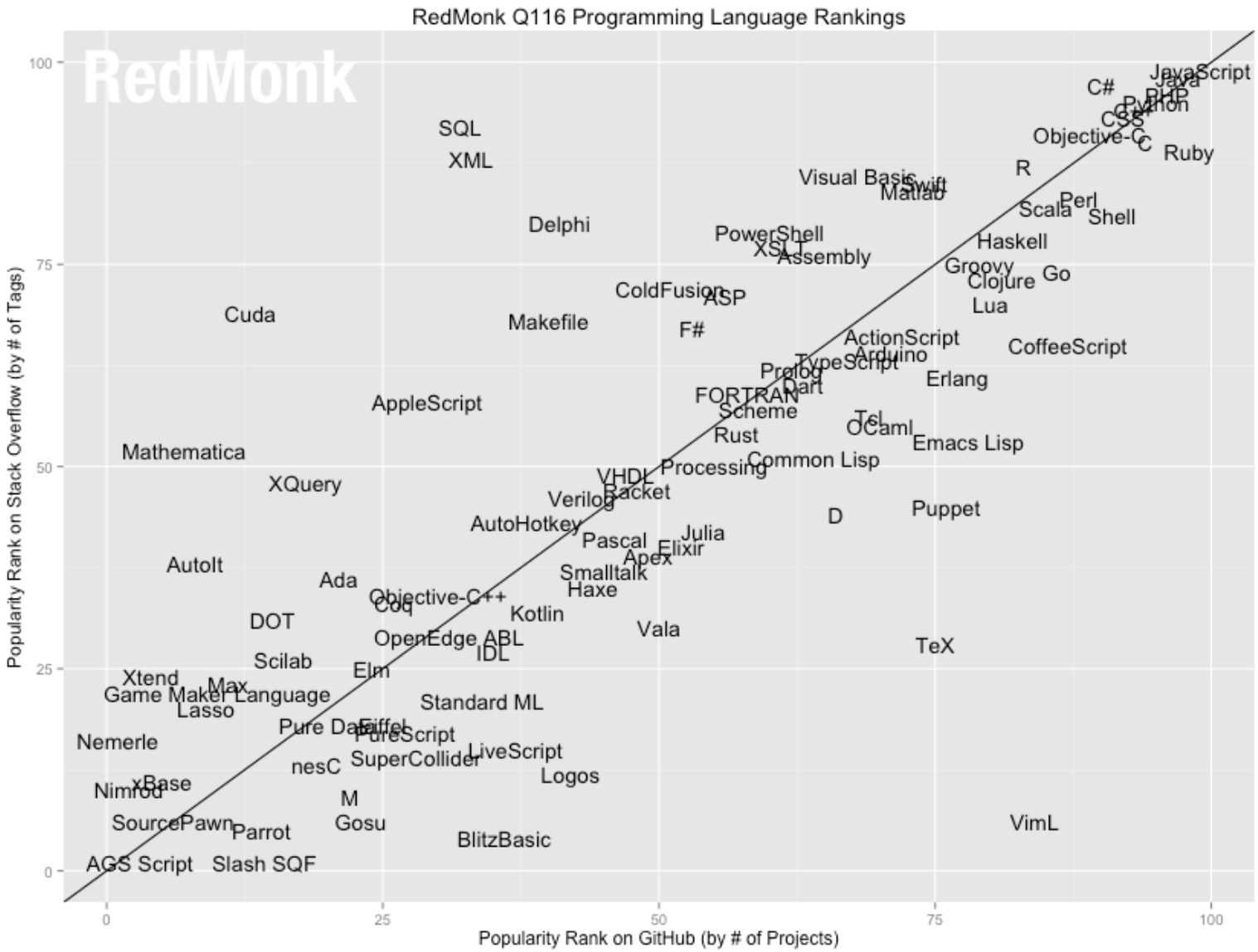


1 JavaScript
2 Java
3 PHP
4 Python
5 C#
5 C++
5 Ruby
8 CSS
9 C
10 Objective-C
11 Shell
12 Perl
13 R
14 Scala
15 Go
15 Haskell
17 Swift
18 Matlab
19 Clojure
19 Groovy
19 Visual Basic

http://redmonk.com/sogrady/2016/02/19/language-rankings-1-16/

CSC 470 – Section 3

Topics in Computer Science: Advanced Browser Technologies

Mark F. Russo, Ph.D.

Spring 2016

Lecture 5

Eloquent JavaScript: Chapters 14

Handling Events

- JavaScript listener functions may be executed in response to certain events, including events raised by the user interacting with the browser
- An event target may be an Element in a document, the Document itself, a Window, and most of the multitude of other DOM objects
- The specific events raised/dispatched depends upon the Element type

Caution

- The browser executes all JavaScript on a single thread (unless specifically requested to use a background thread)
- A function invoked in response to an event will suspend DOM/UI updates until the function completes

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

Browser Event Categories

- Mouse Events
  - `click, mousedown, mouseup, …`
- Touch events
  - `touchstart, touchcancel, touchmove, …`
- Keyboard Events
  - `keypress, keydown, keyup, …`
- Progress Events
  - `loadstart, loadend, load, error, abort, …`
- Form Element Interaction
  - `input, change, focus, blur…`
- Many others…

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

Handling Events

In order to respond to an event, a listening function must be attached.

Registers the specified `listener` function on the EventTarget Object it's called on, in response to the event `type`.

- `anEventTarget.addEventListener(type, listener)`

Removes the event `listener` for event `type` previously registered with `addEventListener()`.

- `anEventTarget.removeEventListener(type, listener)`

- Dispatches an `event` at the specified EventTarget, invoking the affected EventListeners in the appropriate order.
  - `anEventTarget.dispatchEvent(event)`

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

Handling Events - Properties

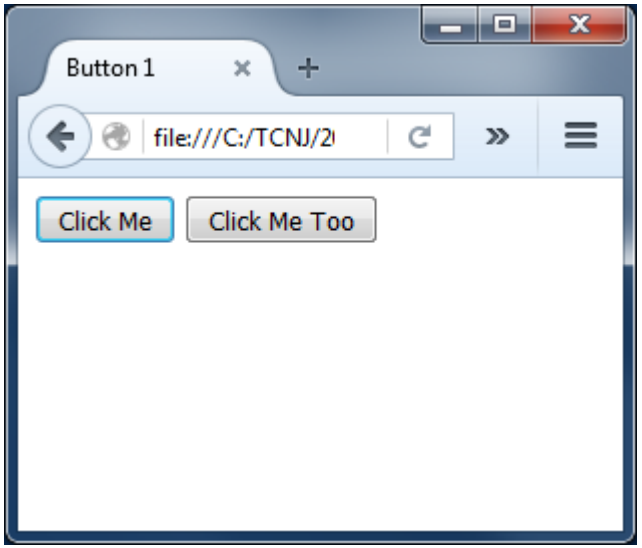
- Alternatively, an event handler may be attached to an Element event by assigning the handler function to a special property
  - Using this approach does not allow multiple handlers to be attached to a single event
- The property may also be set as an HTML attribute of the Element tag

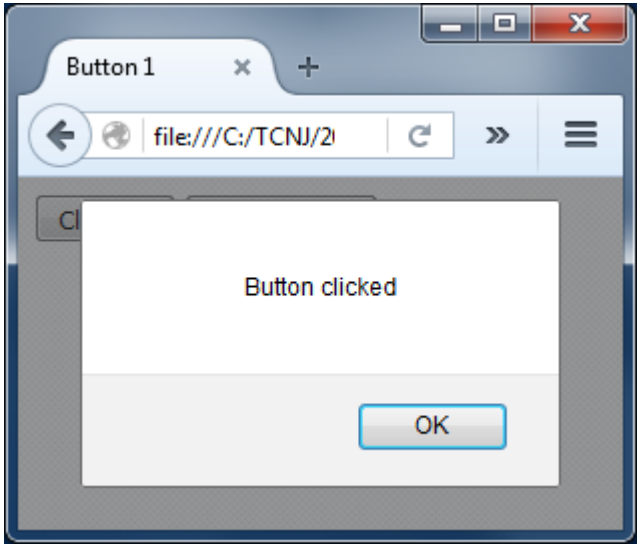
- Examples
  - `onclick, onmousedown, onmouseup, …`
  - `onkeypress, onkeydown, onkeyup, …`
  - `ontouchstart, ontouchcancel, ontouchmove, …`
  - `onload,…`

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

Example: Button Click

```html
<!doctype html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Button 1</title>

    <script type="text/javascript">
      // Button click handler function
      var handleClick = function() {
        alert("Button clicked");
      }
    </script>
  </head>
  <body>
    <!-- event handler assigned in code -->
    <button id="b1">Click Me</button>

    <!-- event handler assigned in HTML attribute -->
    <button onclick="handleClick()">Click Me Too</button>

    <script type="text/javascript">
      var button = document.getElementById('b1');
      button.onclick = handleClick;
    </script>
  </body>
</html>
```





05/button1.html

Example: Button Click

```html
<!doctype html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Button 1</title>

    <script type="text/javascript">
      // Button click handler function
      var handleClick = function() {
        alert("Button clicked");
      }
    </script>
  </head>
  <body>
    <!-- event handler assigned in code -->
    <button id="b1">Click Me</button>

    <!-- event handler assigned in HTML attribute -->
    <button onclick="handleClick()">Click Me Too</button>

    <script type="text/javascript">
      var button = document.getElementById('b1');
      button.onclick = handleClick;
    </script>
  </body>
</html>
```

Event handler function defined before document loads

Event handler function assigned as attribute in <button> tag

Event handler function assigned in code after document loads

No parentheses. What would happen if we added () after function?

