

Intro to PHP

References:

- Programming PHP (O'Reilly)
- PHP Cookbook (O'Reilly)
- www.php.net
- www.google.com/search?q=php

PHP and WWW

- PHP was created with web programming in mind (it was not developed as a general purpose programming language).
 - BUT you can use php from the command line.
- Typically used with Apache (as a module), but can run with any server that supports CGI.
- Close ties with database systems (esp. MySQL)

PHP Development

- Programming model is like ASP:
 - HTML and code are in the same file – special tags used to indicate server-side code.

`<?php . . . ?>`

- Conditional HTML
 - Session support, cookies, form fields, etc.
- Language is similar to Perl.
 - But simpler language – much of the *fancy* language stuff in Perl is provided by libraries in PHP.

PHP Language

- Data Types
- Literals
- Variables
- Operators
- Control Structures
- Functions
- Objects

Syntax Overview

- Case-insensitive (functions, control structures, class names)
- Variable names are case-sensitive.
- Semicolons just like Perl/C/C++;
- Whitespace is insignificant.
- Comments: `/* */`, `//` and `#` all supported

Data Types

- Scalars:
 - Integers
 - Floating-point
 - String
 - Boolean
- Compound
 - Arrays
 - Objects
- Resource
- null

Support functions:

```
is_int(), is_float(), is_string(), is_bool(),  
is_object(), is_resource(), is_null()
```

Scalar Literals

- Integers, floating point and string literals are all pretty much just like C/C++.
 - Hex and octal notation supported for ints
 - Escape sequences inside double quotes.
- Booleans – all the following literals are *false*:
`false 0 0.0 "" "0"`
 - anything else is *true*

Variable *Interpolation*

- You can put a variable name inside a double-quoted string, and it will be replaced with the *value* of the variable:

```
$name="fred";
```

```
$time="11:25";
```

```
$s = "Hello $name, it is now $time";
```

- The variable `$s` now holds the string:

```
"Hello fred, it is now 11:25"
```

Variable Interpolation is very useful!

Instead of this:

```
echo "<img src='" . $imgurl . "' width=" .  
    $width . ">";
```

you find yourself doing this:

```
echo "<img src='$imgurl' width=$width>"
```

Arrays and Array Literals

- There is a single array type that handles both:
 - traditional arrays (indexed by integers – Perl list)
 - associative arrays (Perl hash)
- There are no true array literals, but there is a special array construct:

```
array(1, 2, 4, 8);
```

```
array('red' => '#ff0000',  
      'green' => '#00ff00',  
      'blue' => '#0000ff');
```

Variables

- Variable identifiers follow *the usual* variable naming rules (start with letter, letters, digits, _).
- Variable identifiers are case-sensitive.
- Variable named start with \$ (like Perl).
- Variables don't have pre-determined types (you can assign any type value to any variable).
- There is no variable declaration – just use a variable and it gets created.

PHP Variables (cont.)

- Variable variables:

- `$a="b" , $b=3; echo $$a; → "3"`

- References: use `&` to create a reference:

- `$a=12; $b= & $a; echo $b; → "12"`

- Also called *aliases*

- Functions can return references (but the caller needs to know this):

```
function &foo() { return 12; }      $x = & foo();
```

Variable Scope

- Local: variable created in a function are only visible within the function.
- Global: variables created outside of any function definition. Not available inside a function unless the function uses the `global` keyword.
- Static: declared as static inside a function – retains its value (but is visible only in the function).
- Function parameters: local scope.

Garbage Collection

- Copy-on-write
 - Copy of a value is not allocated until it needs to be (a change is made to one copy).
- Reference counting
 - When there are no references to a value, the memory is released.

Operators

- Arithmetic: the usual gang.
- Autoincrement/decrement: yup (including strings!)
- Comparison: just one set of operators, and automatic type conversion (strings/numbers).
- Bitwise Logical and Logical:
 - `&&` `||` `!`
 - `and` `or` `xor`

More operators

- Casting: `(int)` `(float)` `(string)`
`(bool)` `(array)` `(object)`
- Assignment: like C/C++ `$a = ($b = 3);`
 - Assignment with operation also.
- String concatenation operator: `.`

```
echo "hi" . $name;
```

```
$a .= "</td>\n";
```

And more operators

- Error suppression: `@` `@ ($a = $x / $y) ;`
 - Handles runtime errors (important for WWW!)
 - There are other ways to handle errors in PHP.
- Execution: ```` `$a = `ls -al` ;`
 - Dangerous and generally not portable (but very useful!).

