

# PHP Arrays for Fun and Profit

UPHPU Meeting  
November 17, 2005  
Mac Newbold  
mac@macnewbold.com

## Who am I?

Full-time self-employed computer geek  
Code Greene ([www.codegreene.com](http://www.codegreene.com), partner)  
Wide variety of PHP/MySQL web sites and applications  
Fan of FreeBSD, Firefox, Flyspray, etc.  
Background: B.S. C.S. '01, M.S. C.S. '05  
University of Utah – Go Utes!

## The Plan

### Array Basics

- Lists and Associative Arrays
- Array operators and functions

### Using Arrays

- Looping with Arrays
- Sorting Arrays
- Using arrays as parameters to functions

### Data Structures with Arrays

## The Array Data Type in PHP

[www.php.net/manual/en/-language.types.array.php](http://www.php.net/manual/en/-language.types.array.php)

Two types: Lists and Associative Arrays

- Every array is pairs of keys and values

Consecutive numeric keys (0-based) is a list

- Implied when you don't specify any keys

Other keys (i.e. strings) is an associative array, a.k.a. hash, map, etc.

## Making Arrays

```
$list = array('foo', $bar, "BAZ", 1, 2, 3);
```

```
List: [0]=>'foo', [1]=>$bar, [2]=>... ..  
echo $list[0]; foo  
$map=array("dog"=>"brown","cat"=>"fat")  
Associative array: [dog]=>brown, [cat]=>fat  
echo $map["dog"]; brown  
$list[] = 4;  
$map["mouse"] = "quiet";
```

## Array Operators

5 for comparison, 1 other

==, !=, <> – Test for [in]equality

Same key/value pairs (using “type juggling”)

===, !== – Test for identity

Same pairs, same order, same types

Union: + (see also array\_merge())

Numeric keys get reindexed

For duplicate keys, left hand dominates

## Array Functions

Over 75 array functions

Some are obvious and self-explanatory:

```
array(), is_array(), unset(), in_array(), count(), shuffle(),  
array_keys(), array_values(), array_reverse(), array_sum(),  
array_product()
```

```
array_[push,pop,shift,unshift]()
```

Push and pop at the end

Shift (on) or unshift (off) at the beginning

## Array Functions 2

Set operations: Union, Diff, Intersect

```
array_[u]{diff,intersect}[_[u]{assoc,key}]()
```

Plain: compare values

Key: compare keys

Assoc: compare values and keys

u/uassoc/ukey: Compare with callback function

udiff/uintersect: allow recursive/deep compare

Rarely need anything but plain, key, assoc

## Array Functions 3

```
$str = implode(".", $arr);  
$arr = explode(".", $str);  
$arr = array_combine($keys, $values);  
$arr = array_flip($arr); // val=>key  
$arr = array_pad($arr_in, 10, "");  
array_slice, array_splice – cut/paste  
Extract, compact – array <=> variables
```

## Array Functions 4

Array\_walk() – apply callback to pairs

Array\_map() – return array of results of applying callback to pairs

Array\_unique() – delete duplicate values

Array\_rand() – choose random key(s)

list(\$first, \$second, \$third) = \$arr;

There are a bunch more too...

## The Plan

### Array Basics

- Lists and Associative Arrays

- Array operators and functions

### Using Arrays

- Looping with Arrays

- Sorting Arrays

- Using arrays as parameters to functions

### Data Structures with Arrays

## Looping with Arrays

Arrays have internal pointers for iterating

- Current(), prev(), next(), reset(), end(), key()

- list(\$key, \$val) = each(\$arr) – grabs next pair

```
foreach ($arr as $v) { }
```

```
foreach ($arr as $k => $v) { }
```

```
while (list($k, $v) = each($arr)) { }
```

```
for ($i=0; $i<count($arr); $i++) { $arr[$i];}
```

## Sorting Arrays

sort() – sort by value

ksort() – sort by key  
asort() – sort by value, keep associations  
rsort(), krsort(), arsort() – reverse  
usort(), uksort(), uasort() – custom compare  
natsort(), natcasesort() – “natural order”  
shuffle() – You guessed it...

## Arrays as Parameters

Many string functions can do more complex things when arrays are used as parameters:

```
$arr = array('foo','bar');  
str_replace($arr,"baz",$str);  
str_replace($arr,array('blah','boo'),$str);
```

Passing arrays by reference lets you “return” multiple values from a function:

```
function getUser(&$result) { }
```

## Arrays as Return Values

Rather than returning by reference, you can simply return an array directly:

```
return array("rv1"=>17,"rv2"=>42);
```

Many standard functions return arrays:

MySQL for example: `mysql_fetch_*()`

Assoc gets associative, row gets numeric, array gets either, or both at once

Remember: `list($rv1, $rv2) = getPair();`

## Data Structures with Arrays

Plain arrays already do a whole lot:

List, vector, hash, map, set, stack, queue, etc.

Trees and recursive structures are easy:

Multi-dimensional arrays: `$arr[3][7][9];`

Hash/map of arrays - `$arr["person1"]["name"]`

“Struct” (a la C) or “mock object”

```
$obj = array("name"=>"joe", "age"=>"64");
```

Quicker/simpler than going fully OOP

## Arrays and HTTP/HTML

Make arrays with GET or POST variables:

```
index.php?list[0]=foo&list[1]=bar
```

```
index.php?list[]=blah&list[]=hm&list[]=wow
```

HTML Forms for arrays:

```
<input name="person[name]">
```

```
<input type="checkbox" name="list[yes]">
```

```
<select multiple name="choices[]">
```

Always use [] for multiple select – the only way

## **Grand Finale**

Arrays are pretty simple, yet incredibly powerful at the same time. They're well worth any time you invest in learning to use them well.

Any questions?