05/button1.html

The Event Object

- When an assigned event handling function is invoked, an `Event` object is passed as a single argument
- Different `Event` object types have different properties, appropriate to the event that has occurred

- Example Event object properties include:

  - `target` - the element that dispatched the event
  - `currentTarget` - the element to which the event handler has been attached
  - `type` - a string containing the type of event
  - `clientX` - horizontal coordinate in client at which the event occurred
  - `clientY` - vertical coordinate in client at which the event occur
  - `screenX` - horizontal coordinate of mouse in global (screen) coordinates
  - `screenY` - vertical coordinate of mouse in global (screen) coordinates
  - `button` - which button was pressed on the mouse
  - `key` - the value of a key or keys pressed
  - `code` - the physical key pressed

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

Example: Events 1

In this example we want to …

- Build a table programmatically
- Set dimensions of each cell (`td`) to a fixed `width` and `height`
- Assign the `onclick` property of all `td`s to a function that changes the background color
- Add a `<button>` the clears table cell colors when clicked

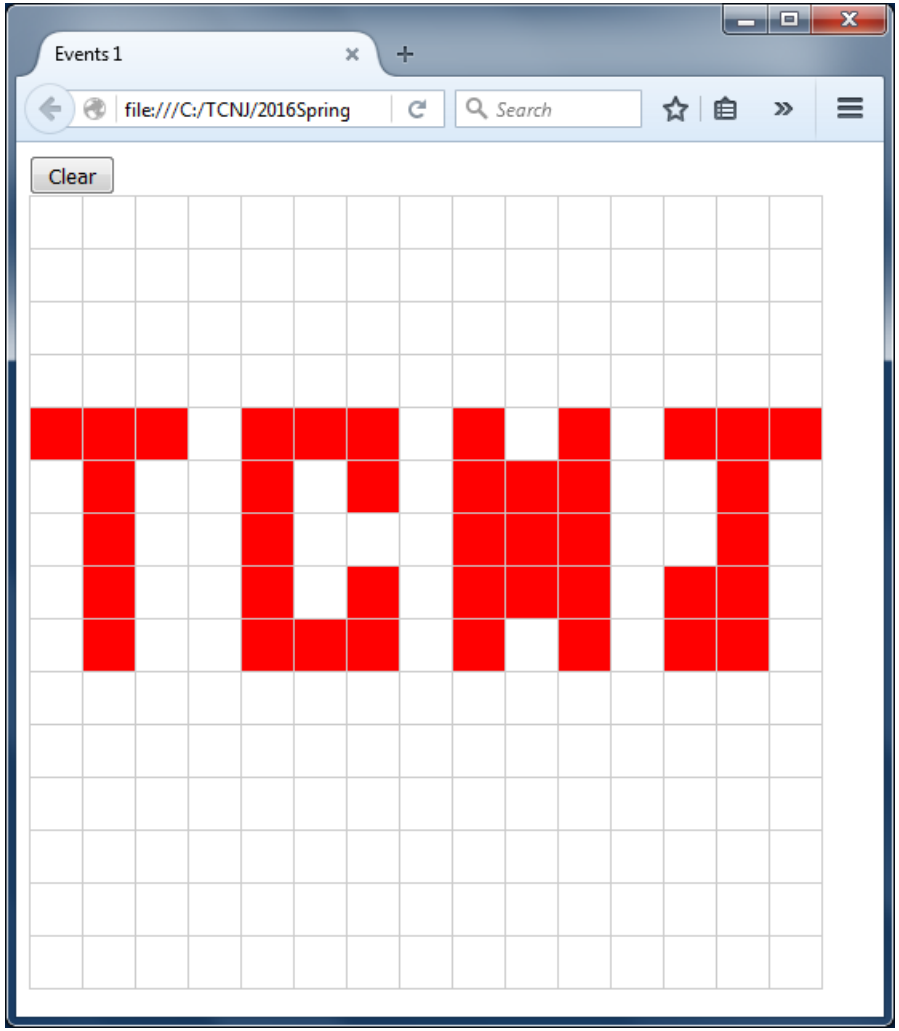
```javascript
// events1.js

// Programmatically add a table to the page
// with cells that change color when clicked
var buildTable = function(nrows, ncols, size) {

  // Create table and tbody elements
  var tbl   = document.createElement('table');
  tbl.style.borderCollapse = 'collapse';
  var tbody = document.createElement('tbody');

  // Create rows
  for (var r=0; r<nrows; r++) {
    var tr = document.createElement('tr');

    // Create cells in each row
    for (var c=0; c<ncols; c++) {
      var td = document.createElement('td');
      td.style.width  = size;
      td.style.height = size;
      td.style.border = '1px solid #ccc';

      // Add element click event handler
      td.onclick = tdClick;

      // Add td to tr
      tr.appendChild(td);
    }
    // Add tr to tbody
    tbody.appendChild(tr);
  }
  tbl.appendChild(tbody);
  // Add table to the document.body
  document.body.appendChild(tbl);
};
```

```javascript
// td click event handler
var tdClick = function(ev) {
  ev.target.style.backgroundColor = 'red';
};

// Clear all table cell colors
var clearTable = function() {
  var tds = document.querySelectorAll('td');

  // Clear td background colors
  for (var i=0; i<tds.length; i++) {
    tds[i].style.backgroundColor = '';
  }
};
```

```html
<!doctype html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Events 1</title>
    <script type="text/javascript"
            src="events1.js"></script>
  </head>
  <body>
    <button onclick="clearTable()">
                          Clear</button>
    <!-- Init table -->
    <script type="text/javascript">
      buildTable(15, 15, '30px');
    </script>
  </body>
</html>
```

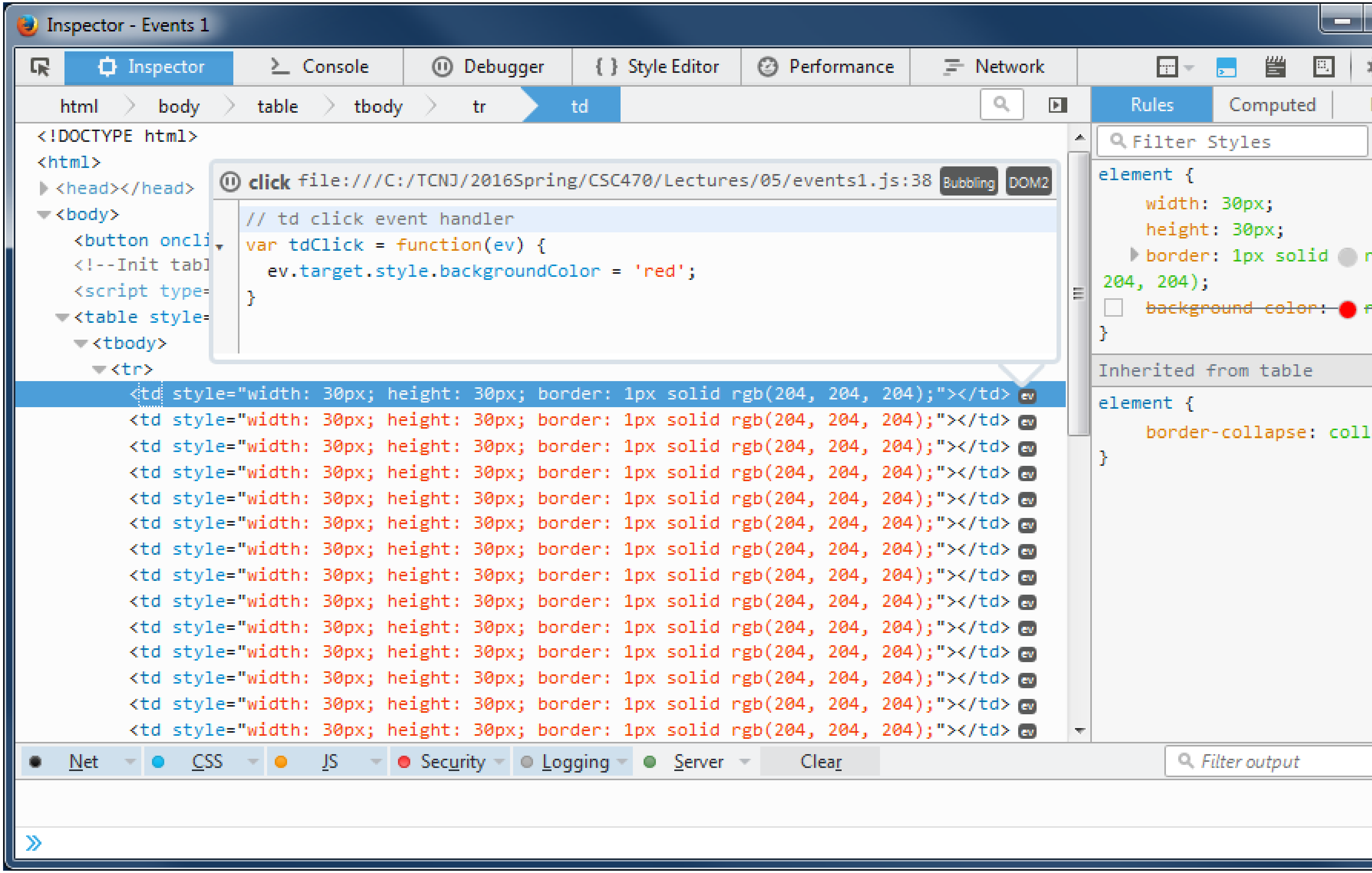
05/events1.html 05/events1.js

Example: Events 1



05/events1.html

Examining Event Listeners in FireFox



```
<!DOCTYPE html>
<html>
<head></head>
<body>
<button oncli
<!--Init tabl
<script type=
<table style=
<tbody>
<tr>
<td style="width: 30px; height: 30px; border: 1px solid rgb(204, 204, 204);"></td> ev
<td style="width: 30px; height: 30px; border: 1px solid rgb(204, 204, 204);"></td> ev
<td style="width: 30px; height: 30px; border: 1px solid rgb(204, 204, 204);"></td> ev
<td style="width: 30px; height: 30px; border: 1px solid rgb(204, 204, 204);"></td> ev
<td style="width: 30px; height: 30px; border: 1px solid rgb(204, 204, 204);"></td> ev
<td style="width: 30px; height: 30px; border: 1px solid rgb(204, 204, 204);"></td> ev
<td style="width: 30px; height: 30px; border: 1px solid rgb(204, 204, 204);"></td> ev
<td style="width: 30px; height: 30px; border: 1px solid rgb(204, 204, 204);"></td> ev
<td style="width: 30px; height: 30px; border: 1px solid rgb(204, 204, 204);"></td> ev
<td style="width: 30px; height: 30px; border: 1px solid rgb(204, 204, 204);"></td> ev
<td style="width: 30px; height: 30px; border: 1px solid rgb(204, 204, 204);"></td> ev
<td style="width: 30px; height: 30px; border: 1px solid rgb(204, 204, 204);"></td> ev
<td style="width: 30px; height: 30px; border: 1px solid rgb(204, 204, 204);"></td> ev
<td style="width: 30px; height: 30px; border: 1px solid rgb(204, 204, 204);"></td> ev
```

