

# CSCB20 – Week 11

*Introduction to Database and  
Web Application Programming*

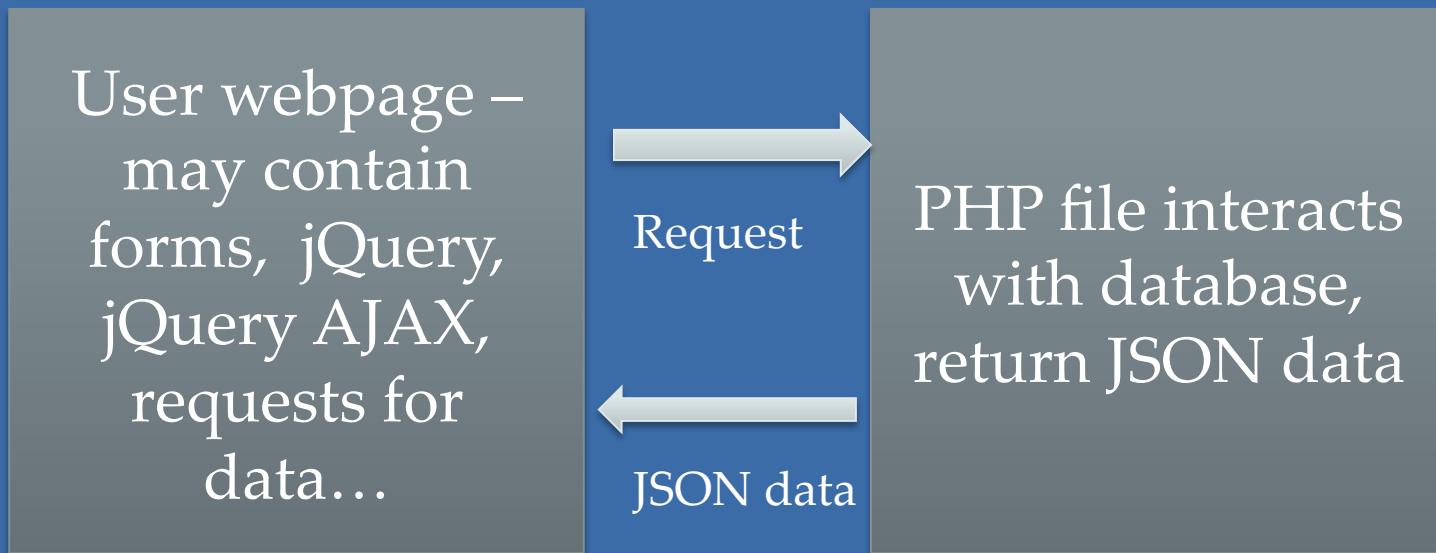
*Anna Bretscher\**

Winter 2017

\*thanks to Alan Rosselet for providing the slides these are adapted from.

-

# This Week



# HTML Forms

Most interesting Web apps are based on data, e.g.:

- Google's compilation of search indices of crawled Web content
- Amazon's product catalog and product reviews
- Facebook's user pages
- Flickr's image catalog
- YouTube's video collection

Many of these apps provide Application Programming Interfaces (API), enabling programs to work with their data, e.g. Amazon product catalog

Many apps allow users to enter their own data, e.g. upload videos to YouTube, upload photos to Flickr

Often control Web app behavior with parameter values

# HTML Forms

- Collect user input
  - for processing by client-side scripts, e.g. jQuery
  - for transmission to server-side programs
- Constructed out of widgets/controls, e.g.:
  - text boxes
  - checkboxes
  - pull-down select menus
  - radio buttons
  - submit and reset buttons
- Each widget/control has a value
- When a form is submitted, widget values are collected together and sent to server
-

# HTML Forms: <form>

<form> element encompasses entire input-form structure

```
<form action="URL" method="GET" >
```

Attributes:

- o action

- identifies URL of server-side program to be invoked in response to submit

- o method

- either “GET” (default) or “POST”

A single Web page may contain many forms

-

# Forms: <form>

```
<form action="search.php" method="GET" >  
  <div>  
    First Name: <input type="text" id="firstname"/>  
    <input type="submit" />  
  </div>  
</form>
```

The image shows a screenshot of a web browser displaying a simple HTML form. The form consists of a single input field labeled "First name:" and a "Submit" button. The input field is a rectangular box with a light gray border. The "Submit" button is a smaller rectangular box with a dark blue border and the word "Submit" in white text.

form fields can be wrapped in a <div> element or a <fieldset> block element.



