Dynamic Webpages

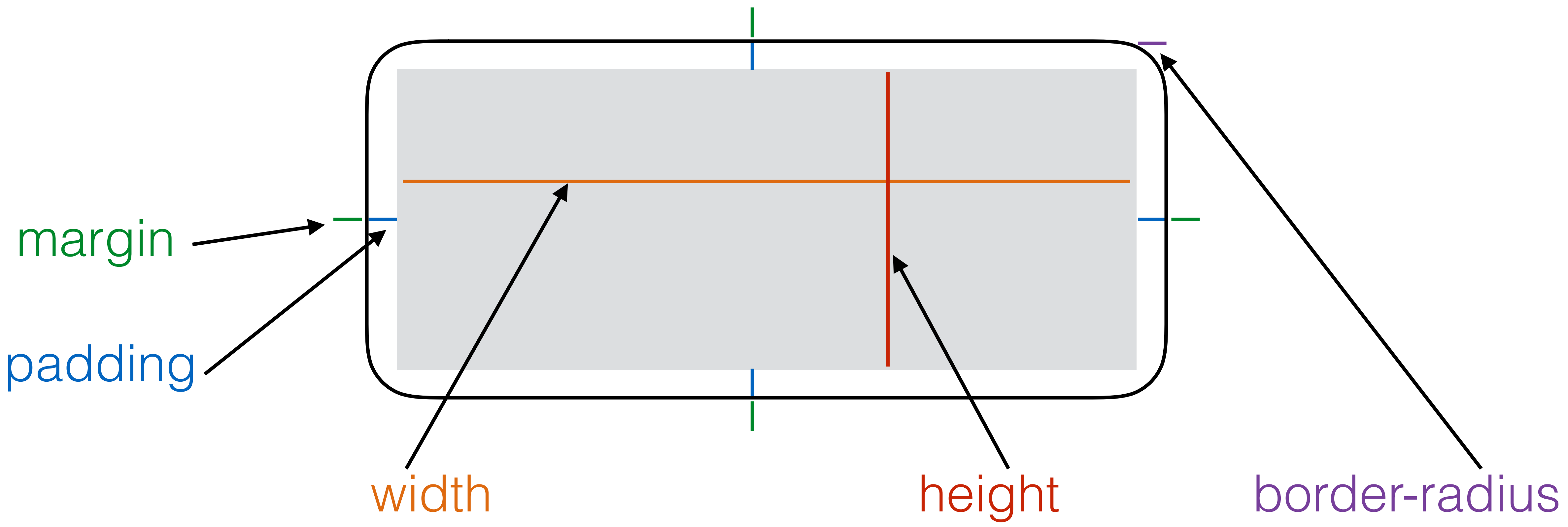
SWE 432, Fall 2017

Design and Implementation of Software for the Web

Today

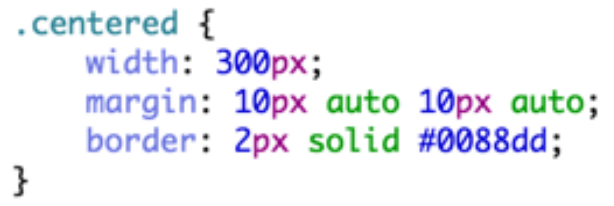
- How to layout elements using CSS
- How to interact with HTML and CSS using frontend JavaScript
- Next time: making and responding to HTTP requests

CSS "Box" Model

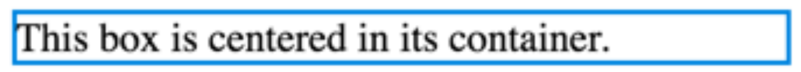


- Boxes, by default, are sized *just* large enough to fit their contents.
- Can specify sizes using px or %
    - % values are relative to the container dimensions
- margin: 10px 5px 10px 5px; (clockwise order - [top] [right] [bottom] [left])
- border: 3px dotted #0088dd; ([width] [style] [color])
    - 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.

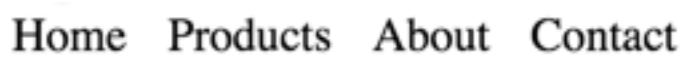
- How do you center an element inside a container?
- Step 1: Must first ensure that element is *narrower* than container.
    - By default, element will expand to fill entire container.
    - So must usually explicitly set width for element.
- Step 2: Use *auto* value for left and right to create equal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

Visibility and layout

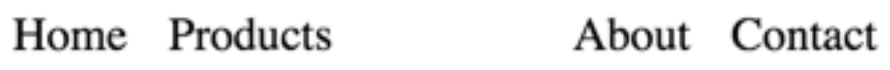
- Can force elements to be inline or block element.
    - display: inline
    - display: block
- Can cause element to not be laid out or take up any space
    - display: none
    - *Very* useful for content that is dynamically added and removed.
- Can cause boxes to be invisible, but still take up space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

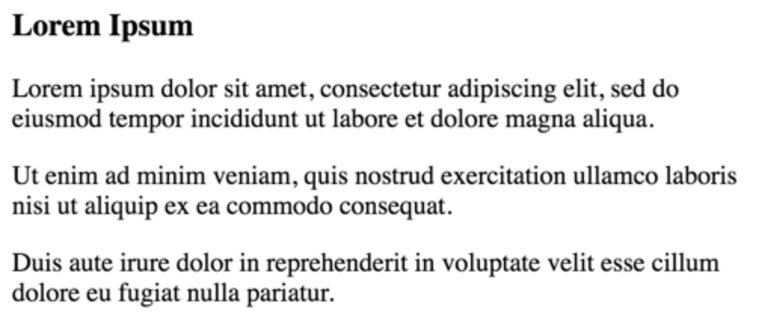

Home   Products   About   Contact

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


Home   Products   About   Contact

Positioning schemes

Normal flow (default)