click file:///C:/TCNJ/2016Spring/CSC470/Lectures/05/events1.js:38 Bubbling DOM2

```
// td click event handler
var tdClick = function(ev) {
  ev.target.style.backgroundColor = 'red';
}
```

```
element {
    width: 30px;
    height: 30px;
    border: 1px solid r
204, 204);
    ~~background-color: r~~
}
Inherited from table
element {
    border-collapse: coll
}
```

Examining Event Objects

Debugger - Events 1

Inspector | Console | Debugger | Style Editor | Performance | Network

tdClick events1.js:39

Search scripts (Ctrl+P)

Sources | Call Stack

file://

events1.html

events1.js

39 ev.target.style.background...

```
25
26        // Add td to tr
27        tr.appendChild(td);
28      }
29      // Add tr to tbody
30      tbody.appendChild(tr);
31    }
32    tbl.appendChild(tbody);
33    // Add table to the document.body
34    document.body.appendChild(tbl);
35  };
36
37  // td click event handler
38  var tdClick = function(ev) {
39    ev.target.style.backgroundColor = 'red';
40  };
41
42  // Clear all table cell colors
43  var clearTable = function() {
44    var tds = document.querySelectorAll('td');
45
46    // Clear td background colors
47    for (var i=0; i<tds.length; i++) {
48      tds[i].style.backgroundColor = '';
49    }
50  };
51
```

Variables | Events

Add watch expression

Function scope [tdClick]

- this: <td>
- ev: MouseEvent click
  - altKey: false
  - bubbles: true
  - button: 0
  - buttons: 0
  - cancelBubble: false
  - cancelable: true
  - clientX: 34
  - clientY: 52
  - ctrlKey: false
  - currentTarget: <td>
  - defaultPrevented: false
  - detail: 1
  - eventPhase: 2
  - explicitOriginalTarget: <td>
  - isChar: false
  - isTrusted: true

Net | CSS | JS | Security | Logging | Server | Clear

Filter output

DOM Event Propagation and Phases

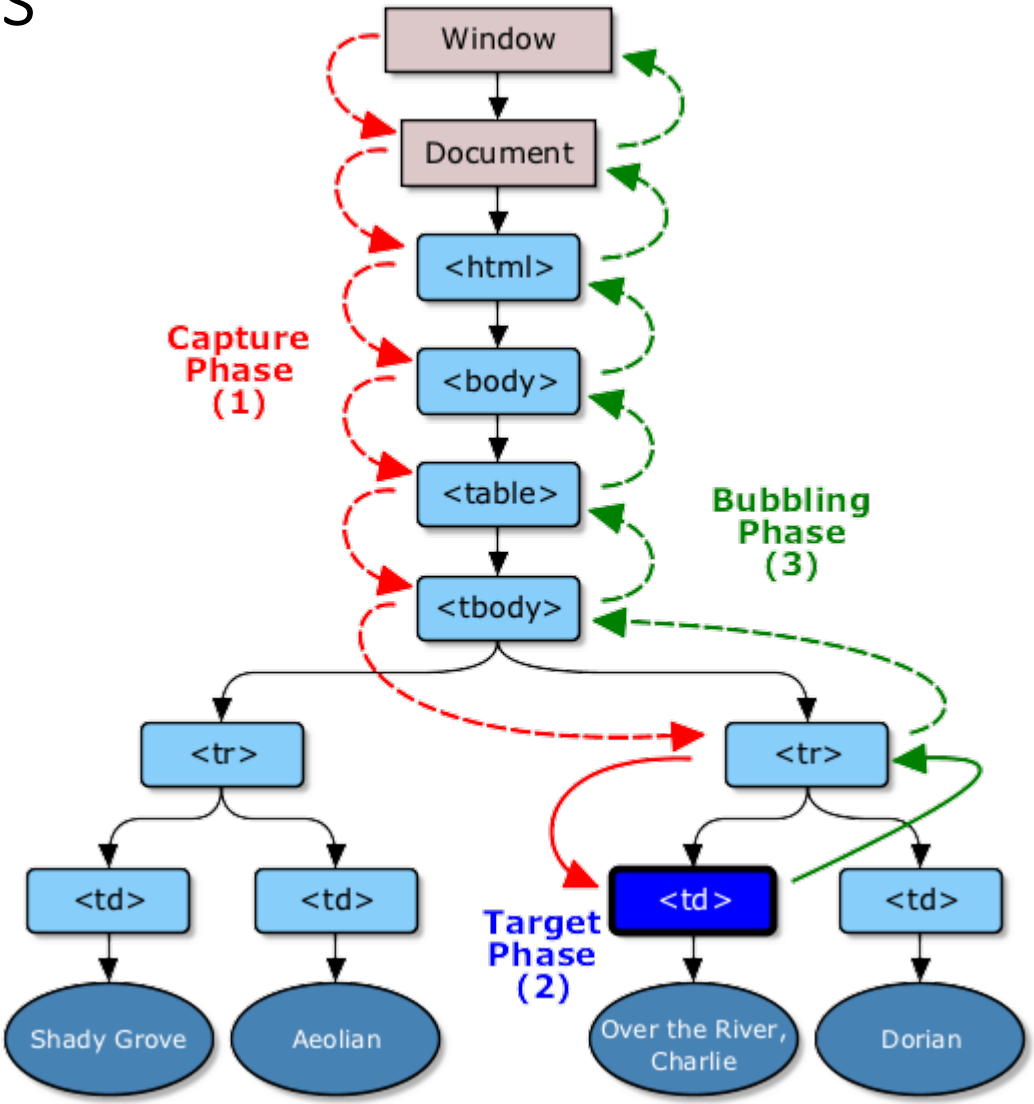
- By default events **bubble up** the DOM tree
  - Events start at the lowest Element in the DOM tree – the one that triggers the event – and moves up
  - At each Element encountered while moving up the DOM, if an event handler has been defined, it is invoked
  - This is called the "***bubbling phase***"

- Alternatively, dispatched events can be instructed to start from the top of the DOM and **trickle down** to the targeted element
  - Events start at the top of the DOM tree and move down toward the event target
  - At each Element encountered while moving down the DOM, if an event handler has been defined, it is invoked
  - This is called the "***capture phase***"

- Event listeners may be registered for the *capturing phase*, *bubbling phase*, or both.

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

DOM Event Propagation and Phases



https://www.w3.org/TR/2013/WD-DOM-Level-3-Events-20131105/

Listening During Capture Phase

- In order to listening for events during the *capture phase* a third parameter is required to be passed to `addEventListener(…)`

```
• target.addEventListener(type, listener[, useCapture]);
```

```
• td.addEventListener('click', tdClick, true);
```

- By default, `useCapture = false`

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

`target` vs. `currentTarget`

Two of the Event object's properties are `target` and `currentTarget`

- `target` is the element that originally dispatched the event
- `currentTarget` is the element in the DOM tree with an appropriate event listener registered for the event phase
  - When encountered during event flow+event phase, the listener is invoked

