CS 142: Section 3

JavaScript and the DOM

Outline

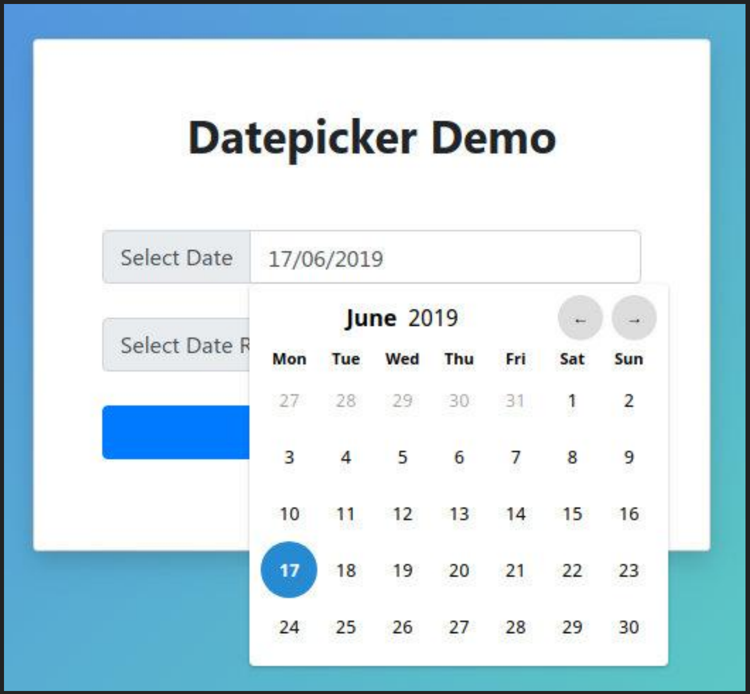
- **Project 3 Tips**
- Document Object Model
- Some JS Concepts
- Events

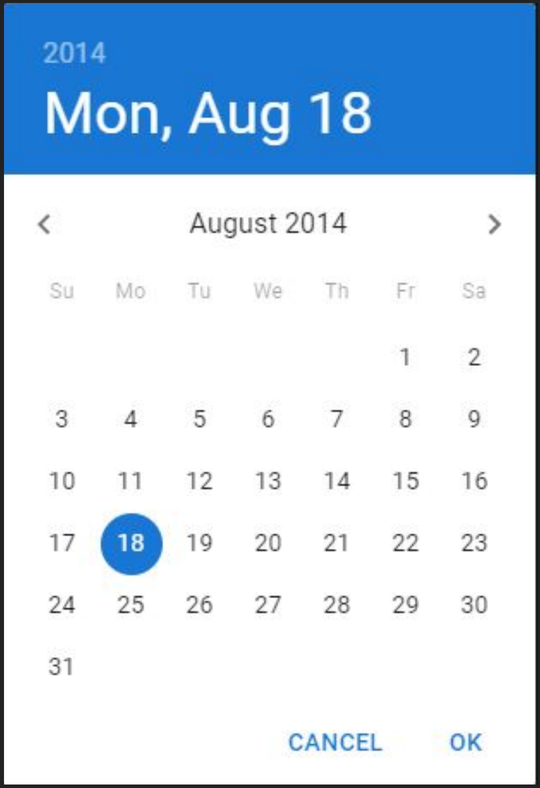
Project 3 - Logistics

- jQuery, et al. is not allowed
  - Built-in JS (e.g. Date object, parseInt) **is** allowed
- Make sure to type `npm run jshint` prior to submission
  - JSHint errors are an easy way to lose style points
  - We will run JSHint checks for projects 2 and up so make it a habit to check for style errors

Project 3 - Problem 1

- Build a Date Picker





Project 3 - Problem 1

- Implement DatePicker.js and datepicker.css
- DatePicker.js exposes a DatePicker constructor

```
<body>
   ...
   <script type="text/javascript" src="DatePicker.js"></script>
   <script type="text/javascript">
     var datePicker = new DatePicker(
       "div1",
       function (id, fixedDate) { ... },
     );
    datePicker.render(new Date("February 20, 2015"));
   </script>
</body>
```

