Today's Goals

Today's lecture will cover:
- Arrays – declaration and use
- Array functions
- String functions

Variable Scope

- Variables declared within a function are typically visible only within the function
- PHP doesn't give an error when an undeclared variable is being used – it just initializes it to null.
- You will not get an error when using variables that are out of scope, only a null value returned.
- Can resolve this by taking advantage of passing parameters to functions and returning a value from a function.

Variable Scope (continued)

- For example, the following code will output "The value is 25"

```
function myfunc()
{
  $ival = 25;
}
myfunc();
print "The value is ". $ival;
```

Global Variables

- Variables can be made global with the `global` keyword.
- For example, the following code will output "The value is 25"

```
function myfunc()
{
  global $ival;
  $ival = 25;
}
myfunc();
print "The value is ". $ival;
```

Static Variables

- Static variables are not visible outside of the function, but the last value stored in a static variable will be available the next time the function is called.
- They are declared using the `static` keyword.
- They must be initialized in the same line where they are declared or they will be reinitialized with each subsequent execution of the function.
Static Variables (continued)

- The following code:
  ```
  function myfunc()
  {
    static $ival=0; // Initial value
    $ival++;
    return($ival);
  }
  print "The value is ".myfunc()."<br />
  print "The value is ".myfunc()."<br />
  
  has the following output:
  The value is 1
  The value is 2
  ```

Arrays in PHP

- Arrays in PHP are much like arrays in JavaScript except that they include some additional "features"
- As with JavaScript, each object within the array is referenced using an index.
- The elements of an array act just like variables in that you can modify them or use them to define other elements.
- Unless otherwise specified, PHP assigns the first object in the list the index/key 0.
- To use the array's index to point to a specific element, use the square brackets [ and ].

Creating Arrays in PHP

- Declared using `array` keyword
- Initialized using list of items in parenthesis after `array` keyword
- Examples:
  ```
  $names = array("Bob", "Larry", "Mr. Lunt");
  $numbers = array(345, 4562, 72, 1, 657);
  ```
- Arrays may contain mixed data types. This will help us when retrieving a record from MySQL.
- Example:
  ```
  $mixed = array(5.2, "apple", true, 42);
  ```

Creating Arrays in PHP (continued)

- Array elements can also be created by assigning values to new, unset indices/keys.
- If no index is specified, the value is assigned to the next available index.
- Example: `$names[] = "Jimmy"`;

Printing Arrays

- Individual elements from an array can be printed simply by referencing their index.
  ```
  print $names[2]; // Should print "Mr. Lunt"
  ```
- When printing arrays as part of a string, the curly brackets should be used. Take for instance the PHP code:
  ```
  print "Array element 2 is $names[2].";
  ```
- Some PHP engines would output:
  Array element 2 is $names[2].
- To fix this, use the curly brackets:
  ```
  print "Array element 2 is {$names[2]}.";
  ```

Printing Arrays (continued)

- To debug an array, you can also print out the entire array using the `print_r()` function.
- Example:
  ```
  print_r($names);
  ```
- Output from previous code:
- You must use parenthesis with print_r() as it is a function.
Alternate Indices for Arrays

- A nice, but sometimes confusing feature is that PHP allows the use of index values other than integers starting at 0.
- By using the => operator, a different index can be used to identify an array element.
- Syntax: `array(index1=>value1, index2=>value2,...);`
- Example: an array with indices equal to the first four powers of 2.
  ```php
  $sp2 = array(1=>23, 2=>45, 4=>13, 8=>96);
  ```

Strings as Array Indices

- This same method works if you want to use strings as indices to an array.
- Example:
  ```php
  $si = array("first"=>23, "second"=>45, "third"=>13, "fourth"=>96);
  print $si["second"]; // will output "45"
  print_r($si); // will output "Array ( [first] => 23 [second] => 45 [third] => 13 [fourth] => 96 )"
  ```
- This will be helpful when accessing database records.

