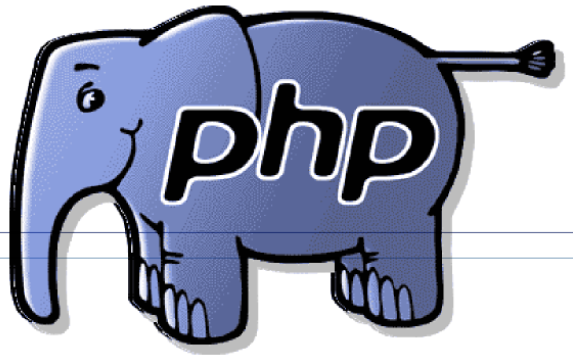
COMP 2021

Unix and Script Programming



PHP Introduction

PHP Introduction

PHP is a recursive acronym for "PHP: Hypertext Preprocessor" -- It is a widely-used open source general-purpose scripting language that is especially suited for web development and can be embedded into HTML.

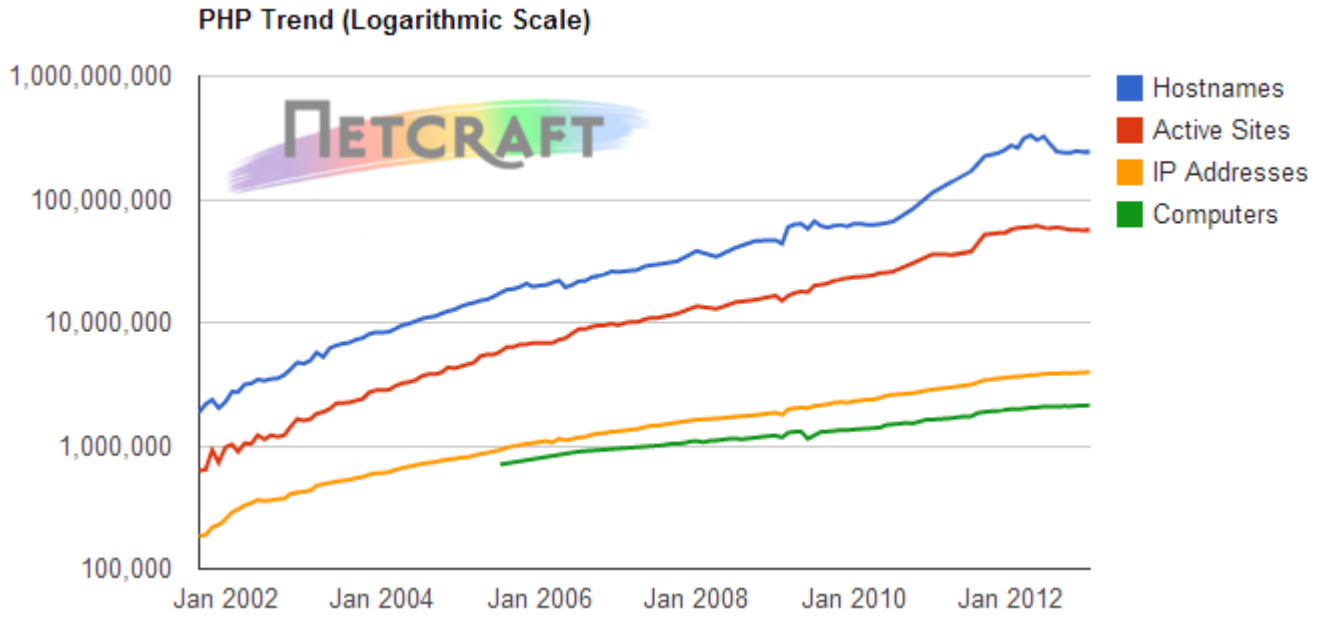
Free to download from http://www.php.net/

What can it do?

- It can work as a stand-alone script
- Most commonly used as: Server-side scripting language
- Used to make web pages dynamic:
  - Provide different content depending on context
  - Interface with other services: database, e-mail, etc.
  - Authenticate users
  - Process form information
- PHP code can be embedded in XHTML code

Why PHP?