Project 3 - Problem 1: Date class

Example

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

```
var xmas95 = new Date('December 25, 1995 23:15:30');
var day = xmas95.getDate();
console.log(day); //prints 25
```

Hints

- `getDate()` to retrieve day (1-31) of the month
- `setDate(0)` will set to last day of previous month
- the fixedDate object passed into the callback is not a Date object. (Read datepicker.html and the project spec to get a sense for why)

Project 3 - Problem 1: Editing DOM

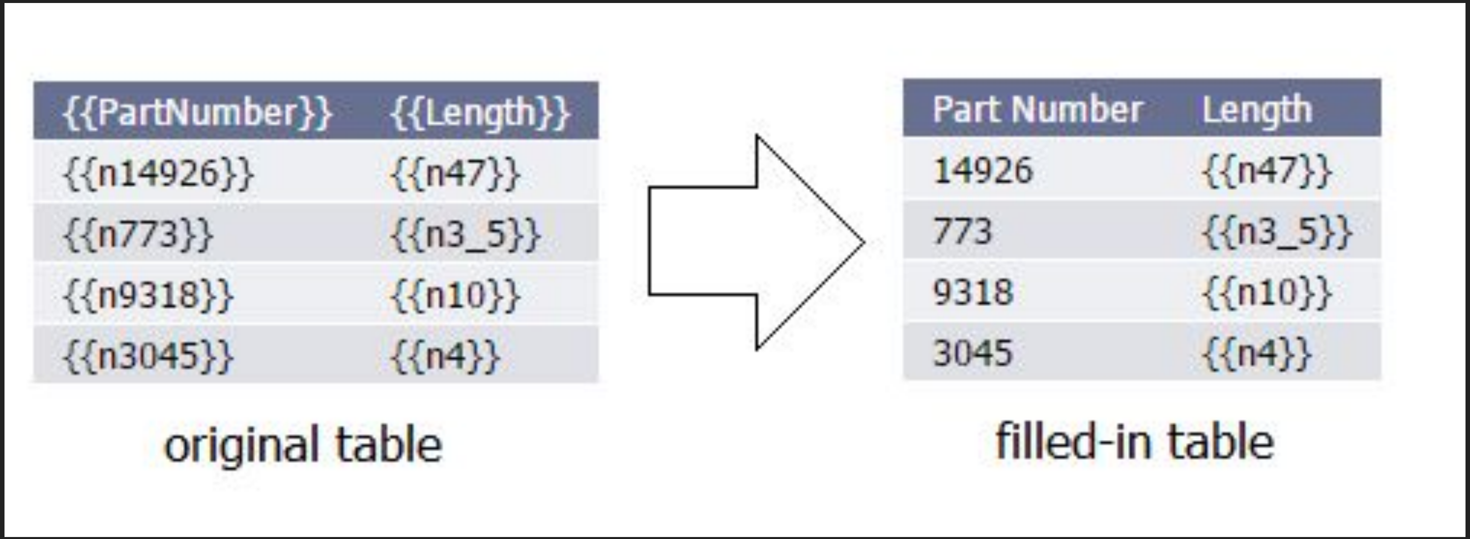
1. DOM API to actually create html elements:
   - `myDiv.innerHTML = '<a href="http://google.com">Google</a>';`
     Output: Google

2. DOM API to just add text:
   - `myDiv.textContent = '<a href="http://google.com">Google</a>';`
     Output: <a href="http://google.com">Google</a>
   - textContent is faster & more secure than innerHTML

3. Also an option: `createElement`, `appendChild`
   - http://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model

Project 3 - Problem 2

- You will be filling out TableTemplate.js, which exposes a TableTemplate class with a static method `fillIn`
- `TableTemplate.fillIn('table', dict, 'Part Number')` should produce*:



{{PartNumber}}	{{Length}}
{{n14926}}	{{n47}}
{{n773}}	{{n3_5}}
{{n9318}}	{{n10}}
{{n3045}}	{{n4}}

original table

Part Number	Length
14926	{{n47}}
773	{{n3_5}}
9318	{{n10}}
3045	{{n4}}

filled-in table

* "table" is the id of the table, dict is a dictionary mapping template string names to string values

