October 18-21, 2005
San Francisco, CA

Zend PHP Certification Tutorial

Marco Tabini
php | architect

marcot@tabini.ca

www.phparch.com

October 18, 2005

Hyatt Regency San Francisco Airport Burlingame, CA

Welcome!

- **A few words about me**
- **A few words about what we'll be covering**
  - This is **not** a PHP tutorial!
  - I expect that you already have some PHP experience
  - Goals of this tutorial
  - Structure

A bit about the exam

- We'll talk about the exam proper at the end of the tutorial
- The exam covers only PHP 4 — *not* PHP 5
- If you are taking the exam here, it will be on paper, not on a computer
- The exam tests your knowledge of PHP, not your knowledge of programming

Part I - The PHP Language

- **What we'll cover in this section:**
  - PHP Tags
  - File inclusion
  - Data types & typecasting
  - Variables and constants
  - Operators
  - Conditionals
  - Iteration
  - Functions
  - Objects

Tags

- **Tags "drop" you out of HTML and into PHP mode**
- **PHP recognizes several types of tags:**
  - Short tags: <? ?>
  - Special tags: <?= ?>
  - Regular tags: <?php ?>
  - ASP tags: <% %>
  - HTML script tags: <script language="PHP"> </script>

File Inclusion

- **External files can be included in a script using either include() or require()**
- **Both are *constructs*, not *functions*:**
  - include ('myfile.php'); *or* include 'myfile.php';
- **They behave in exactly the same way, except for how they handle failure**
  - include generates a warning
  - require throws an error
  - Upon inclusion, the parser "drops off" of PHP mode and enters HTML mode again
- **Variants: include_once()/require_once()**
  - Prevent multiple inclusions from within the same script

Data Types

- PHP is not a typeless language
- It supports many different data types
- It is *loosely typed*
- The interpreter automatically "juggles" data types as most appropriate
- "Most appropriate" doesn't necessarily mean *always* appropriate

Data Types — Numeric/Boolean

- **PHP recognizes two types of numeric values:**
  - Integers
  - Floats
- **Boolean values are used for logic operations**
  - True / False
  - Easily converted to integers: non-zero / zero
- **Result type of operations depends on types of operands**
  - For example: int + int == int — int / float == float
    int / int == int or float
- **Numbers can be specified in a number of ways:**
  - Decimal (123), Hexadecimal (0x123) and Octal (0123)

Data Types — Strings

- **Strings are heterogeneous collections of single-byte characters**
  - They *don't* necessary have to be text
  - They *can* represent Unicode as well, but cannot be manipulated by the standard PHP functions
- **PHP supports three ways of declaring strings:**
  - Single quotes: 'test 1 2 3'
  - Double quotes: "test 1 2 3\n"
  - Heredoc syntax: <<<EOT test 1 2 3
    EOT;
- **Main differences:**
  - Support for variable substitution / escape sequences
  - *All* strings support newline characters

Data Types — Arrays

- **Arrays are ordered structures that map a key to a value**
- **Values can be of any type—including other arrays**
- **Keys can be either integer numeric or strings**
  - Keys are unique
  - Negative numbers *are* valid keys

Data Types — Resources / Null

- **Resources are special containers that identify external resources**
  - They can only be operated on directly as part of logical operations
  - They are usually passed to C-level functions to act on external entities
  - Examples: database connections, files, streams, etc.
- **NULL is a special value that indicates... no value!**
  - NULL converts to Boolean false and Integer zero

Data Types — Objects

- **Objects are containers of data *and* functions**
  - The individual data elements are normally called *properties*
  - The functions are called *methods*
  - Individual members (methods / properties) of an object are accessed using the -> operator
  - We'll cover objects in more depth later in this section

Typecasting

- PHP's ability to juggle among different data types is not *entirely* dependable
- There are circumstances in which you will want to control how and when individual variables are converted from one type to another
- This is called *Typecasting*

Typecasting — Integers

- **You can typecast any variable to an integer using the (int) operator:**
  - echo (int) “test 1 2 3”;
- **Floats are automatically truncated so that only their integer portion is maintained**
  - (int) 99.99 == 99
- **Booleans are cast to either one or zero:**
  - (int) TRUE == 1 — (int) FALSE == 0
- **Strings are converted to their integer equivalent:**
  - (int) “test 1 2 3” == 0 , (int) “123” == 123
  - (int) “123test” == 123 // String begins with integer
- **Null always evaluates to 0**