- Free and open source
- Simple
- Compatible all major Operating System
- Support for most of the web servers



PHP: Hello World!

```
# helloworld.php
<?php
print "Hello, world!";
?>
```

PHP

```
Hello world!
```

output

PHP Syntax Template

```
<?php
PHP code
?>
```
PHP

```
HTML content
<?php
PHP code
?>
HTML content
<?php
PHP code
?>
HTML content ...
```
PHP embedded in HTML

- Contents of a .php file between `<?php` and `?>` are executed as PHP code
- All other contents are output as pure HTML
- We can switch back and forth between HTML and PHP "modes"

Execute PHP Script

- Run PHP from the command line
  - Output simply comes out on the terminal
  - PHP parser needed

```
ras1.cse.ust.hk:lixin:129> php helloworld_cli.php
Hello world!
```

- Run PHP as a stand-alone script
  - Shebang line
  - Make the script executable: `chmod u+x`

```
ras1.cse.ust.hk:lixin:125> cat helloworld.php
#!/usr/bin/env php

<?php
print "Hello world!";
?>

ras1.cse.ust.hk:lixin:126> chmod u+x helloworld.php
ras1.cse.ust.hk:lixin:127> helloworld.php

Hello world!
```

PHP Embedded in HTML

```
ras1.cse.ust.hk:lixin:140> cat helloworld.php
#!/usr/bin/env php

<html>
 <head>
  <title>PHP Test</title>
 </head>
 <body>
 <?php echo "<h2>Hello World</h2>"; ?>
 </body>
</html>
ras1.cse.ust.hk:lixin:141> chmod 755 /homes/lixin/public_html/cgi-bin/helloworld.php
ras1.cse.ust.hk:lixin:142>
```

Name is *.php

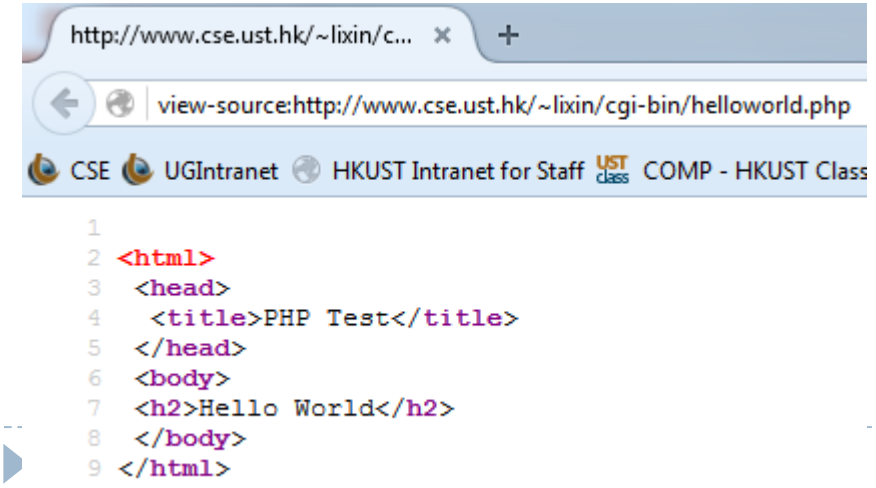
Shebang line, might vary for different servers