Lorem Ipsum

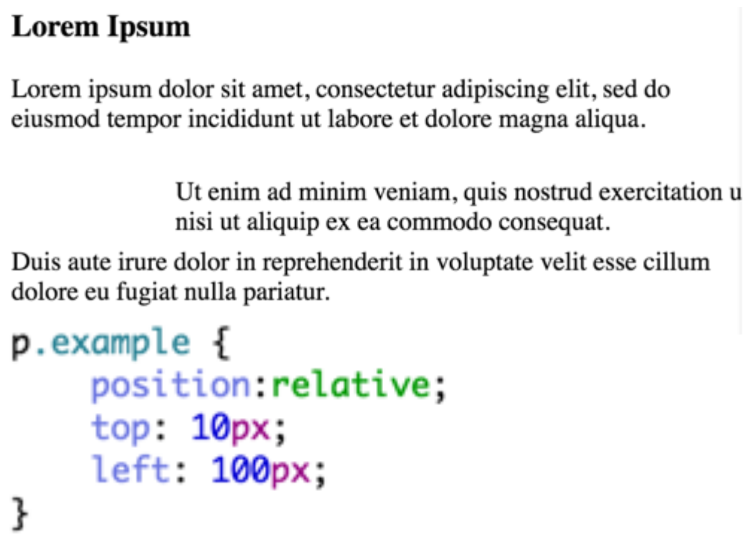
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

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.

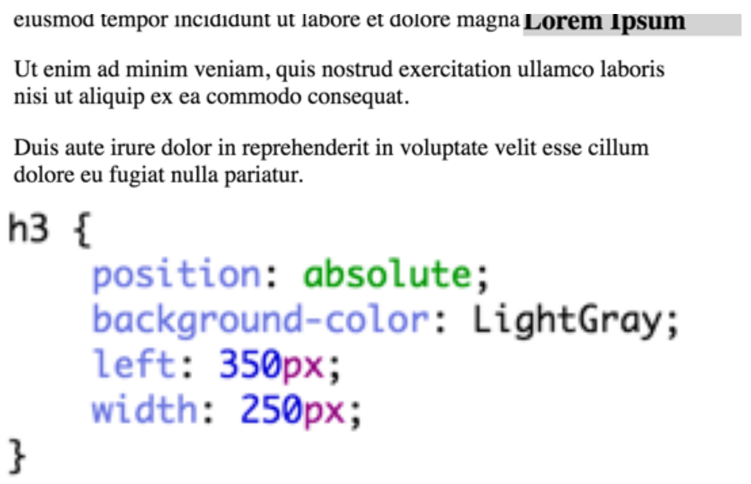
Ut enim ad minim veniam, quis nostrud exercitation u
nisi ut aliquip ex ea commodo consequat.

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**

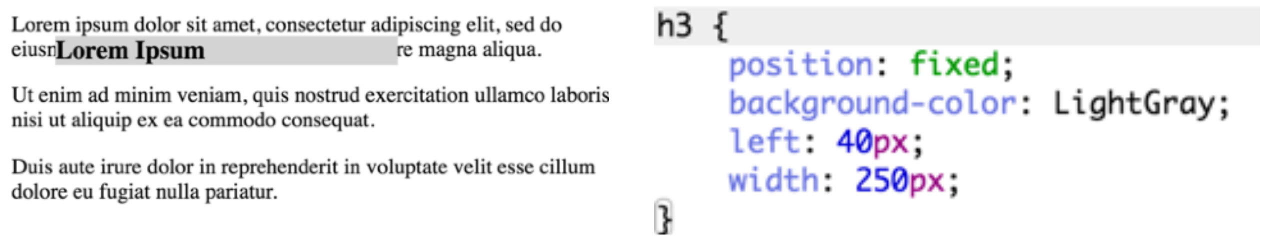
Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

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
eiusn**Lorem Ipsum** re magna aliqua.

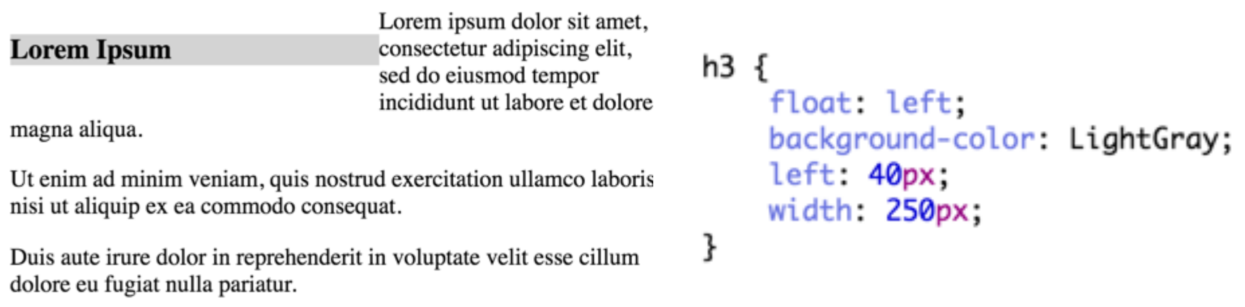
Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

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.

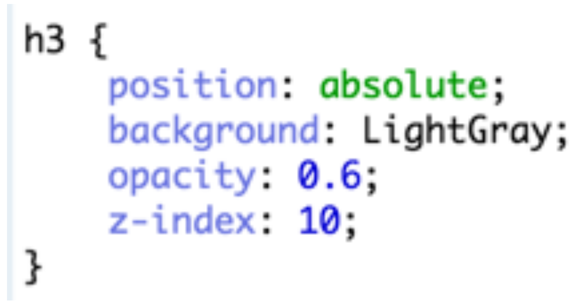
Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