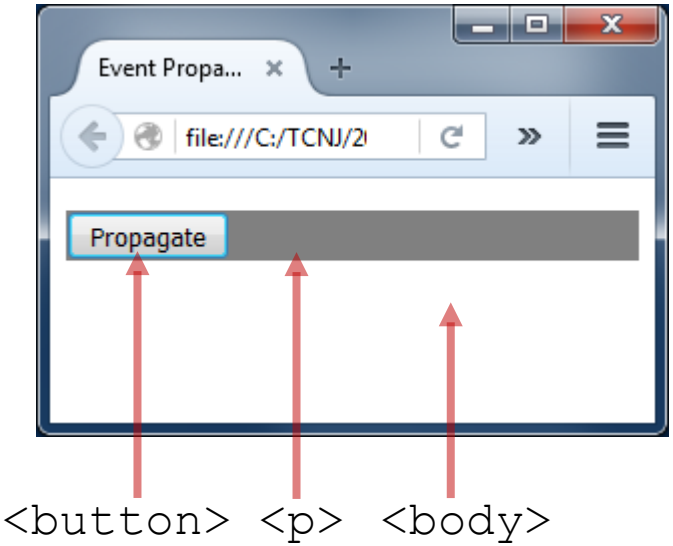
Example: Event Phases

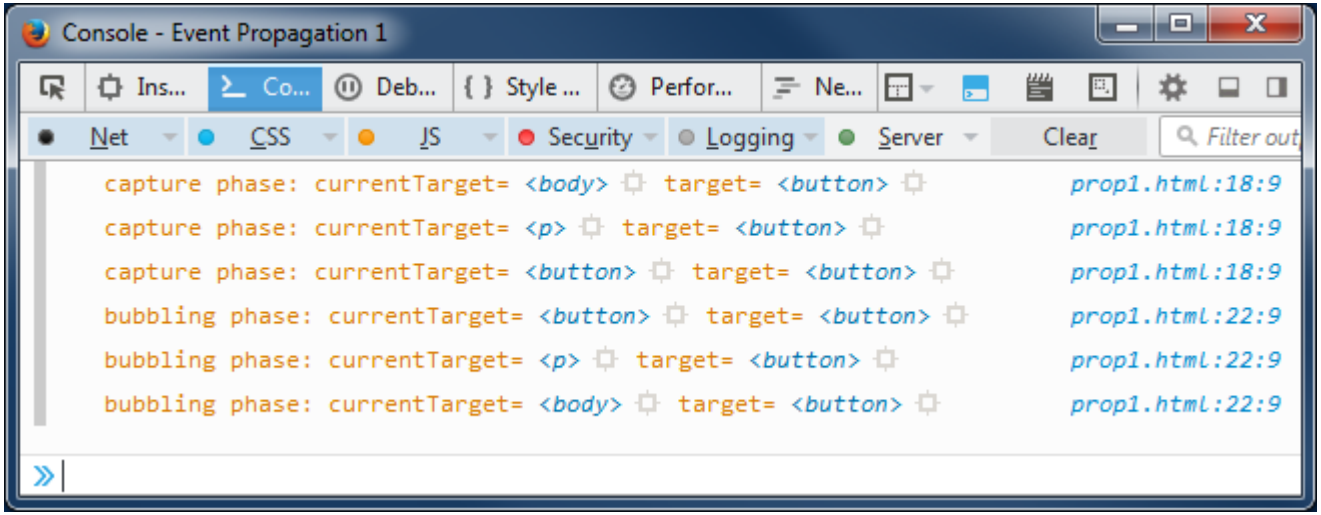
```html
<body>
  <p>
    <button>Propagate</button>
  </p>
</body>
```

```javascript
// Event handlers
var handleCapture = function(ev) {
  console.log('capture phase:', 'currentTarget=', ev.currentTarget, 'target=', ev.target);
};

var handleBubble = function(ev) {
  console.log('bubbling phase:', 'currentTarget=', ev.currentTarget, 'target=', ev.target);
};

// Connect event listeners

// body event listeners
document.body.addEventListener('click', handleCapture, true);
document.body.addEventListener('click', handleBubble);

// <p> event listeners
var p = document.querySelector('p');
p.addEventListener('click', handleCapture, true);
p.addEventListener('click', handleBubble);

// <button> event listeners
var but = document.querySelector('button');
but.addEventListener('click', handleCapture, true);
but.addEventListener('click', handleBubble);
```

05/prop1.html

Example: Event Phases





05/prop1.html

Stopping Event Propagation

- Event propagation may be stopped at any time
- Invoke the `event.stopPropagation()` method

- Ex. you have a button inside another clickable element and you don’t want clicks on the button to activate the outer element’s click behavior

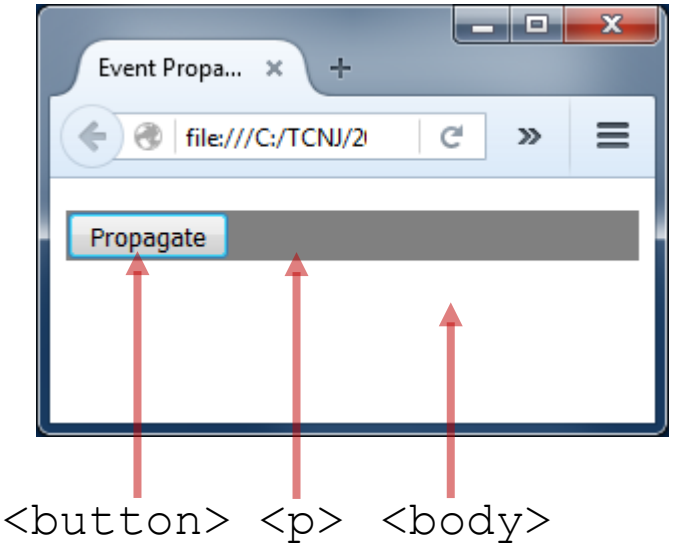
```html
<body>
  <p>
    <button>Propagate</button>
  </p>
</body>
```

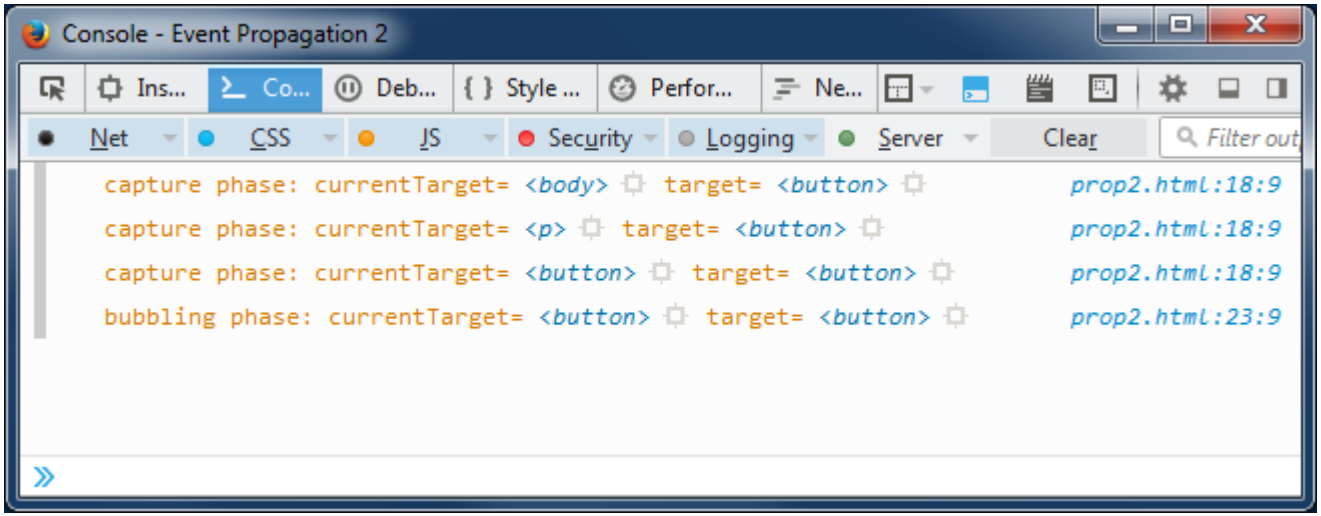
Example: Stop Propagation

```javascript
// Event handlers
var handleCapture = function(ev) {
  console.log('capture phase:', 'currentTarget=', ev.currentTarget, 'target=', ev.target);
  // ev.stopPropagation();
};

var handleBubble = function(ev) {
  console.log('bubbling phase:', 'currentTarget=', ev.currentTarget, 'target=', ev.target);
  ev.stopPropagation();
};

// Connect event listeners

// body event listeners
document.body.addEventListener('click', handleCapture, true);
document.body.addEventListener('click', handleBubble);

// <p> event listeners
var p = document.querySelector('p');
p.addEventListener('click', handleCapture, true);
p.addEventListener('click', handleBubble);

// <button> event listeners
var but = document.querySelector('button');
but.addEventListener('click', handleCapture, true);
but.addEventListener('click', handleBubble);
```

05/prop2.html

Example: Stop Propagation





05/prop2.html

Example: Events 2

In this example we want to …

- Build a table with HTML
- Assign the `onmouseover` property of all `tr`s to a function that changes the background color to a highlight
- Assign the `onmouseout` property of all `tr`s to a function that changes the background color back to the default

Example: Events 2

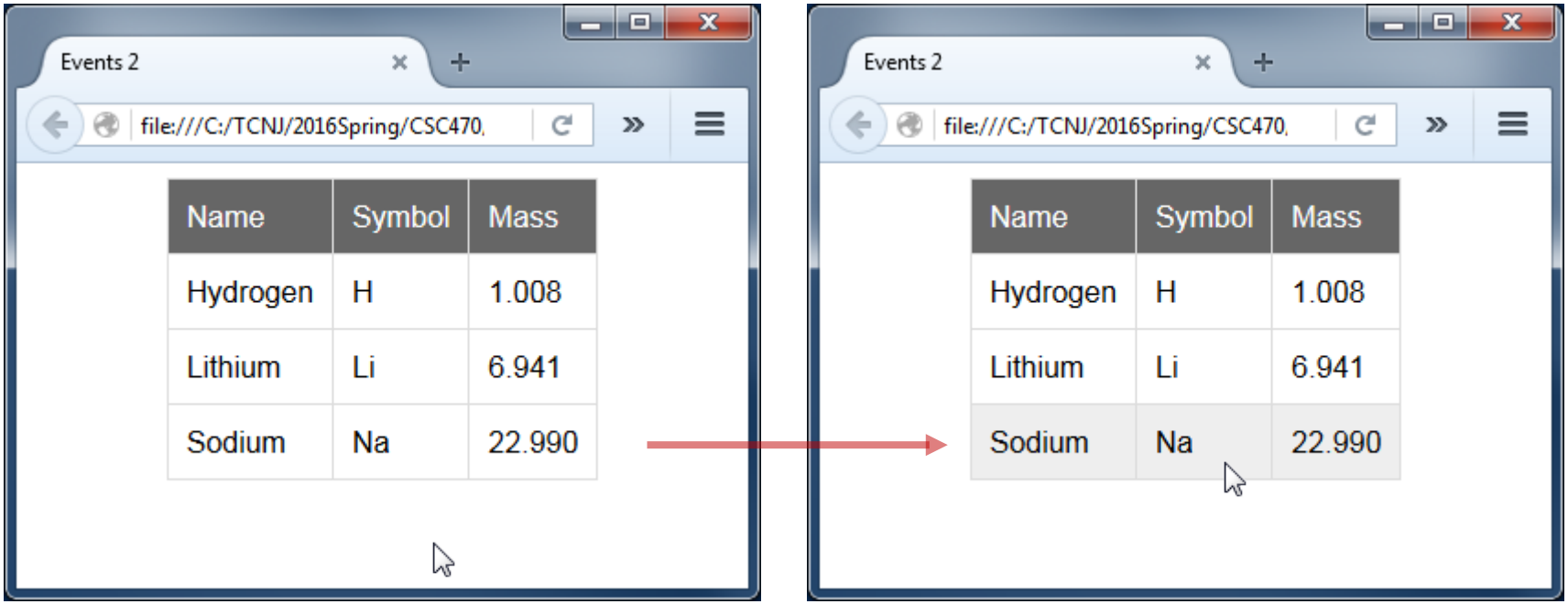
```html
<!doctype html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Events 2</title>
    <link rel="stylesheet" type="text/css"
            href="events2.css">
    <script type="text/javascript"
            src="events2.js"></script>
  </head>
  <body>
    <table id="tab1">
      <tbody>
        <tr id="header">
          <td>Name</td><td>Symbol</td><td>Mass</td>
        </tr>
        <tr>
          <td>Hydrogen</td><td>H</td><td>1.008</td>
        </tr>
        <tr>
          <td>Lithium</td><td>Li</td><td>6.941</td>
        </tr>
        <tr>
          <td>Sodium</td><td>Na</td><td>22.990</td>
        </tr>
      </tbody>
    </table>
    <script type="text/javascript">
      setEventHandlers();
    </script>
  </body>
</html>
```

```javascript
// events2.js

var hightlightRow = function(ev) {
  var tr = ev.currentTarget;
  tr.style.backgroundColor = "#eee";
};

var unhighlightRow = function(ev) {
  var tr = ev.currentTarget;
  tr.style.backgroundColor = "";
};

var setEventHandlers = function() {
  var trs = document.getElementsByTagName('tr');
  for (var i=0; i<trs.length; i++) {
    trs[i].onmouseover = hightlightRow;
    trs[i].onmouseout  = unhighlightRow;
  }
};
```