Embedded PHP code

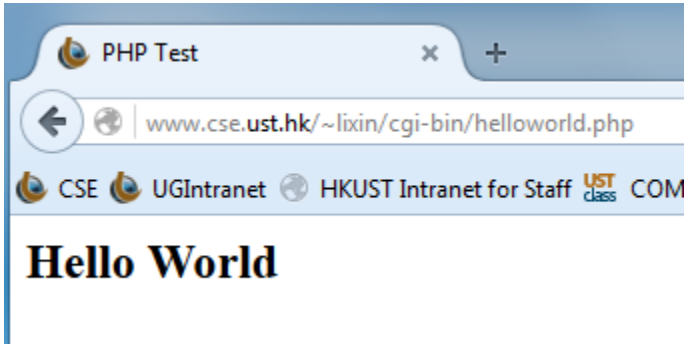
Make it executable

Place under /public_html/cgi-bin directory

html output generated by PHP



html interpreted and shown in web browser



Web Server and IDE

- Web service from ITSC
  - Personal homepage
  - https://itsc.ust.hk/services/general-it-services/communication-collaboration/ihome
  - Check "Running CGI programs“
  - Transfer file to/from server using FTP (e.g. FileZilla)
- Personal web server on your computer
  - XAMPP (available free for Windows, Linux and OS X versions)
  - https://www.apachefriends.org/
- IDE (Integrated Development Environment)
  - Simple PHP code: any plain text editor will do
  - More support: PHPStorm https://www.jetbrains.com/phpstorm/
  - 1-year free license for university students if registered with ITSC email

PHP Cheat Sheet

```
$s = $s[$i];  // a character from a string
```

```
$n++;  ++$n;  // post- and pre-increment
$n--;  --$n;  // post- and pre-decrement
```

```
$i = ~$i;   // bitwise complement
$n = -$n;
$Z = @$Z;   // hide error messages
$b = (bool)$z;   $b = (boolean)$z;
$i = (int)$z;   $i = (integer)$z;
$x = (double)$z;   $x = (float)$z;   $x = (real)$z;
$s = (string)$z;
$a = (array)$Z;
$o = (object)$Z;
NULL = (unset)$Z;
```

Specify the class name, a string containing the class name, or an object instance of the class.

```
$b = $Z instanceof C;
$b = $Z instanceof $sClassName;
$b = $Z instanceof $o;
```

Is $Z an object of the class? Or a class that extends or implements the class?

```
$b = !$b;  // boolean not
```

```
$n = $n * $n;
$n = $n / $n;
$i = $i % $i;  // integer modulo
```

```
$a = $a + $a;  // union by keys, not values
$n = $n + $n;
$n = $n - $n;
$s = $s . $s;  // string concatenate, à la Perl
```

```
$i = $i << $i;  // shift bits left
$i = $i >> $i;  // shift bits right
```

```
$z = $z <  $z;  // less
$z = $z <= $z;  // less or equal
$z = $z >= $z;  // more or equal
$z = $z >  $z;  // more
```

all these are CScs

```
$b = $Z == $Z;  // loose equal   not = assignment
$b = $Z != $Z;  // not loosely equal
$b = $Z <> $Z;  // inequal
$b = $Z === $Z;  // strictly equal (same type)
$b = $Z !== $Z;  // not strictly equal
```

```
$i = ($i & $i);  // bitwise 'and' (use parens)
$Z = &$Z;  // assign (or pass or return) by reference
```

Integer ⇦ String

```
assert(44 === (int)'44');
assert(44 === (integer)'44');
assert(44 === 0+'44');
assert(2.0 == doubleval('2'));
assert(-2.0 == floatval('-2'));
assert(-3 == intval('-3garbage'));
```

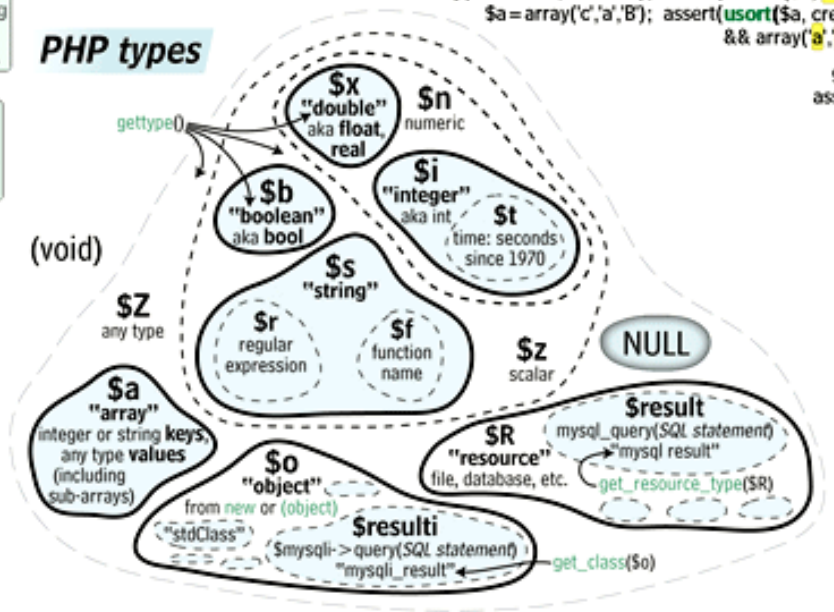
String ⇦ Integer

```
assert('44' === (string)44);
assert('44' === '' . 44);  // (concatenate empty string)
assert('3.14' == strval(3.140));
```