Typecasting — Booleans

- **Data is cast to Boolean using the (bool) operator:**
  - echo (bool) “1”;
- **Numeric values are always TRUE unless they evaluate to zero**
- **Strings are always TRUE unless they are empty**
  - (bool) “FALSE” == true
- **Null always evaluates to FALSE**

Typecasting — Strings

- **Data is typecast to a string using the (string) operator:**
  - echo (string) 123;
- **Numeric values are converted to their decimal string equivalent:**
  - (string) 123.1 == “123.1”;
- **Booleans evaluate to either “1” (TRUE) or an empty string (FALSE)**
- **NULL evaluates to an empty string**
- **Numeric strings are *not* the same as their integer or float counterparts!**

Typecasting — Arrays / Objects

- **Casting a non-array datum to an array causes a new array to be created with a single element whose key is zero:**
  - var_dump ((array) 10) == array (10);
- **Casting an object to an array whose elements correspond to the properties of the object**
  - Methods are discarded
- **Casting a scalar value to an object creates a new instance of stdClass with a single property called "scalar"**
  - Casting an array to an object create an instance of stdClass with properties equivalent to the array's elements

Identifiers / Variables / Constants

- **Identifiers are used to identify entities within a script**
  - Identifiers **must** start with a letter or underscore and can contain only letters, underscores and numbers
- **Variables**
  - Containers of data
  - Only one data type at any given time
  - Variable names are case-sensitive identifiers prefixed with a dollar sign ($my_var)
  - Variables can contain *references* to other variables
- **Constants**
  - Assigned value with declare(), cannot be modified
  - User-defined constants are not case-sensitive

Substitution / Variable variables

- **Variables can be substituted directly within a double-quoted or Heredoc string**
  - $a = 10;
    echo "\$a is: $a"; // Will output $a is: 10
- **Variables values can be used to access other variables (variable variables):**
  - $a = "b";
    $b = 10;
    echo $$a; // will output 10

Statements

- **Statements represent individual commands that the PHP interpreter executes**
  - Assignment: $a = 10;
  - Construct: echo $a;
  - Function call: exec ($a);
- **Statements must be terminated by a semicolon**
  - Exception: the last statement before the end of a PHP block

Operations

- **PHP supports several types of operations:**
  - Assignment
  - Arithmetic
  - Bitwise
  - String
  - Comparison
  - Error control
  - Logical

Operations — Assignment

- **The assignment operator '=' makes it possible to assign a value to a variable**
  - $a = 10;
- **The left-hand operand must be a variable**
  - Take advantage of this to prevent mistakes by "reversing" logical operations (as we'll see later)
  - 10 = $a; // Will output error

Operations — Arithmetic

- **These operators act on numbers and include the four basic operations:**
  - Addition: $a + $b
  - Subtraction: $a - $b
  - Multiplication: $a * $b
  - Division: $a / $b
    - Remember that dividing by zero is *illegal*
- **They also include the modulus operator**
  - Determines the remainder of the integer division between two numbers: 10 % 4 = 2
  - Unlike proper modulus, PHP allows a negative right-hand operand
    - 10 % -4 = 2

Operations — Bitwise

- **Bitwise operations manipulate numeric values at the bit level**
  - AND (&) — set bit if it is set in both operands
    - 1 & 0 == 0
  - OR (|) — set bit if is is set in either operand
    - 1 | 0 == 1
  - XOR (^) — set bit if it is set in either, but not both
    - 1 ^ 1 == 0
  - NOT — invert bits
    - ~0 == -1
  - Shift left/right (<</>>) - shift bits left or right
    - 1 << 2 ==4 == 8 << 1
    - Excellent shortcuts for integer multiplications by powers of two

Operators — Combined

- **Numeric and bitwise operators can be combined with an assignment:**
  - $a += 10 is equivalent to $a = $a + 10;
- **This does not apply to the NOT operator, since it's unary**