- Note the use of `currentTarget` instead of `target`
- What would happen if the `target` property was used instead?

05/events2.html

Example: Events 2



05/events2.html

Mouse Events

`click`
- occurs when the user clicks on an element

`contextmenu`
- occurs when the user right-clicks on an element to open a context menu

`dblclick`
- occurs when the user double-clicks on an element

`mousedown`
- occurs when the user presses a mouse button over an element

`mouseenter`
- occurs when the pointer is moved onto an element

`mouseleave`
- occurs when the pointer is moved out of an element

`mousemove`
- occurs when the pointer is moving while it is over an element

`mouseover`
- occurs when the pointer is moved onto an element, or onto one of its children

`mouseout`
- occurs when a user moves the mouse pointer out of an element, or out of one of its children

`mouseup`
- occurs when a user releases a mouse button over an element

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

MouseEvent - Specific Properties

`altKey`

- Returns whether the "ALT" key was pressed when the mouse event was triggered

`button`

- Returns which mouse button was pressed when the mouse event was triggered

`buttons`

- Returns which mouse buttons were pressed when the mouse event was triggered

`clientX`

- Returns the horizontal coordinate of the mouse pointer, relative to the current window, when the mouse event was triggered

`clientY`

- Returns the vertical coordinate of the mouse pointer, relative to the current window, when the mouse event was triggered

`ctrlKey`

- Returns whether the "CTRL" key was pressed when the mouse event was triggered

`metaKey`

- Returns whether the "META" key was pressed when an event was triggered

`pageX`

- Returns he horizontal coordinate of the event relative to the whole document

`pageY`

- Returns the vertical coordinate of the event relative to the whole document

`screenX`

- Returns the horizontal coordinate of the mouse pointer, relative to the screen, when an event was triggered

`screenY`

- Returns the vertical coordinate of the mouse pointer, relative to the screen, when an event was triggered

`shiftKey`

- Returns whether the "SHIFT" key was pressed when an event was triggered

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

**Does not include Event properties

Default Event Actions

- Many elements have a default action associated with them.
  - For example, if you click a link, you will be taken to the link's target.

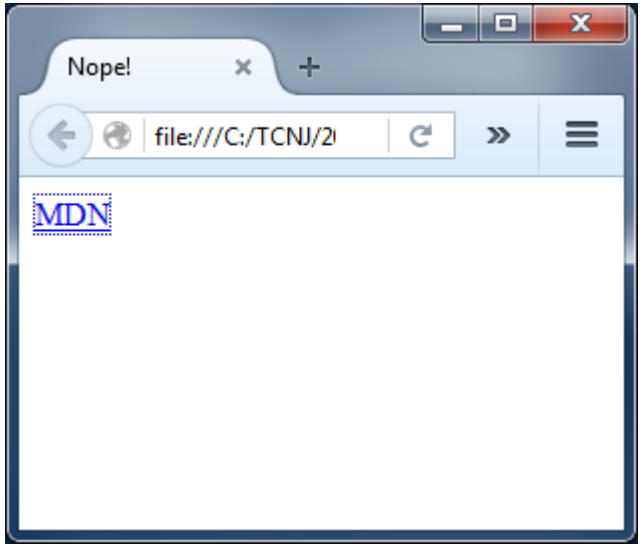
- For most, the default behavior is invoked **AFTER** any registered JavaScript event listener.

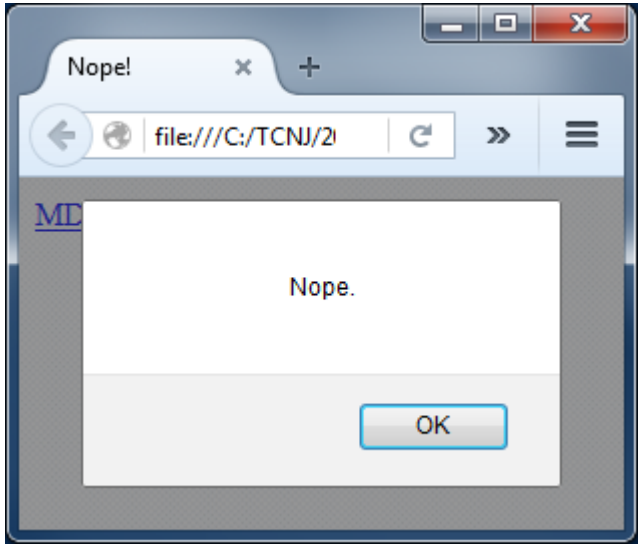
- If the handler doesn't want the normal behavior, typically because it has already taken care of handling the event, it can call the `event.preventDefault()` method on the event object.

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

Example: Un-navigatable Links

```
<a href="https://developer.mozilla.org/">MDN</a>
<script>
  var link = document.querySelector("a");

  link.addEventListener("click", function(event) {
    event.preventDefault();
    alert("Nope.");
});

</script>
```





** Must use addEventListener() in this case to attach event handler functions.

05/nope.html

Example: Drag

In this example we want to …

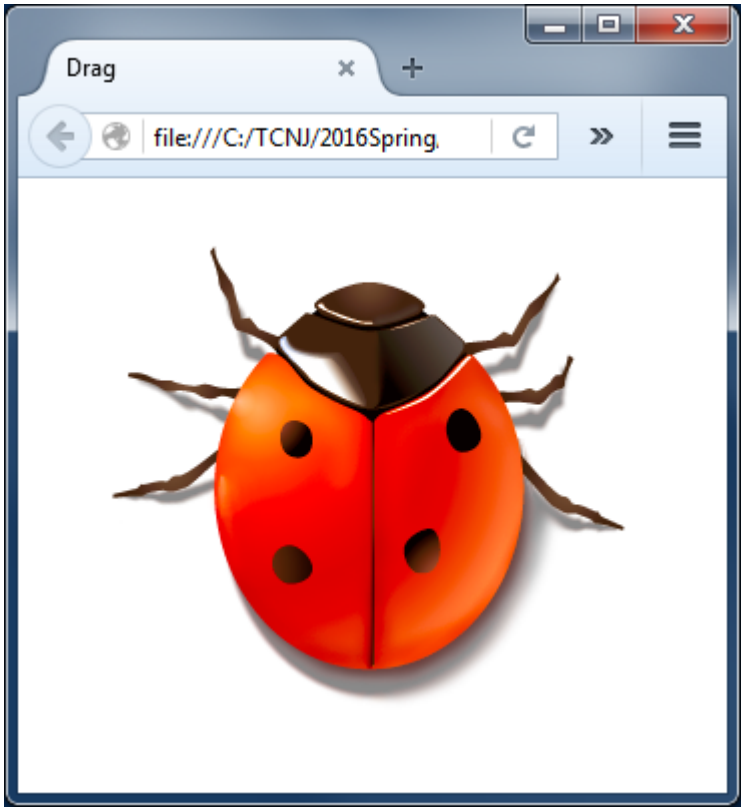
- Display an image on the page with absolute positioning
- Be able to drag the image around the page using the mouse

We focus on..

