Agenda

• This week’s expected outcomes
• This week’s topics
• This week’s homework
• Upcoming deadlines
• Questions and answers
Week 5 Outcomes

• Employ string functions to manipulate character-based data
• Employ date and time functions to manipulate date-based data
• Discuss reasons to avoid and alternatives to user-entered HTML markup in web-applications.
Strings

- Single quoted strings: 'Hello $i\n' – no interpolation, no escape sequences
- Double quoted strings: "Hello $i\n" – interpolation, escape sequences
Strings - Heredocs

- Heredocs and nowdocs

```php
<?php
$arr = array('heredoc', 'double-quoted');
$message = <<< END
This is a ${arr[0]} that acts like a ${arr[1]} string, and so interpolation and escape sequences are significant as are line breaks.
END;
print(nl2br($message));
?>
```
Strings - Heredocs

• Heredocs and nowdocs

```php
<?php
$arr = array('heredoc', 'double-quoted');
$message = <<<END
This is a ${arr[0]} that acts like a ${arr[1]} string, and so interpolation and escape sequences are significant as are line breaks.
END;
print(nl2br($message));
?>
```

Notice that NetBeans doesn’t syntax-highlight the heredoc properly.
Strings - Nowdocs

• Heredocs and nowdocs

```php
<?php
$arr = array('nowdoc', 'single-quoted');
$message = <<< 'END'
This is a ${arr[0]} that acts like
a ${arr[1]} string, and so
interpolation and escape sequences
are not significant but line breaks are.
END;
print(nl2br($message));
?>
```
Strings - Nowdocs

• Heredocs and nowdocs

```php
<?php
$arr = array('nowdoc', 'single-quoted');
$message = <<< 'END'
This is a ${arr[0]} that acts like a ${arr[1]} string, and so interpolation and escape sequences are not significant but line breaks are.
END;
print(nl2br($message));
?>
```

Notice that NetBeans doesn’t syntax-highlight the nowdoc properly either.
String Escape codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>\</code></td>
<td>Backslash</td>
</tr>
<tr>
<td><code>'</code></td>
<td>Single quote</td>
</tr>
<tr>
<td><code>&quot;</code></td>
<td>Double quote</td>
</tr>
<tr>
<td><code>\$</code></td>
<td>Dollar sign</td>
</tr>
<tr>
<td><code>\n</code></td>
<td>Newline</td>
</tr>
<tr>
<td><code>\t</code></td>
<td>Tab</td>
</tr>
<tr>
<td><code>\r</code></td>
<td>Carriage return</td>
</tr>
<tr>
<td><code>\xhh</code></td>
<td>Hexadecimal char</td>
</tr>
</tbody>
</table>

HTML ignores whitespace, so you’d only see `\t`, `\n`, `\r` in “view source”
Strings and Characters

• ASCII values
  - Each character maps to an integer value
    - Ex: ‘A’ is 65, ‘Z’ is 90, etc. (see www.asciiitable.com)
  - Use ord() with a character parameter to get the ASCII value back.
  - Use chr() with an integer parameter to get the character value back.
Looping and Strings

- Looping through strings
  - Use `str_split()` to convert a string to an array of 1-character strings.

```php
function asciiEncode($str) {
    $result = "";
    $chars = str_split($str, 1);
    foreach ($chars as $char) {
        $result .= '\&#'. ord($char) . ';';
    }
    return $result;
}

$encoded = asciiEncode("todd.whittaker@franklin.edu");
```
Looping and Strings

- Looping through strings
  - Use str_split() to convert a string to an array of 1-character strings.

```php
function asciiEncode($str) {
    $result = "";
    $chars = str_split($str, 1);
    foreach ($chars as $char) {
        $result .= '&#' . ord($char) . ';
    }
    return $result;
}

$encoded = asciiEncode("todd.whittaker@franklin.edu");
```

Produce:
```
&#116;&#111;&#100;&#100;&#46;&#119
;;&#104;&#105;&#107;&#97;&#101;&#114;&#64;&#102;&#114;&#97;&#110;&#107;&#108;&#105;&#110;&#46;&#101;&#100;&#117;
```
Learning a Language

• Two basic parts to learning any new programming language
  ➢ Syntactical constructs
    ➢ Control structures, key words, punctuation, data types, etc. I.e. rules of the language
  ➢ Libraries
    ➢ Pre-written routines (functions, objects) that you can use without writing them yourself.
Common String Functions


