6.1: Form Basics

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

- most interesting web pages revolve around data
  - examples: Google, IMDB, Digg, Facebook, YouTube, Rotten Tomatoes
  - can take many formats: text, HTML, XML, multimedia
- many of them allow us to access their data
- some even allow us to submit our own new data
- most server-side web programs accept parameters that guide their execution

Query strings and parameters (6.1.1)

URL?name=value&name=value...


- query string: a set of parameters passed from a browser to a web server
  - often passed by placing name/value pairs at the end of a URL
  - above, parameter username has value stepp, and sid has value 1234567
- PHP code on the server can examine and utilize the value of parameters
**HTML forms**

- **form**: a group of UI controls that accepts information from the user and sends the information to a web server
- forms use HTML UI controls (buttons, checkboxes, text fields, etc.)
- the information is sent to the server as a **query string**
- JavaScript can be used to create interactive controls (seen later)

**HTML form: `<form>` (6.1.2)**

```html
<form action="web service URL">
  form controls
</form>
```

- required `action` attribute gives the URL of the server web service that will process this form's data
Form example

```html
<form action="http://www.google.com/search">
  <div>
    Let's search Google:
    <input name="q" />
    <input type="submit" />
  </div>
</form>
```

- should wrap the form's controls in a block element such as `div`

Form controls: `<input>`

```html
<input type="text" name="q" value="Colbert Report" />
<input type="submit" value="Booyah!" />
```

- `input` element is used to create many UI controls
  - an inline element that MUST be self-closed
- `name` attribute specifies name of query parameter to pass to server
- `type` can be `button`, `checkbox`, `file`, `hidden`, `password`, `radio`, `reset`, `submit`, `text`, ...
- `value` attribute specifies control's initial text
6.2: Form Controls

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

**Text fields: `<input>` (6.2.1)**

```html
<input type="text" size="10" maxlength="8" /> NetID<br />
<input type="password" size="16" /> Password
<input type="submit" value="Log In" />
```

- `input` attributes: `disabled`, `maxlength`, `readonly`, `size`, `value`
- `size` attribute controls onscreen width of text field
- `maxlength` limits how many characters user is able to type into field
Text boxes: `<textarea>` (6.2.2)

A multi-line text input area (inline)

```
<textarea rows="4" cols="20">
Type your comments here.
</textarea>
```

- initial text is placed inside `textarea` tag (optional)
- required `rows` and `cols` attributes specify height/width in characters
- optional `readonly` attribute means text cannot be modified

Checkboxes: `<input>` (6.2.3)

Yes/no choices that can be checked and unchecked (inline)

```
<input type="checkbox" name="lettuce" /> Lettuce
<input type="checkbox" name="tomato" checked="checked" /> Tomato
<input type="checkbox" name="pickles" /> Pickles
```

- none, 1, or many checkboxes can be checked at same time
- when sent to server, any checked boxes will be sent with value `on`:
- use `checked="checked"` attribute in HTML to initially check the box
Radio buttons: `<input>` (6.2.4)

Sets of mutually exclusive choices (inline)

```html
<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
```

- grouped by `name` attribute (only one can be checked at a time)
- must specify a `value` for each one or else it will be sent as `value on`

Text labels: `<label>` (6.2.5)

```html
<label>
<input type="radio" name="cc" value="visa" checked="checked" /> Visa
</label>
<label>
<input type="radio" name="cc" value="mastercard" /> MasterCard
</label>
<label>
<input type="radio" name="cc" value="amex" /> American Express
</label>
```

- associates nearby text with control, so you can click text to activate control
- can be used with checkboxes or radio buttons
- `label` element can be targeted by CSS style rules
Drop-down list: `<select>, <option>` (6.2.6)

menus of choices that collapse and expand (inline)

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

- Option element represents each choice
- `<select>` optional attributes: `disabled`, `multiple`, `size`
- May need to specify a value for each option on IE6

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
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 italic?

Reset buttons (6.2.7)

```html
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" />
```

- 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)

```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>
```

- `fieldset` groups related input fields; `legend` supplies an optional caption

Common UI control errors

- “I changed the checkbox's `checked` property, the `textarea`'s inner text, the text box's `value` ... but when I refresh, the page doesn't reflect this change!”
  - By default, when you refresh a page in your browser, it leaves the previous values in all UI controls
  - it does this in case you were filling out a long form and needed to refresh it, but didn't want it to clear out all the info you'd entered
  - 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)

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

```css
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)

Styling Text Boxes

```html
<textarea rows="3" cols="40"></textarea>
```

```css
body { height: 100%; }  
textarea {  
    position: absolute;  
    width: 50%;  
    height: 15%;  
}
```

- XHTML validator requires rows and cols on a textarea
- if you want a textarea at a specific width/height in pixels or %, you must specify rows/cols in the XHTML and width/height in the CSS
  - the rows/cols will be ignored but must be there anyway...
  - sometimes specifying a height on the page's body helps
  - sometimes using absolute/fixed positioning on the textarea helps
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