Project 3 - Problem 2: Tips

- Reuse functionality from project 2: cs142-template-processor.js
- DOM helper functions for `<table>`
  - https://developer.mozilla.org/en-US/docs/Web/API/HTMLTableElement
  - Example:

```
var table = document.getElementById("myTable");
var tbody = table.tBodies[0];
var first_tr = tbody.rows[0];
```

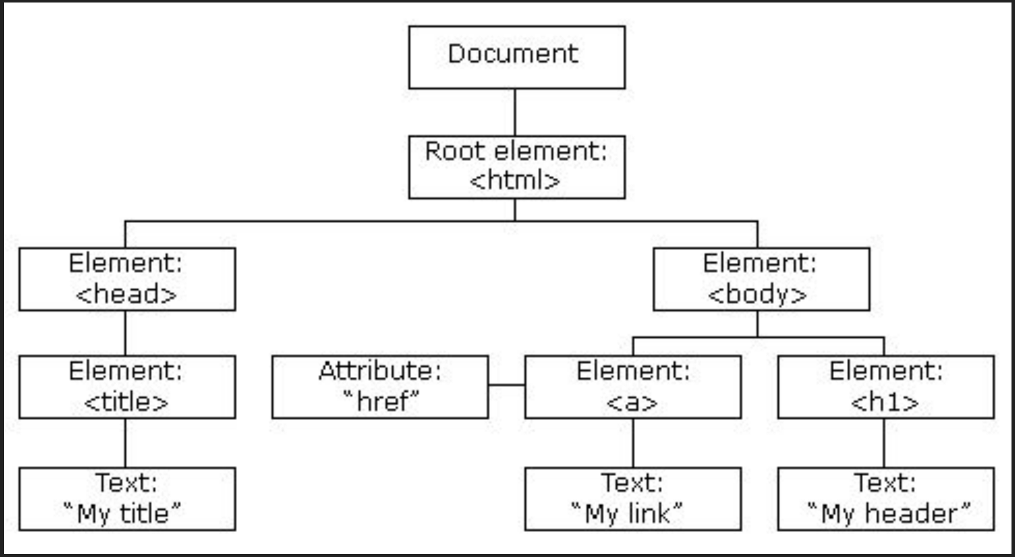
Outline

- Project 3 Tips
- **Document Object Model**
- Some JS Concepts
- Events

Document Object Model (DOM)

- Representation of HTML as Javascript objects
- Can use Javascript to manipulate DOM and events
- Sample selector: `document.getElementsByTagName`

