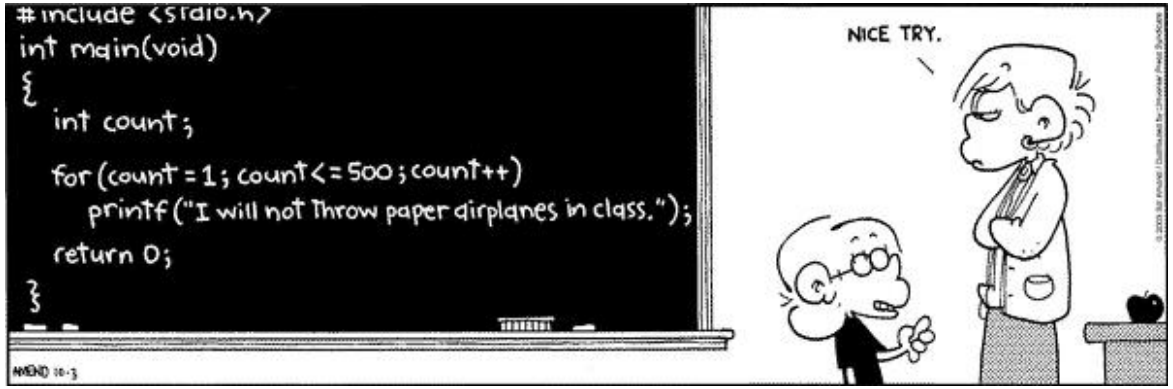
# CSE 154




## LECTURE 15: *MORE PHP*

# Arrays

```php
$name = array();                              # create
$name = array(value0, value1, ..., valueN);

$name[index]                                  # get element value
$name[index] = value;                         # set element value
$name[] = value;                              # append
```

```php
$a = array();      # empty array (length 0)
$a[0] = 23;        # stores 23 at index 0 (length 1)
$a2 = array("some", "strings", "in", "an", "array");
$a2[] = "Ooh!";    # add string to end (at index 5)
```

- to append, use bracket notation without specifying an index
- element type is not specified; can mix types

# Array functions

| function name(s) | description |
|---|---|
| count | number of elements in the array |
| print_r | print array's contents |
| array_pop, array_push, array_shift, array_unshift | using array as a stack/queue |
| in_array, array_search, array_reverse, sort, rsort, shuffle | searching and reordering |
| array_fill, array_merge, array_intersect, array_diff, array_slice, range | creating, filling, filtering |
| array_sum, array_product, array_unique, array_filter, array_reduce | processing elements |

# Array function example

```
$tas = array("MD", "BH", "KK", "HM", "JP");
for ($i = 0; $i < count($tas); $i++) {
  $tas[$i] = strtolower($tas[$i]);
}                                    # ("md", "bh", "kk", "hm", "jp")
$morgan = array_shift($tas);         # ("bh", "kk", "hm", "jp")
array_pop($tas);                     # ("bh", "kk", "hm")
array_push($tas, "ms");              # ("bh", "kk", "hm", "ms")
array_reverse($tas);                 # ("ms", "hm", "kk", "bh")
sort($tas);                          # ("bh", "hm", "kk", "ms")
$best = array_slice($tas, 1, 2); # ("hm", "kk")
```

- the array in PHP replaces many other collections in Java
  - list, stack, queue, set, map, ...

# The foreach loop

```
foreach ($array as $variableName) {
   ...
}
```
PHP

```
$stooges = array("Larry", "Moe", "Curly", "Shemp");
for ($i = 0; $i < count($stooges); $i++) {
  print "Moe slaps {$stooges[$i]}\n";
}
foreach ($stooges as $stooge) {
  print "Moe slaps $stooge\n";  # even himself!
}
```

- a convenient way to loop over each element of an array without indexes

# Math operations

```php
$a = 3;
$b = 4;
$c = sqrt(pow($a, 2) + pow($b, 2));
```

| | | | | | | |
|---|---|---|---|---|---|---|
| abs | ceil | cos | floor | log | log10 | max |
| min | pow | rand | round | sin | sqrt | tan |

math functions

| | | |
|---|---|---|
| M_PI | M_E | M_LN2 |

math constants

- the syntax for method calls, parameters, returns is the same as Java

# NULL

```
$name = "Victoria";
$name = NULL;
if (isset($name)) {
  print "This line isn't going to be reached.\n";
}
```

- a variable is NULL if
  - it has not been set to any value (undefined variables)
  - it has been assigned the constant NULL
  - it has been deleted using the unset function
- can test if a variable is NULL using the isset function
- NULL prints as an empty string (no output)

# Functions

```
function name(parameterName, ..., parameterName) {
  statements;
}
```
PHP

```
function bmi($weight, $height) {
  $result = 703 * $weight / $height / $height;
  return $result;
}
```
PHP

- parameter types and return types are not written
- a function with no return statements is implicitly "void"
- can be declared in any PHP block, at start/end/middle of code

# Calling functions

```
name(expression, ..., expression);
```
PHP

```
$w = 163;   # pounds
$h = 70;    # inches
$my_bmi = bmi($w, $h);
```
PHP

- if the wrong number of parameters are passed, it's an error

# Variable scope: global and local vars

```php
$school = "UW";                    # global
...

function downgrade() {
  global $school;
  $suffix = "(Wisconsin)";         # local

  $school = "$school $suffix";
  print "$school\n";
}
```

- variables declared in a function are local to that function; others are global
- if a function wants to use a global variable, it must have a global statement
  - but don't abuse this; mostly you should use parameters

