# LBSC 690: Information Technology
# Lecture 08
# Developing Web Applications

William Webber
CIS, University of Maryland

Spring semester, 2012

# Today's coverage

- Javascript in the browser (make user-page interactive)
- Server-side programming (last piece of the web-application puzzle) (with PHP)
- AJAX (make page-server interactive)

# Coverage depth

These are quite advanced topics; each worth a mini-course.
Today's scope:

- Get a feeling for features of each.
- Understand the architecture involved.
- Develop confidence in example-based development.
- Appreciate scope for projects.

# Javascript in the browser

- Already seen that:
  - Web browsers contain Javascript interpreters.
  - We can run simple programs in them.
- Javascript code most useful when interacts with page and user.
- Interaction relies upon two concepts:
  1. Event-driven programming (code responds to user events)
  2. The document object model (code is able to manipulate web page)

# Event-driven programming

- Without user interaction, Javascript programs mostly toys

  → `http://codalism.com/~wew/lbsc690/auton.html`
- To interact with user, Javascript must repond to user-initiated actions or events:
  - Clicks
  - Key presses
  - Mouse role-overs
  - Form submissions
- Later, we'll see that Javascript can also initiate, respond to events from the server (arrival of new data, etc.)

# Events, HTML, and Javascript

```
<html>
  <head>
    <title>onClick() example</title>
  </head>

  <body>
    <input type="button" value="Click me!"
           onClick="alert('Stop clicking me!')">
  </body>

</html>
```

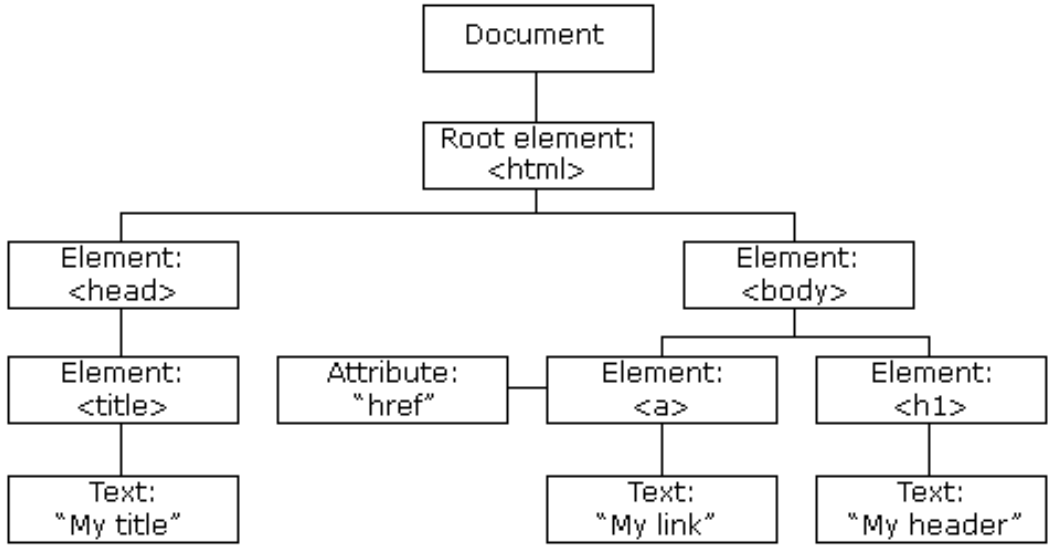
- Events are captured by on[EVENT] attributes (e.g. onClick, onMouseOver, etc.)
- Value of on[EVENT] is Javascript code that gets run when the event occurs.

→ `http://codalism.com/~wew/lbsc690/onclick.html`

# Interactive display

- Event-driven programming gives interactivity user $\rightarrow$ page.
- We want interactivity page $\rightarrow$ user.
- While a page is being output, we can use document.write("").
- After it has already been complete, we need to modify it in place.
- That ability is offered by the Document Object Model (DOM).