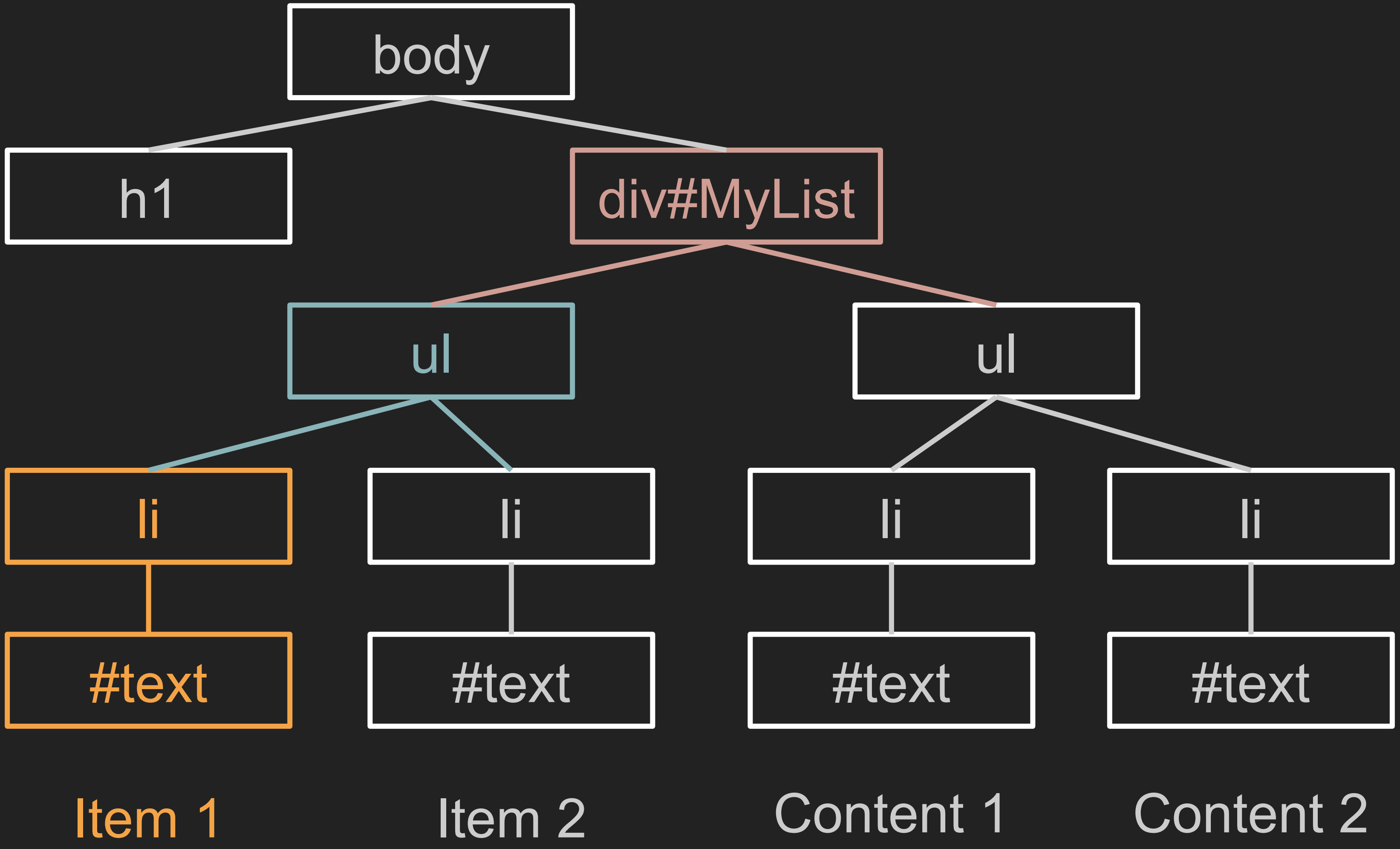
```
<html>
  <head>
    <title>My title</title>
  </head>
  <body>
    <a href="#">My link</a>
    <h1>My header</h1>
  </body>
<html>
```



DOM Traversal: Markup

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en"
lang="en">
<head><title>My Lists</title></head>
<body>
    <h1 id="Header">My Big Header</h1>
    <div id="MyList">
        <ul><li>Item 1</li><li>Item 2</li></ul>
        <ul><li>Content 1</li><li>Content 2</li></ul>
    </div>
</body>
</html>
```

DOM Traversal: DOM Tree



Outline

- Project 3 Tips
- Document Object Model
- **Some JS Concepts**
- Events

Functions

Functions as First Class Values

```
// Functions can be assigned to variables
var squared = function(x) { return x*x; };

// And can be passed as arguments to other functions
var applyTwice = function(num, fn) {
   return [fn(num), fn(num)];
};

applyTwice(2, squared); // Will return [4, 4]
```

Prototypes: Define new classes

Pair class

```
// constructor
function Pair(x, y) {
  // instance variables, public
  this.x = x;
  this.y = y;
};

// instance method: uses current object
Pair.prototype.sum = function() {
  return this.x + this.y;
};