Array Sort

```
$a = array(2,1,1);  $b = array(0,8,9);  assert(array_multisort($a,$b) && array(1,1,2) === $a && array(8,9,0) === $b);
                    $a = array(7,8,9);  assert(arsort($a) && array(2 => 9, 1 => 8, 0 => 7) === $a);
                    $a = array(9,8,7);  assert(asort($a) && array(2 => 7, 1 => 8, 0 => 9) === $a);
$a = array(8 => 1, 7 => 1, 9 => 1);     assert(krsort($a) && array(9 => 1, 8 => 1, 7 => 1) === $a);
$a = array(8 => 1, 7 => 1, 9 => 1);     assert(ksort($a) && array(7 => 1, 8 => 1, 9 => 1) === $a);
                  $a = array('A222', 'a99');  assert(natcasesort($a) && array(1 => 'a99', 0 => 'A222') === $a);
                  $a = array('a222', 'a99');  assert(natsort($a) && array(1 => 'a99', 0 => 'a222') === $a);
                    $a = array(8,7,9);  assert(rsort($a) && array(9,8,7) === $a);
                    $a = array(8,7,9);  assert(sort($a) && array(7,8,9) === $a);
                $a = array('K12','k9');  assert(uasort($a, 'strnatcasecmp') && array(1 => 'k9', 0 => 'K12') === $a);
$a = array('b12' => 0,'b4' => 0);  assert(uksort($a, 'strnatcmp') && array('b4' => 0,'b12' => 0) === $a);
                  $a = array('c','a','B');  assert(usort($a, create_function('$L,$R', 'return strcasecmp($L,$R);'))
                                            && array('a','B','c') === $a);
```

PHP types



Type Test

```
guess('stdClass' == get_class($o));
assert('mysql link' == get_resource_type($link));
assert('integer' == gettype(9));
assert(is_array(array(1,2,3)));
assert(is_bool(FALSE));
assert(is_callable('strlen'));
assert(is_double(7.5));
assert(is_float(7.5));
assert(is_integer(3) && is_int(3));
assert(is_null(NULL));
assert(is_numeric(2.55e2));
assert(is_object(new stdClass));
assert(is_real(7.5));
guess(is_resource(mysql_connect('x.com')));
assert(is_scalar('s') && !is_scalar(array()));
assert(is_string('abc'));
```

Variables are function-local. The two ways to use a global variable within a function:

```
global $variable;  // once at function top
$GLOBALS['variable']  // anywhere.
```

Every function behaves as if it had these automatically:

```
global $GLOBALS, $_COOKIE, $_ENV, $_FILES, $_GET;
global $_POST, $_REQUEST, $_SERVER, $_SESSION;
```

Forms

```
guess('file.jpg' == $_FILES['pic']['name']);  // e.g. from <input type='file' name='pic'>
guess('value' == $_GET['field']);  // aka $HTTP_GET_VARS[]  e.g. <input name=\'field\'>
guess('value' == $_POST['field']);  // aka $HTTP_POST_VARS[]
guess('value' == $_REQUEST['field']);  // merged $_GET[] and $_POST[] etc.
```

