POLI
[TECH>
NIKA

Czestochowa
University
of Technology

Faculty of Computer Science
and Artificial Intelligence

SCRIPTING LANGUAGES IN WEB APPLICATIONS

L07 – PHP for web applications

Web Programming Step by Step

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

6.3: Submitting Data

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 />
```

HTML

○ Visa ○ MasterCard
Favorite Star Trek captain: James T. Kirk
Submit Query

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

```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 />
```

HTML

○ Visa ○ MasterCard
Favorite Star Trek captain: James T. Kirk
Submit Query

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

```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

Name:
Food:
Meat?
Submit Query

output

GET or POST?

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

- 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

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)

```
$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:

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

Uploading files (6.3.4)

```html
<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... No file selected. 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

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" />
```

Browse... No file selected. Submit Query

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

Extra stuff about associative arrays

- 6.1: Form Basics
- 6.2: Form Controls
- 6.3: Submitting Data
- 6.4: Processing Form Data in PHP
- **More about associative arrays**

Creating an associative array

```
$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");
```
PHP

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

Printing an associative array

```
print_r($blackbook);
```
PHP

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

- `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']}\n";
} else {
  print "No phone number found for Marty Stepp.\n";
}
```

name(s)	category
isset, array_key_exists	whether the array contains value for given key
array_keys, array_values	an array containing all keys or all values in the assoc.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

Web Programming Step by Step

Lecture 11
Form Validation
References: PHP.net, webcheatsheet.com, roscripts, PHPro

A form that submits to itself

```html
<form action="" method="post">
  ...
</form>
```

- a form can submit its data back to itself by setting the `action` to the page's own URL (or blank)
- benefits
  - fewer pages/files (don't need a separate file for the code to process the form data)
  - can more easily re-display the form if there are any errors

Processing a self-submitted form

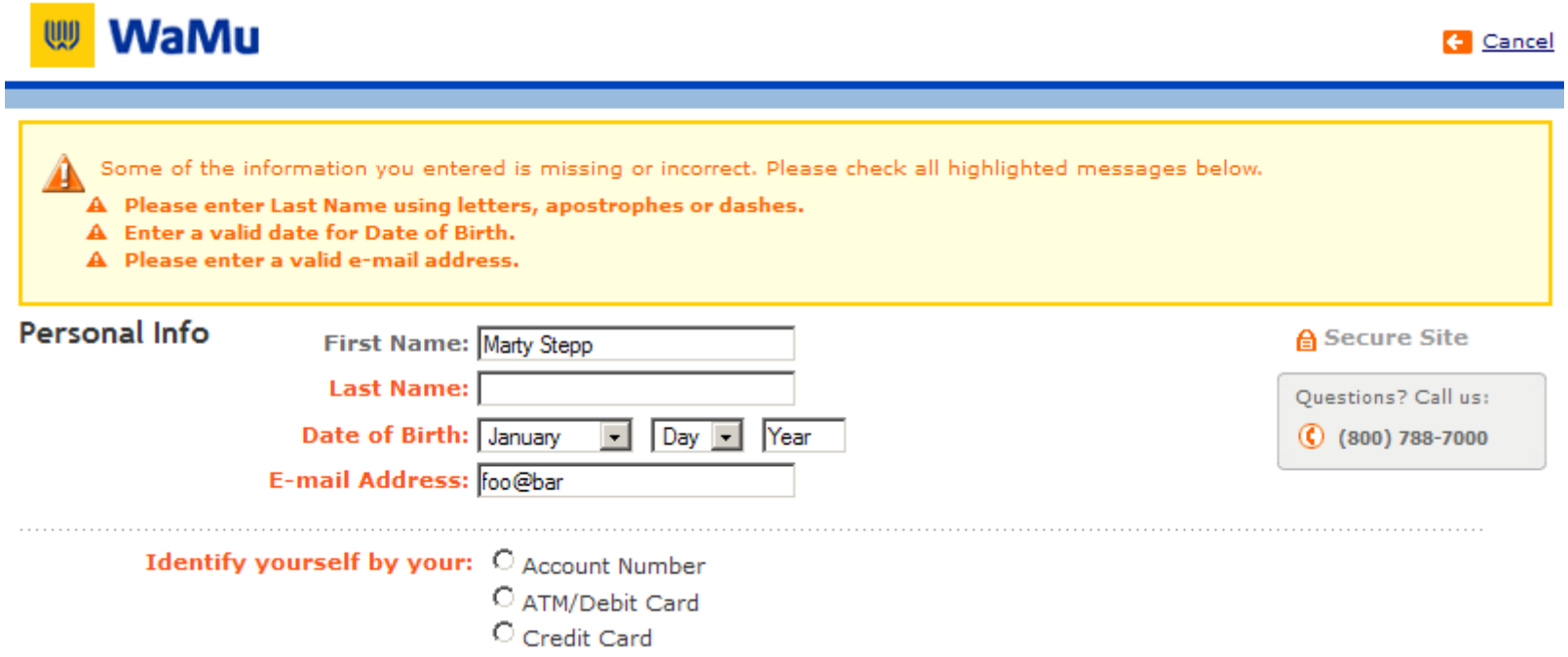
```
if ($_SERVER["REQUEST_METHOD"] == "GET") {
  # normal GET request; display self-submitting form
  ?>
  <form action="" method="post">...</form>
  <?php
} elseif ($_SERVER["REQUEST_METHOD"] == "POST") {
  # POST request; user is submitting form back to here; process it
  $var1 = $_REQUEST["param1"];
  ...
}
```