// static method: independent of object
Pair.distance = function(p1, p2) {
  return Math.sqrt(Math.pow(p1.x - p2.x, 2)
     + Math.pow(p1.y - p2.y, 2));
};
```

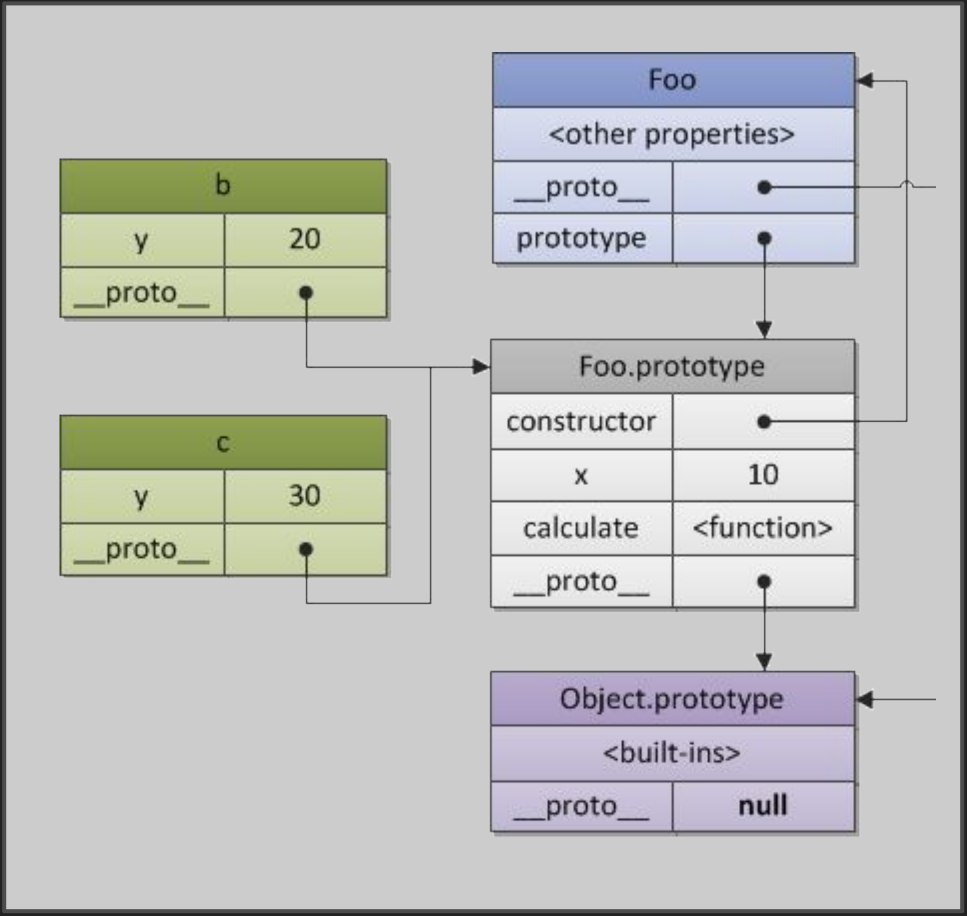
Sample usage

```
var p1 = new Pair(3, 4);

p1.x; // returns 3

p1.sum(); // returns 7

Pair.distance(p1, p1);
```

Prototype Diagram

From http://dmitrysoshnikov.com/ecmascript/javascript-the-core/

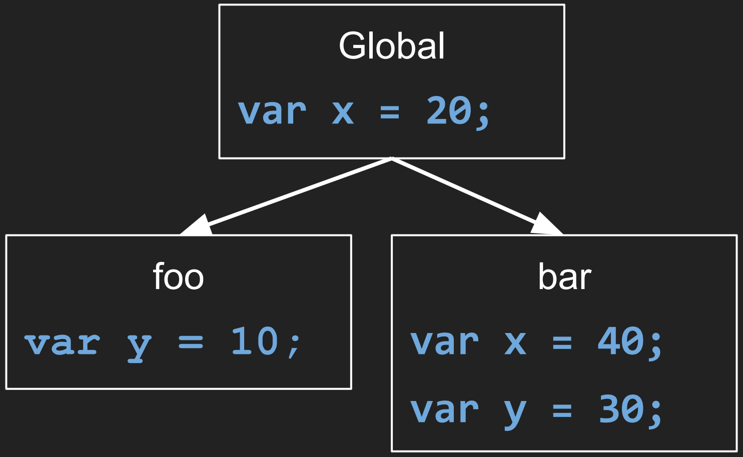
```
function Foo(y) {
  this.y = y;
}

Foo.prototype.x = 10;

Foo.prototype.calculate =
function(z) {
  return this.x + this.y + z;
};

var b = new Foo(20);
var c = new Foo(30);

b.calculate(30); // 60
c.calculate(40); // 80
```



Scope

When nesting functions, inner functions contain the scope of parent functions. **This forms a tree of scopes.**