Control Structures

- Similar to C/C++/JavaScript.
 - Alternate syntax for control structures that doesn't use curly braces!
- A few new control structures not in C or JavaScript.

if / else / elseif

```
if ($blah==3)
    $foo="bar";

if (is_int($a)) {
    $b=(string)$a;
    echo "in an if";
} else
    echo "else it is";
```

```
if ($g > 90)
    record("A");
elseif ($g > 80)
    record("B");
elseif ($g > 70)
    record("C");
else
    record("F");
```

Alternate syntax for `if`

- Similar syntax for other control structures (but I won't go over all of them...).

```
if (expression) :
```

```
    statement1;
```

```
    statement2;
```

```
    . . .
```

```
else :
```

```
    statements . . .;
```

```
endif;
```

switch

- Like the C switch statement, but not restricted to integral types!
- fall-through execution, break, default all like C.

```
switch($query) {  
    case 'edit':  
        do_edit(); break;  
    case 'delete':  
        do_del(); break;  
    case 'add':  
        do_add(); break;  
    default:  
        unknown_q();  
}
```

`while / do while / for`

- Just like C / C++ / JavaScript.
 - Nothing to see here – move on...

foreach

- iterate over elements in an array.
 - *operates on a copy of the array*

```
foreach ( $array as $current ) {  
    statements...  
}
```

```
foreach ( $array as $key => $val ) {  
    statements...  
}
```

foreach example

```
echo "<table>";  
foreach ($_REQUEST as $fname=>$fval) {  
    echo "<tr><td>$fname</td>";  
    echo "<td>$fval</td></tr>\n";  
}  
echo "</table>\n";
```

Functions

- Define using `function` keyword.
- Named parameters (treated as local variables).
- Can return value or reference.
- Parameters can be values or references.
- All variables are local (unless declared `global`)
- Explicit return statement.

Sample Functions

```
function sum( $a, $b ) {  
    return($a+$b);  
}
```

```
function increment_variable( & $a ) {  
    $a++;  
}
```

```
function & select_arrayelem(& $arr, $i) {  
    return $arr[$i];  
}
```

Returning/Passing References

- A variable becomes a reference only when assigned a reference value.

`$blah` will just get a copy of the value of `$a[2]`;
`$blah = select_arrayelem($a, 2);`

`$blah` will be a reference to `$a[2]`;
`$blah = &select_arrayelem($a, 2);`

Local/Global variables

- All variables used in a function are local unless explicitly declared as global.

```
global $foo;
```

- variable scope is "the entire function", there is no block scope.

```
for ($i=0;$i<3;$i++) {  
    echo "i is now $i\n";  
}
```

 "i is now 3"

Static Variables

- Like C/C++:
 - initialization happens only once.
 - variable retains value between function calls.

```
function counter() {  
    static $count=0;  
    return $count++;  
}
```

Pass By Reference

- Function parameter is declared as a reference.
- Caller doesn't do anything special!

```
function incarray( & $arr ) {  
    for ( $i=0; $i<count( $arr ); $i++ )  
        $arr[ $i ]++;  
}  
  
$blah = array( 1, 2, 3 );  
incarray( $blah );
```

Default Parameters

- Each parameter can have a default value.
 - these need to be declared after all other parameters.

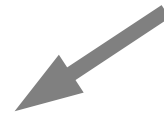
```
function inc( $x, $inc=1) {  
    return($x+$inc);  
}
```

Variable Functions

- The name of a function can be stored in a variable:

```
switch($form) {  
  case 'login':  
    login(); break;  
  case 'logout':  
    logout(); break;  
  case 'showcart':  
    showcart(); break;  
  case 'emptycart':  
    emptycart(); break;  
}
```

these do the same thing!



```
$form();
```

PHP Objects

- Fairly simple object model:
 - single inheritance
 - no protection (public/private) for data members or methods.
- Generally used to encapsulate related data and code (some libraries are provided as classes).
- Familiar syntax for accessing data members and methods: `$a->foo() ;` `$a->blah ;`

Creating Objects

- Keyword `new`.
- Constructor can support parameters.
- PHP assignment semantics – a copy is made!

```
$foo = new MyClass(13, "hello");
```

```
$bar = & new MyClass();
```

```
$blah = new SomeClass;
```

Defining Classes

- Keywords: class var extends

```
class Student extends Person {
    var $stolen_music_collection;
    function Student($n,$a,$m) {
        $this->Person($n,$a);
        $stolen_music_collection = $m;
    }
    function stealMoreMusic($newsong) {
        $stolen_music_collection->steal($newsong);
    }
}
```

```
class Person {
    var $name,$age;
    function Person($n,$a) {
        $name=$n; $age = $a;
    }
}
```

Object Support in PHP

- Introspection
 - determining methods, data members, inheritance
- Serialization
 - conversion to/from bytestream
 - used to support persistent objects.
 - used to support PHP session and session variables.

PHP function library

- Tons of stuff... check out

<http://www.php.net/manual/en/funcref.php>

- Some important groups of functions:

- regexp POSIX and Perl

- array functions

- sorting, searching, extraction, counting, etc.

- string functions (similar to C library)

- database access

- network protocols (ftp, mail, ldap, http...)

Web Support

- Some global arrays:
 - `$_COOKIE`
 - `$_GET`
 - `$_POST`
 - `$_REQUEST` (holds everything!)
 - `$_SERVER`
- `register_globals` option in `php.ini` config file is dangerous (most people turn it off).