<table>
<thead>
<tr>
<th>Function</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>strlen($str)</td>
<td>Returns the length of the string</td>
</tr>
<tr>
<td>empty($str)</td>
<td>Returns TRUE if the string is empty, null, or '0'.</td>
</tr>
<tr>
<td>substr($str, $i [, $len])</td>
<td>Returns a substring of $str starting at position $i (0-based indexing) and containing $len characters (at most).</td>
</tr>
<tr>
<td>strpos($str1, $str2)</td>
<td>Searches $str1 for $str2 and returns the integer value of where it is found or FALSE if it is not found. See also stripos, strrpos, strripos.</td>
</tr>
</tbody>
</table>
Common String Functions


<table>
<thead>
<tr>
<th>Function</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>str_replace($old, $new, $orig)</code></td>
<td>Replace all occurrences of <code>$old</code> with <code>$new</code> in the string <code>$orig</code>. See also <code>str_ireplace</code>.</td>
</tr>
<tr>
<td><code>ltrim($str), rtrim($str), trim($str)</code></td>
<td>Trims whitespace from the string on the left, right, and both sides.</td>
</tr>
<tr>
<td>`str_pad($str, $len[, $pad[, $type]]))</td>
<td>Pads a string up to be up to <code>$len</code> in length using <code>$pad</code>.</td>
</tr>
<tr>
<td><code>strtolower($str), strtoupper($str)</code></td>
<td>Converts a string to lower or upper case respectively.</td>
</tr>
</tbody>
</table>
Common String Functions


<table>
<thead>
<tr>
<th>Function</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>explode($sep, $str)</td>
<td>Splits a string into an array based on the $sep delimiter.</td>
</tr>
<tr>
<td>implode($sep, $arr)</td>
<td>Produces a single string from the array with $sep between elements.</td>
</tr>
<tr>
<td>strcmp($str1, $str2),</td>
<td>Compares two strings, returning -1 if $str1 &lt; $str2, 0 if $str1 ==</td>
</tr>
<tr>
<td>strcasecmp($str1, $str2),</td>
<td></td>
</tr>
<tr>
<td>strnatcmp($str1, $str2),</td>
<td></td>
</tr>
</tbody>
</table>
# Common Math Functions


<table>
<thead>
<tr>
<th>Function</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>abs($num)</code></td>
<td>Returns the absolute value of <code>$num</code>.</td>
</tr>
<tr>
<td><code>ceil($num)</code></td>
<td>Returns the next integer greater than or equal to <code>$num</code>.</td>
</tr>
<tr>
<td><code>floor($num)</code></td>
<td>Returns the next integer less than or equal to <code>$num</code>.</td>
</tr>
<tr>
<td><code>round($num[, $prec])</code></td>
<td>Rounds <code>$num</code> to <code>$prec</code> decimal places.</td>
</tr>
</tbody>
</table>
# Common Math Functions


<table>
<thead>
<tr>
<th>Function</th>
<th>Purpose</th>
</tr>
</thead>
</table>
| `max($n1, $n2[, $n3...])` | Returns the maximum of all parameters. See also `min()`.
| `pow($base, $exp)`     | Raises $base to the power $exp.                                         |
| `sqrt($num)`           | Computes the square root of $num.                                       |
| `mt_rand($low, $high)` | Returns a random integer between [$low, $high]                          |
Formatting Output

• `sprintf($format, $val1[, $val2 ...])`
  - Returns a string with values inserted at given locations, using the format specified

```php
$result = sprintf("Hello, %s, you have %10.2f dollars", 'Fred', 13.245);
```

Hello, Fred, you have 13.24 dollars
Dates and Times

- Timestamp: an integer number of seconds since 12:00 AM, January 1, 1970 GMT.
- Can use functions to generate timestamps, format output, compute differences, etc.

```php
$seconds = time();
$str = date("n/j/Y", $seconds);
```

$seconds is 1328123445, $str is 2/1/2012
Dates and Times

• Use `strtotime` to parse date strings into timestamps

```php
$seconds = strtotime("2012-02-01 4:35:21pm");
$str = date("g:i:s A, n/j/Y", $seconds);
```

$seconds is 1328110521 ,
$str is 4:35:21 PM, 2/1/2012
Dates and Times

- Use `strtotime` to parse date strings into timestamps

```php
$seconds = strtotime("2012-02-01 4:35:21pm");
$str = date("g:i:s A, n/j/Y", $seconds);

$seconds = strtotime("+2 weeks 8am", time());
$str = date("g:i:s A, n/j/Y", $seconds);
```

$seconds is 1329289200, $str is 8:00:00 AM, 2/15/2012
Dates and Times

- Can also use a DateTime object to manipulate dates.

```php
$dueDate = new DateTime();
$dueDate -> modify("next Sunday 11:59:59pm");
$str = $dueDate -> format("g:i:s A, n/j/Y");
```