```
var x = 20;
function foo() {
  var y = 10;
  // global x accessed
  console.log(x+y); // 30
};
// y is not accessible here
function bar() {
  var x = 40; // bar uses this x, not 20
  var y = 30; // Not foo’s y; foo not a parent
  console.log(x+y); // 70
}
```



Exception to tree of scopes: this

- `this` does NOT check its parents
- All that matters is how the function was invoked
  - If the function was invoked on an object like `obj.myMethod()`, "`this`" will refer to the object `obj`
  - If the function was invoked on its own like `myMethod()`, "`this`" will be the global Window object. (Unless strict mode)

Guide:
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/this

this and events

```
function Student(givenName){
    this.givenName = givenName;
    this.element = document.getElementById(givenName + "_button");

    this.element.onclick = function() {

        console.log(this.givenName);
    }
}

<button id="John_button">John</button>

var s1 = new Student("John");
```

When you click on “John_button”, what gets printed?

this and events

```
function Student(givenName){
    this.givenName = givenName;
    this.element = document.getElementById(givenName + "_button");

    this.element.onclick = function() {
        // this == this.element == <button id="John_button">
        console.log(this.givenName);
    }
}

<button id="John_button">John</button>

var s1 = new Student("John");
// returns undefined
```

this takes on the value of the object the function is invoked on.

The caller can also explicitly set the value of this via a utility function like `Function.prototype.call`

Outline

- Project 3 Tips
- Document Object Model
- Some JS Concepts
- **Events**

Events

- Attaching events
- The **event** object
- List of common events

Attaching Event Listeners

addEventListener

```
var elem = document.getElementById("myButton");
elem.addEventListener("click", function(evt) { alert("clicked"); });
```

attribute in HTML

```
<input type=”button” onclick="alert('clicked');">
```

property of DOM element

```
var elem = document.getElementById("myButton");
elem.onclick = function(evt) { alert("clicked"); };
```

Attaching Event Listeners

- Using `addEventListener` is preferred
  - Allows you to add more than one listener per event
  - Allows finer control over the phases where listener gets activated

```
elem.addEventListener('click', handler)

elem.removeEventListener('click', handler)
```

In addition: `elem.addEventListener(‘click’, handler, useCapture)`
If useCapture is set to true, handler will be activated during capture (trickle-down) phase instead of bubble down phase when false (default). More info at: http://bit.ly/1LSyATE

Attaching Event Listeners

- Difference between `addEventListener` and DOM properties - overwriting DOM properties

```
function clickListener(evt) { alert("clicked 1"); };
function clickListener2(evt) { alert("clicked 2"); };