- `onmousedown`
- `onmousemove`
- `onmouseup`

Example: Drag

```
<!doctype html>
<html>

  <head>
    <meta charset="UTF-8">
    <title>Drag</title>
    <script type="text/javascript"
            src="drag.js"></script>
    <style>
      /* Position bug relative to page */
      #bug {
        position : absolute;
        left     : 100px;
        top      : 100px;
      }
    </style>
  </head>

  <body>
    <img id="bug" src="bug.png">
  </body>

  <script type="text/javascript">
    setup();
  </script>

</html>
```



https://commons.wikimedia.org/wiki/Category:Crystal_Project#/media/File:Crystal_Project_bug.png

05/drag.html

```javascript
// drag.js

// Globals to be tracked
var target  = null; // The element being dragged
var offsetX = 0;    // Offset of mouse from left edge
var offsetY = 0;    // Offset of mouse from top edge

// Initialize dragging
var startDrag = function(ev) {
  ev.preventDefault();                // Stop default dragging behavior
  target   = ev.target;               // Save target element
  var rect = target.getBoundingClientRect();
  offsetX  = ev.clientX - rect.left; // Save mouse offset from corner
  offsetY  = ev.clientY - rect.top;  // of image
};

// Perform the drag
var doDrag = function(ev) {
  if (target === null) return;     // Do nothing of no target
                                   // Move target, accounting for offset
  target.style.left = (ev.clientX - offsetX) + 'px';
  target.style.top  = (ev.clientY - offsetY) + 'px';
};

// Stop dragging
var stopDrag = function(ev) {
  target = null;                   // No target, nothing to drag
};

// Set up dragging of image
var setup = function() {
  var bug = document.getElementById("bug");
  bug.onmousedown = startDrag;
  bug.onmouseup   = stopDrag;
  bug.onmousemove = doDrag;
};
```

05/drag.js

Example: Drag 2

- Problem – if we drag too fast we lose the image as the event target

- One solution:
  - On mouse down, dynamically display an invisible `<div>` covering entire window
  - `<img>` handles `mousedown` event to set up drag
  - `<div>` handles `mousemove` and `mouseup` events
  - Hide `<div>` on mouse up

```html
<!doctype html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Drag 2</title>
    <script type="text/javascript"
            src="drag2.js"></script>
    <style>
      /* Position bug relative to page */
      #bug {
        position: absolute;
        left: 100px;
        top: 100px;
      }
      /* Invisible, whole-window overlay */
      #overlay {
        position: absolute;
        left: 0px;
        right:0px;
        top: 0px;
        bottom: 0px;
        background-color: rgba(0,0,0,0.1);
        display: none;
        z-index: 1000;
      }
    </style>
  </head>

  <body>
    <img id="bug" src="bug.png">
    <div id="overlay"></div>
  </body>

  <script type="text/javascript">
    setup();
  </script>
</html>
```

Add a that covers entire window
Initially is not displayed

05/drag3.html

```javascript
// drag2.js

// Globals to be tracked
var target = null;  // The element being dragged
var offsetX = 0;    // Offset of mouse from left edge
var offsetY = 0;    // Offset of mouse from top edge

// Initialize dragging
var startDrag = function(ev) {
  ev.preventDefault();                // Stop default dragging behavior
  target = ev.currentTarget;          // Save target element
  var rect = target.getBoundingClientRect();
  offsetX  = ev.clientX - rect.left; // Save mouse offset from corner
  offsetY  = ev.clientY - rect.top;  // of image
  var overlay = document.getElementById("overlay");
  overlay.style.display = 'initial';
};

// Perform the drag
var doDrag = function(ev) {
  if (target === null) return;     // Do nothing of no target
                                   // Move target, accounting for offset
  target.style.left = (ev.clientX - offsetX) + 'px';
  target.style.top  = (ev.clientY - offsetY) + 'px';
};

// Stop dragging
var stopDrag = function(ev) {
  target = null;                   // No target, nothing to drag.
                                   // Hide overlay
  var overlay = document.getElementById("overlay");
  overlay.style.display = 'none';
};

// Set up dragging of image
var setup = function() {
  // Image handles mousedown event
  var bug = document.getElementById("bug");
  bug.onmousedown      = startDrag;

  // overlay div handles mousemove and mouseup events
  var overlay = document.getElementById("overlay");
  overlay.onmouseup    = stopDrag;
  overlay.onmousemove = doDrag;
};
```

Overlay is displayed and can now receive events

Overlay is hi dden when dragging completes

Overlay handles `mousemove` and `mouseup` events

05/drag3.js

Event Handler Functions and Context

All functions belong to an object

- Functions that appear to be "free floating," actually belong to the global object
- In this case in a browser the global object is `window`
- The keyword `this` inside a global function refers to `window`

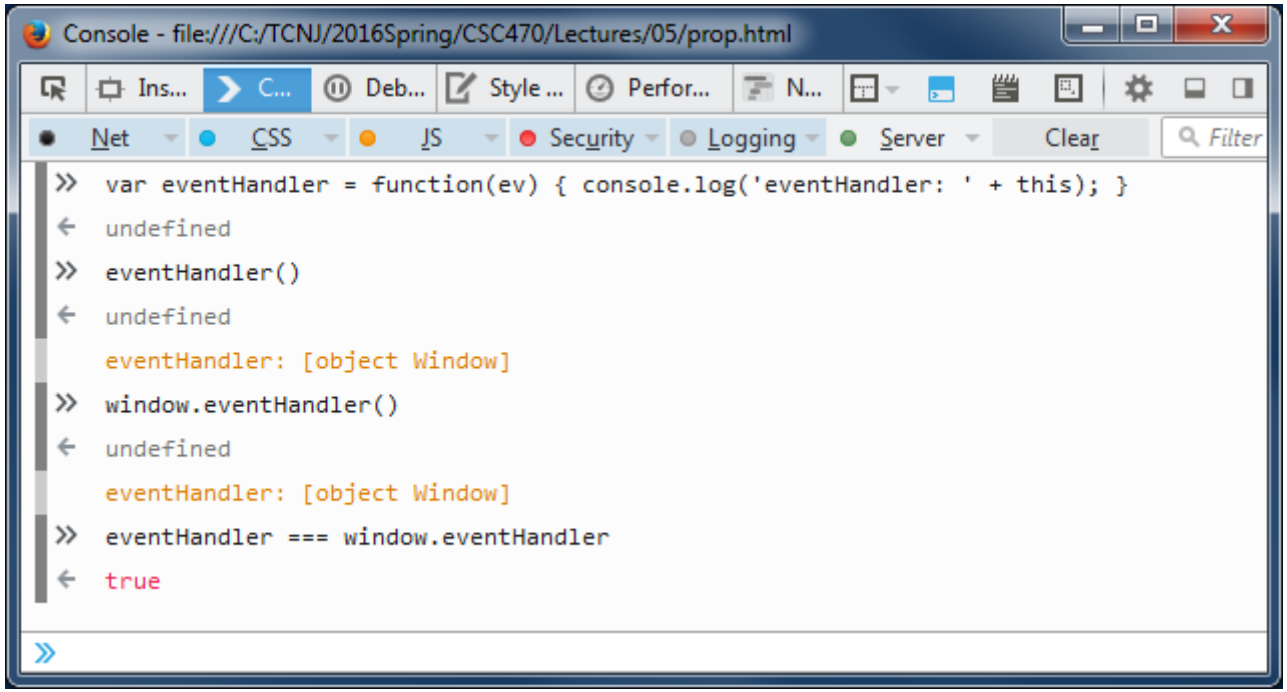
In response to an event, an event handler function runs in the scope of the event target

- In this case the variable `this` will refer to the object that invoked the event handler function

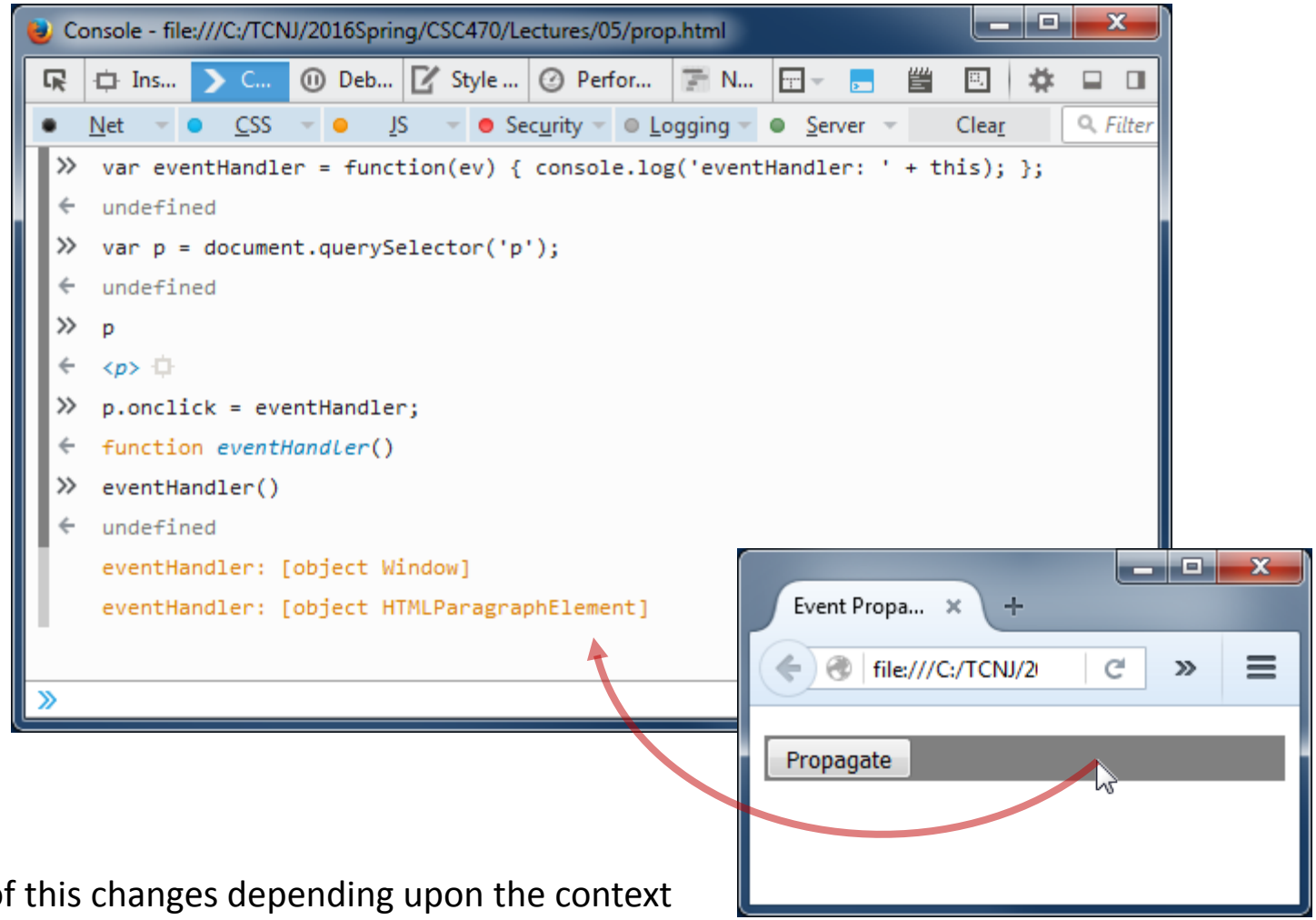
It is very common for novice JavaScript developers to develop and debug a function in the global scope, but be baffled when they observe completely different behavior when the function runs as an event handler