Toolbox

```
assert(4 == 2+2);
assert('4' == 2+2');  // more informative failure
define('THREE', 3);  assert(3 == THREE);
```

```
srand(0);  assert(12345 == rand());
```

```
define('TWO', 2);  assert(defined('TWO'));  // remember quotes
```

Syntax and Philosophy

- Syntax inspired by C and Perl
  - C: Curly braces, semicolons, no significant whitespace
  - Perl: $ to start variable names, associative arrays
- Extends HTML to add segments of PHP within an HTML file
- Philosophy
  - You are a responsible and intelligent programmer
  - You know what you want to do
  - Some flexibility in syntax is OK - style choices are OK
  - Lets make this as convenient as possible
  - Sometimes errors fail silently

PHP Basics

Comments

```
# single-line comment
// single-line comment
/*
multi-line comment
*/
```

- Similar to C, but # is also allowed
  - A lot of PHP code uses # comments instead of //

Console output: `print`

```
print "text";
```

Syntax

```
<?php
   print "Hello, World!\n";
   print "Escape \"chars\" are the SAME as in C!\n";
   print "You can have
   line breaks in a string. ";
   print 'A string can use "single-quotes". It\'s cool!';
?>
```

PHP

```
Hello, World!
Escape "chars" are the SAME as in C!
You can have
line breaks in a string. A string can use "single-quotes".
It's cool!
```

output

Variables

```
$name = expression;
```

```
<?php
    $user_name = "cindyli";
    $age = 18;
    $drinking_age = $age + 3;
    $this_class_rocks = TRUE;
?>
```

- Names are case sensitive
- Names always begin with $, on both declaration and usage
- Always implicitly declared by assignment (type is not written)
- A loosely typed language (like JavaScript or Python)

Variables (cont.)

- Basic types
  - `int`, `float`, `boolean`, `string`, `array`, `object`, `NULL`
  - Test type of variable with `is_type` functions, e.g. `is_string`
  - `gettype` function returns a variable's type as a string
- PHP converts between types automatically in many cases:
  - `string` → `int` auto-conversion on `+`
  - `int` → `float` auto-conversion on `/`
- type-cast with `(type)`
  - `$age = (int) "21";`

`Int` and `Float` Types

```
<?php
   $a = 7 / 2; # float: 3.5
   $b = (int) $a; # int: 3
   $c = round($a); # float: 4.0
   $d = "123"; # string: "123"
   $e = (int) $d; # int: 123
?>
```

- `int` for integers and float for `reals`
- division between two `int` values can produce a `float`

Arithmetic Operators

- `+` `-` `*` `/` `%` `.` `++` `--`
- `=` `+=` `-=` `*=` `/=` `%=` `.=`
- many operators auto-convert types: `5 + "7"` is `12`

Math Operations

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

Math functions

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

String Type

```
<?php
   $favorite_food = "Ethiopian";
   print $favorite_food[2];
   $favorite_food = $favorite_food . " cuisine";
   print $favorite_food;
?>
```

- Zero-based indexing using bracket notation
- There is no char type; each letter is itself a String
- String concatenation operator is . (period), not +
  - `5 + "2 turtle doves" === 7`
  - `5 . "2 turtle doves" === "52 turtle doves"`
- Can be specified with " " or ' '

String Functions

```
<?php
   # index 0123456789012345
   $name = " Leonardo DiCaprio";
   $length = strlen($name);
   $cmp = strcmp($name, "Leo");
   $index = strpos($name, "e");
   $first = substr($name, 9, 5);
   $name = strtoupper($name);
?>
```

Name	Function
strlen	length
strpos	indexOf
substr	substring
strtolower, strtoupper	toLowerCase, toUpperCase
trim	trim
explode, implode	split, join
strcmp	compareTo

Interpreted Strings

```
<?php
$age = 16;
print "You are " . $age . " years old.\n";
print "You are $age years old.\n"; # You are 16 years old.
?>
```

- Strings inside " " are interpreted
  - Variables that appear inside them will have their values inserted into the string
- Strings inside ' ' are not interpreted

```
<?php
print ' You are $age years old.\n '; # You are $age years
old. \n
?>
```