# Forms: <input>

Example :

```
<input type="radio" id="cardtype"  
       name="cardtype" value="amex" />
```

Attributes:

- **type**: text boxes, checkboxes, radio buttons, user-defined buttons, file, hidden, password, reset and submit buttons
- **name, id**: for reference e.g. by script, HTTP query parameter
- **value**: default for checkboxes and radio buttons, initial text for others
- 
-

# Forms: <input>

Text inputs can have a placeholder value displayed within the text-input field to prompt the user, e.g.

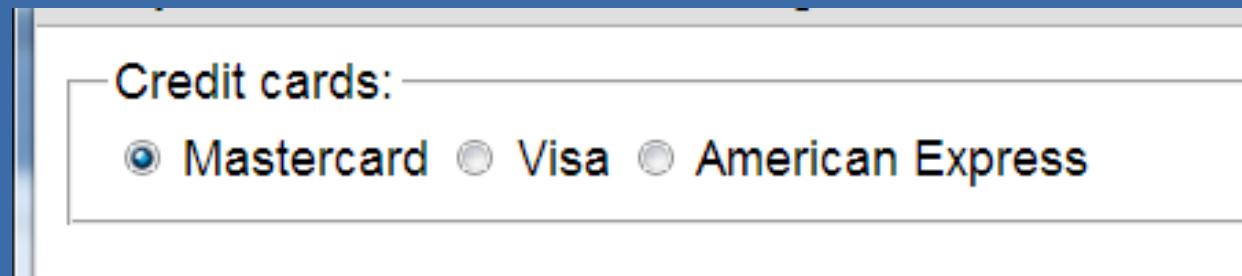
```
<input type="text" id="firstname"  
name="firstname" placeholder="First Name"/>
```

controls can be labelled with <label> element. “for” attribute connects label to matching input-field “id”.  
e.g.:

```
<label for="cardtype">Card Type</label>  
<input type="radio" id="cardtype"  
name="cardtype" value="amex" />
```

# Forms: <fieldset>, <legend>

```
<fieldset>
  <legend>Credit cards:</legend>
  <input type="radio" name="cc" value="visa"
         checked="checked" /> Visa
  <input type="radio" name="cc"
         value="mastercard" /> MasterCard
  <input type="radio" name="cc"
         value="amex" /> American Express
</fieldset>
```



**fieldset** groups related-input set and puts a border around them

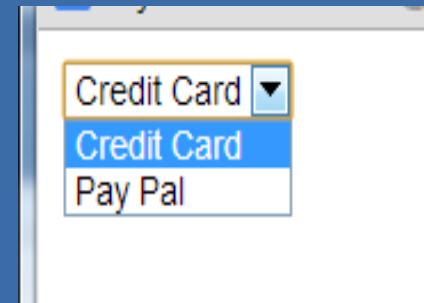
**legend** provides a caption for the form fields

- 
-

# Forms: <select>

- <select> element encompasses:
  - menus
- attributes:
  - name, id – for referencing, e.g. by script
  - size – for drop down and scrolling lists
  - multiple – allow multiple values to be selected
- <option> element children of <select> define the allowed values for the menu
- e.g.:

```
<select>
    <option value="1">Credit Card</option>
    <option value="2">Pay Pal</option>
</select>
```



# Forms: type hidden

```
<label>Firstname  
    <input type="text" name="firstname" /></label><br/>  
<label>Lastname  
    <input type="text" name="lastname" /></label><br/>  
<input type="hidden" name="fullname" />  
<input type="submit" value="go" />
```

The image shows a simple HTML form within a browser window. It contains two text input fields, one for 'Firstname' and one for 'Lastname', both of which are currently empty. Below these fields is a single button labeled 'go'.

Note hidden form fields are still passed to the server as parameter values, referenced by their name fields

-

# Forms: submit and reset types

```
<input type="reset" ... />
```

- clears all user selections/data and returns form to initial state

```
<input type="submit" value="filter" />
```

- encode all widget/control values into a string value (essentially a parameter list), called the “query string”; example:

name1=value1&name2=value2&name3=value3

- pass that string value (parameter list) to an application on the server, as specified in form “action” attribute

URL?name1=value1&name2=value2&name3=value3

- In PHP on server, e.g.:   \$\_REQUEST ["name1"]



# URL-encoding

Form data appended to URL as

?name1=value1&name2=value2&...