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

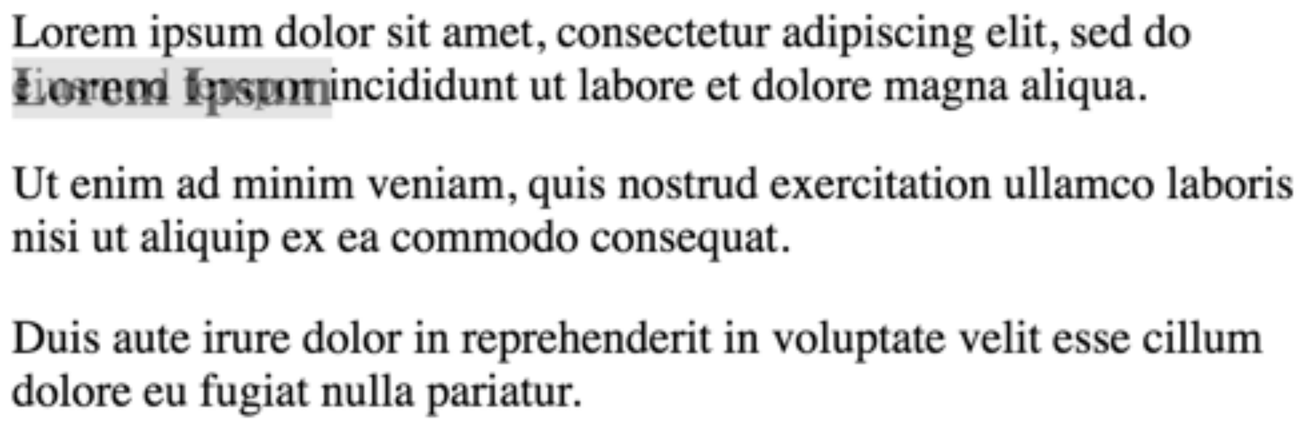
```
h3 {
    float: left;
    background-color: LightGray;
    left: 40px;
    width: 250px;
}
```

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 Lorem Ipsum incididunt ut labore et dolore magna aliqua.

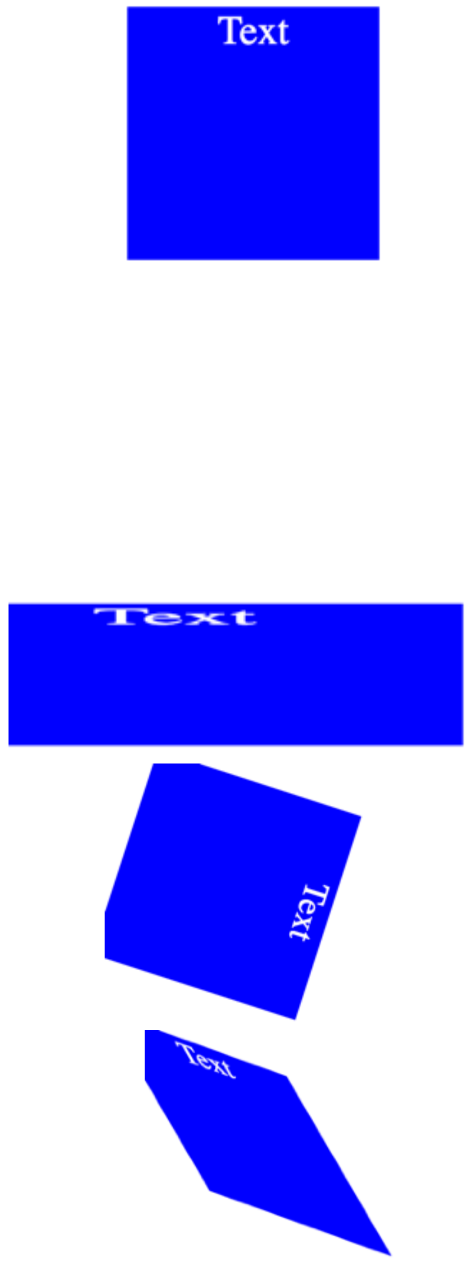
Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

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

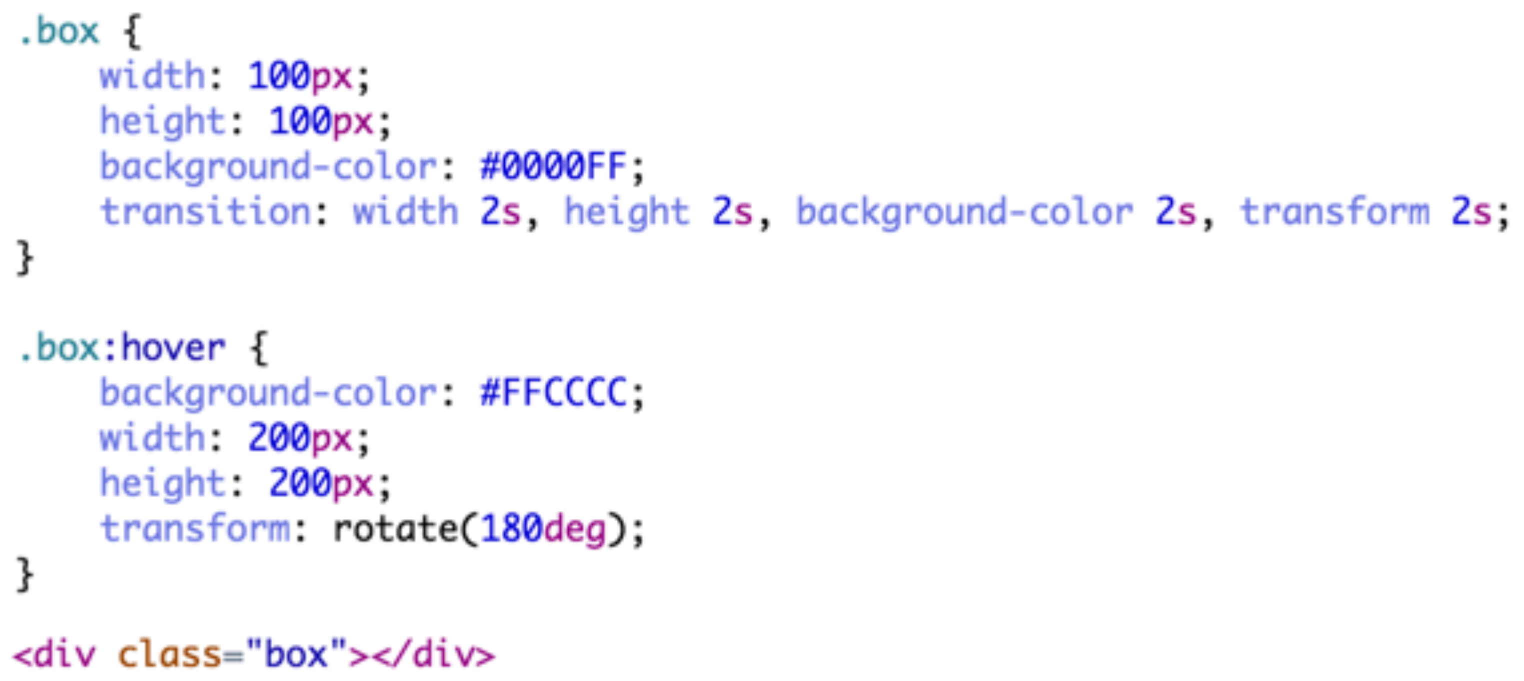
Transform - examples

```
.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>
```