Operators — Error Control

- **PHP support several different levels of errors**
- **Error reporting can be tweaked either through PHP.INI settings or by calling error_reporting().**
- **Remember that the exam assumes the default "recommended" INI file**
  - Warning and Notices are **not** reported!
- **Error reporting can be controlled on a statement-by-statement basis using the @ operator:**
  - @fopen ($fileName, "r");
  - This only works if the underlying functionality uses PHP's facilities to report its errors

Operators — Inc/Dec and String

- **Incrementing and decrementing operators are special unary operators that increment or decrement a numeric variable:**
  - Postfix: $a++
  - Prefix: ++$a
  - You cannot perform two unary operations on the same variable at the same time— ++$a-- will throw an error
- **The only string operation is the concatentaion (.), which "glues" together two strings into a third one**
  - "a" . 'b' == 'ab'

Operators — Comparison / Logical

- **Comparison operators are used to compare values:**
  - Equivalence: == !=
    - Equivalence operators do **not** require either of their operands to be a variable
  - Identity: === !==
  - Relation: <, <=, >=, >
- **Logical operators are used to manipulate Boolean values:**
  - AND (&&) — TRUE if both operands are TRUE
  - OR (||) — TRUE if either operand is TRUE
  - XOR (xor) — TRUE if either operand is TRUE, but not both
  - NOT (!) — Reverses expression

Operator Precedence

- **The precedence of most operators follows rules we are used to—but not *all* of them**
  - Example: “test ” . 1 + 10 . “ 123” == “1 123”
- **There are two variants of logical operators**
  - The “letter” operators AND, OR differ from their “symbol” equivalents &&, | | in the fact that they have lower precedence

Conditionals — if-then-else

- **Conditionals are used to direct the execution flow of a script**
  - if (condition) {

    ... statements ...

    } else {

    ... statements ...

    }
- **Alternative short form:**
  - $a = (cond) ? yesvalue : novalue;

Conditionals — case/switch

- **Case/switch statements allow you to verify a single expression against multiple expressions:**
  - switch (expr) {
    case expr1 :
    ... statements ...
    break;

    case expr2:
    ... statements ...
    break;

    default:
    ... statements ...
    break;
    }

Iterators — While

- **While loops are the simplest form of iterator; they allow you to repeat a set of statements while a condition evaluates to TRUE:**
  - while (expr) {

    ... statements ...

    }

Iterators — Do...while

- **Do...while loops are equivalent to while loops, but the condition is evaluated *at the end* of the loop, instead of the beginning:**
  - do {

    ... statements ...

    } while (expr);
  - This means that the statement block is executed at least once

Iterators — For and Foreach

- **While and do...while are the only *indispensible* iterators in any language.**
- **For convenience, PHP includes for loops:**
  - for (initial; condition; incremental) {
    ... statements ...
    }
- **Foreach loops can be used to iterate through an aggregate value:**
  - foreach ($array as $k => $v) {
    ... statements ...
    }
  - Important: $k and $v are assigned by value!
  - Works on objects, too!

Iterators: continuing/breaking

- **Loops can be continued using the continue construct:**
  - while ($a == 1) { if ($b == 2) continue; }
- **Loops can be interrupted using the break construct:**
  - while ($a == 1) { if ($b == 2) break; }
- **Multiple nested loops can be continued/broken at once:**
  - continue 2;
  - Remember the semicolon at the end of the break or continue statement!

Functions

- **Functions allow for code isolation and reuse**
  - function myfunc (&$arg1, $arg2 = 10)
    {
      global $variable;

      ... statements ...
    }

    echo myfunc (10);
- **Pay attention to variable scope!**
- **Functions can support variable parameters:**
  - func_num_args();
  - fung_get_arg();

OOP: Classes and Objects

- **Classes define the structure of objects:**
  - class myClass {
    var $myVar;

    function myClass() { // constructor
      $this->myVar = 10;
    }
- **Objects represent individual instances of a class:**
  - $a = new myClass;
    $a->myVar = 11;
- **Objects support dynamic methods and properties:**
  - $obj->$var();

OOP: Classes as Namespaces

- **PHP does not support namespaces (this is true also of PHP 5), but classes can simulate their behaviour:**
  - class class encode {
    function base64($str)
    {
    return base64_encode($str);
    }
    }

    echo encode::base64("my string");