Example: Functions of Window

- Functions defined in the global scope are actually owned by `window`



Example: Event Context



*The value of this changes depending upon the context in which the `eventHandler()` function is invoked

Setting Function Context

- The context of a function can be set if invoked using the `call()` or `apply()` methods of a function object.
- In either case the value of `this` within the function will be modified.
- Non-local variables referenced within the function are expected to be properties of the `this` context

```
method.call(  newThisContext, param1, ..., paramN )
method.apply( newThisContext, [ param1, ..., paramN ] );
```

Example: Setting Function Context

Console - file:///C:/TCNJ/2016Spring/CSC470/Lectures/05/prop.html

```
var eventHandler = function(ev) { console.log('eventHandler: ' + this); };
undefined
var p = document.querySelector('p');
undefined
p
<p>
p.onclick = eventHandler;
function eventHandler()
eventHandler()
undefined
eventHandler: [object Window]
eventHandler: [object HTMLParagraphElement]
var button = document.querySelector('button')
undefined
eventHandler.call(button)
undefined
eventHandler: [object HTMLButtonElement]
```

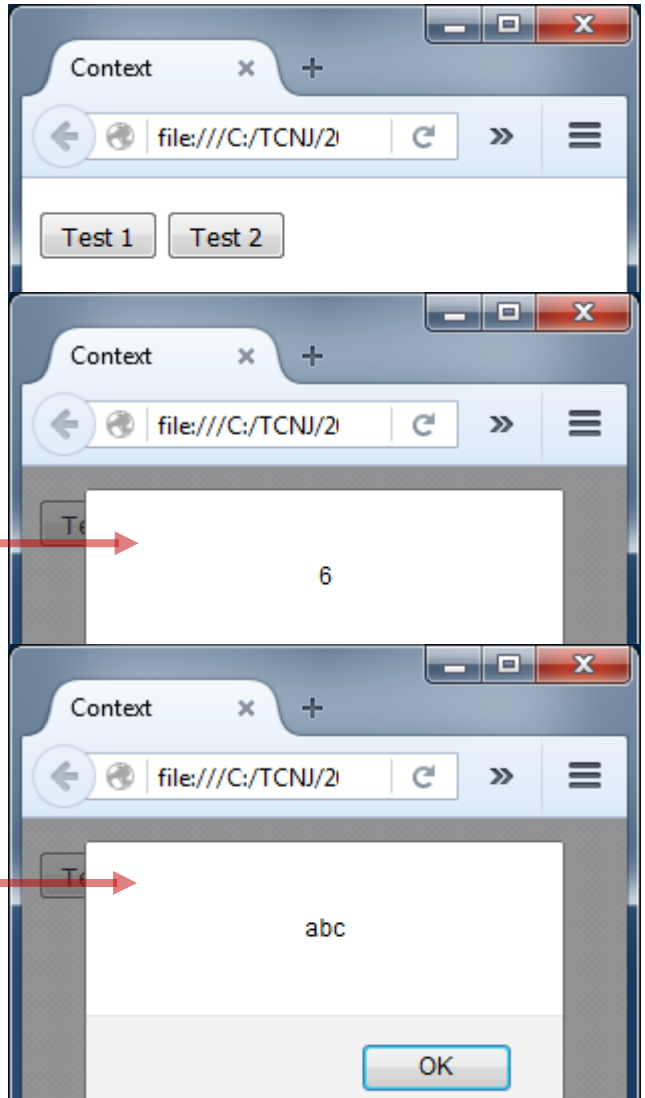
`eventHandler()` function explicitly invoked in the context of the `<button>` element using the `call()` method of the function object

Example: Custom Function Context

```
<!doctype html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Context</title>
    <script type="text/javascript">

      var addABC = function() {
        return this.A + this.B + this.C;
      };

      var test1 = function() {
        var ob = {'A':1, 'B':2, 'C':3};
        var result = addABC.call( ob );
        alert(result);
      }

      var test2 = function() {
        var ob = {'A':'a', 'B':'b', 'C':'c'};
        var result = addABC.call( ob );
        alert(result);
      }
    </script>
  </head>
  <body>
    <p>
      <button onclick="test1();">Test 1</button>
      <button onclick="test2();">Test 2</button>
    </p>
  </body>
</html>
```

05/context.html

Any object may be provided to `call()` or `apply()` as a function context, including a custom Object.



Keyboard Events

Events are raised when keys on the keyboard are pressed and may be handled by JavaScript functions

- Key events are not raised on a mobile device browser

`keydown`

- occurs when the user is pressing a key

`keyup`

- occurs when the user releases a key

`keypress`

- occurs when the user presses a key

- May be assigned handler functions using "on" attributes of an Element
  - `onkeydown, onkeyup, onkeypress`

http://www.w3schools.com/jsref/dom_obj_event.asp

KeyboardEvent-Specific Properties

`key`

- the value of a key or keys pressed by the user.

`code`

- represents a physical key, that is value not changed neither by the modifier state, nor by keyboard layout.
- useful when you attempt to distinguish keys physically

`altKey`

- a Boolean indicates if the alt key (Option or ⌥ on OS X) was pressed (true) or not (false) when the event occurred

`ctrlKey`

- a Boolean that indicates if the control key was pressed (true) or not (false) when the event occured

`shiftKey`

- a Boolean that indicates if the shift key was pressed (true) or not (false) when the event occurred

`metaKey`

- a Boolean that indicates if the Meta key was pressed (true) or not (false) when the event occurred

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

**Does not include Event properties

Example: Etch A Sketch

- Develop an "Etch A Sketch" style program made from a large table of `<td>` elements
- Drive stylus using arrow keys
- Wrap at edges
- Similar to events1.html

```
<!doctype html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Etch A Sketch</title>
    <script type="text/javascript"
            src="etchASketch.js"></script>
  </head>
  <body>
    <button onclick="clearTable()">Clear</button>
    <!-- Init table -->
    <script type="text/javascript">
      setup();
    </script>
  </body>
</html>
```

05/etchASketch.html

```javascript
// Programmatically add a table to the page
// with cells that change color when clicked
var buildTable = function(nrows, ncols, size) {

  // Reset grid array
  grid_tds = [];

  // Create table and tbody elements
  var tbl   = document.createElement('table');
  tbl.style.borderCollapse = "collapse";
  var tbody = document.createElement('tbody');
  tbl.appendChild(tbody);

  // Create rows
  for (var r=0; r<nrows; r++) {
    var row_tds = [];
    var tr = document.createElement('tr');

    // Create cells in each row
    for (var c=0; c<ncols; c++) {
      var td = document.createElement('td');
      td.style.width  = size;
      td.style.height = size;
      td.style.border = '1px solid #ccc';

      // Add td to tr
      tr.appendChild(td);
      row_tds.push(td);
    }
    // Add tr to tbody
    tbody.appendChild(tr);
    grid_tds.push(row_tds);
  }
  // Add table to the document.body
  document.body.appendChild(tbl);

  // Color first td
  current_row = 0;
  current_col = 0;
  grid_tds[0][0].style.backgroundColor = 'red';
};
```

- Very similar to `events1.js`
- A global variable named `grid_tds` is assigned to an array of arrays of `<td>` elements that make up the grid
- Also, `current_row` and `current_col` globals are assigned – hold the position of the stylus

05/etchASketch.js

Example: Etch A Sketch

```
// Move the highlighted cell by amount specified
var move = function(delta_row, delta_col) {
  // Clear current td, leaving a trace
  var td = grid_tds[current_row][current_col];
  td.style.backgroundColor = 'rgba(255, 0, 0, 0.1)';

  // Compute nrows and ncols of grid
  var nrows = grid_tds.length;
  var ncols = grid_tds[0].length;

  // Increment location and update globals
  current_row = (current_row + delta_row + nrows) % nrows;
  current_col = (current_col + delta_col + ncols) % ncols;

  // Modify color of new td
  td = grid_tds[current_row][current_col];
  td.style.backgroundColor = 'red';
};

// Read key pressed and move in appropriate directlion
var move_cell = function(ev) {
  switch (ev.code) {
    case "ArrowRight":
      move(0, 1);
      break;
    case "ArrowLeft":
      move(0, -1);
      break;
    case "ArrowUp":
      move(-1, 0);
      break;
    case "ArrowDown":
      move(1, 0);
      break;
  }
};
```

- The `move()` function updates the table cells and global variables to simulate moving the stylus in the direction specified by `delta_row` and `delta_col`
- The `move_cell()` function reads the key pressed from the event and invokes `move()` with appropriate arguments

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

05/etchASketch.js

Example: Etch A Sketch

```
var clearTable = function() {
  var tds = document.querySelectorAll('td');

  // Clear td background colors
  for (var i=0; i<tds.length; i++) {
    tds[i].style.backgroundColor = '';
  }

  // Reset selected cell
  td = grid_tds[current_row][current_col];
  td.style.backgroundColor = 'red';
};


// Track grid and current location
var grid_tds = [];
var current_row = 0;
var current_col = 0;

// Set up the example program
var setup = function() {
  // Build the table
  buildTable(30, 60, '10px');

  // Set up key event handlers
  document.body.onkeypress = move_cell;
};
```