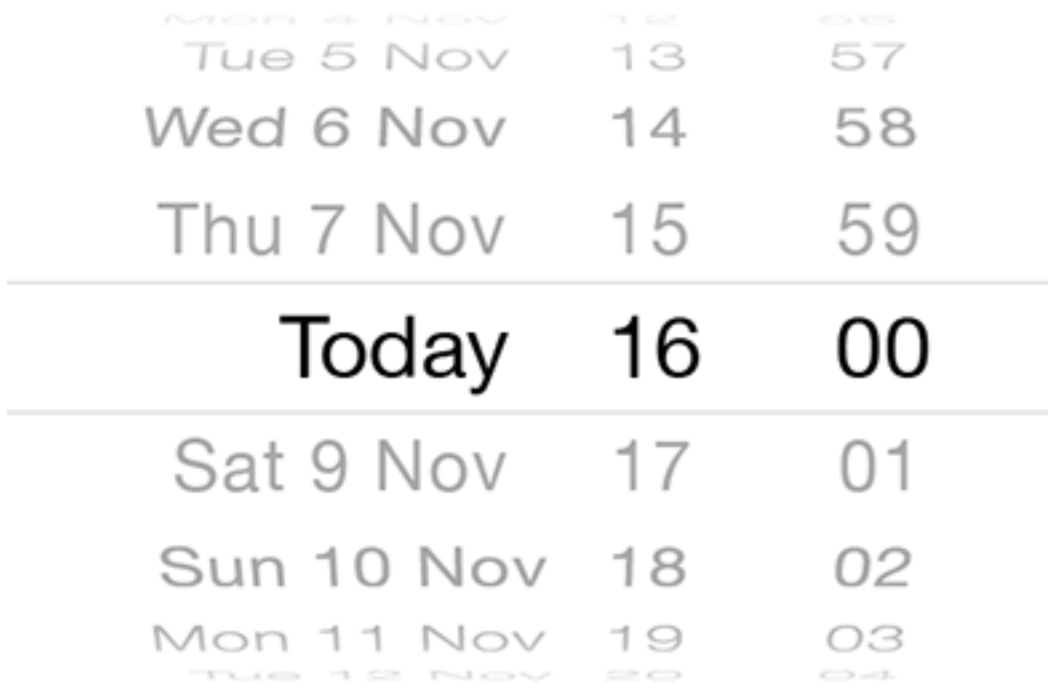
- transition: [property time], …, [property time]
  - When new class is applied, specifies the time it will take for each property to change
  - Can use *all* to select all changed properties

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.

Activity: Build a simple homepage

- In groups of 2
- Build a simple static homepage
    - Should have
        - A title
        - Tags: <table><div><span><a>
        - Use CSS selectors to apply styles

Deployment: serving static content from Node

```
const express = require('express');
const app = express();

app.use(express.static('public'));

app.listen(3000, function () {});
```

- Usually have specific directory where static content is located
  - ONLY want content in the folder to be directly visible to clients
  - Security vulnerability to enable clients to download server side scripts, as it makes it possible to build targeted attacks
  - Directory can be called anything. Often called public or client

Demo: Hello world static content

Frontend JavaScript

- Static page
  - Completely described by HTML & CSS
  - May have interactivity (e.g., CSS transforms, hover pseudo-classes)
  - But described in HTML & CSS
- Dynamic page
  - Adds interactivity, updating HTML based on user interactions

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 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

