events & handlers

For now, all rights reserved, Daniel Jackson, 2013.
purposes of today’s class

after today’s class, you should understand

• the handler idiom, and how DOM and events interact
• how closures are used as handlers to process events
• the asynchronous processing model for events
DOM overview
what is the DOM?

an API for manipulating web pages
› elements are mutable objects

what you can do
› add and delete elements
› change properties of elements
› attach event listeners to elements

what you can’t do
› modify CSS rules: no CSS rule object in DOM
› but can modify CSS properties of elements
The new, faster MacBook Air

The ultimate everyday notebook. Now up to 2.5x faster, featuring high-speed Thunderbolt I/O and OS X Lion.
the standard DOM

- standard DOM API
  - standardized by W3C
  - native to all browsers

Gecko (Mozilla) documentation
The nice thing about standards is that you have so many to choose from.

— Tom Knight
jQuery

jQuery library
› remove platform dependences
› smooth over some non-uniformities
› 6.170 will use jQuery in place of standard DOM API

jQuery elements
› jQuery wraps DOM elements
› provide their own DOM manipulation methods

how to use jQuery
› include library in page header
› then just use jQuery functions ($) in any JS

```html
<script src="http://code.jquery.com/jquery-1.11.3.min.js"></script>
```

using jQuery CDN

Point out that DOM API is becoming better (e.g. no need to do string concatenation on className anymore).
the event queue
reactive programming: two idioms

**stream idiom** (e.g. used in old Win32 applications)
› sequential thread of control
› one big, infinite "event loop"
› each time: reads input, acts on input

```javascript
while (true) {
  var e = getInput();
  if (e instanceof click) {
    // handle click event
    var t = e.target();
    // do something with e, t
  } else if (e instanceof drag) {
    // handle drag event
  }
}
```

(this idiom is **not** explicitly used in web programming)
reactive programming: two idioms

event idiom

› code registers functions with elements for particular events
› these functions are called “handlers” (a kind of “callback” function)
   May also hear the term “listener”, although this is technically a DOM API interface
› when event occurs, all callbacks are automatically invoked

```html
<head>
  <script>
    $(function () {
      var b = $('button');
      b.click(function () {alert("Pressed!");});
    });
  </script>
</head>
<body>
<button id=button>Press me!</button>
</body>
```

› how many handlers here? (clue: more than one)

http://codepen.io/eirikbakke/pen/jbrMJa?editors=101
behind the scenes

event idiom uses stream idiom in background
› but a generic event loop
› reads event, then runs all the handlers for that event

```javascript
while (true) {
    var e = getInput();
    // execute handlers for event e
    e.handlers.each(function (f) {f(e);});
}
```
For the second part of this example, I add “for (var i = 0; i < 5000000000; i++);” in the button event handler (counting to 5 billion). Point out that the button stops working properly, too. Then put the counting loop in the interval handler instead. Note that the _entire_ page freezes up, including the CodePen editor. Show the CPU usage too (although it would be 100% even if we had concurrency).
"concurrency" model

- single thread
- event loop: run event at front of queue
- events added by: user actions, server responses, timers

http://dev.w3.org/html5/spec/Overview.html#event-loop
http://dev.w3.org/2006/webapi/DOM-Level-3-Events/html/DOM3-Events.html

Note: this slide might not be entirely accurate; I think timers are handled specially by the event loop.
“concurrency” model

key fact
› each event runs to completion before next event

good news
› no need to worry about mutual exclusion

bad news
› long-running event freezes the UI
› timers and timeouts may run late
› server responses may come back long after initial AJAX requests (covered later), when context has changed

really no concurrency problem?
events and handlers
events & handlers

“The by-design purpose of JavaScript was to make the monkey dance when you moused over it.”
—Eric Lippert
attaching handlers in jQuery

execute handler when document DOM is ready
› $(document).ready(handler) or just $(handler)

execute handler when element is clicked
› element.click(handler)
› element.bind(‘click’, handler)

execute handler depending on event type
› element.bind({keydown: handler1, keyup: handler2})

can also trigger event manually
› element.trigger(‘myevent’)

DOM events, along with values from input elements, are the main way to get user input in JavaScript.
events and input elements

http://codepen.io/eirikbakke/pen/meEOYp?editors=101
handler uses event property

```html
<head>
  <script>
    $(function () {
      $(document).bind('mousemove',
        function(e){
          $('#log').text("x: " + e.pageX
            + ", y: " + e.pageY));
        })
    </script>
</head>
<body>
  <div id=log></div>
</body>
```