A few complications:

- o e.g. “=”, “&”, “|”, “/”, or “<“ character in value
- o space character in value

Browser automatically “encodes” these special cases, and on server side, PHP automatically “decodes” them

- o e.g. space character is encoded as “%20”

# HTTP: the Web's protocol

Used “behind the scenes” to send browser requests and server responses

Provides “methods” that effect how client data is passed to the server – forms have a “method” attribute to set the HTTP method.

**GET:** intended for *read-only requests*, such as static Web page retrieval, or simple server-program calls

**POST:** intended for transmitting general-case client data to the server

-

# HTTP: GET and POST

```
<form action="URL" method="GET" >
```

or

```
<form action="URL" method="POST" >
```

GET appends URL-encoded data to the end of the server URL as shown on previous slide

- o not private
- o easily tampered by user
- o length limits

POST places URL-encoded data in the “body” of the request

- o more private (not visible in browser)
- o harder to tamper
- o no length limits



# jQuery and Forms



# Detecting event types

Suppose a user makes a choice from a <select> element; how do you detect that they made a choice, and how can you determine what they chose?

```
... header with script tags ...
$( "mymenu" ).change(showMe);
function showMe() {
    alert(this.value + "was chosen");
}
</script>
<select id="mymenu">
    <option value="volvo">Volvo</option>
    <option value="saab">Saab</option>
    <option value="opel">Opel</option>
    <option value="audi">Audi</option>
</select>
```



# Dealing with forms

- An HTML **form element** wraps a **set of input types**, such as buttons, input-text fields, “radio” buttons, checkboxes

```
<form method="GET" action="search.php">  
... <input type="submit" value="go"> ... other fields  
</form>
```

- When the **user clicks** on the input element with **type “submit”**, which is displayed as a **button**, the values of the form’s fields are **submitted** to the **URL** named by the “**action**” parameter.
- Sometimes you may need to **interrupt** that **normal submit** behavior, e.g. to perform validation or some other action using the form fields.
- 
-

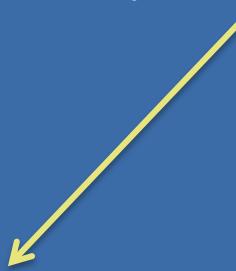
# Dealing with forms

```
<form id="f1" method="GET" action="search.php">  
... <input type="submit" value="go"> ... other fields  
</form>
```

```
$ (function () {  
    $('#f1').submit(getSomething);  
});
```

```
function getSomething(event) {  
    event.preventDefault(); // suspend submit  
    /* code to invoke myquery.php script to get  
       info from database to create <select> menu */  
}
```

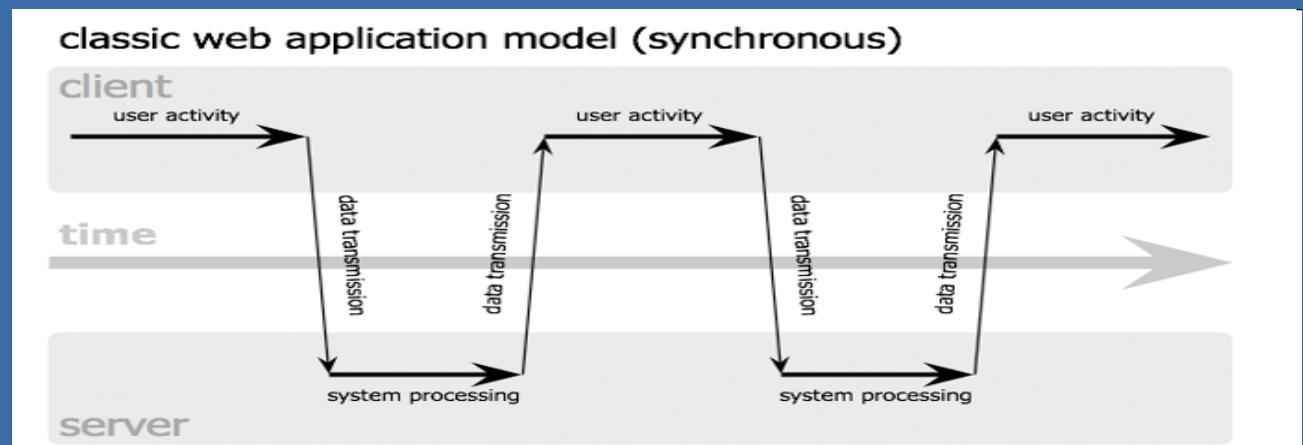
You may  
need this in  
your A3



# Ajax: Asynchronous Requests

- Traditional Web applications follow the underlying HTTP protocol model of *request and response* page-flow.
- A client Web page makes a request to the server, which results in a **new Web page** being returned by the server, replacing the original page in the browser.

