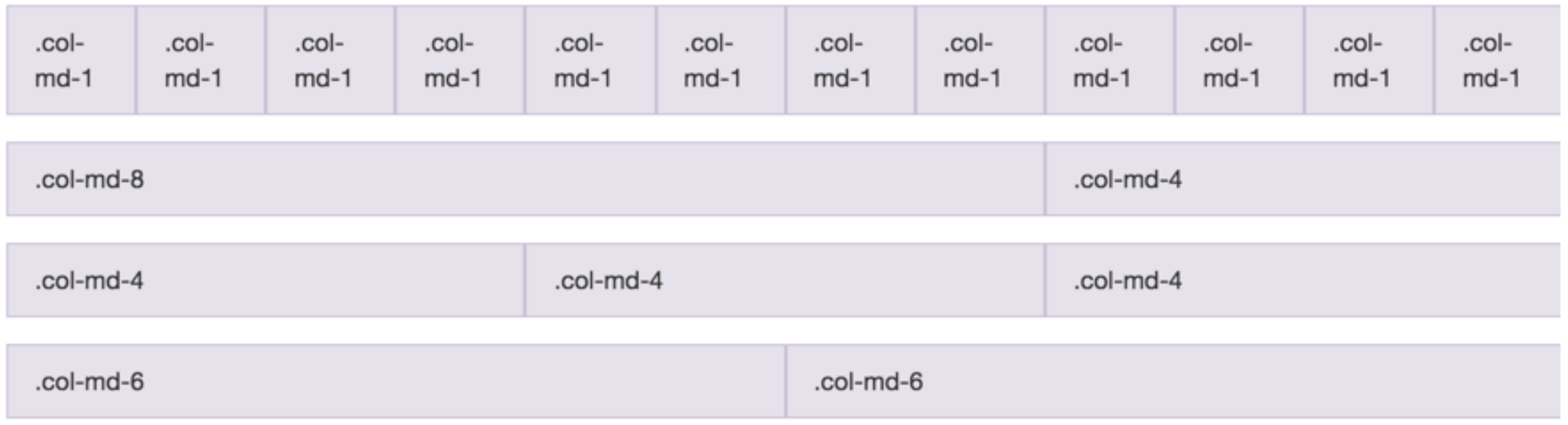
```
<div class="modal fade" tabindex="-1" role="dialog">
  <div class="modal-dialog" role="document">
    <div class="modal-content">
      <div class="modal-header">
        <button type="button" class="close" data-dismiss="modal" aria-label="Close"><span aria-
hidden="true">&times;</span></button>
        <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 -->
```

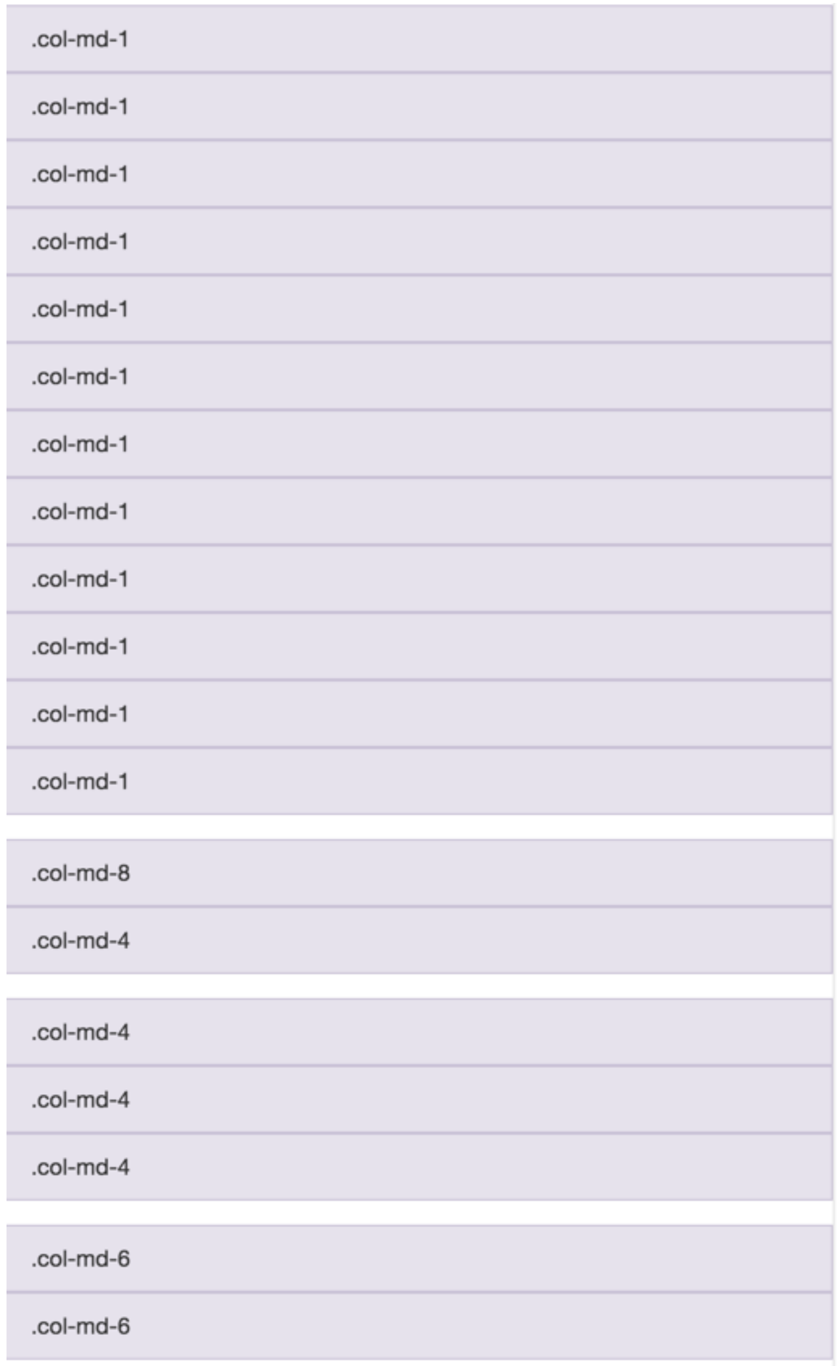
Bootstrap Grid Layout

- 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





```
<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>
```

http://getbootstrap.com/css/

Exercise: Build Tab Panes

- Requirements
  - Click on tab to switch between two visible divs
- Allowed technologies
  - HTML, CSS, ES6, jQuery
- Not allowed
  - Bootstrap (it already does this)

What’s next?

- Organizing code in web apps
- Interacting with remote hosts over HTTP
- Asyncrhonous programming
- And more!