listener uses $(this)

```html
<ul>
  <li>List item 1.</li>
  <li>List item 2.</li>
  <li>List item 3.</li>
</ul>
<div id="log"></div>
```

```js
$\("li\"\).click(function (e) {
  $(this).append($("<span> Hello!</span>"));
});
```
handler uses “global” variables

```html
<head>
<script>
(function () {
    var ds = $('#dollars');
    var es = $('#euros');
    var EUROS_PER_DOLLAR = 0.755;
    var convert = function (x, rate) {
        return (x * rate).toFixed(2);};
    ds.change(function () {
        es.val(convert(ds.val(), EUROS_PER_DOLLAR));});
    es.change(function () {
        ds.val(convert(es.val(), 1/EUROS_PER_DOLLAR));});
});
</script>
</head>
<body>
    Dollars:<input id=dollars></input><br>
    Euros:<input id=euros></input>
</body>

http://codepen.io/eirikbakke/pen/YyWGoX?editors=101
Not truly a “global” variable, though, just declared at the level of the outer $(window).ready handler function level...
```
After first showing how this works, rewrite to declare i using “var” instead of taking the extra function argument.
<head>
<script>
$(function () {
    var fromTo = function (from, to, f) {
        for (var i = from; i <= to; i = i+1) f(i);
    };

    fromTo(0,3, function (i) {
        var bi = $('#<button>').text(i);
        $('body').append(bi);
        bi.click(function () {
            $('#log').text('Pressed ' + i);
        });
    });
});
</script>
</head>
<body>
<div id=log>What did you press?</div>
</body>

http://codepen.io/eirikbakke/pen/EVyNYM?editors=101
doesn't have the problem of the next slide
what’s wrong with this?

```html
<head>
<script src="jquery-1.7.2.min.js"></script>
<script>
$(function() {
    for (var i = 0; i <= 3; i += 1) {
        var bi =$("<button>".vaadin);
        bi.text(i);
        $('body').append(bi);
        bi.click(function() {
            $('#log').text("Pressed " + i);
        });
    }
});
</script>
</head>
<body>
  <div id=log>What did you press?</div>
</body>
```

› http://codepen.io/eirikbakke/pen/vNKyYL?editors=101
› same as multipliers example in live JS slides closure exercises
› http://people.csail.mit.edu/dnj/teaching/6170/javascript-live/modules/functions/slides.html#/8/4
Fix: http://codepen.io/eirikbakke/pen/meEOyr?editors=101
Inlined fix: http://codepen.io/eirikbakke/pen/BozQyZ?editors=101
For the inlined fix, point out shadowing of “i” (could call inner one “i2” instead to illustrate effect).
more DOM events...
exercise: bunny jump

Template code at [http://shoutkey.com/watermelon](http://shoutkey.com/watermelon)

Objective: Make the bunny jump to a random other `<li>` when element you click on it.

Hint: `$('<li>').eq(Math.floor(Math.random()*5))` selects a random `<li>` element.

Point out what happens if bunnyElm.remove() is done before it is appended (jQuery removes the listener) vs. bunnyElm.detach().
event propagation
Event propagation

- Netscape (olden days): capturing only
- < IE9: bubbling only
- W3C, modern browsers: support both (developer selects when adding listener using the DOM API)

- jQuery: bubbling only

Capture, bubble, target: call capture handler e.g. when going down
IE8 doesn’t support capture phase, so this is probably why JQ only does bubble and target
event propagation — capturing

Not used in jQuery

(illustration from https://signalvnoise.com/posts/3137-using-event-capturing-to-improve-basecamp-page-load-times)
event propagation — bubbling

Always used in jQuery

(Illustration from https://signalvnoise.com/posts/3137-using-event-capturing-to-improve-basecamp-page-load-times)
event bubbling example

http://codepen.io/eirikbakke/pen/jbrVmp
Click in the padding area as well
Add e.stopPropagation()
Show the effect of stopPropagation() vs. preventDefault() in someLink. Also show jQuery's shortcut for invoking both by returning false. Safer to use preventDefault at beginning of function than returning false, in case of exceptions.
**event delegation**

- “Event delegation allows us to attach a single event listener, to a parent element, that will fire for all descendants matching a selector, whether those descendants exist now or are added in the future.”
  
  [http://learn.jquery.com/events/event-delegation/](http://learn.jquery.com/events/event-delegation/)

- Common technique that takes advantage of event bubbling.

- Supported in jQuery by the “on” function.

- Note terminology:
  
  - event *propagation* (intrinsic capture/bubble behavior of DOM) vs.
  - event *delegation* (technique that takes advantage of bubbling)

---

[http://learn.jquery.com/events/event-delegation/](http://learn.jquery.com/events/event-delegation/)

event delegation

What's wrong with the Item #5?

http://codepen.io/eirikbakke/pen/GpqNQv?editors=101 (as in previous slide)
event delegation
(Fixed it!)

Put the event handler on the #list instead, giving jQuery’s “on” the “a” selector as a second argument. Note that $this still refers to the <a> element.
Conclusion

• DOM events, along with values from input elements, are the main way to get user input in JavaScript.

• JavaScript is *single-threaded*
  • Even though user actions, server responses, or timers may trigger multiple events at nearly the same time, the browser’s *event loop* will call each handler in turn, and never at the same time.

• When event handlers are attached at multiple levels in the DOM, the rules of *event propagation* decide which ones get called first (and which one may override others).