```html
<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 />
```

- the following form may look correct, but when you submit it...
- 
- `[$cc] => on, [startrek] => Jean-Luc Picard`
**The value attribute**

```html
<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 />
```

- value attribute controls 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 that contains one of these, it is **URL-encoded**
  - "Marty's cool!?" → "Marty%27s+cool%3F%21"
- you don't usually need to worry about this:
  - the browser automatically URL-encodes parameters before sending them
  - PHP scripts that accept query parameters automatically URL-decode them
  - ... but occasionally the weird encoded version does pop up
    (e.g. when debugging in Firebug)
Hidden input parameters (6.3.2)

- 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

Submitting data to a web server

- though web browsers mostly retrieve data from servers, sometimes they also want to send new data onto the 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 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 request has parameters, they are embedded in the request 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

```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 />
  </div>
  <input type="submit" />
</form>
```

**HTML**

**output**

Name: 
Food: 
Meat? 

Submit Query
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>
```

- 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
PHP **superglobal** arrays (global variables) contain information about the current request, server, etc.:

<table>
<thead>
<tr>
<th>Array</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$_GET, $_POST</td>
<td>parameters passed to GET and POST requests</td>
</tr>
<tr>
<td>$_REQUEST</td>
<td>parameters passed to any type of request</td>
</tr>
<tr>
<td>$_SERVER, $_ENV</td>
<td>information about the web server</td>
</tr>
<tr>
<td>$_FILES</td>
<td>files uploaded with the web request</td>
</tr>
<tr>
<td>$_SESSION, $_COOKIE</td>
<td>&quot;cookies&quot; used to identify the user (seen later)</td>
</tr>
</tbody>
</table>

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): an array that 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";
  ```
Creating an associative array

```php
$name = array();
$name["key"] = value;
...
$name["key"] = value;
```

```php
$name = array(key => value, ... , key => value);
```

```php
$blackbook = array("marty" => "206-685-2181",
"stuart" => "206-685-9138",
"jenny" => "206-867-5309");
```

- an associative array can be declared either initially empty, or with a set of predeclared key/value pairs

Printing an associative array

```php
print_r($blackbook);
```

```
Array
(
    [jenny] => 206-867-5309
    [stuart] => 206-685-9138
    [marty] => 206-685-2181
)
```

- `print_r` function displays all keys/values in the array
- `var_dump` function is much like `print_r` but prints more info
- unlike `print`, these functions require parentheses
Associative array **functions**

```php
if (isset($blackbook['marty'])) {
    print "Marty's phone number is {$blackbook['marty']}
";
} else {
    print "No phone number found for Marty Stepp.\n";
}
```

- `isset, array_key_exists`: whether the array contains value for given key
- `array_keys, array_values`: list of all keys or all values in the array
- `asort, arsort`: sorts by value, in normal or reverse order
- `ksort, krsort`: sorts by key, in normal or reverse order

foreach loop and associative arrays

```php
foreach ($blackbook as $key => $value) {
    print "$key's phone number is $value\n";
}
```

```
output

jenny's phone number is 206-867-5309
stuart's phone number is 206-685-9138
marty's phone number is 206-685-2181
```

- both the key and the value are given a variable name
- the elements will be processed in the order they were added to the array
Query parameters: $\_REQUEST (6.4.2)

```php
$user_name = $\_REQUEST["username"];
@student_id = (int) $\_REQUEST["sid"];
$eats_meat = FALSE;
if (isset($\_REQUEST["meat"})) {
    $eats_meat = TRUE;
}
```

- $\_REQUEST["parameter name"] returns param’s value as a string
- if no such parameter was passed, you’ll get a warning when trying to access it; test for this with isset

Form response pages

```php
<?php
$name = $\_REQUEST["name"];
$email = $\_REQUEST["emailaddress"];
...
print("Thank you, $name, for creating an account with address $email.\n");
?>
```

```output
Thank you, Marty, for creating an account with address foo@bar.com.
```

- users expect an HTML response page when they submit forms
- the above code is not a complete page...
Embedded PHP and response pages

```php
$name = $_REQUEST['name'];
$email = $_REQUEST['emailaddress'];

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head><title>Account Creation</title></head>
  <body>
    <h1>New account created.</h1>
    <p>Thank you, <?= $name ?>, for creating an account with address <?= $email ?>.</p>
  </body>
</html>
```

- expression blocks get rid of `print` statement in previous example

Example: Exponents

```php
<?php
$base = $_REQUEST['base'];
$exp = $_REQUEST['exponent'];
$result = pow($base, $exp);

3 ^ 4 = 81
```

http://example.com/exponent.php?base=3&exponent=4
Example: Print all parameters

```php
<?php
foreach ($_REQUEST as $param => $value) {
    ?><p>Parameter $param has value $value</p><?php
}
?>
```

http://example.com/print_params.php?name=Marty+Stepp&sid=1234567

Parameter name has value Marty Stepp
Parameter sid has value 1234567

GET or POST?

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

- some PHP web services process both GET and POST requests
- can find out which kind of request we are currently processing by looking at the
  "REQUEST_METHOD" key of the global $_SERVER array
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

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

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