# Document object model (DOM)




- An HTML page can be represented as a “tree” (a kind of graph):
  - Elements are nodes
  - Enclosed elements are children of enclosing elements
  - The root element is <html>
- The DOM provides a fully general way to traverse and manipulate the whole tree
- . . . and therefore change the appearance, contents of the page

# getElementById()

- The easiest way to access an HTML element through the DOM
- . . . is to give the element an id, say 
- . . . then access it via document.getElementById("myid")

# innerHTML

```
<html>
  <head>
    <title>onClick() example</title>
  </head>

  <body>
    <h1 id="title">[[SET ME]]</h1>
    <input type="button" value="Foo"
      onClick="document.getElementById('title').innerHTML = 'Foo';">
    <input type="button" value="Bar"
      onClick="document.getElementById('title').innerHTML = 'Bar';">
  </body>

</html>
```

- To access or change the text inside an element, reference its innerHTML property.

→ `http://codalism.com/~wew/lbsc690/clickchange.html`

# Element properties

```
<html><head><title>Image shifter</title></head>
  <body>
    <script language="javascript">
      function swImg(name) {
        document.getElementById("image").src = name;
      }
    </script>
    <center>
      <img src="kitten.jpg" id="image" height="400px">
    </center>
    <div style="height: 30px"> </div>
    <center>
    <img src="kitten.jpg" height="50px" onMouseOver="swImg('kitten.jpg
    <img src="owlet.jpg" height="50px" onMouseOver="swImg('owlet.jpg')
    <img src="dog.jpg" height="50px" onMouseOver="swImg('dog.jpg')">
    <img src="flea.jpg" height="50px" onMouseOver="swImg('flea.jpg')">
    </center>
  </body></html>
```

- Attributes of HTML element generally available as properties of the DOM object
- Note use of function to handle extended or repeated code

→ http://codalism.com/~wew/lbsc690/imageshift.html

# Three-tier Web Application Development

Revisiting the three-tier web application

| | |
|---|---|
| Presentation | *HTML, CSS, Javascript* |
| Logic | *???* |
| Data | *RDBMS, SQL* |

We still have to fill in the “logic” layer.

# Responsibilities of logic layer

In simple application, logic layer is responsible for coordinating interaction between presentation (web pages) and data (database), namely:

- Pulling information out of database, sending to display (HTML)
- Taking information from display (HTML forms), entering into database

More complicated logic is also possible (e.g. interacting with other web services)

# Implementing logic level

- Logic level runs (generally) on the server side:
  - Closer to data
  - Less code to download to client
  - Can’t trust client to run code securely
- To implement logic level, we need to program
  - because programming is a systematic encoding of logic
  - just as HTML is a systematic encoding of presentation
  - and SQL is a systematic encoding of data (and its manipulations)
- Web server must be equipped and configured to handle server-side programming.
  - The terpconnect server is not (apart from SSI)

# PHP as server-side programming language

- In theory, Javascript could be used for server-side programming, too
- In practice, for historical reasons, it is poorly suppoted for general server-side web app development
  - though it is gaining popularity in certain uses; see `node.js`
- A much more popular server-side web programming language is PHP
- We’re going to learn (to read) PHP by example (deep breath!)

# PHP in HTML

```
<html>
  <head>
    <title>Hello, world!</title>
  </head>

  <body>
    <?php echo "Hello, world!"; ?>
  </body>
</html>
```

- PHP code can be embedded in HTML.
  - (although it is more accurate to say that the HTML is embedded in PHP)
- The embedded code goes in <?php ... ?>
- This code is executed when read by the PHP interpreter
- Note similarity to processing of SSI includes

→ `http://codalism.com/~wew/lbsc690/helloworld1.php`

# Filesystem, server-side and client-side

```
<html>
  <head>
    <title>Hello, world!</title>
  </head>

  <body>
    <?php echo "Hello, world!"; ?>
    <br>
    <script language="javascript">
    document.write("Goodbye, world!")</script>
  </body>
</html>
```

- The file sits on the server filesystem as above.
- Webserver reads file, invokes the PHP processor, which replaces the <?php ... ?> expressions, on the server side.
- The HTML, with the Javascript embedded, is then sent to the web browser.
- The web browser invokes the Javascript interpreter, which execues the Javascript statement, on the client side.