- a page with a self-submitting form can process both GET and POST requests
- look at the global `$_SERVER` array to see which request you're handling
- handle a GET by showing the form; handle a POST by processing the submitted form data

What is form validation?

- **validation**: ensuring that form's values are correct
- some types of validation:
  - preventing blank values (email address)
  - ensuring the type of values
    - integer, real number, currency, phone number, Social Security number, postal address, email address, date, credit card number, ...
  - ensuring the format and range of values (ZIP code must be a 5-digit integer)
  - ensuring that values fit together (user types email twice, and the two must match)

A real form that uses validation



Client vs. server-side validation

Validation can be performed:

- **client-side** (before the form is submitted)
  - can lead to a better user experience, but not secure (why not?)
- **server-side** (in PHP code, after the form is submitted)
  - needed for truly secure validation, but slower
- both
  - best mix of convenience and security, but requires most effort to program

An example form to be validated

```html
<form action="http://foo.com/foo.php" method="get">
  <div>
    City:  <input name="city" /> <br />
    State: <input name="state" size="2" maxlength="2" /> <br />
    ZIP:   <input name="zip" size="5" maxlength="5" /> <br />
    <input type="submit" />
  </div>
</form>
```

HTML

City:
State:
ZIP:
Submit Query

output

- Let's validate this form's data on the server...

Basic server-side validation code

```
$city  = $_REQUEST["city"];
$state = $_REQUEST["state"];
$zip   = $_REQUEST["zip"];
if (!$city || strlen($state) != 2 || strlen($zip) != 5) {
  print "Error, invalid city/state/zip submitted.";
}
```

- *basic idea:* examine parameter values, and if they are bad, show an error message and abort. But:
  - How do you test for integers vs. real numbers vs. strings?
  - How do you test for a valid credit card number?
  - How do you test that a person's name has a middle initial?
  - (How do you test whether a given string matches a particular complex format?)

Regular expressions

```
/^[a-zA-Z_\-]+@(([a-zA-Z_\-])+\.)+[a-zA-Z]{2,4}$/
```

- **regular expression** ("regex"): a description of a pattern of text
  - can test whether a string matches the expression's pattern
  - can use a regex to search/replace characters in a string
- regular expressions are extremely powerful but tough to read
  (the above regular expression matches email addresses)
- regular expressions occur in many places:
  - Java: `Scanner`, `String`'s `split` method (CSE 143 sentence generator)
  - supported by PHP, JavaScript, and other languages
  - many text editors (TextPad) allow regexes in search/replace

Basic regular expressions

```
/abc/
```

- in PHP, regexes are strings that begin and end with `/`
- the simplest regexes simply match a particular substring
- the above regular expression matches any string containing `"abc"`:
  - YES: `"abc"`, `"abcdef"`, `"defabc"`, `".=.abc.=."`, ...
  - NO: `"fedcba"`, `"ab c"`, `"PHP"`, ...

Wildcards: .

- A dot `.` matches any character except a `\n` line break
  - `/.oo.y/` matches `"Doocy"`, `"goofy"`, `"LooNy"`, ...
- A trailing `i` at the end of a regex (after the closing `/`) signifies a case-insensitive match
  - `/mart/i` matches `"Marty Stepp"`, `"smart fellow"`, `"WALMART"`, ...

Special characters: |, (), \

- `|` means *OR*
  - `/abc|def|g/` matches `"abc"`, `"def"`, or `"g"`
  - There's no *AND* symbol. Why not?
- `()` are for grouping
  - `/(Homer|Marge) Simpson/` matches `"Homer Simpson"` or `"Marge Simpson"`
- `\` starts an **escape sequence**
  - many characters must be escaped to match them literally: `/ \ $ . [ ] ( ) ^ * + ?`
  - `/<br \/>/` matches lines containing `<br />` tags

Quantifiers: *, +, ?