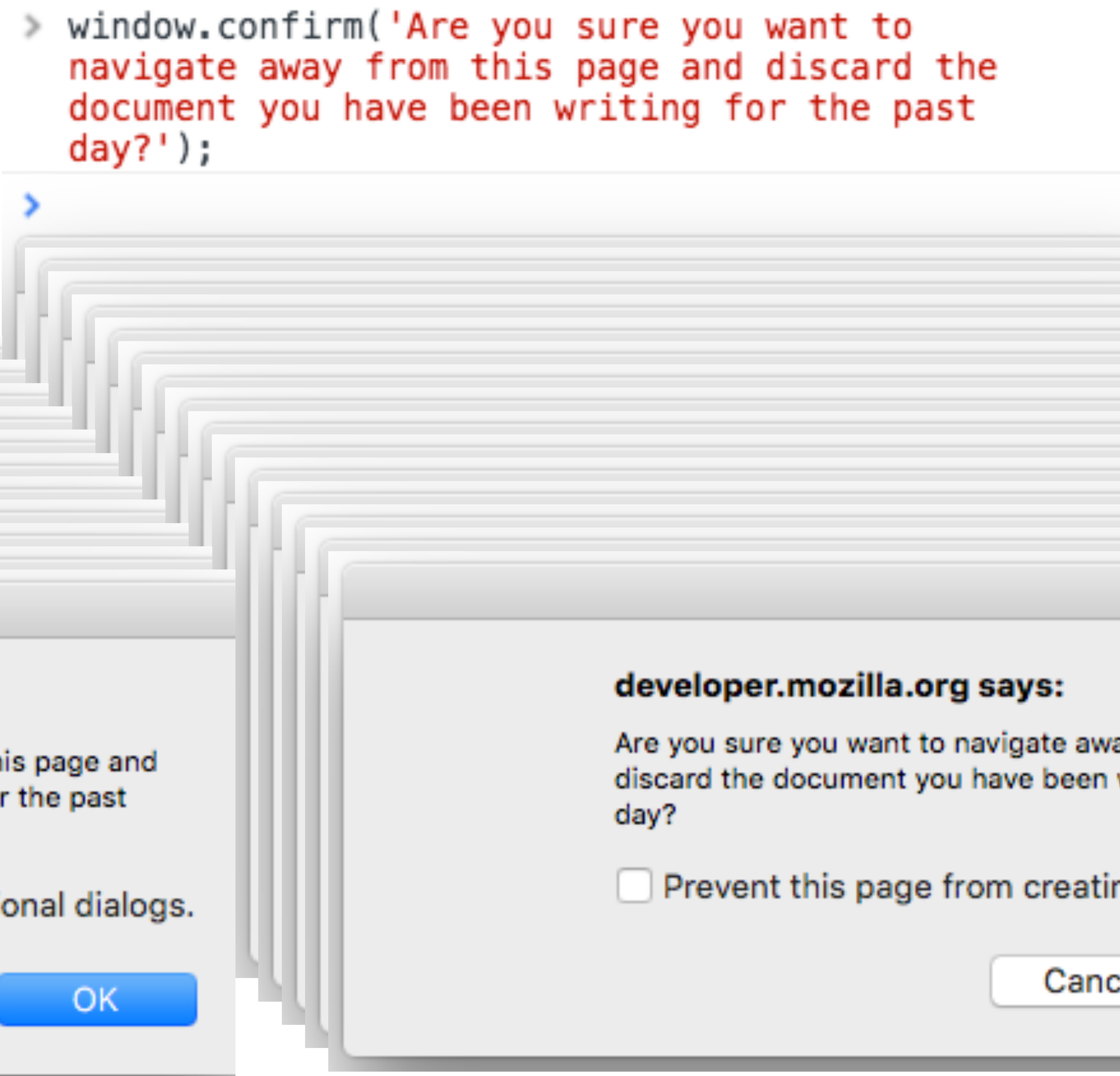
Reference and tutorials
https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model

Global DOM objects

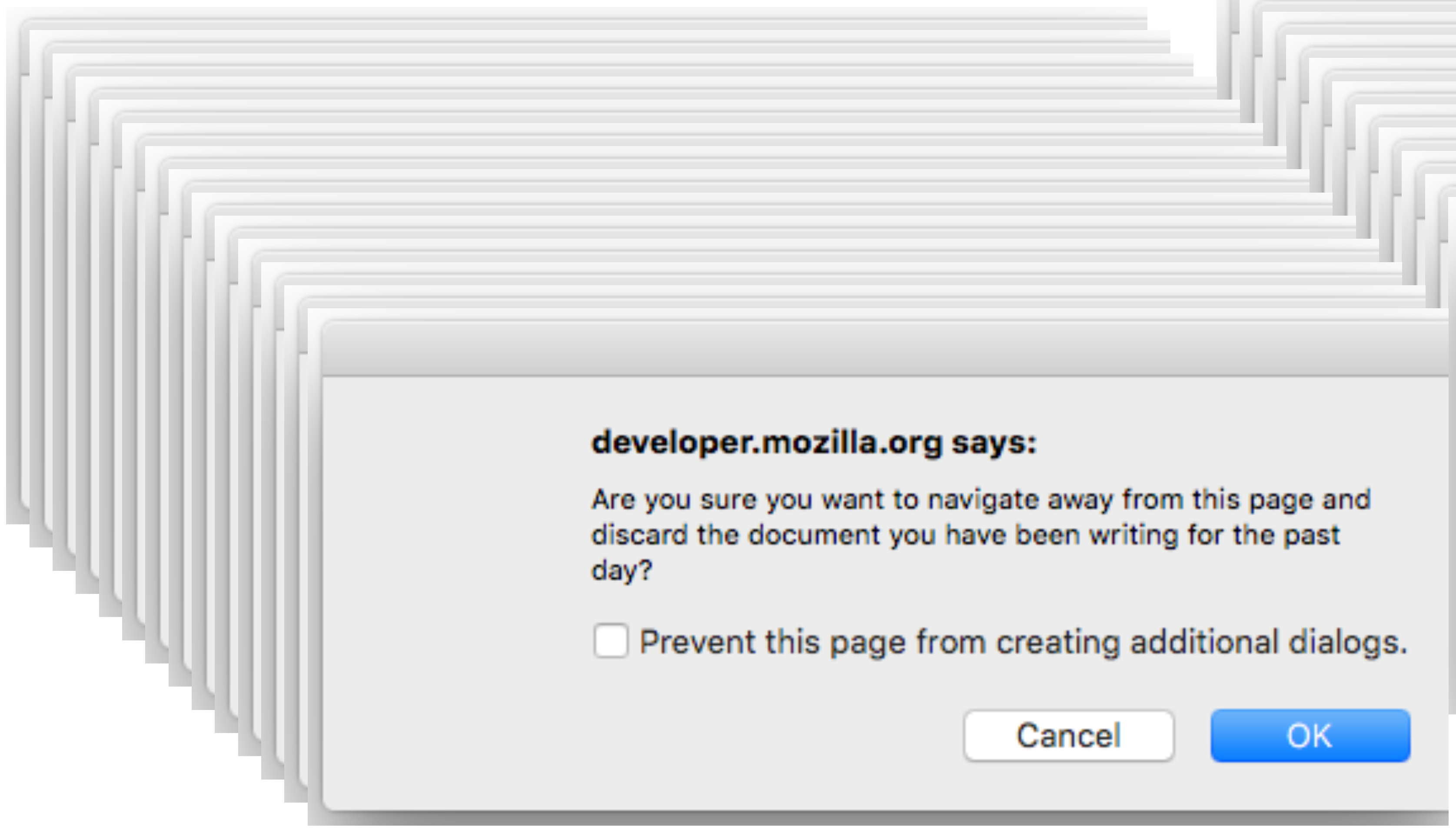
- window - the browser window
  - Has properties for following objects (e.g., window.document)
  - Or can refer to them directly (e.g., document)
- document - the current web page
- history - the list of pages the user has visited previously
- location - URL of current web page
- navigator - web browser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