→ http://codalism.com/~wew/lbsc690/helloworld.php

# From database to HTML

Server-side PHP program to display data from database has following basic form:

1. Connext to database
2. Construct query in SQL
3. Submit query to database and read results
4. Write results out in HTML

# From database to HTML

```
<html>
  <head> <title >Drawing</title > </head>
  <body>
    <h1>Drawings</h1>

    <table cellpadding="2px" border="1">
      <tr>
        <th>Title </th> <th>Student</th> <th>Date</th> <th>Image</th>
       </tr>
<?php
$db = mysql_connect('localhost', 'william', 'pw4william');
if (!$db) {
    die("Cannot connect to DBMS: " . mysql_error());
}
mysql_select_db('william', $db);

$qry = "SELECT Drawing.title, Drawing.date_drawn, Drawing.url, Student.given, "
  . "Student.family  FROM Drawing, Student WHERE Drawing.student_id=Student.id";
$res = mysql_query($qry);

while ( ($drawing = mysql_fetch_assoc($res))) {
?>
      <tr>
        <td><?php echo $drawing["title"]; ?></td>
        <td><?php echo $drawing["given"] . " " . $drawing["family"]; ?></td>
        <td><?php echo $drawing["date_drawn"]; ?></td>
        <td><img height="50px" src="<?php echo $drawing["url"]; ?>"></td>
       </tr>
<?php } ?>
    </table>
  </body>
</html>
```

→ http://codalism.com/~wew/lbsc690/drawings.php

# HTML forms

- User input in HTML through HTML forms
- <form> tag bounds form
- Contents of form given by <input> tags (<select> for drop-down list; <textarea> for text)
- Each input has a name attribute
- When submitted, name:value pairs sent by browser to server

# HTML forms

```
<html>
  <head>
    <title>Add Drawing</title>
  </head>
  <body>
    <h1>Add Drawing</h1>

    <span style="color:red"><?php if (!empty($error)) { echo $error; }

    <form action="do_add_drawing.php" method="post">
      Student id: <input type="text" name="student_id"><br>
      Title: <input type="text" name="title"><br>
      Date drawn: <input type="text" name="date_drawn"><br>
      URL: <input type="text" name="url" size="40"><br>
      <input type="submit">
    </form>

  </body>
</html>
```

→ http://codalism.com/~wew/lbsc690/add_drawing.php

# Browser to database

- Name:value pairs of form are made available to PHP script in $_REQUEST object
- PHP script:
  1. extracts form properties request
  2. performs error checking
  3. create SQL insert or update query
  4. submits to database

# Browser to database

```
<?php
$db = mysql_connect('localhost', 'william', 'pw4william');
$res = mysql_select_db('william', $db);

if (empty($_REQUEST["title"])) {
    $error = "You must supply a title!";
    include "add_drawing.php";
    exit;
}

$qry = "INSERT INTO Drawing (title, student_id, date_drawn, url) "
    . "VALUES ('" . mysql_real_escape_string($_REQUEST["title"])
    . "', " . mysql_real_escape_string($_REQUEST["student_id"])
    . ", '" . mysql_real_escape_string($_REQUEST["date_drawn"])
    . "', '" . mysql_real_escape_string($_REQUEST["url"]) . "')";

$res = mysql_query($qry);
if (!$res) {
    die("Error with query: " . mysql_error());
}

include "drawings.php";
?>
```

→ http://codalism.com/~wew/lbsc690/do_add_drawing.php

# The request-response model

- HTTP is a request-response mechanism
- In traditional operation:
  - The request was for a web page
  - The response was the full web page
- Every time the user and server want to interact, a new page is loaded
- Becomes very clunky or impossible for interactive web applications.

# Beyond request-response

Alternative approach is AJAX (Asynchronous Javascript And XML)

- Web page is loaded only once (with sizeable amount of Javascript code)
- When data on page needs to change:
  - Javascript in page sends request (over HTTP) to server for new data
  - Server sends response (to Javascript) containing data
  - Javascript in page rewrites just the part of the page that is updated
- Note that response comes to Javascript as an event (“asynchronously”), rather than Javascripts waiting, to maximize interactivity

# AJAX example: Google Maps

Google Maps is an example of application written with AJAX.

In traditional request-response model

- Either whole page would have to change when scrolling, zooming
- Or all map panes for entire world at every resolution would have to be loaded when page loaded

In AJAX model

- Only map panes for initial location loaded immediately
- Newly required panes loaded on demand, when user scrolls, zooms

# AJAX example: Google search typeahead

Google search provides type-ahead suggestions

- As you type, current query goes to backend server
- Server calculates, send suggestions based on prefix
- Browser adds these to suggestion list
- Also, dynamically updates page with results for partial query

Note that typing is not delayed, and results don't appear immediately.