Web Programming Step by Step

Lecture 10
More HTML Forms; Posting Data
Reading: 6.3 - 6.5

Reset buttons (6.2.7)

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

Name:
Food: pizza
Meat?
Reset Submit Query

output

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

Grouping input: <fieldset>, <legend> (6.2.8)

groups of input fields with optional caption (block)

```
<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>
```

HTML

Credit cards:

Visa MasterCard American Express

Submit Query

output

- `fieldset` groups related input fields, adds a border; `legend` supplies a caption

Common UI control errors

- "I changed the form's HTML code ... but when I refresh, the page doesn't update!"
  - By default, when you refresh a page, it leaves the previous values in all form controls
  - it does this in case you were filling out a long form and needed to refresh/return to it
  - if you want it to clear out all UI controls' state and values, you must do a **full refresh**
    - Firefox: Shift-Ctrl-R
    - Mac: Shift-Command-R

Styling form controls (6.2.9)

```
element[attribute="value"] {
  property : value;
  property : value;
  ...
  property : value;
}
```
CSS

```
input[type="text"] {
  background-color: yellow;
  font-weight: bold;
}
```
CSS

Borat

output

- **attribute selector**: matches only elements that have a particular attribute value
- useful for controls because many share the same element (`input`)

Hidden input parameters (6.3.2)

```
<input type="text" name="username" /> Name <br />
<input type="text" name="sid" /> SID <br />
<input type="hidden" name="school" value="UW" />
<input type="hidden" name="year" value="2048" />
```
HTML

Name
SID
Submit Query

output

- 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

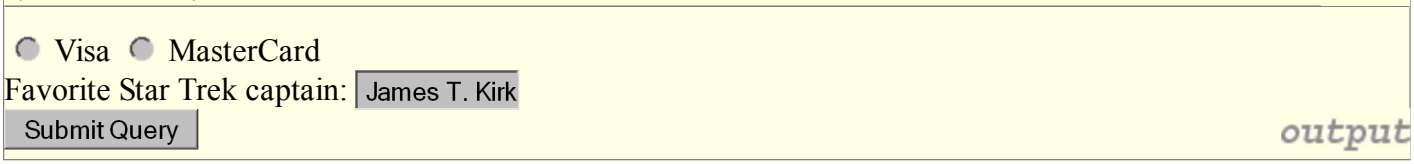
6.3: Submitting Data

- 6.1: Form Basics
- 6.2: Form Controls
- **6.3: Submitting Data**
- 6.4: Processing Form Data in PHP

Problems with submitting data

```
<label><input type="radio" name="cc" /> Visa</label>
<label><input type="radio" name="cc" /> MasterCard</label> <br />
Favorite Star Trek captain:
<select name="startrek">
  <option>James T. Kirk</option>
  <option>Jean-Luc Picard</option>
</select> <br />
```

HTML



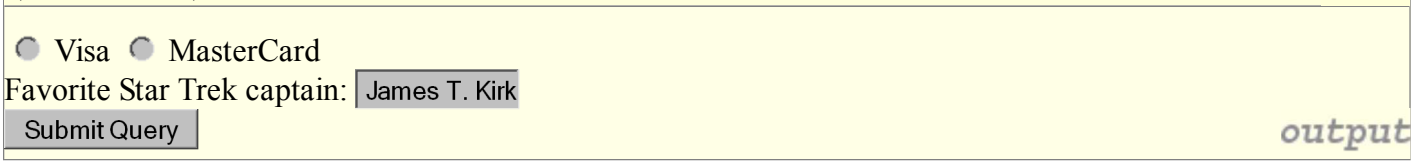
output

- this form submits to our handy params.php tester page
- the form may look correct, but when you submit it...
- `[cc] => on, [startrek] => Jean-Luc Picard`

The `value` attribute

```
<label><input type="radio" name="cc" value="visa" /> Visa</label>
<label><input type="radio" name="cc" value="mastercard" /> MasterCard</label> <br />
Favorite Star Trek captain:
<select name="startrek">
  <option value="kirk">James T. Kirk</option>
  <option value="picard">Jean-Luc Picard</option>
</select> <br />
```