More About Arrays

- The stored order of elements in arrays corresponds to the order in which they are declared.
  ```php
  $ul[1] = "Keith";
  $ul[3] = "Mick";
  $ul[2] = "Brian";
  $ul[9] = "Charlie";
  $ul[0] = "Ron";
  ```

Multidimensional Arrays

- There are times when arrays must contain data in more than one dimension.
- For example, you would need a 2-dimensional array to represent a matrix.
  ```plaintext
  23 19 -4 3
  42 -9 9 5
  51 33 1 -8
  ```

Initializing 2-Dimensional Array

- The matrix from the following slide would be initialized with code similar to that shown below:
  ```php
  $matrix = Array(
    0 => array(23, 19, -4, 3),
    1 => array(42, -9, 9, 5),
    2 => array(51, 33, 1, -8),
  );
  ```
- The same technique would be used to create arrays of even more dimensions.

"foreach" Loops

- Because of the possibility for arrays with unusual indices, PHP provides a simple method for "visiting" each element of an array.
- The "foreach" loop steps through an array one index at a time based on their stored order.
- Syntax:
  ```php
  foreach(arrayname as [indexname =>] varname) {
    // Code where current array element is referenced using varname with an index of indexname
  }
  ```
"foreach" Example

• The code:

```php
$names = array("Bob", "Larry", "Mr. Lunt");
foreach($names as $thisname)
    print $thisname."<br />";
```

outputs:

Bob
Larry
Mr. Lunt

"foreach" with Associated Indices

• The `foreach` syntax allows the programmer to access the index an array too.

• The code:

```php
$names = array("Bob", "Larry", "Mr. Lunt");
foreach($names as $num => $thisname)
    print "Name ". $num." is ". $thisname."<br />";
```

outputs:

Name 0 is Bob
Name 1 is Larry
Name 2 is Mr. Lunt

In-Class Exercise

• Given the following array, create a PHP script that prints a list of office hours.

```php
$_office_hours = array(
    "Monday" => "2:45 PM to 3:45 PM",
    "Tuesday" => "2:15 PM to 4:15 PM",
    "Wednesday" => "2:45 PM to 3:45 PM",
    "Thursday" => "2:15 PM to 4:15 PM",
    "Friday" => "By appointment"
);
```

Array Functions


• Many of these functions are a result of the flexibility PHP offers by having non-standard indices/keys/

• The next few slides offer some examples of these functions.

array_key_exists()

• As was stated earlier, PHP needs to have additional functionality when navigating arrays since it allows atypical indexing.

• `array_key_exists()` checks for an array index within an array and returns true if it exists.

• Syntax:

```php
boolean array_key_exists(index, array)
```

• For example, from the in-class exercise, the following function call would return a false.

```php
array_key_exists("Saturday", $_office_hours)
```

array_keys()

• Programmers can also get a list of the keys/indices used in an array using `array_keys()`.

• The keys are returned as an array.

• For example, if the following function were run on the in-class exercise array:

```php
array_keys($_office_hours);
```

it would create the following array:

```php
```
count()
• The function count() returns the number of elements in an array.
• The code:

```
$names = array("Bob", "Larry", "Mr. Lunt");
print "Number of elements = ".count($names);
```

outputs:
Number of elements = 3

array_fill()
• The function array_fill() creates and returns an array filled with a designated value.
• Syntax:
  ```
  array_name = array_fill(integer start, integer count, mixed fill_value)
  ```
• The code:

```
$new_array = array_fill(2, 4, "a");
print_r ($new_array);
```

outputs:

range()
• The function range() creates and returns an array filled with a sequence of values starting with a value low and ending at a value high with an optional step. (Our server doesn’t appear to like step.)
• Syntax:
  ```
  array_name = range(mixed low, mixed high[, integer step])
  ```
• The code:

```
$new_array = range("a", "e");
print_r ($new_array);
```

outputs:

max() and min()
• The functions max() and min() can be used to return the maximum and minimum elements of an array. The elements must be numbers
• Syntax:
  ```
  number max(array_of_numbers)
  number min(array_of_numbers)
  ```
• The code:

```
$numbers = array(345, 4562, -72, 1, 657);
print "Maximum = ".max($numbers)."
print "Minimum = ".min($numbers);
```

outputs:
Maximum = 4562
Minimum = -72

in_array()
• To simplify the process of checking an array for a specific element, PHP offers the in_array() function.
• in_array() returns a boolean true if it finds the element in the array.
• Syntax:
  ```
  boolean in_array(mixed element, arrayname)
  ```
• The following would print "betsy is a valid user."

```
$username = "betsy";
$users = array("adam", "betsy", "carl");
if(in_array($username, $users))
  print $username." is a valid user."
```

array_search()
• The problem with in_array() is that frequently, you want the index returned if it is in the array.
• array_search() returns the array index instead of a boolean true if value is found and a false if the value is not found.
• By the way, since a false can act like a 0 which would be the typical index of the first element, use the is-identical to operator "===". This will force the type to match in addition to value.
• The following code would output "2".

```
$users = array("adam", "betsy", "carl");
print array_search("carl", $users);
```
More Array Functions

- `array_key_exists()` – returns true if an element with a specific index/key exists. Returns false otherwise.
- `array_merge()` – returns an array which is the result of combining 2 or more arrays.
- `array_reverse()` – reverses the order of the elements in an array. It can also be told to preserve the indices/keys which would result in the indices/keys also be reversed.

Array Element Sorting Functions

- `sort()` – rearranges array elements in ascending order.
- `rsort()` – rearranges array elements in descending order.
- `asort()` – rearranges array elements in ascending order keeping keys associated with elements.
- `arsort()` – rearranges array elements in descending order keeping keys associated with elements.
- `ksort()` – rearranges array elements in ascending order of keys.
- `krsort()` – rearranges array elements in descending order of keys.

String Functions

- Strings have plenty of functions in PHP too. (Almost 100 according to http://www.php.net/manual/en/ref.strings.php)
- Mercifully, we will not be responsible for them all.

`join()` or `implode()`

- Syntax:
  ```php
  string join(string delimiter, arrayname)
  ```
- The code:
  ```php
  $words = array("The", "dog", "chased", "the", "ball.");
  $sentence = join(" ", $words);
  print $sentence;
  ```
  will produce the string "The dog chased the ball."

explode()

- Syntax:
  ```php
  array explode ( string separator, string string [, int limit] )
  ```
- `explode()` returns an array of strings, each of which is a substring of string of the original string divided at the string separator.
- If limit is used, the maximum number of elements will be set to limit, the last element of which will contain the rest of string.
- The code:
  ```php
  $words = explode ".", "423.439.6404";
  print_r ($words);
  ```

strlen()

- `strlen()` returns the length of the string in characters.
- Syntax: `integer strlen(string)`
- The code:
  ```php
  print strlen("The quick brown fox jumps over the lazy dog.");
  ```
  outputs "44".
**Formatted Output**
- When using print, the programmer is at the mercy of the PHP engine in terms of how elements such as variables will be output.
- For example, print M_PI; will output "3.1415926535898"
- printf() gives formatting control to the programmer.
- Syntax: `printf(string_w_formatting, arguments)`
- Specifiers located within the "string_w_formatting" identify where the arguments are to be placed and the format they are to follow.
- Multiple arguments are separated with commas.