var element = document.getElementById("mybutton");
element.onclick = clickListener;
element.onclick = clickListener2;
element.click(); // Alerts “clicked 2”
```

Attaching Event Listeners

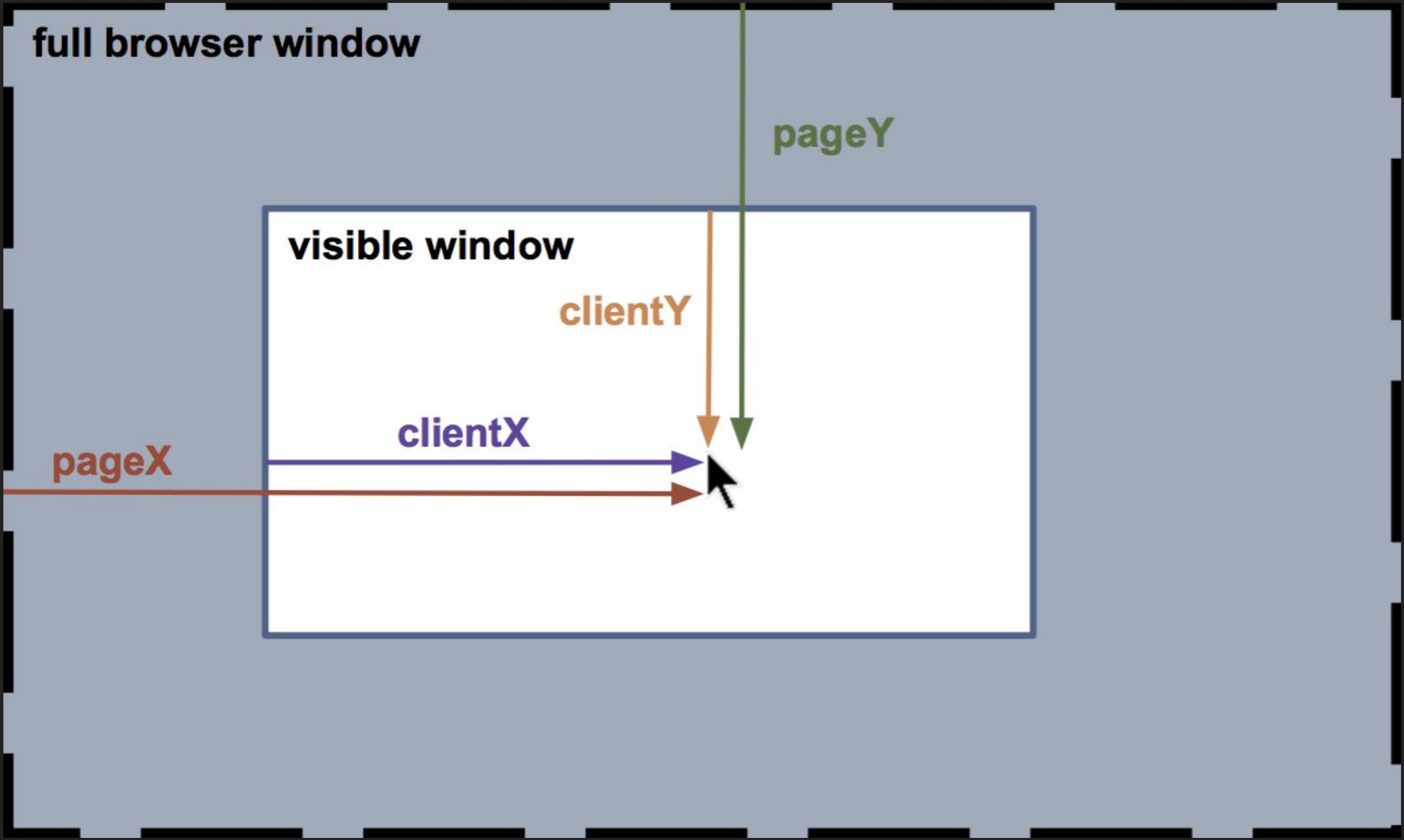
- Difference between `addEventListener` and DOM properties - Multiple event listeners

```
function clickListener(evt) { alert("clicked 1"); };
function clickListener2(evt) { alert("clicked 2"); };

var element = document.getElementById("mybutton");
element.addEventListener("click", clickListener);
element.addEventListener("click", clickListener2);
element.click(); // Alerts "clicked 1" and "clicked 2" in that
order
```

The Event Object

- The event object is passed to event listener as an argument when the event occurs
- Properties:
  - **button** - an integer indicating which mouse button is pressed
  - **clientX, clientY** - the mouse coordinates relative to the upper left corner of the current window
    - be careful about scrolling!
  - **pageX, pageY** - the mouse coordinates relative to the whole document

Event Coordinates



The Event Object

Potentially Useful Methods

- `preventDefault()` - cancel the default action of the event, e.g. `<a href="...">`
- `stopPropagation()` - stops the bubbling of an event to parent elements

```
function handler(evt) { evt.preventDefault(); };

var img = document.getElementById("image");
// With this listener the image won't be dragged
img.addEventListener('mousedown', handler);
```

Example: http://jsfiddle.net/W6zT7/

Side note: JS Script Tag Placement

- JS should wait until DOM has been created: put JS script tags at the bottom of <body>

```
<body>
   ...
   <script type="text/javascript" src="DatePicker.js"></script>
   <script type="text/javascript">
      //<![CDATA[
         var datePicker = new DatePicker("div1",
          function (id, fixedDate) { ... });
         ...
      //]]>
   </script>
</body>
```

A list of Events (for your reference)

- Mouse events
    - `onclick`
    - `onmousedown`
    - `onmousemove`
    - `onmouseup`
- Keyboard events
    - `onkeydown`
    - `onkeypress`
    - `onkeyup`
- Others
    - `onload`
    - `onsubmit`