- `*` means 0 or more occurrences
  - `/abc*/` matches `"ab"`, `"abc"`, `"abcc"`, `"abccc"`, ...
  - `/a(bc)*/` matches `"a"`, `"abc"`, `"abcbc"`, `"abcbcbc"`, ...
  - `/a.*a/` matches `"aa"`, `"aba"`, `"a8qa"`, `"a!?xyz__9a"`, ...
- `+` means 1 or more occurrences
  - `/a(bc)+/` matches `"abc"`, `"abcbc"`, `"abcbcbc"`, ...
  - `/Goo+gle/` matches `"Google"`, `"Gooogle"`, `"Goooogle"`, ...
- `?` means 0 or 1 occurrences
  - `/a(bc)?/` matches `"a"` or `"abc"`

More quantifiers: `{min,max}`

- `{`*min*`,`*max*`}` means between *min* and *max* occurrences (inclusive)
  - `/a(bc){2,4}/` matches `"abcbc"`, `"abcbcbc"`, or `"abcbcbcbc"`
- *min* or *max* may be omitted to specify any number
  - `{2,}` means 2 or more
  - `{,6}` means up to 6
  - `{3}` means exactly 3

Anchors: ^ and $

- `^` represents the beginning of the string or line;
  `$` represents the end
  - `/Jess/` matches all strings that contain `Jess`;
    `/^Jess/` matches all strings that *start with* `Jess`;
    `/Jess$/` matches all strings that *end with* `Jess`;
    `/^Jess$/` matches the exact string `"Jess"` only
  - `/^Mart.*Stepp$/` matches `"MartStepp"`, `"Marty Stepp"`, `"Martin D Stepp"`, ...
    but NOT `"Marty Stepp stinks"` or `"I H8 Martin Stepp"`
- (on the other slides, when we say, `/PATTERN/` matches `"text"`, we really mean that it matches any string that contains that text)

Character sets: []

- `[]` group characters into a **character set**; will match any single character from the set
  - `/[bcd]art/` matches strings containing `"bart"`, `"cart"`, and `"dart"`
  - equivalent to `/(b|c|d)art/` but shorter
- inside `[]`, many of the modifier keys act as normal characters
  - `/what[!*?]*/` matches `"what"`, `"what!"`, `"what?**!"`, `"what??!"`, ...
- What regular expression matches DNA (strings of A, C, G, or T)?
  - `/[ACGT]+/`

Character ranges: `[start-end]`

- inside a character set, specify a range of characters with `-`
  - `/[a-z]/` matches any lowercase letter
  - `/[a-zA-Z0-9]/` matches any lower- or uppercase letter or digit
- an initial `^` inside a character set negates it
  - `/[^abcd]/` matches any character other than a, b, c, or d
- inside a character set, `-` must be escaped to be matched
  - `/[+\-]?[0-9]+/` matches an optional `+` or `-`, followed by at least one digit
- What regular expression matches letter grades such as A, B+, or D- ?
  - `/[ABCDF][+\-]?/`

Escape sequences

- special escape sequence character sets:
  - `\d` matches any digit (same as `[0-9]`); `\D` any non-digit (`[^0-9]`)
  - `\w` matches any "word character" (same as `[a-zA-Z_0-9]`); `\W` any non-word char
  - `\s` matches any whitespace character (, `\t`, `\n`, etc.); `\S` any non-whitespace
- What regular expression matches dollar amounts of at least $100.00 ?
  - `/\$\d{3,}\.\d{2}/`

Regular expressions in PHP (PDF)

- regex syntax: strings that begin and end with `/`, such as `"/[AEIOU]+/"`

function	description
`preg_match(`*regex*`, `*string*`)`	returns `TRUE` if *string* matches *regex*
`preg_replace(`*regex*`, `*replacement*`, `*string*`)`	returns a new string with all substrings that match *regex* replaced by *replacement*
`preg_split(`*regex*`, `*string*`)`	returns an array of strings from given *string* broken apart using given *regex* as delimiter (like `explode` but more powerful)

Regular expression example

```
# replace vowels with stars
$str = "the quick    brown        fox";

$str = preg_replace("/[aeiou]/", "*", $str);
                          # "th* q**ck    br*wn        f*x"

# break apart into words
$words = preg_split("/[ ]+/", $str);
                          # ("th*", "q**ck", "br*wn", "f*x")

# capitalize words that had 2+ consecutive vowels
for ($i = 0; $i < count($words); $i++) {
  if (preg_match("/\\*{2,}/", $words[$i])) {
    $words[$i] = strtoupper($words[$i]);
  }
}                         # ("th*", "Q**CK", "br*wn", "f*x")
```

- notice how \ must be escaped to \\

PHP form validation w/ regexes

```php
$state = $_REQUEST["state"];
if (!preg_match("/[A-Z]{2}/", $state)) {
  print "Error, invalid state submitted.";
}
```

- `preg_match` and regexes help you to validate parameters
- sites often *don't* want to give a descriptive error message here (why?)

POLI
[TECH]
NIKA

Czestochowa
University
of Technology

[@]

Faculty of Computer Science
and Artificial Intelligence