OOP: Objects and References

- **In PHP 4, objects receive no special treatment: they are essentially arrays with embedded functions**
  - This means that references to objects must be handled with care.
- **Passing/assigning an object is normally done by value, not by reference, even when using new**

OOP: Objects and References

- **The $this special variable *cannot* be passed by reference, even if you use the & operator**
  - However, you can embed $this in a global array and circumvent this problem (albeit in a horrible way):
    - ```
      class obj {
          var $prop;
          function obj($arg)
          {
              global $obji; // import variable into local scope
              $obji[] = $this; // get a copy of current class
              $this->prop = $arg;
          }
      }
      $obj = new obj(123);
      var_dump($obj->prop != $obji[0]->prop); // FALSE
      ```
      ```

OOP: Inheritance

- **Inheritance makes it possible to create classes ("subclasses") that are based on other classes ("superclasses"):**
  - ```
    class base {
        function base()
        {
        }
    }

    class main extends base {
        function main()
        {
            parent::base();
        }
    }
    ```
    ```

OOP: Object Serialization

- **Serialization is the process of reducing an aggregate (array or object) to a scalar (string)**
- **Serialization is a mostly automatic process, but for objects it is possible to exercise a certain amount of control:**
  - __sleep()
  - __wakeup()
  - Useful for dynamically-generated properties, such as database connections and file descriptors
  - Classes must be declared before their instances are unserialized

Q&A Time

- What is the difference between print and echo?
- Under what circumstance is it impossible to assign a default value to a parameter while declaring a function?
- How does the identity operator === compare two values?

Answers

- **What is the difference between print and echo?**

- **echo is a construct**
- **print is a function**

Answers

- **Under what circumstance is it impossible to assign a default value to a parameter while declaring a function?**
- **Always, as long as the parameter is not being passed by reference**

Answers

- How does the identity operator === compare two values?
- It first compares the type, then the value

Part II — Strings and Arrays

- **What we'll cover in this section:**
  - Comparisons
  - Basic search and replace
  - Regular Expressions
  - String functions and formatting
  - Accessing arrays
  - Single- and multidimensional arrays
  - Array iteration
  - Array sorting
  - Array functions and manipulation
  - Serialization

String Comparison

- **String comparison is mostly trivial, but can sometimes be tricky**
  - The equivalence operator should be used when you know that you are comparing two strings—or when you don't care about cases like this:
    - "123test" == 123 == TRUE!
  - The identity operator should be otherwise used every time you know that you *want* to compare two strings without letting PHP juggle types
- **PHP also provides function-based comparison:**
  - strcmp()
  - strcasecmp()
  - strncmp() and strncasecmp()

Basic String Searching

- **strstr() (aliased into strchr()) determines whether a substring exists within a string:**
  - strstr ("PHP is a language", "PHP") == true
  - stristr() provides a case-insensitive search
- **strpos() will return the location of a substring inside a string, optionally starting from a given position:**
  - strpos ($haystack, $needle, $pos)
  - Beware of zero return values!
  - There is no stripos() in PHP 4!
- **Reverse search is done with strrchr() / strrpos()**

Counting Strings

- **The length of a string is determined with strlen()**
  - Do **not** use count()!
- **You can count words inside a string using str_word_count():**
  - str_word_count ($str, $n);
  - $n == 1 — Returns array with words in order
  - $n == 2 — Returns array with words and positions
- **substr_count() can be used to count the number of occurrences of a given substring:**
  - substr_count (“phpphpPHP”, “php”) == 2

Formatting Strings

- **Most of the time, strings can be formatted using a combination of concatenations**
- **In some cases, however, it is necessary to use special functions of the *printf()* family**
  - printf() — outputs formatted strings to STDOUT
    - printf ("%d", 10);
  - sprintf() — returns the formatted string
    - $a = sprintf ("%d", 10);
  - fprintf() — outputs formatted strings to a file descriptor
    - fprintf ($f, "%d", 10);
  - vprintf(), vsprintf() — take input from array
    - vprintf ("%d", array (10));
    - $a = vsprintf ("%d", array (10));

Formatting Strings

- % - a literal percent character.
- b – integer presented as a binary number
- c – integer (ASCII value)
- d – integer (signed decimal number)
- e – number in scientific notation (Ex. 1.2e+2)
- u – integer (unsigned decimal number)
- f – float as a floating-point number.
- o – integer (octal number).
- s – string
- x – hexadecimal number (lowercase letters).
- X – hexadecimal number (uppercase letters).

Accessing Strings as Arrays

- **You can access individual characters of a string as if it were an array**