Interpreted Strings (cont.)

```
<?php
print "Today is your $ageth birthday.\n";
# $ageth not found
print "Today is your {$age}th birthday.\n";
?>
```

- if necessary to avoid ambiguity, can enclose variable in { }

Interpreted Strings (cont.)

```
<?php
$name = "Xenia";
$name = NULL;
if (isset($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)

`bool` (Boolean) type

```
$feels_like_summer = FALSE;
$php_is_great = TRUE;
$student_count = 31;
$nonzero = (bool) $student_count; # TRUE
```

- the following values are considered to be FALSE (all others are TRUE):
  - 0 and 0.0 (but NOT 0.00 or 0.000)
  - " ", "0", and NULL (includes unset variables)
  - Arrays with 0 elements
- FALSE prints as an empty string (no output); TRUE prints as a 1

`for` loop (same as C)

```
for (initialization; condition; update) {
      statements;
}
```

```
<?php
for ($i = 0; $i < 10; $i++) {
      print "$i squared is " . $i * $i . ".\n";
}
?>
```

`if/else` statement (similar to C)

```
if (condition) {
      statements;
} elseif (condition) {
      statements;
} else {
      statements;
}
```

`while` loop (same as C)

```
while (condition) {
      statements;
}
```

```
do {
      statements;
} while (condition);
```

Create Array

```
// An array called $dinner with numeric keys
$dinner[0] = 'Sweet Corn and Asparagus';
$dinner[1] = 'Lemon Chicken';
$dinner[2] = 'Braised Bamboo Fungus';

//associative array with (key, value) pairs
//An array called $vegetables with string keys
$vegetables['corn'] = 'yellow';
$vegetables['beet'] = 'red';
$vegetables['carrot'] = 'orange';


// An array called $computers with numeric and string keys
$computers['trs-80'] = 'Radio Shack';
$computers[2600] = 'Atari';
$computers['Adam'] = 'Coleco';
```

Create Array with `array()`

```
$name = array();          # create
$name = array(value0, value1, ..., valueN);
$name[index]              # get element value
$name[index] = value;     # set element value
$name[] = value;          # append
```

```
$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)
```

- Append: use bracket notation without specifying an index
- Element type is not specified; can mix types

`array()` with Associative Array

```
$vegetables = array('corn' => 'yellow',
                    'beet' => 'red',
                    'carrot' => 'orange');

$dinner = array(0 => 'Sweet Corn and Asparagus',
                1 => 'Lemon Chicken',
                2 => 'Braised Bamboo Fungus');

$computers = array('trs-80' => 'Radio Shack',
                   2600 => 'Atari',
                   'Adam' => 'Coleco');
```

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

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

foreach loop

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

```
<?php
$fellowship = array("Frodo", "Sam", "Gandalf", "Strider",
"Gimli", "Legolas", "Boromir");

print "The fellowship of the ring members are: \n";
for ($i = 0; $i < count($fellowship); $i++) {
        print "{$fellowship[$i]}\n";
}

print "The fellowship of the ring members are: \n";
foreach ($fellowship as $fellow) {
        print "$fellow\n";
}
?>
```

String Comparison Functions

Name	Function
strcmp	Compare to
strstr, strchr	Find string/char within a string
strpos	Find numerical position of string
str_replace, substr_replace	Replace string

- Comparison can be:
  - Partial matches
  - Others
- Variations with non case sensitive functions
  - strcasecmp

String Comparison Functions Examples

```
<?php
// Provides: Hll Wrld f PHP
$vowels = array("a", "e", "i", "o", "u", "A", "E", "I",
"O", "U");
$onlyconsonants = str_replace($vowels, "", "Hello World of
PHP");

// Provides: You should eat pizza, beer, and ice cream
every day
$phrase  = "You should eat fruits, vegetables, and fiber
every day.";
$healthy = array("fruits", "vegetables", "fiber");
$yummy   = array("pizza", "beer", "ice cream");

$newphrase = str_replace($healthy, $yummy, $phrase);
?>
```

Multidimensional Arrays

- Creating multidimensional arrays with array( )