HTML



output

- `value` attribute sets what will be submitted if a control is selected
- `[cc] => visa, [startrek] => picard`

URL-encoding (6.3.1)

- certain characters are not allowed in URL query parameters:
  - examples: `" "`, `"/"`, `"="`, `"&"`
- when passing a parameter, it is **URL-encoded** (reference table)
  - `"Marty's cool!?"` → `"Marty%27s+cool%3F%21"`
- you don't usually need to worry about this:
  - the browser automatically encodes parameters before sending them
  - the PHP `$_REQUEST` array automatically decodes them
  - ... but occasionally the encoded version does pop up (e.g. in Firebug)

Submitting data to a web server

- though browsers mostly retrieve data, sometimes you want to submit data to a server
  - Hotmail: Send a message
  - Flickr: Upload a photo
  - Google Calendar: Create an appointment
- the data is sent in HTTP requests to the server
  - with HTML forms
  - with **Ajax** (seen later)
- the data is placed into the request as parameters

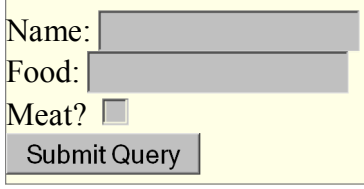
HTTP `GET` vs. `POST` requests (6.3.3)

- **`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, a `POST` request is more appropriate than a `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

```
<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

GET or POST?

```
if ($_SERVER["REQUEST_METHOD"] == "GET") {
  # process a GET request
  ...
} elseif ($_SERVER["REQUEST_METHOD"] == "POST") {
  # process a POST request
  ...
}
```

PHP

- some PHP pages process both GET and POST requests
- to find out which kind of request we are currently processing, look at the global `$_SERVER` array's `"REQUEST_METHOD"` element

Uploading files (6.3.4)

```
<form action="http://webster.cs.washington.edu/params.php"
      method="post" enctype="multipart/form-data">
  Upload an image as your avatar:
  <input type="file" name="avatar" />
  <input type="submit" />
</form>
```

HTML

Upload an image as your avatar: Browse... Submit Query

output

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

- it makes sense that the form's request method must be `post` (an entire file can't be put into a URL!)
- form's `enctype` (data encoding type) must be set to `multipart/form-data` or else the file will not arrive at the server

6.4: Processing Form Data in PHP

- 6.1: Form Basics
- 6.2: Form Controls
- 6.3: Submitting Data
- **6.4: Processing Form Data in PHP**

"Superglobal" arrays (6.4.1)

Array	Description
`$_REQUEST`	parameters passed to any type of request
`$_GET`, `$_POST`	parameters passed to GET and POST requests
`$_SERVER`, `$_ENV`	information about the web server
`$_FILES`	files uploaded with the web request
`$_SESSION`, `$_COOKIE`	"cookies" used to identify the user (seen later)

- PHP **superglobal** arrays contain information about the current request, server, etc.:
- These are special kinds of arrays called **associative arrays**.

Associative arrays (6.4.1)

```php
$blackbook = array();
$blackbook["marty"] = "206-685-2181";
$blackbook["stuart"] = "206-685-9138";
...
print "Marty's number is " . $blackbook["marty"] . ".\n";
```

- **associative array** (a.k.a. **map**, **dictionary**, **hash table**) : uses non-integer indexes
- associates a particular index "key" with a value
  - key `"marty"` maps to value `"206-685-2181"`
- syntax for embedding an associative array element in interpreted string:

```php
print "Marty's number is {$blackbook['marty']}.\n";
```

Processing an uploaded file in PHP (6.4.3)

- uploaded files are placed into global array `$_FILES`, not `$_REQUEST`
- 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

```
<input type="file" name="avatar" />
```
HTML

output

- 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 = $_REQUEST["username"];
if (is_uploaded_file($_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`

Including files: `include` (5.4.2)

```php
include("filename");
```

```php
include("header.php");
```

- inserts the entire contents of the given file into the PHP script's output page
- encourages modularity
- useful for defining reused functions needed by multiple pages