Working with popups

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



```
> 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.
Cancel
OK

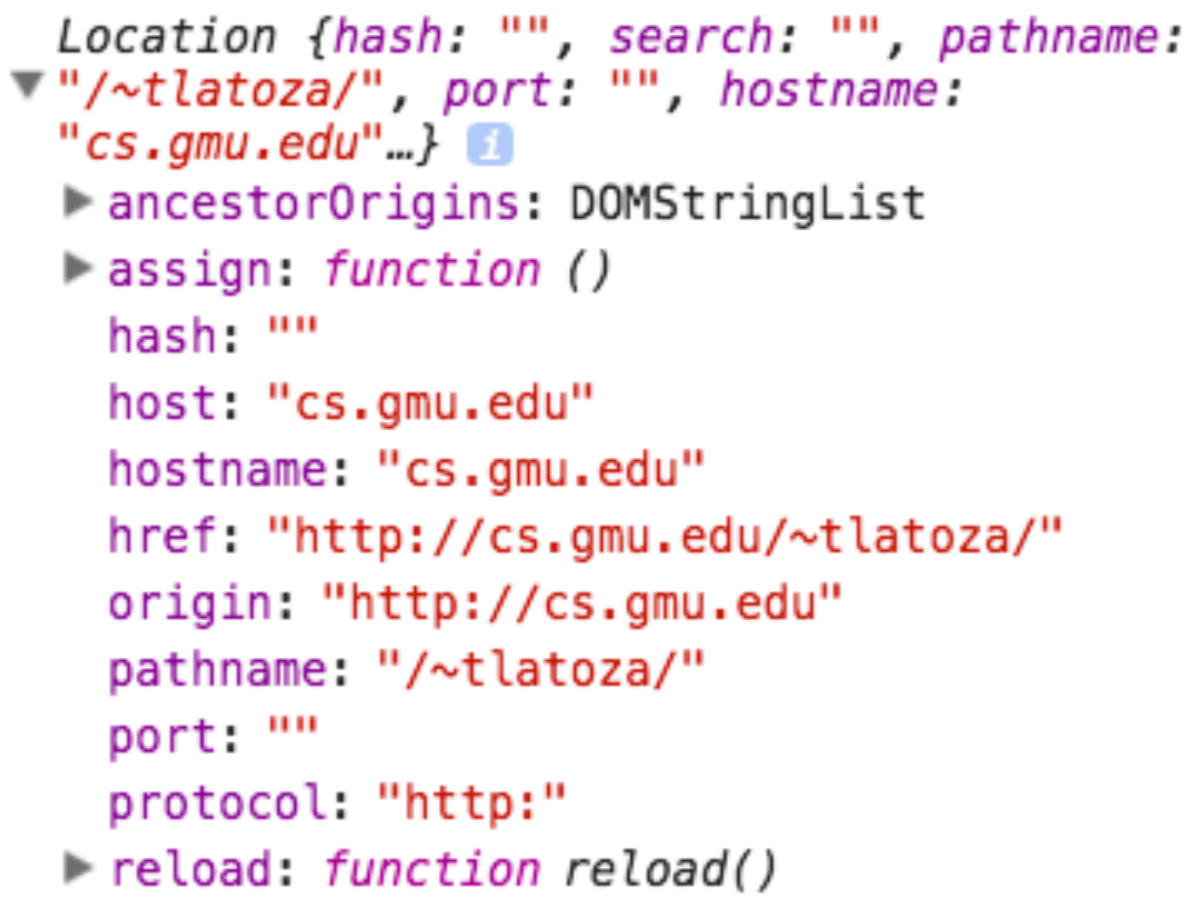
developer.mozilla.org says:

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

Prevent this page from creatir
Canc

Working with location

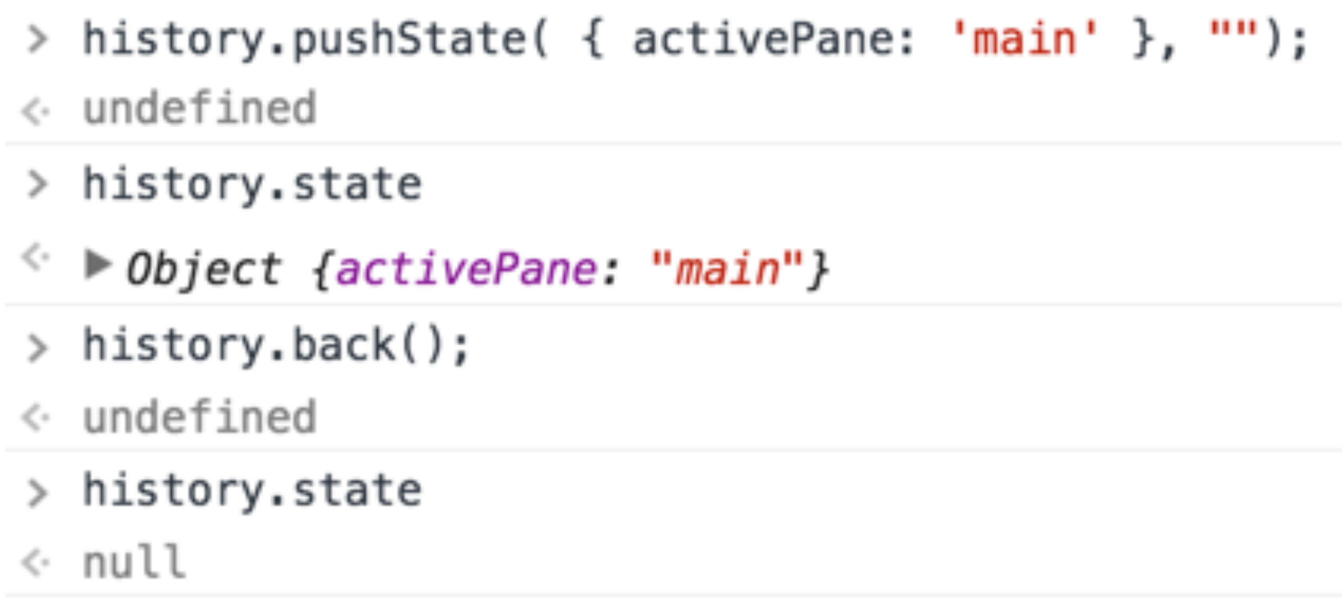
- Some properties
  - location.href - full URL of current location
  - location.protocol - protocol being used
  - location.host - hostname
  - location.port
  - location.pathname
- Can navigate to new page by updating the current location
  - location.href = ‘[new URL]’;



