CSE 154

LECTURE 18: FORMS AND UPLOADING FILES

“This is interesting, 70% of the respondents to our survey said they don’t respond to surveys.”
Exercise: Baby name web service JSON

- Modify our `babynames.php` service to produce its output as JSON. For the data:

<table>
<thead>
<tr>
<th>Name</th>
<th>Gender</th>
<th>Rankings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Morgan</td>
<td>m</td>
<td>[375, 410, 392, 478, 579, 507, 636, 499, 446, 291, 278, 332, 518]</td>
</tr>
</tbody>
</table>

- The service should output the following JSON:

  ```json
  {
    "name": "Morgan",
    "gender": "m",
    "rankings": [375, 410, 392, 478, 579, 507, 636, 499, 446, 291, 278, 332, 518]
  }
  ```
Emitting JSON data manually

```php
...  
header("Content-type: application/json");
print "\n";
print " \"books\": [\n";
foreach ($books as $book) {
    print "   \"author\": \"{$book[\'author\']}\", \"title\": \"{$book[\'title\']}\"\n";
}
print "\n";
```

- specify a content type of **application/json**
- messy, just like when manually printing XML (not recommended)
PHP's JSON functions

PHP includes the following global functions for interacting with JSON data:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>json_decode(string)</code></td>
<td>parses the given JSON data string and returns an equivalent associative array object (like JSON.parse in JavaScript)</td>
</tr>
<tr>
<td><code>json_encode(object)</code></td>
<td>returns JSON equivalent for the given object or array or value (like JSON.stringify in JavaScript)</td>
</tr>
</tbody>
</table>

- `json_encode` will output associative arrays as objects and normal arrays as arrays
<?php
$data = array(
    "library" => "Odegaard",
    "category" => "fantasy",
    "year" => 2012,
    "books" => array(
        array("title" => "Harry Potter", "author" => "J.K. Rowling"),
        array("title" => "The Hobbit", "author" => "J.R.R. Tolkien"),
        array("title" => "Game of Thrones", "author" => "George R. R. Martin"),
        array("title" => "Dragons of Krynn", "author" => "Margaret Weis"),
    )
);

header("Content-type: application/json");
print json_encode($data);
PHP JSON example - output

```json
{
    "library": "Odegaard",
    "category": "fantasy",
    "year": 2012,
    "books": [
        {
            "title": "Harry Potter", "author": "J.K. Rowling"
        },
        {
            "title": "The Hobbit", "author": "J.R.R. Tolkien"
        },
        {
            "title": "Game of Thrones", "author": "George R. R. Martin"
        },
        {
            "title": "Dragons of Krynn", "author": "Margaret Weis"
        }
    ]
}
```
HTML forms

- **form**: a group of UI controls that accepts information from the user and sends the information to a web server

- The information is sent to the server as a **query string**

- JavaScript can be used to create interactive controls (seen later)
HTML form: <form>

```
<form action="destination URL">
  form controls
</form>
```

- required action attribute gives the URL of the page that will process this form's data
- when form has been filled out and submitted, its data will be sent to the action's URL
- one page may contain many forms if so desired
Reset buttons

- when clicked, returns all form controls to their initial values
- specify custom text on the button by setting its value attribute

```html
Name: <input type="text" name="name" />
Food: <input type="text" name="meal" value="pizza" />
<label>Meat? <input type="checkbox" name="meat" /></label>
<input type="reset" />
```
Hidden input parameters

- an invisible parameter that is still passed to the server when form is submitted
- useful for passing on additional state that isn't modified by the user
HTTP GET vs. POST requests

- GET: asks a server for a page or data
  - if the request has parameters, they are sent in the URL as a query string

- POST: submits data to a web server and retrieves the server's response
  - if the request has parameters, they are embedded in the request's HTTP packet, not the URL

- For submitting data to be saved, POST is more appropriate than GET
  - GET requests embed their parameters in their URLs
  - URLs are limited in length (~ 1024 characters)
  - URLs cannot contain special characters without encoding
  - private data in a URL can be seen or modified by users
Form POST example

```html
<form action="http://foo.com/app.php" method="post">
  <div>
    Name: <input type="text" name="name" /> <br/>
    Food: <input type="text" name="meal" /> <br/>
    <label>Meat? <input type="checkbox" name="meat" /></label> <br/>
    <input type="submit" />
  </div>
</form>
```

HTML output

Name: 
Food: 
Meat?  
Submit Query
The htmlspecialchars function

htmlspecialchars returns an HTML-escaped version of a string

- text from files / user input / query params might contain <, >, &, etc.
- we could manually write code to strip out these characters
- better idea: allow them, but escape them

$text = "<p>hi 2 u & me</p>";
$text = htmlspecialchars($text);   # "&lt;p&gt;hi 2 u &amp; me&lt;/p&gt;"
Uploading files

```
    Upload an image as your avatar:
    <input type="file" name="avatar" />
    <input type="submit" />
</form>
```

- add a file upload to your form as an input tag with type of file
- must also set the enctype attribute of the form
Processing an uploaded file in PHP

- uploaded files are placed into global array $_FILES, not $_POST
- each element of $_FILES is itself an associative array, containing:
  - name : the local filename that the user uploaded
  - type : the MIME type of data that was uploaded, such as image/jpeg
  - size : file's size in bytes
  - tmp_name : a filename where PHP has temporarily saved the uploaded file
  - to permanently store the file, move it from this location into some other file
### Uploading details

<table>
<thead>
<tr>
<th>HTML</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;input type=&quot;file&quot; name=&quot;avatar&quot; /&gt;</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Browse... No file selected. Submit Query</td>
</tr>
</tbody>
</table>

- example: if you upload borat.jpg as a parameter named avatar,
  - `$_FILES["avatar"]["name"]` will be "borat.jpg"
  - `$_FILES["avatar"]["type"]` will be "image/jpeg"
  - `$_FILES["avatar"]["tmp_name"]` will be something like "/var/tmp/phpZtR4TI"
Processing uploaded file, example

```php
$username = $_POST["username"];
if (is_uploaded_file($_FILES["avatar"]['tmp_name'])) {
    move_uploaded_file($_FILES["avatar"]['tmp_name'],
                        "$username/avatar.jpg");
    print "Saved uploaded file as $username/avatar.jpg\n";
} else {
    print "Error: required file not uploaded";
}
```

- functions for dealing with uploaded files:
  - is_uploaded_file(filename)
  - returns TRUE if the given filename was uploaded by the user
  - move_uploaded_file(from, to)
  - moves from a temporary file location to a more permanent file
- proper idiom: check is_uploaded_file, then do move_uploaded_file