Processing Form Data

Common client-side scripting task. Very easy in PHP. Often involves using collected data in producing further output (search engine) or interacting with a database (store customer info).

Client/Server dividing line
Creating an XHTML Form

Key elements:

Input fields must be contained inside a **form tag**.
All input fields must have a **name**.
   Names cannot have spaces in them. Fields should be **named well** for clear identification.
Form **action** should be URL to PHP processing script.
Appropriate form transmission method selected: **GET** or **POST**.

http://einstein.etsu.edu/~pittares/CSCI2910/examples/3-1.htm

---

GET vs. POST

**GET:**

- Name/value pairs appended in clear text to the URL of the receiving page/script.
- Each name/value pair separated by '&'. Value data automatically URL encoded.
- Names are taken from the form field names.
- GET URLs can be saved, bookmarked, etc. and used to recall the script with the same data.
- GET strings provide 'transparency' that may/may not be desired.
- Long GET strings may be problematic.
GET vs. POST

POST:

Data is encoded in the page request body sent by the browser, but not shown in the URL. Unseen to user. Since data not part of URL, bookmarking and reusing URL to recall the script with the same data is not possible.

Large POST packets not a problem.

On most browsers, hitting 'refresh' causes post data to be retransmitted.

Manual GET encoding

Since GET string is part of URL, a web programmer can manually build a GET string to use as a method of data passing without a form being involved.

http://einstein.etsu.edu/~pittares/CSCI2910/examples/3-2.php?val1=17&val2=35
http://einstein.etsu.edu/~pittares/CSCI2910/examples/3-2.php?val1=22&val2=885

Technically, post data can be transmitted without a form as well, but it requires the programmer to open and write request packets using network sockets. Almost never worth the effort.
Processing received GET or POST data

Data stored by web server in superglobal array **$_GET** or **$_POST**, using field name as index.

http://einstein.etsu.edu/~pittares/CSCI2910/examples/3-3.htm
http://einstein.etsu.edu/~pittares/CSCI2910/examples/3-5.htm

**$_GET** and **$_POST** only available to receiving script (not persistent).

Form field data passing specifics

Text fields, password fields, and textareas that make up a form will *always* have their name (and value if field is completed) passed in a form.

Radio buttons and checkboxes will only have their name passed (with the appropriate value) if the button or checkbox has been set.

http://einstein.etsu.edu/~pittares/CSCI2910/examples/3-7.htm
Form processing logistics

Never assume a form:
- Is filled out completely
- Contains the type of information requested
- Has been submitted by a benign user
- Only contains the fields and values or value ranges expected

Check all form data to verify that it is complete and valid.

Conditional Statements

PHP supports traditional if construction with elseif being a valid keyword.

```php
if (condition)
{
    // conditional code here
}

if (condition)
{
    // conditional code here
} else {
    // conditional code here
}
```
Conditional Statements

if (condition)
{
   // conditional code here
} elseif (condition) {
   // conditional code here
} else {
   // conditional code here
}

Notice and follow code alignment illustrated.

Comparison Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>Equal in value</td>
<td>$a == $b</td>
</tr>
<tr>
<td>===</td>
<td>Equal in value and type</td>
<td>$a === $b</td>
</tr>
<tr>
<td>!= or &lt;&gt;</td>
<td>Not equal</td>
<td>$a != $b or $a &lt;&gt; $b</td>
</tr>
<tr>
<td>!==</td>
<td>Not equal in value and type</td>
<td>$a !== $b</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
<td>$a &lt; $b</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
<td>$a &gt; $b</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal</td>
<td>$a &lt;= $b</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal</td>
<td>$a &gt;= $b</td>
</tr>
</tbody>
</table>

```php
<?php
   $a = 4;
   $b = "4";
   $a == $b; //true
   $a === $b; //false
?>
```
Logical Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>Not</td>
<td>!$a</td>
</tr>
<tr>
<td>&amp;&amp;, and</td>
<td>And</td>
<td>$a &amp;&amp; $b</td>
</tr>
<tr>
<td></td>
<td></td>
<td>, or</td>
</tr>
<tr>
<td>xor</td>
<td>exclusive or</td>
<td>$a xor $b</td>
</tr>
</tbody>
</table>

```php
<?php
$a = true;
$b = true;
$a || $b;  //true
$a xor $b; //false
?>
```

Helpful form validation functions

Functions exist for testing data types: `is_numeric($x)`, etc., but use can be challenging.

- `isset($var)`—does $var exist?
- `empty($var)`—returns false unless $var contains an empty string, 0, "0", NULL, or FALSE.

```php
<?php
$testvar = "";
echo isset($testvar)."<br/>";
echo empty($testvar);
?>
```
What kind of validation is needed here?

http://einstein.etsu.edu/~pittares/CSCI2910/examples/3-9.htm

Name:

Age:

Gender:

http://einstein.etsu.edu/~pittares/CSCI2910/examples/3-10.php?firstname=0&age=one+million&gender=martian&button=Submit

String manipulation functions useful with forms

- **trim($s)** returns string $s with whitespace from beginning and end of a string removed.
- **ltrim($s)** returns string $s with leading white space removed.
- **chop($s)** returns string $s with trailing white space removed.
- **strToUpper($s)** —returns uppercase of string $s.
- **strToLower($s)** —returns lowercase of string $s.
- **ucfirst($s)** —returns capitalized first character of string $s, if alphabetic.
- **ucWords($s)** —returns capitalized first character of every word in string $s, if alphabetic.
String manipulation functions useful with forms

`htmlspecialchars($s)` returns $s with &, ',', <, and > turned into XHTML special characters.

`htmlentities($s)` identical to above, except all characters with a special XHTML special character are translated.

`strip_tags($s)` returns $s with any HTML and PHP tags removed.

Other string-related functions:

- [http://einstein.etsu.edu/~pittares/CSCI2910/examples/3-11.htm](http://einstein.etsu.edu/~pittares/CSCI2910/examples/3-11.htm)

---

**switch**

When doing repeated tests on the same variable, the `switch` statement may be useful.

```php
switch ($gender) {
    case 'M':
        echo "Hello, Sir!";
        break;
    case 'F':
        echo "Hello, Ma'am!";
        break;
    default:
        echo "You didn't enter your gender!";
        break;
}
```

Client-side Form Validation

JavaScript, etc., permits client-side form validation, however you should never rely on it to work.

Browser may have scripting turned off.

User may save and edit page with client-side scripting edited.

Never trust any form data! Check to make sure it fits the profile for valid data.

Copyrights

Presentation prepared by and copyright of Dr. Tony Pittarese, East Tennessee State University, Computer and Information Sciences Dept. (pittares@etsu.edu)

Podcast lecture related to this presentation available via ETSU iTunesU.