  - $s = “12345”;
    echo $s[1]; // Outputs 2
    echo $s{1}; // Outputs 2
  - This works for both reading and writing
  - Remember that you **cannot** use count() to determine the number of characters in a string!

Extracting and Replacing

- **Substrings can be extracted using the substr() function:**
  - echo substr (“Marco”, 2, 1); // Outputs r
  - echo substr (“Marco”, -1); // Outputs o
  - echo substr (“Marco”, 1, -1); // Outputs arc
- **Substrings can be replaced using substr_replace ():**
  - substr_replace (‘Marco’, ‘acr’, 1, -1) == “Macro”
- **The sscanf() function can be used to extract tokens formatted à la printf() from a string:**
  - sscanf(“ftp://127.0.0.1”, "%3c://%d.%d.%d.%d:%d");
  - Returns array (‘ftp’, ‘127’, ‘0’, ‘0’, ‘1’);

Multiple Replacements

- **str_replace() replaces instances of a substring with another:**
  - str_replace (“.net”, “arch”, “php.net”) == “phparch”
- **You can perform multiple replacements by passing arrays to str_replace():**
  - str_replace(array('apples', 'applesauce', 'apple'), array('oranges', 'orange-juice', 'cookie'), “apple apples applesauce”)
  - Returns “cookie oranges orangesauce”

PCRE — Perl Regular Expressions

- **Perl Regular Expressions (PCRE) make it possible to search (and replace) variable patterns inside a string**
- **PCRE is usually fast and simple to understand, but it can also be complicated or slow (or both)**
- **Regular expressions are matched using the preg_match() function:**
  - preg_match ($pcre, $search, &$results)
  - preg_match_all ($pcre, $search, &$results)
- **Search-and-replace is performed using preg_replace():**
  - preg_replace ($pcre, $replace, $search)

PCRE — Meta Characters

- **Meta characters are used inside a regex to represents a series of characters:**
  - \d — digits 0–9
  - \D — not a digit
  - \w — alphanumeric character or underscor
  - \W — opposite of \w
  - \s — any whitespace (space, tab, newline)
  - \S — any non-whitespace character
  - . — any character except for a newline
- **Meta characters only match one character at a time (unless an operator is used to change this behaviour)**

PCRE — Operators / Expressions

- **PCRE operators indicate repetition:**
  - ? — 0 or 1 time
  - * — 0 or more times
  - + — 1 or more times
  - {,n} — at more n times
  - {m,} — m or more times
  - {m,n} — at least m and no more than n times
- **Parentheses are used to group patterns**
  - (abc)+ — means "abc" one more times
- **Square brackets indicate character classes**
  - [a-z] means "any character between a and z
  - The caret negates a class: [^a-z] is the opposite of the expression above

PCRE — An example

- **Here's an example of a PCRE:**
  - $string = '123 abc';
    preg_match ('/\d+\s\[a-z]+/', $string);

    preg_match ('/\w\s\s/', $string);

    preg_match ('\d{3}\s[a-z]{3}'/, $string);

PCRE — Another Example

- **Here's an example of how to retrieve data from a regex:**
  - $email = 'marcot@tabini.ca";
    preg_match ('/(\w+)@(\w+)\.(\w+)/');
  - Will return array ('marcot@tabini.ca', 'marcot', 'tabini', 'ca')

String Splitting and Tokenization

- **The explode() function can be used to break up a string into an array using a common delimiter:**
  - explode ('.', 'www.phparch.com');
  - Will return array ('www', 'phparch', 'com');
- **The preg_split() function does the same thing, but using a regex instead of a fixed delimiter:**
  - explode ('[@.]', 'marcot@tabini'ca');
  - Will return array ('marcot', 'tabini', 'ca');

Word Wrapping

- **The wordwrap() function can be used to break a string using a specific delimiter at a given length**
  - wordwrap ($string, $length, $delimiter, $break);
- **If the $break parameter evaluates to TRUE, the break occurs at the specified position, even if it occurs in the middle of a word**

Arrays

- **Arrays are created in a number of ways:**
  - Explicitly by calling the array() function
    - array (1, 2, 3, 4);
    - array (1 => 1, 2, 3, 5 => “test”);
    - array (“2” => 10, “a” => 100, 30);
  - By initializing a variable using the array operator:
    - $x[] = 10;
    - $x[-1] = 10;
    - $x[‘a’] = 10;
- **The count() function is used to determine the number of elements in an array**
  - Executing count() against any other data type (including objects), it will return 1 (or 0 for NULL)

Array Contents

- **Array can contain *any* data type supported by PHP, including objects and other arrays**
- **Data can be accessed using the array operator**
  - $x = $array[10];
- **Multiple elements can be extracted using the list function:**
  - $array = (1, 2, 3);
    list ($v1, $v2, $v3) = $array

Array Iteration

- **It's possible to iterate through arrays in a number of ways. Typically:**
- **for ($i = 0; $i < count ($array); $i++) // WRONG!**
  - $cnt = count ($array)
    for ($i = 0; $i < $cnt; $i++)
  - Storing the **invariant** array count in a separate variable improves performance
- **foreach ($array as $k => $v)**
  - $k and $v are assigned by value—therefore, changing them won't affect the values in the array
  - However, you can change the array directly using $k:
  - $array[$k] = $newValue;

Array Iteration

- **You can also iterate through an array using the internal array pointer:**
  - $a = array(1,2,3);