```
<?php
$meals = array('breakfast' => array('Walnut Bun','Coffee'),
               'lunch'     => array('Cashew Nuts', 'White
                                    Mushrooms'),
               'snack'     => array('Dried Mulberries','Salted
                                    Sesame Crab'));

$lunches = array( array('Chicken','Eggplant','Rice'),
                  array('Beef','Scallions','Noodles'),
                  array('Eggplant','Tofu'));

$flavors = array('Japanese' => array('hot' => 'wasabi',
                                     'salty' => 'soy sauce'),
                 'Chinese'  => array('hot' => 'mustard',
                                     'pepper-salty' => 'prickly
                                                       ash'));
?>
```

Multidimensional Arrays (cont.)

- Access elements in these arrays of arrays by using more sets of square brackets to identify elements.
- Each set of square brackets goes one level into the entire array.

```
//continue from the example in previous page
<?php
print $meals['lunch'][1];          // White Mushrooms
print $meals['snack'][0];          // Dried Mulberries
print $lunches[0][0];              // Chicken
print $lunches[2][1];              // Tofu
print $flavors['Japanese']['salty'];  // soy sauce
print $flavors['Chinese']['hot'];    // mustard
?>
```

Multidimensional Arrays (cont.)

- Iterating through a multidimensional array with foreach( )

```
<?php
$flavors = array('Japanese' => array('hot' => 'wasabi',
                                     'salty' => 'soy sauce'),
                 'Chinese'  => array('hot' => 'mustard', 'pepper-salty' =>
                                                          'prickly ash'));

// $culture is the key and $culture_flavors is the value (an array)
foreach ($flavors as $culture => $culture_flavors) {

    // $flavor is the key and $example is the value
    foreach ($culture_flavors as $flavor => $example) {
        print "A $culture $flavor flavor is $example.\n";
    }
}
?>
// output
// A Japanese hot flavor is wasabi.
// A Japanese salty flavor is soy sauce.
// A Chinese hot flavor is mustard.
// A Chinese pepper-salty flavor is prickly ash.
```

Multidimensional Arrays (cont.)

- Iterating through a multidimensional array with `for()`

```
<?php
$specials = array( array('Chestnut Bun', 'Walnut Bun', 'Peanut Bun'),
                   array('Chestnut Salad','Walnut Salad', 'Peanut Salad')
);

// $num_specials is 2: the number of elements in the first dimension of
$specials
for ($i = 0, $num_specials = count($specials); $i < $num_specials; $i++) {
    // $num_sub is 3: the number of elements in each sub-array
    for ($m = 0, $num_sub = count($specials[$i]); $m < $num_sub; $m++) {
        print "Element [$i][$m] is " . $specials[$i][$m] . "\n";
    }
}
?>
// output
// Element [0][0] is Chestnut Bun
// Element [0][1] is Walnut Bun
// Element [0][2] is Peanut Bun
// Element [1][0] is Chestnut Salad
// Element [1][1] is Walnut Salad
// Element [1][2] is Peanut Salad
```

User-defined Functions

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

```
<?php
function writeMsg() {
    echo "Hello world!";
}

writeMsg(); // call the function
?>
```

- Argument types and return types are not written
- A function with no return statements implicitly returns NULL

Default Argument Value

```
<?php
function setHeight($minheight = 50) {
    echo "The height is : $minheight <br>";
}

setHeight(350);
setHeight(); // will use the default value of 50
setHeight(135);
setHeight(80);
?>
```

- if no value is passed, the default will be used

Return Value

```
<?php
function sum($x, $y) {
    $z = $x + $y;
    return $z;
}

echo "5 + 10 = " . sum(5, 10) . "<br>";
echo "7 + 13 = " . sum(7, 13) . "<br>";
echo "2 + 4 = " . sum(2, 4);
?>
```

- To let a function return a value, use the return statement

Useful Online Resources

- http://php.net
- http://www.w3schools.com/PHP/default.asp
- https://www.codecademy.com/