```
▼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()
```

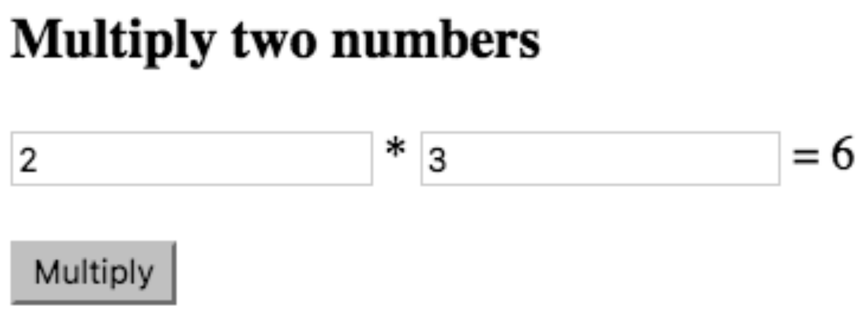
Traveling through history

- 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



```
<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>
```

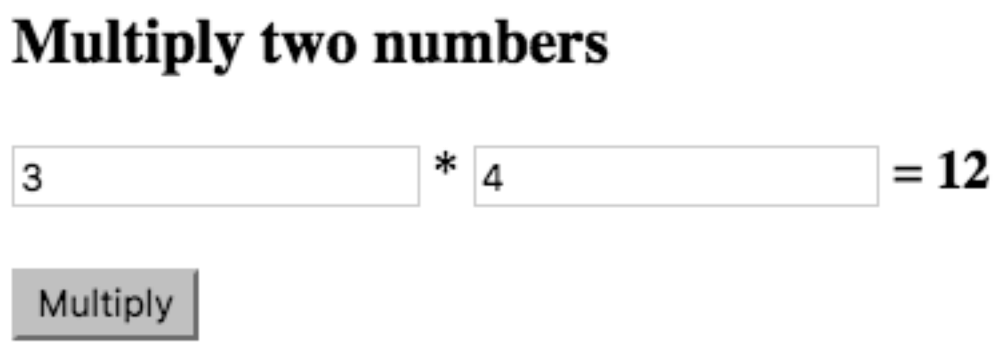
```
document.getElementById('compute')
        .addEventListener("click", multiply);
function multiply()
{
    var x = document.getElementById('num1').value;
    var y = document.getElementById('num2').value;
    var productElem = document.getElementById('product');
    productElem.innerHTML = x * y;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

“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



```
<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

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

Loading pages

- What is the output of the following?

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

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

Loading pages

- Code in script tags will run in the order in which it is contained in the page
- Solution: should put script tags at the bottom of the body after elements in the document.

The Event Loop

- Remember that JS is **event-driven**

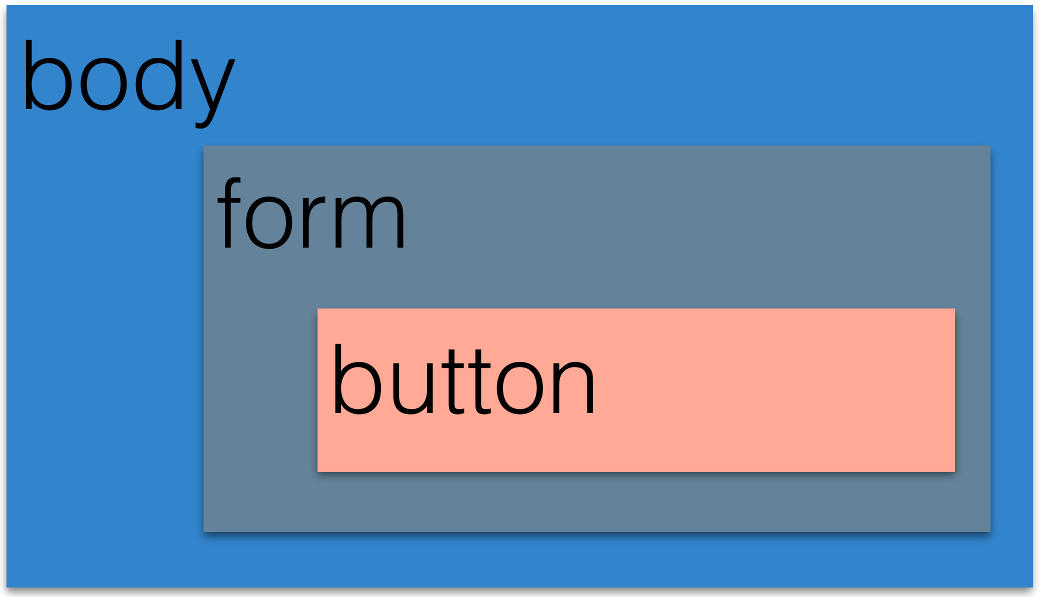
```
$(window).on('hashchange', function () {
  show(location.hash);
});
```

- Event loop is responsible for dispatching events when they occur
- Main thread for event loop:

```
while(queue.waitForMessage()){
  queue.processNextMessage();
}
```

Event Dispatching

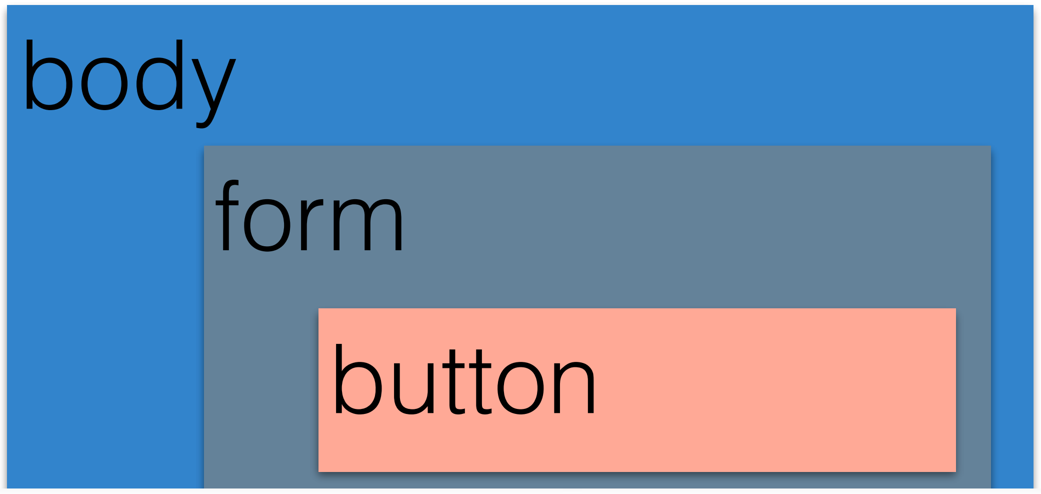
- Each event target can have (0…n) listeners registered for any given event type, called in arbitrary order
- What happens with nested elements?



```
Listener1: body onClick
Listener2: form onClick
Listener3: button onClick
```

What happens when we click in **`button`**?

Event Bubbling



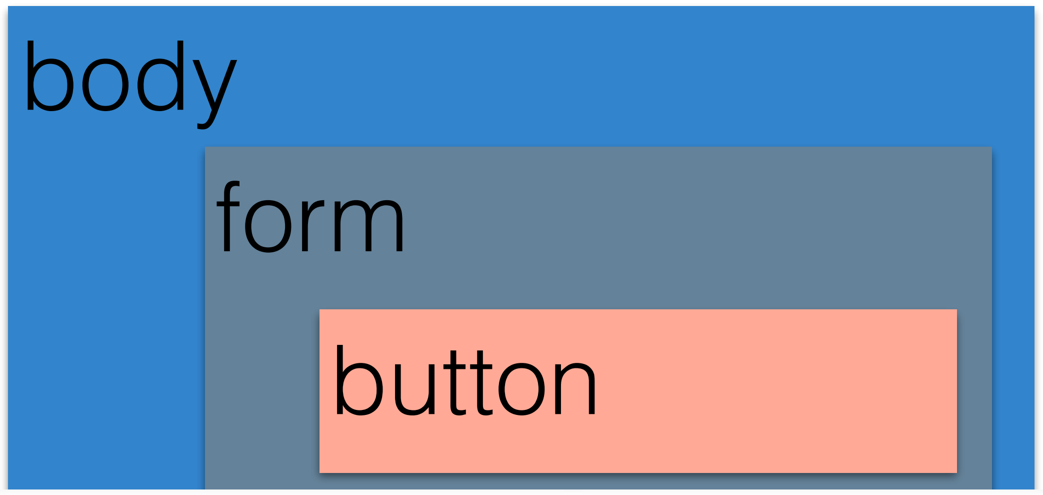
What happens when we click in **`button`**?

```
Listener1: body onClick
Listener2: form onClick
Listener3: button onClick
```

Called →

This is the default behavior

Event Capturing



What happens when we click in **`button`**?

Called →

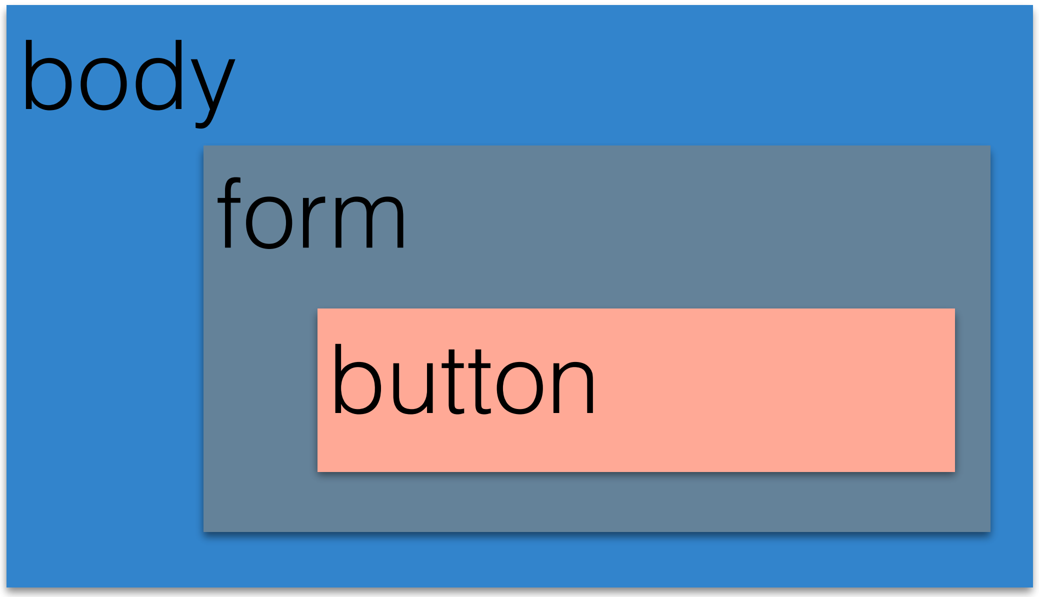
```
Listener1: body onClick
Listener2: form onClick
Listener3: button onClick
```

Enable event capturing when you register your listener:

```
element.addListener('click', myListener, true);
```

Event Dispatching

- An individual listener can stop bubbling/capturing by calling
- `event.stopPropagation();`
  - Assuming that `event` is the name of your handler's parameter
- Or in jQuery, simply `return false;`



```
Listener1: body onClick
Listener2: form onClick
Listener3: button onClick
```

Activity: Build an interactive page

- In groups of 2 or 3
  - Build a 4 function calculator page that lets users add, delete, multiply divide