The client blocks while awaiting the server response. We refer to this kind of request as **synchronous**



# Ajax: Asynchronous Requests

Nothing wrong with synchronous requests when building Web apps that consist of a sequence of page views (traditional view-click-view-click cycle)

Sometimes we want Web apps that should not block (freeze) while waiting for the server – we need a different model.

Leads to AJAX...



# Web 2.0, RIA (Rich Internet Apps)

RIA/Web-2.0 apps attempt to bridge gap between  
“native” (desktop) apps and traditional Web apps

## Goals:

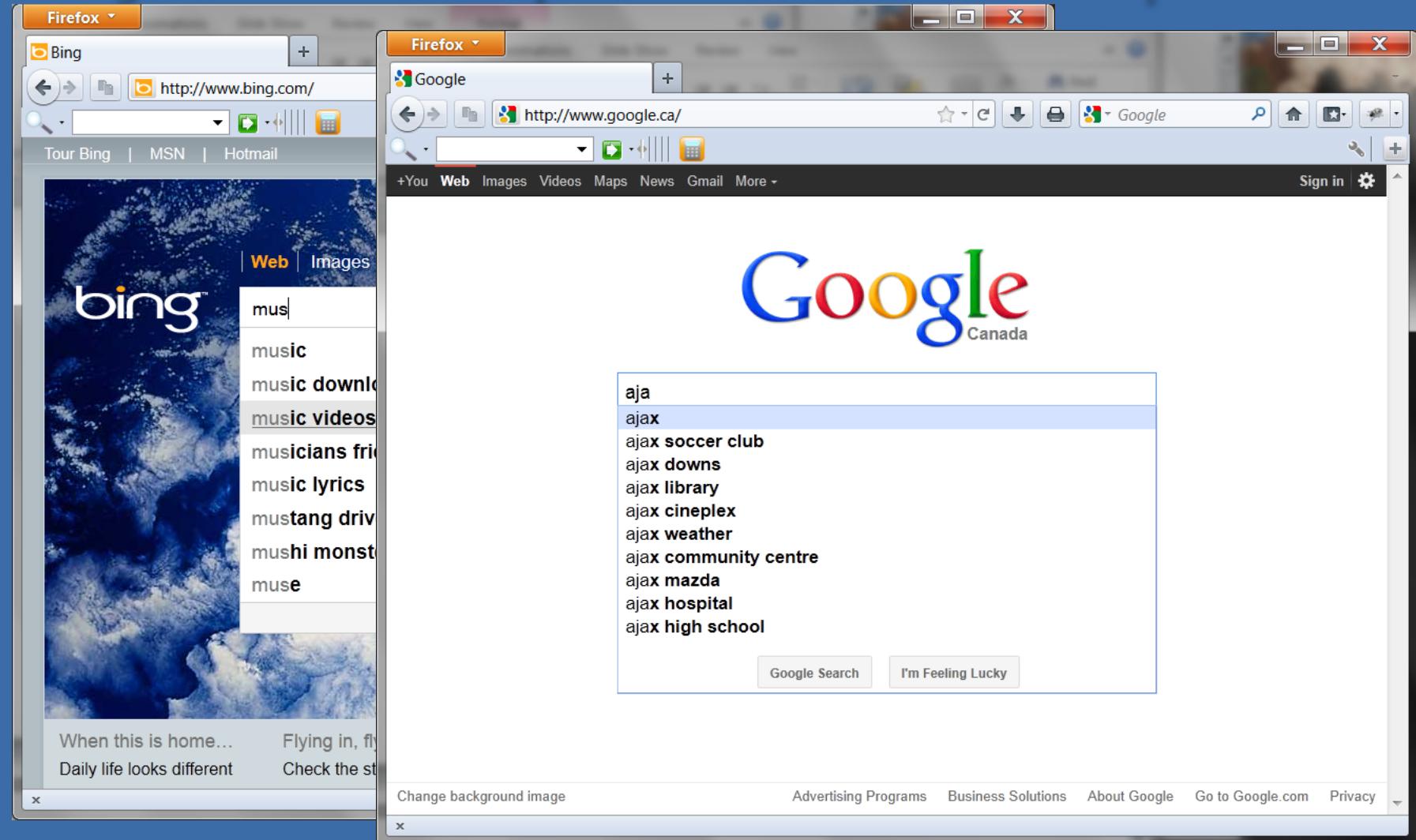
- more interactivity (can achieve some of this with JS and the DOM without server-side interaction)
- would like to achieve performance similar to native (desktop) applications

A major change is a shift from page-oriented interaction to  
“*event-driven*” programming where events result in  
updates to *portions* of pages.

