CSE 154

LECTURE 9: SUBMITTING DATA (POST)
Drop-down list: `<select>`, `<option>`

*menus of choices that collapse and expand (inline)*

```html
<select name="favoritecharacter">
  <option>Jerry</option>
  <option>George</option>
  <option selected="selected">Kramer</option>
  <option>Elaine</option>
</select>
```

- option element represents each choice
- select optional attributes: disabled, multiple, size
- optional selected attribute sets which one is initially chosen

**Output**
Using `<select>` for lists

```html
<select name="favoritecharacter[]" size="3" multiple="multiple">
  <option>Jerry</option>
  <option>George</option>
  <option>Kramer</option>
  <option>Elaine</option>
  <option selected="selected">Newman</option>
</select>
```

- optional multiple attribute allows selecting multiple items with shift- or ctrl-click
  - must declare parameter's name with [] if you allow multiple selections
- option tags can be set to be initially selected

```
Kramer
Elaine
Newman
```

Submit Query
Option groups: `<optgroup>`

```html
<select name="favoritecharacter">
  <optgroup label="Major Characters">
    <option>Jerry</option>
    <option>George</option>
    <option>Kramer</option>
    <option>Elaine</option>
  </optgroup>
  <optgroup label="Minor Characters">
    <option>Newman</option>
    <option>Susan</option>
  </optgroup>
</select>
```

- What should we do if we don't like the bold appearance of the optgroups?
Grouping input: `<fieldset>`, `<legend>`

Groups of input fields with optional caption (block)

```html
<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>
```

- Visa ○ MasterCard ○ American Express

- fieldset groups related input fields, adds a border; legend supplies a caption
Styling form controls

```css
/* attribute selector: matches only elements that have a particular attribute value */
element[attribute="value"] { 
    property : value;
    property : value;
    ...
    property : value;
}

/* useful for controls because many share the same element (input) */
input[type="text"] { 
    background-color: yellow;
    font-weight: bold;
}
```

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

• certain characters are not allowed in URL query parameters:
  - examples: "", "/", ";", "&"

• when passing a parameter, it is URL-encoded (reference table)
  - “Allison's cool!?" → “Allison%27s+cool%3F%21"

• you don't usually need to worry about this:
  - the browser automatically encodes parameters before sending them
  - the PHP $_GET and $_POST arrays automatically decode them
  - ... but occasionally the encoded version does pop up (e.g. in Firebug)
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>
```
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

```html
  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

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`