PHP Functions
and
Arrays

Orlando PHP Meetup
Zend Certification Training
March 2009
Functions

- Minimal syntax
  - `function name() { }`
- PHP function names are *not* case-sensitive.
- All functions in PHP return a value—even if you don’t explicitly cause them to.
  - No explicit return value returns NULL
  - “return $value;” exits function immediately.
Variable Scope

✦ Global scope
✦ A variable declared or first used outside of a function or class has global scope.
✦ NOT visible inside functions by default.
✦ Use “global $varname;” to make a global variable visible inside a function.

✦ Function scope
✦ Declared or passed in via function parameters.
✦ A visible throughout the function, then discarded when the function exits.

✦ Case scope
✦ Stay tuned for Object Oriented chapter…
$a = "Hello";
$b = "World";

function hello() {
    global $a, $b;
    echo "$a $b";
    echo "$a $b";
    echo $GLOBALS[’a’] . ’ ’ . $GLOBALS[’b’];
}

Global
Scope (continued)

✦ Regular Parameters
  ✦ function hello($who = "World") {
      echo "Hello $who";
  }

✦ Variable Length Parameters
  ✦ func_num_args(), func_get_arg() and func_get_args()
  ✦ function hello() {
      if (func_num_args() > 0) {
          $arg = func_get_arg(0); // The first argument is at position 0
          echo "Hello $arg";
      } else {
          echo "Hello World";
      }
  }
Passing Arguments by Reference

- Prefix a parameter with an “&”
  - Function func (&$variable) {}
  - Changes to the variable will persist after the function call

```php
function cmdExists($cmd, &$output = null) {
    $output = 'whereis $cmd';
    if (strpos($output, DIRECTORY_SEPARATOR) !== false) {
        return true;
    } else {
        return false;
    }
}
```
Function Summary

- Declare functions with parameters and return values
- By default, parameters and variables have function scope.
- Pass parameters by reference to return values to calling function.
- Parameters can have default values. If no default identified, then parameter must be passed in or an error will be thrown.
- Parameter arrays are powerful but hard to debug.
Arrays

- All arrays are ordered collections of items, called *elements*.
  - Has a value, identified by a unique key (integer or case sensitive string)

- Keys
  - By default, keys start at zero
  - String can be of any length
  - Value can be any variable type (string, number, object, resource, etc.)
Declaring Arrays

**Explicit**

- $a = \textbf{array} (10, 20, 30);
- $a = \textbf{array} (’a’ => 10, ’b’ => 20, ’cee’ => 30);
- $a = \textbf{array} (5 => 1, 3 => 2, 1 => 3,);
- $a = \textbf{array}(); //Empty

**Implicit**

- $x[] = 10; //Gets next available int index, 0
- $x[] = 11; //Next index, 1
- $x[’aa’] = 11;
Printing Arrays

- **print_r()**
  - Recursively prints the values
  - May return value as a string to assign to a variable.

- **Var_dump()**
  - Outputs the data types of each value
  - Can only echo immediately
Enumerative vs. Associative

Enumerative
- Integer indexes, assigned sequentially
- When no key given, next key = \( \text{max} (\text{integer keys}) + 1 \)

Associative
- Integer or String keys, assigned specifically
- Keys are case sensitive but type insensitive
  - ‘a’ != ‘A’ but ‘1’ == 1

Mixed
- \$a = \text{array} (‘4’ => 5, ’a’ => ’b’);
- \$a[] = 44; // This will have a key of 5
Multi-dimensional Arrays

- **Array item value is another array**

  ```
  $array = array();
  $array[] = array('foo','bar');
  $array[] = array('baz','bat');
  echo $array[0][1] . $array[1][0];
  ```
Unraveling Arrays

- list() operator
  - Receives the results of an array
  - List($a, $b, $c) = array(‘one’, ‘two’, ‘three’);
  - Assigns $a = ‘one’, $b = ‘two’, $c = ‘three’
- Useful for functions that return an array
  - list($first, $last, $last_login) = mysql_fetch_row($result);
Comparing Arrays

- **Equal (==)**
  - Arrays must have the same number and value of keys and values, but in any order

- **Exactly Equal (===)**
  - Arrays must have the same number and value of keys and values and in the exact same order.
Counting, Searching and Deleting Elements

- `count($array)`
  - Returns the number of elements in the first level of the array.

- `array_key_exists($key, $array)`
  - If array element with key exists, returns TRUE, even if value is NULL

- `in_array($value, $array)`
  - If the value is in the array, returns TRUE

- `unset($array[$key])`
  - Removes the element from the array.
Flipping and Reversing

- `array_flip($array)`
  - Swaps keys for values
  - Returns a new array

- `array_reverse($array)`
  - Reverses the order of the key/value pairs
  - Returns a new array

- Don’t confuse the two, like I usually do
Array Iteration

The Array Pointer
- reset(), end(), next(), prev(), current(), key()

An Easier Way to Iterate
- foreach($array as $value) {}
- foreach($array as $key=>$value){}

Walk the array and process each value
- array_walk($array, ‘function name to call’);
- array_walk_recursive($array, ‘function’);
Sorting Arrays

There are eleven sorting functions in PHP:

- **`sort($array)`**
  - Sorts the array in place and then returns.
  - All keys become integers 0,1,2,…

- **`asort($array)`**
  - Sorts the array in place and returns
  - Sorts on values, but leaves key assignments in place.

- **`rsort()` and `arsort()` to reverse sort.**
- **`ksort()` and `krsort()` to sort by key, not value.**
Sorting (continued)

usort($array, ‘compare_function’)

Compare_function($one, $two) returns

- 1 if $one < $two
- 0 if $one == $two
- 1 if $one > $two

Great if you are sorting on a special function, like length of the strings or age or something like that.

uasort() to retain key=>value association.

shuffle($array) is the anti-sort
Arrays as Stacks, Queues and Sets

Stacks (Last in, first out)
- array_push($array, $value [, $value2] )
- $value = array_pop($array)

Queue (First in, first out)
- array_unshift($array, $value [, $value2] )
- $value = array_shift($array)

Set Functionality
- $array = array_diff($array1, $array2)
- $array = array_intersect($a1, $a2)
Summary

- Arrays are probably *the* single most powerful data management tool available to PHP developers. Therefore, learning to use them properly is essential for a good developer.
- Some methods are naturally more efficient than others. Read the function notes to help you decide.
- Look for ways in your code to use arrays to reduce code and improve flexibility.
Thank You

Shameless Plug

- 15 years of development experience, 14 of that running my own company.
- I am currently available for new contract work in PHP and .NET.
- cchubb@codegurus.com