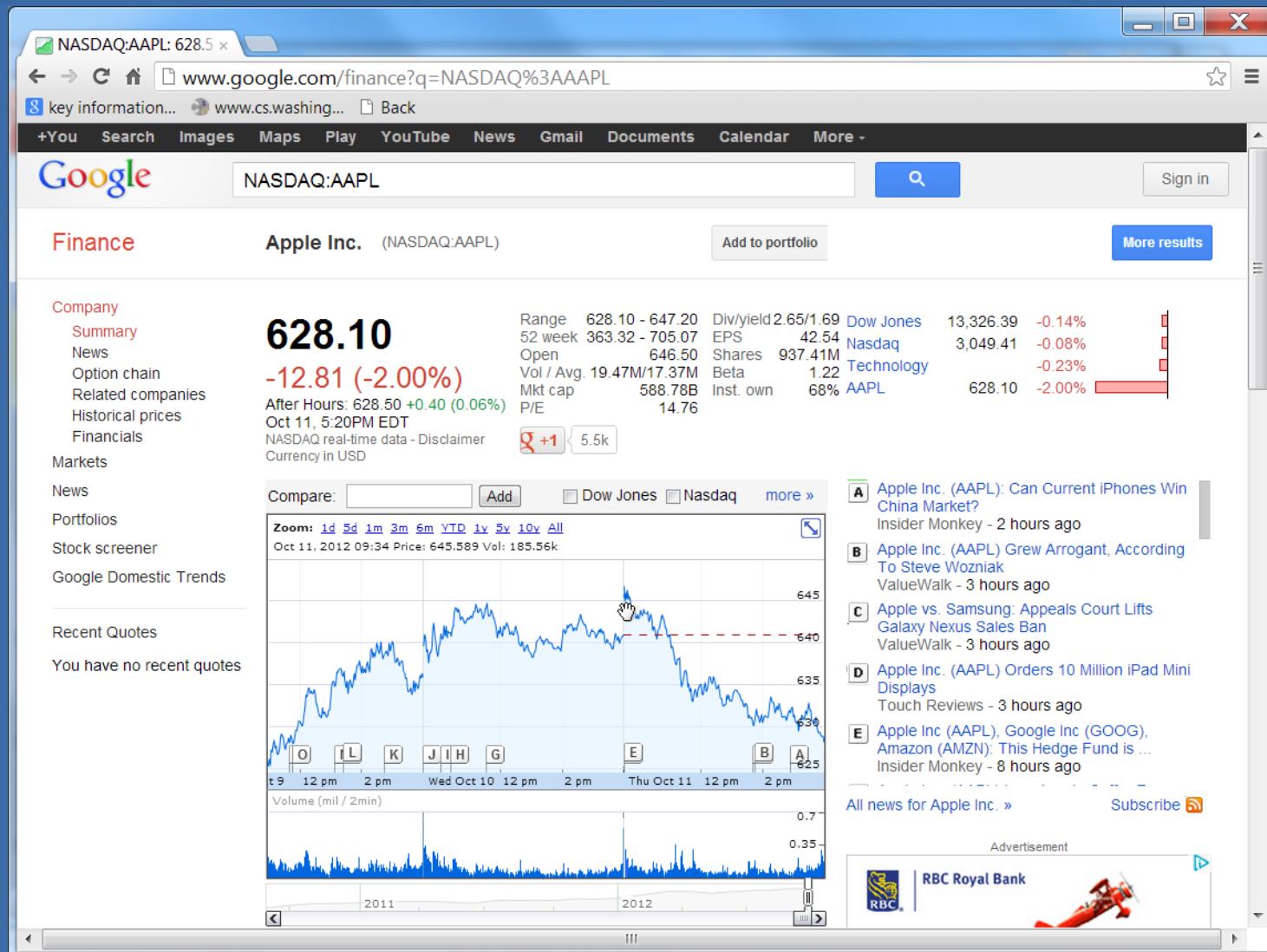
# Ajax Use

- Many mainstream apps now utilize Ajax to improve the user experience, e.g. GMail, Google maps, Google finance page, Yahoo search, Flickr
- What is Ajax used for?
  - real-time *semantic form-data validation* (not just regex checks, but actual value checks against e.g. a server DB)
  - Auto-completion (e.g. Google suggest, Yahoo search)
  - dynamic page interaction, e.g. user drag/pan results in new data display (e.g. Google maps, Google finance)
  - auto-refresh of page data (e.g., real-time stock quotes embedded within document)
  - interactive user-to-user communications, e.g. Google talk
-