**Specifiers**

<table>
<thead>
<tr>
<th>Specifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%%</td>
<td>&quot;Escape&quot; sequence to print '%'</td>
</tr>
<tr>
<td>%b</td>
<td>Binary integer</td>
</tr>
<tr>
<td>%c</td>
<td>ASCII character</td>
</tr>
<tr>
<td>%d</td>
<td>Signed decimal integer</td>
</tr>
<tr>
<td>%u</td>
<td>Unsigned decimal integer</td>
</tr>
<tr>
<td>%o</td>
<td>Octal integer</td>
</tr>
<tr>
<td>%x</td>
<td>Hexadecimal integer</td>
</tr>
<tr>
<td>%f</td>
<td>Float w/specific decimal point placement</td>
</tr>
<tr>
<td>%s</td>
<td>String</td>
</tr>
</tbody>
</table>

**printf() Examples**
- Code: `printf("Pi = %5.3f", M_PI);`
  Output: "Pi = 3.142"
- Code: `printf("%d in binary is %b", 25, 25);`
  Output: "25 in binary is 11001"
- Code: `printf("The ASCII value of %c is %x hex", 72, 72);`
  Output: "The ASCII value of H is 48 hex"
- Code: `printf("%s owns %d computers", "Tom", 5);`
  Output: "Tom owns 5 computers"

**Modifying Case**
- `strtolower()` – returns a copy of the string argument in all lower case
- `strtoupper()` – returns a copy of the string argument in all upper case
- `ucfirst()` – returns a copy of the string argument with the first character in upper case. Doesn’t affect rest of string, therefore to verify sentence case, use strtolower() first.
- `ucwords()` – returns a copy of the string argument with the first character of each word in upper case. Doesn’t affect rest of string, therefore to verify title case, use strtolower() first.

**Trimming Whitespace**
- There are three functions used to trim leading and/or trailing whitespace
- Whitespace includes spaces, tabs, newlines, and carriage returns
  - `trim(string, character list)` – returns string with leading and trailing whitespace removed
  - `ltrim(string, character list)` – returns string with trailing (right) whitespace removed
  - `rtrim(string, character list)` – returns string with leading (left) whitespace removed
- `string` is the string to be modified
- `character list` allows the programmer to specify a string of the exact characters to trim
- A range of characters is represented with "."
### Comparing Strings
- The most reliable way to compare to strings is with the functions `strcmp()` and `strncmp()`.
- **Syntax:**
  ```c
  integer strcmp(string1, string2)
  integer strncmp(string1, string2)
  ```
- **Return values:**
  - 0 – strings are equal
  - 1 – string2 comes alphabetically before string1
  - –1 – string1 comes alphabetically before string2

### Comparing Strings (continued)
- `strcmp()` and `strncmp()` differ only in that `strncmp()` allows user to limit number of characters compared.
- `strcmp()` and `strncmp()` are case sensitive – lowercase is considered as coming before uppercase
- Use `strcasecmp()` and `strncasecmp()` for case insensitive comparisons.

### Substrings
- There are a number of string functions that operate on substrings.
- In order to use these functions properly, it is important to understand that the index of a character identifies its position within the string
- An index of 0 points to the first character in a string.

### Substring Functions
- **string substr(source, start[, length])** – returns a substring of `source` starting at `start` with length `length`. If `length` is left out, substring ends at end of `source`.
- **integer strpos(source, substring[, offset])** – returns the index of the position where the `substring` first appears in the `source`. If `offset` is included, search starts from that index. Returns false if not found. (Remember `===` operator!)
- **substr_replace(source, replace, start[, length])** – starting at position `start`, inserts `replace` into `source`. `length` identifies the number of characters being replaced, and when omitted, replaces to end of `source`.

### Substr_replace() examples
- ```
  print substr_replace("abcdefgij", "DEF", 3);
  outputs "abcDEF"
  ```
- ```
  print substr_replace("abcdefgij", "DEF", 3, 3);
  outputs "abcDEFghij"
  ```
- ```
  print substr_replace("abcdefgij", "DEF", 3, 0);
  outputs "abcDEfghij"
  ```

### In-class Exercise
Use the string functions to do the following:
- Retrieve the area code from a phone number in format (423)439-6404
- Retrieve just the user name from an e-mail address
- See how many times the letter 't' appears in a string.
- Find "&" and replace it with "&amp;" in a string.