$str is 11:59:59 PM, 2/5/2012 based on today being Wednesday, 2/1/2012
Dates and Times

• A DateInterval object holds a difference between dates.

```php
$date911 = new DateTime("2001-09-11 9:59:00am");
$today = new DateTime();
$delta = $date911 -> diff($today);
$delta = $delta -> format("%R%yy %mm %dd %H:%I:%S");

$str has +10y 4m 21d 10:43:10 based on today being 2/1/2012

Given a DateInterval object, you can add or subtract that from a DateTime object as well.
```
Dates and Times

Handling text

• Have used htmlentities and htmlspecialchars to avoid injection vulnerabilities
  . But, it is desirable to allow some formatting, just not all formatting.
  . Special mini-languages for formatting
    . BBCode
    . Markdown
Handling text

• Have used `htmlentities` and `htmlspecialchars` to avoid injection vulnerabilities
  • But, it is desirable to allow some formatting, just not all formatting.
• Special mini-languages for formatting
  • BBCode
  • Markdown

What can injection in a web page let you do?
function markdown($str) {
    $str = htmlspecialchars(ltrim($str), ENT_QUOTES);
    $str = preg_replace('/\*\*(.+)/u', '<b>$1</b>', $str);
    $str = preg_replace('/\*(.+)/u', '<i>$1</i>', $str);
    $str = preg_replace('/#### (.+)/n','<h4>$1</h4>', $str);
    $str = preg_replace('/### (.+)/n','<h3>$1</h3>', $str);
    $str = preg_replace('/## (.+)/n','<h2>$1</h2>', $str);
    $str = preg_replace('/# (.+)/n','<h1>$1</h1>', $str);
    $str = preg_replace('/\[(.+)/\](.+)/u', '<a href="$2">$1</a>', $str);
    $str = preg_replace('/\(\(\{2,}\)(?:\r\n){2,}|\r{2,}\n{2,}|$)/u', '<p>$1</p>', $str);
    return $str;
}
Handling text

**Mini-markdown**

This form lets you submit a mini-markdown document that will be rendered into HTML. This is safer than allowing HTML markup in your web applications. See [Wikipedia](https://en.wikipedia.org) for a complete Markdown syntax. Note, this is merely a demonstration, and not production-ready code. There are some complete Markdown libraries that are available for PHP.

```
# My Simple Markdown

This is a test of my simple markdown. You can *emphasize* things with asterisks, or **really emphasize** things with two asterisks.

Paragraphs are separated by two newlines.

You can even embed [simple links](http://en.wikipedia.org/wiki/Markdown).
```

Submit
function markdown($str) {
    $str = htmlspecialchars(ltrim($str), ENT_QUOTES);
    $str = preg_replace('/\*(.+)\*/u', '<b>$1</b>', $str);
    $str = preg_replace('/\*\*\*\*(.+)*\*/u', '<i>$1</i>', $str);
    $str = preg_replace('/#### \([^\n]*\)\n/', '<h4>$1</h4>', $str);
    $str = preg_replace('/### \([^\n]*\)\n/', '<h3>$1</h3>', $str);
    $str = preg_replace('/## \([^\n]*\)\n/', '<h2>$1</h2>', $str);
    $str = preg_replace('/# \([^\n]*\)\n/', '<h1>$1</h1>', $str);
    $str = preg_replace('/\[(\[^\]*\])\]\((\[^\]*\))\]/', '<a href="$2">$1</a>', $str);
    $str = preg_replace('/\(\([^\n\r]{2,})\)?(\([^\n\r]{2,}\)?|\(\[^\n\r]{2,}\)\)?/u', '<p>$1</p>', $str);
    return $str;
}

“Mini-markdown” for the simplest of formatting. See “minimarkdown.zip” example. Full markdown parsers are much better.
Handling text

• General rule
  • Escape all HTML markup
  • Store Markdown (or BBCode) text in the DB
  • Convert to HTML only when sent back to the browser.
Handling text

• Alternatives
  . Use a WYSIWYG HTML editor (such as TinyMCE or CKEditor) combined with…
  . An HTML sanitizer library (such as http://htmlpurifier.org/) to limit tags.
  . Store HTML directly in the DB without escaping.
Upcoming Deadlines

• Readings for next week
  ➢ Chapters 11 and 12 in *PHP and MySQL*

• Assignments
  ➢ Homework 4 due end of week 5
  ➢ Lab 2 due end of week 7

• Next week:
  ➢ Arrays, cookies, sessions
General Q & A

• Questions?
• Comments?
• Concerns?