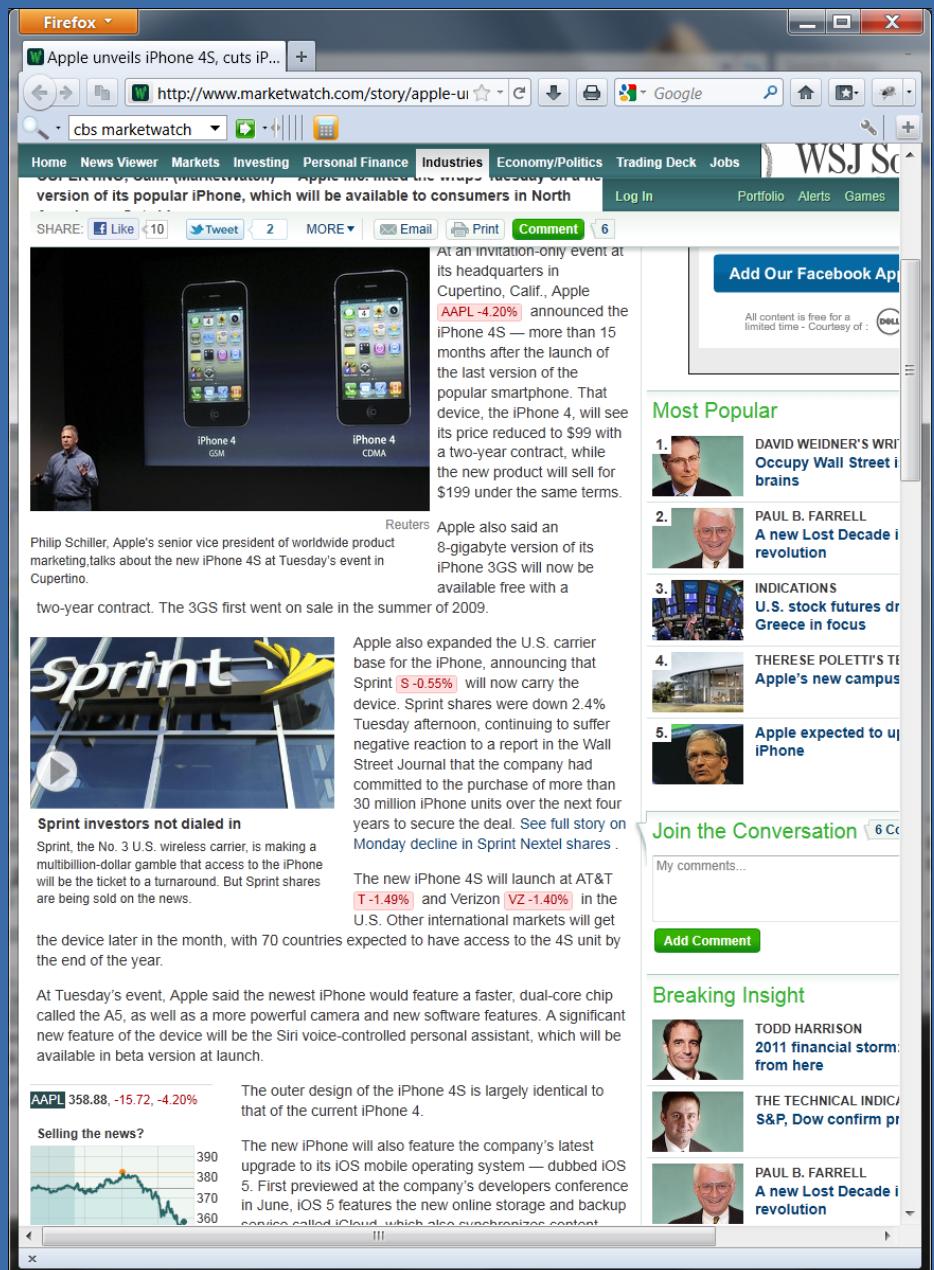
# Ajax example: autocomplete



# Ajax example: live data updates



# Ajax example: live document updates



# Ajax Interaction

What's different with Ajax?