# Default parameter values

```
function name(parameterName = value, ..., parameterName = value) {
  statements;
}
```
PHP

```
function print_separated($str, $separator = ", ") {
  if (strlen($str) > 0) {
    print $str[0];
    for ($i = 1; $i < strlen($str); $i++) {
      print $separator . $str[$i];
    }
  }
}
```
PHP

```
print_separated("hello");          # h, e, l, l, o
print_separated("hello", "-");     # h-e-l-l-o
```
PHP

- if no value is passed, the default will be used (defaults must come last)

# Web Services

**web service**: software functionality that can be invoked through the internet using common protocols

- like a remote function(s) you can call by contacting a program on a web server
  - many web services accept parameters and produce results
- can be written in PHP and contacted by the browser in HTML and/or Ajax code
- service's output might be HTML but could be text, XML, JSON or other content

# Setting content type with header

```
header("Content-type: type/subtype");
```
PHP

```
header("Content-type: text/plain");
print "This output will appear as plain text now!\n";
```
PHP

- by default, a PHP file's output is assumed to be HTML (text/html)
  - However, in our class we aren't using PHP to generate HTML
- So, we use the header function to specify non-HTML output
  - must appear before any other output generated by the script
  - (doesn't have to be the first line of code, though)

# Recall: Content ("MIME") types

| MIME type | related file extension |
|---|---|
| **text/plain** | .txt |
| **text/html** | .html, .htm, ... |
| **text/xml** | .xml |
| **application/json** | .json |
| text/css | .css |
| text/javascript | .js |
| image/gif | .gif |

- Lists of MIME types: by type, by extension

# Query strings and parameters

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

```
http://www.google.com/search?q=Romney
http://example.com/student_login.php?username=obourn&id=1234567
```

- **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 obourn, and sid has value 1234567
- PHP code on the server can examine and utilize the value of parameters
- a way for PHP code to produce different output based on values passed by the user

# Query parameters: $_GET, $_POST

```php
$user_name = $_GET["username"];
$id_number = (int) $_GET["id"];
$eats_meat = FALSE;
if (isset($_GET["meat"])) {
  $eats_meat = TRUE;
}
```

- $_GET["parameter name"] or $_POST["parameter name"] returns a GET/POST parameter's value as a string
- parameters specified as http://....?name=value&name=value are GET parameters
- test whether a given parameter was passed with isset

# Query parameters: $_POST

```
$username = $_POST["username"];
$password = $_POST["password"];

$users_pw_hash = db_lookup_hashed_pw($username);
if (password_hash($password) == $users_pw_hash) {
  print("Successfully logged in!");
}
```

- POST parameters come in through the body of the request, not through the URL.
- However, on the server, we get access to them the same way we use the GET params, but with a different array: $_POST
- This means that client side POST requests look different than GET requests, but server-side POST-request handling looks similar to GET-request handling

# Example: Exponent web service

Write a web service that accepts a `base` and `exponent` and outputs `base` raised to the `exponent` power. For example, the following query should output `81` :

```
http://example.com/exponent.php?base=3&exponent=4
```

solution:

```php
<?php
header("Content-type: text/plain");
$base = (int) $_GET["base"];
$exp  = (int) $_GET["exponent"];
$result = pow($base, $exp);
print $result;
?>
```

# Embedded PHP

Embedded PHP is a strategy for generating HTML pages on the server side using PHP.

The textbook assumes that we're using PHP in this way, but we don't. This quarter, we are focusing on using PHP for data generation.

The next couple slides are about embedded PHP if you are interested in learning a little bit more about it

# Embedded PHP syntax template

```
HTML content

  <?php
  PHP code
  ?>

HTML content

  <?php
  PHP code
  ?>

HTML content ...      PHP
```

- any contents of a .php file between <?php and ?> are executed as PHP code
- all other contents are output as pure HTML

# Printing HTML tags in PHP = bad style

```php
<?php
print "<!DOCTYPE html>\n";
print "<html>\n";
print "  <head>\n";
print "    <title>Geneva's web page</title>\n";
...
for ($i = 1; $i <= 10; $i++) {
  print "<p class=\"count\"> I can count to $i! </p>\n";
}
?>
```

PHP

- printing HTML tags with print statements is bad style and error-prone:
  - must quote the HTML and escape special characters, e.g. \"
- but without print, how do we insert dynamic content into the page?

# PHP expression blocks

```
<?= expression ?>
```
PHP

```
<h2> The answer is <?= 6 * 7 ?> </h2>
```
PHP

**The answer is 42**

output

- **PHP expression block:** evaluates and embeds an expression's value into HTML
- `<?= expr ?>` is equivalent to `<?php print expr; ?>`

# Expression block example

```
<!DOCTYPE html>
<html>
  <head><title>CSE 154: Embedded PHP</title></head>
  <body>
    <?php for ($i = 99; $i >= 1; $i--) { ?>
      <p> <?= $i ?> bottles of beer on the wall, <br />
          <?= $i ?> bottles of beer. <br />
          Take one down, pass it around, <br />
          <?= $i - 1 ?> bottles of beer on the wall. </p>
    <?php } ?>
  </body>
</html>
```

PHP

# Complex expression blocks

```
  <body>
    <?php for ($i = 1; $i <= 3; $i++) { ?>
      <h<?= $i ?>>This is a level <?= $i ?> heading.</h<?= $i ?>>
    <?php } ?>
  </body>
```

PHP


output

- expression blocks can even go inside HTML tags and attributes

# Common errors: unclosed braces, missing = sign

```
<body>
    <p>Watch how high I can count:
      <?php for ($i = 1; $i <= 10; $i++) { ?>
        <? $i ?>
    </p>
  </body>
</html>
```
PHP

- </body> and </html> above are inside the for loop, which is never closed
- if you forget to close your braces, you'll see an error about 'unexpected $end‘
- if you forget = in <?=, the expression does not produce any output