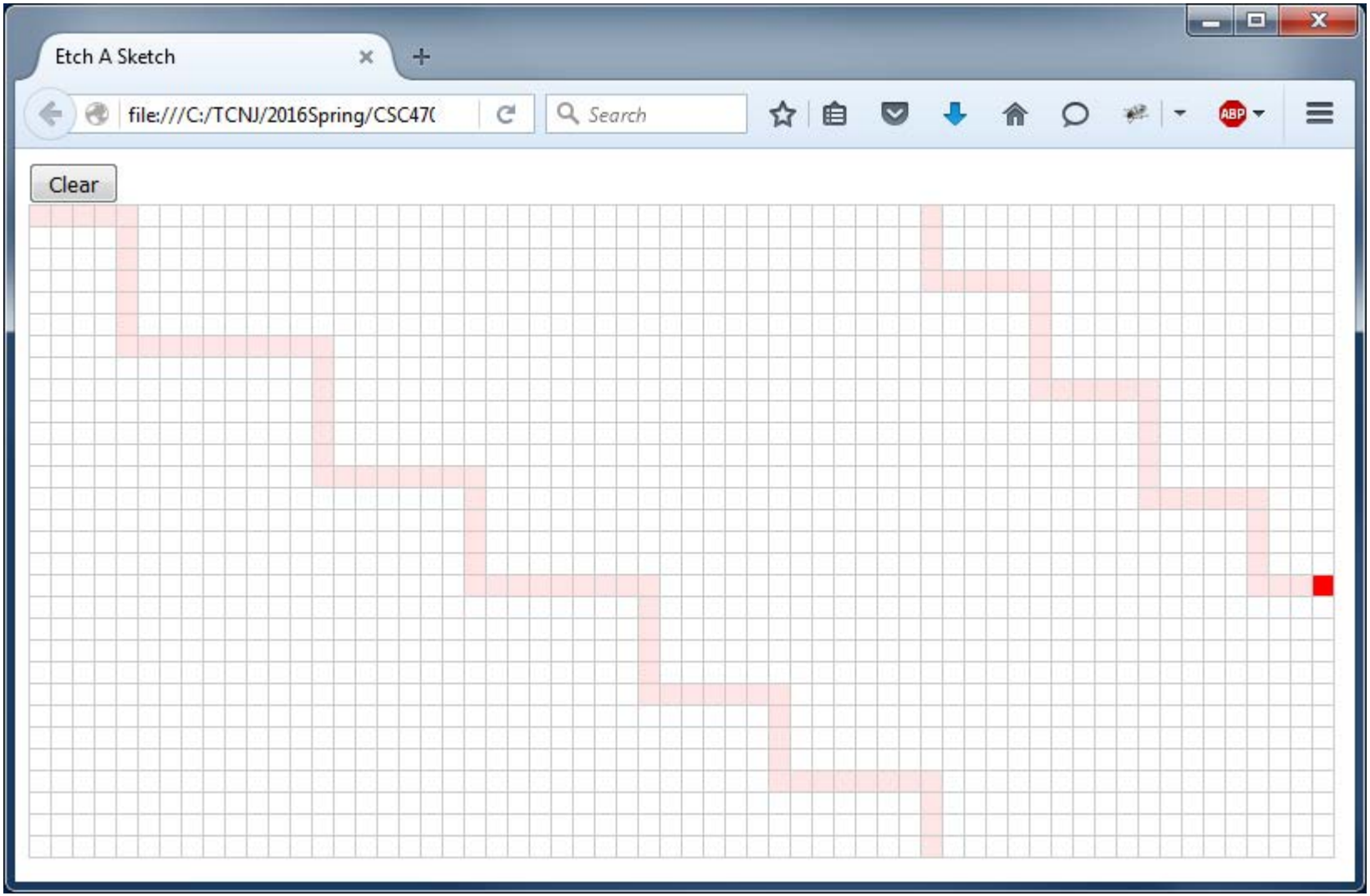
- The `clearTable()` function is very similar to `events1.js`
- The `setup()` builds the main table and sets up an `onkeypress` handler for `document.body`

- Globals declared:

`grid_tds`:	array of array of `<td>` elements
`current_row`:	row of highlighted cell
`current_col`:	column of highlighted cell

05/etchASketch.js

Example: Etch A Sketch



05/etchASketch.html

Load Event

- Often, a page must be loaded completely before configuration can be finished
  - Programmatically adding Elements
  - Attaching event handlers
  - Etc…
- When a resource finishes loading, the "load" event fires
- When fired by the `document` object, implies that the main document has loaded
- When fired by the `window` object, implies that the main document and all other resources have been loaded
  - Images, scripts, CSS, …
  - Could take some time
- The `document.load` event is often used to schedule initialization actions that require the depend upon the document to exist
- The content of `<script>` tags is run immediately when the tag is encountered. This is often too soon.
  - Many recommend putting `<script>` tags to be executed when document loads at bottom of main HTML file
- Use the `onload` property to attach a load event handler

Example: Use `window.load` to Init Program

```html
<!doctype html>
<html>

  <head>
    <meta charset="UTF-8">
    <title>Etch A Sketch</title>
    <script type="text/javascript"
            src="etchASketch.js"></script>

  </head>

  <body>
    <button onclick="clearTable()">Clear</button>

    <!-- Init table -->
    <script type="text/javascript">
      setup();
    </script>
  </body>

</html>
```

```html
<!doctype html>
<html>

  <head>
    <meta charset="UTF-8">
    <title>Etch A Sketch</title>
    <script type="text/javascript"
            src="etchASketch.js"></script>

    <script type="text/javascript">
      window.onload = setup;
    </script>

  </head>

  <body>
    <button onclick="clearTable()">Clear</button>

  </body>

</html>
```

05/etchASketch2.html

Timers

Although not exactly events, JavaScript functions can be scheduled to run after a period of time, on a regular time interval, or prior to the next repaint of the window

`var id = setTimeout(func, delay)`

- Calls a function or executes a code snippet after specified delay

`clearTimeout(id)`

- Clears the delay set by `setTimeout()`

`var id = setInterval(func, delay)`

- Calls a function or executes a code snippet repeatedly, with a fixed time delay between each call

`clearInterval(id)`

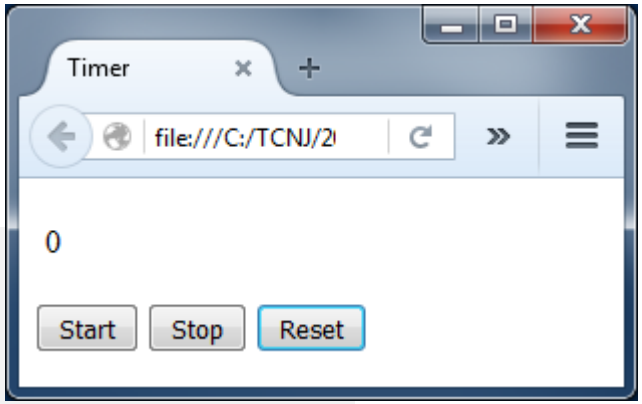
- Clears the delay set by `setTimeout()`

`window.requestAnimationFrame(callback)`

- Requests that the browser call a specified function before the next repaint.
- Provides callback function with a timestamp argument.
- The method takes as an argument a callback to be invoked before the repaint.
- Can reach 60 fps

https://developer.mozilla.org/en-US/Add-ons/Code_snippets/Timers

Example: Simple Timer




```
<!doctype html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Timer</title>
    <script type="text/javascript" src="timer.js">
    </script>
  </head>

  <body>
    <p style="padding: 5px;" id="readout">0</p>
    <p>
      <button onclick="startTimer();">Start</button>
      <button onclick="stopTimer();"> Stop </button>
      <button onclick="resetTimer();">Reset</button>
    </p>
  </body>
</html>
```

05/timer.html

Example: Simple Timer

```
// timer.js

// globals
var tenths = 0.0;
var timerID = null;

// Start the timer running at a 100 millisec interval
var startTimer = function() {
  timerID = setInterval(update, 100);
};

// The function called on each interval
// Add one tenth to counter and update display
var update = function(time) {
  tenths += 1;
  updateDisplay();
};

// Stop the timer using the saved timerID
var stopTimer = function() {
  if (timerID !== null) clearInterval(timerID);
  timerID = null;
}

// Reset the tenths counter and update display
var resetTimer = function() {
  tenths = 0.0;
  updateDisplay();
};

// Reset the timer display
var updateDisplay = function() {
  var p = document.getElementById('readout');
  p.innerHTML = (tenths/10).toString();
}
```

05/timer.js

Example: Animation

```html
<!doctype html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Animated Bee</title>
  </head>

  <body>
    <img id="bee" src="bee.png">
  </body>

  <script type="text/javascript">
    var start = null;

    // Get the image Element
    var element = document.getElementById("bee");
    element.style.position = 'absolute';

    function step(timestamp) {
      // Init start the first time in step()
      if (!start) start = timestamp;

      // Calculate a time value representing progress
      var t = (timestamp - start)/1000;

      // Set position of image using t
      element.style.left = 250+200*Math.sin(t) + "px";
      element.style.top  = 100+50*Math.cos(5*t) + "px";

      // Schedule another update prior to next refresh
      window.requestAnimationFrame(step);
    }

    // Kick off the animation
    window.requestAnimationFrame(step);
  </script>
</html>
```

The Element to animate has absolute positioning

Animation parameter are calculate based don a timing parameter in case updates are not uniform

Before finishing one frame of the animation, request another

Kick off the animation

https://openclipart.org/image/90px/svg_to_png/221154/Cartoon-Bee.png
https://developer.mozilla.org/en-US/docs/Web/API/Window/requestAnimatioanFrame

05/animate.html