Traditional web

- user-initiated HTTP requests
  - typing on navigation window, or clicking on form or hyperlink
- response from server replaces existing page
- low request rate, and random amount of time between requests

Ajax application

- new type of request that does not trigger page reload
- not initiated by user (at least not directly)
- requests are typically small, but more frequent



# Ajax and JSON

We will only use a few specific *jQuery Ajax* methods related to managing **JSON** data.

What is **JSON** data?

- **JSON**, **JavaScript Object Notation**, is a *string-based* representation of JavaScript array and dictionary objects
- A popular way to **communicate structured data** to (input) and from (output) Web services

# JSON Data

- A JSON dictionary is written:

```
{  
    "field0": "value0",  
    "field1": "value1",  
    ...  
}
```

- A JSON array is written:

```
[ value1, value2, ... ]
```

- Values can be primitive types such as numbers and strings, and can also be arrays and dictionaries nested arbitrarily



# JSON Data

- For example:

```
{  
  "id": "cd1",  
  "title": "Sgt. Pepper",  
  "band": "The Beatles",  
  "price": 22.00  
}
```

- Note that JSON is strict about the presence of quotes around **keys** and **values** that are strings.
- A malformed JSON string **will** not be interpreted by JavaScript and **will** fail silently



# JSON Data

JSON array of dictionaries:

```
Music = [ {"id": "cd1", "title": "Sgt. Pepper",  
          "band": "The Beatles", "price": 22.00},  
          {"id": "cd2", "title": "Hey Jude", "band":  
          "The Beatles", "price": 20.00},  
          {"id": "cd3", "title": "Greatest Hits",  
          "band": "Neil Young", "price": 21.00}]
```

Accessing the JSON data:

```
// Both ways return "Hey Jude"  
Music[1].title
```

```
Music[1]["title"]
```



# When do we use JSON Data?

When our webpage interacts with a database, we call a **php file** to read from the database.

The resulting data is returned as **JSON** data.

How do we do this?

PHP provides global functions for converting **JSON** string representation to/from a PHP (associative) array object:

```
json_decode(string)    # returns PHP object  
json_encode(object)   # returns JSON string value
```

- 
-

# Outputting JSON Data

For example, suppose the php needs to return the following:

```
<?php  
$json = array(  
    "title" => "Sgt. Pepper",  
    "band" => "The Beatles", "price" => 22.00);  
  
header ("Content-type: application/json");  
print json_encode($json);  
?>
```

The `print` statement returns the JSON data.

In our case, the calling file will be a `jQuery` script.



# jQuery AJAX

We will only use a few specific *jQuery Ajax* methods related to managing JSON data.

One AJAX method is `.getJSON` :

```
(selector).getJSON(url, param_data, my_func(data, status))
```

Function applied to data

Data sent to the url

Data returned by url

Example:

```
$ (document) .ready(function () {
    $ ("button") .click(function () {
        $.getJSON("demo_ajax_json.js", function(result) {
            $.each(result, function(i, field) {
                $ ("div") .append(field + " ");
            }) ;
        }) ;
    }) ;
}) ;
```



# jQuery AJAX

What is `$.each`?

```
$ (document) .ready(function() {  
    $("button") .click(function() {  
        $.getJSON("demo_ajax_json.js", function(result) {  
            ————→ $.each(result, function(i, field) {  
                $("div") .append(field + " ");  
            } );  
        } );  
    } );  
});
```

The `$.each(result, func())` method specifies a function to run for each element in `result`.

Another variation is:

```
$(selector) .each(function(index, element))
```

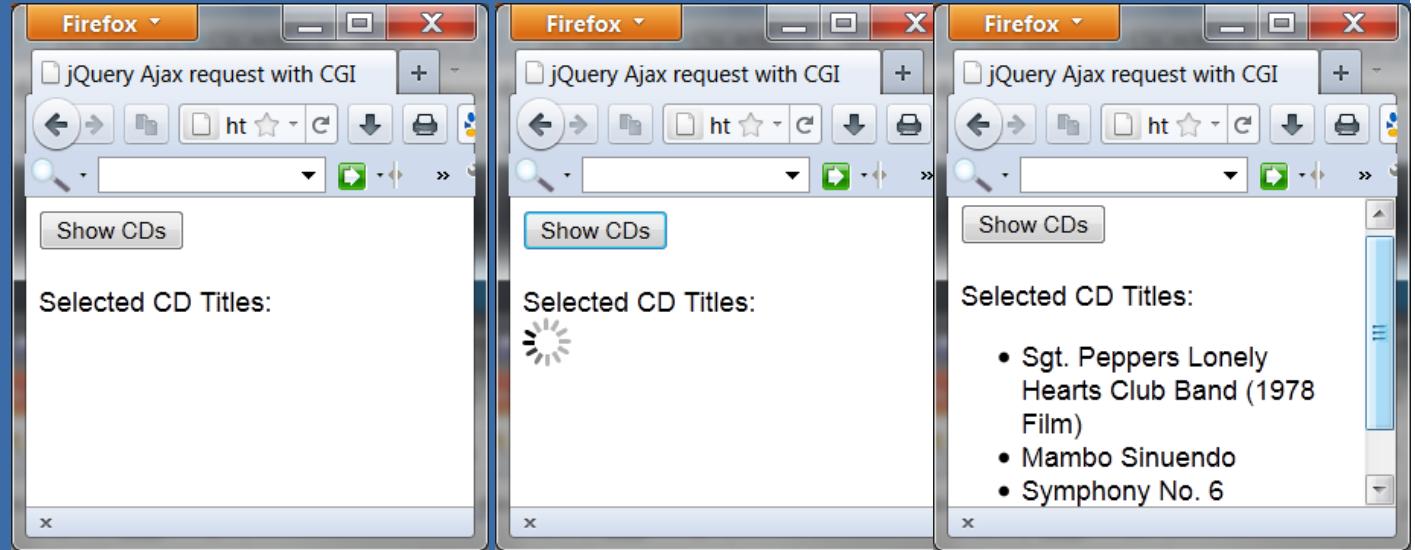
Where `function` is applied to each element matching the `selector`.



# jQuery

:

# events



- Another task, possibly useful for assignment 3: provide the user with a button, that when clicked will initiate an Ajax request to a server-side PHP service to retrieve JSON data
- We use jQuery's `getJSON()` method to call the server, retrieve the JSON result, and define a “callback” function to handle this JSON.



# jQuery Ajax callbacks

- A typical jQuery Ajax callback function binds the Ajax result value to the callback parameter, as in:

```
$.getJSON(... URL ... , function(data) { ... });
```

- Within the anonymous callback function, **data** refers to the value returned by the getJSON() Ajax call.

- If the Ajax result is a dictionary or array, you can iterate over the items in this dictionary/array using \$.each():

```
$.each(data, function(key, value) { ... }); // dict
```

```
$.each(data, function(index, value) { ... }); // array
```

- Putting the parts together:

```
$.getJSON(... URL ... , function(data) { ...
    $.each(data, function(index, value) {
        ... // index refers to current array index });
    ...
});
```

-

# Using the JSON data

```
<script type="text/javascript">
    $(document).ready(function(){
        $('.getCD').click(function() {
            $("#cdinfo").html('');
            $.getJSON('cd.php',function(data) {

                /* Iterate over the data values using 'each', invoking a function
                   to process each dictionary entry in turn */
                var display = "<ul>"
                $.each(data, function(key, val) {
                    display += '<li>' + val + '</li>';
                });
                display += '</ul>';
                $("#cdinfo").html(display);
            });
        });
    });
</script>
</head>
<body>
    <input type="button" class="getCD" value="CD Info"/>
    <br/> <br/>
    CD Info: <br/>
    <!-- div "cdinfo" is a container to hold the Ajax output -->
    <div id="cdinfo"></div>
</body>
```

# Example: DB Results as JSON

Using a database query to produce JSON results:

```
<?php  
    $conn = mysqli_connect($dbhost, $dbname,  
                           $dbuser, $dbpass);  
    $stmt = mysqli_query($conn, "SELECT * FROM  
                          Cities WHERE population > 9000000;");  
    $cities = array();  
    // create an array of all the results  
    while ($row = mysqli_fetch_assoc($stmt)) {  
        $cities[] = $row;  
  
    $json = json_encode($cities);  
    print $json;  
?>
```

```
[ {"id": "206", "name": "Sao Paulo", "country_code": "BRA",  
  "district": null, "population": "9968485"},  
  {"id": "939", "name": "Jakarta", "country_code": "IDN", "district":  
  "Jakarta Raya", "population": "9604900"}, ... ]
```

# The End