```
while (list($k, $v) = each($a)) {
  echo "{$k} => {$v} ";
  if ($k % 2) { // add entry if key is odd
    $a[] = $k + $v;
  }
} // 0 => 1 1 => 2 2 => 3 3 => 3 4 => 6
```

  - With this approach, operations take place directly on the array
- **Finally, you can use array_callback() to iterate through an array using a user-supplied function**

Array Keys and Values

- **You can check if an element exists in one of two ways:**
  - array_key_exists ($array, $key); // Better, but slower
  - isset ($array[$key]); // Faster, but has pitfalls
    - $a[1] = null;
      echo isset ($a[1]);
- **You can also check whether a value exists:**
  - in_array ($value, $array)
- **You can extract all the keys and values from an array using specialized functions:**
  - array_keys ($array);
  - array_value ($array);

Sorting Arrays

- **The sort() and rsort() functions sort an array in-place**
  - sort ($array); — rsort ($array)
  - Key association is lost—you can use asort() and arsort() to maintain it
- **A more "natural" sorting can also be performed:**
  - natsort ($array);
  - natcasesort ($array);
- **Sorting by key is also a possibility:**
  - ksort();
  - krsort();

Array Functions

- **Changing key case:**
  - array_change_key_case ($a, CASE_LOWER)
  - array_change_key_case ($a, CASE_UPPER)
- **Randomizing the contents of an array:**
  - shuffle($array)
- **Extracting a random value:**
  - array_rand ($array, $qty);

Merge, Diff and Sum

- **Merging arrays:**
  - array_merge ($a, $b[, ...]);
  - Later values with the same key overwrite earlier ones
- **Diff'ing arrays:**
  - array_diff ($a, $b[, ...]);
  - Returns keys that are *not* common to all the arrays
  - Key association is lost—you can use array_diff_assoc() to maintain it
- **Intersecting:**
  - array_intersect ($a, $b[, ...]);
- **Calculating arithmetic sum:**
  - array_sum ($array);

Unique Array Values

- **The array_unique() function retrieves all the unique array values**
  - array_unique ($array)
  - Requires traversal of entire array and therefore hampers performance

Arrays as stacks or queue

- **The array_push() function pushes a new value at the end of an array**
  - array_push ($array, $value)
  - Essentially equivalent to $array[] = $value;
- **The array_pop() retrieves the last value from an array:**
  - $x = array_pop ($array);
- **This allows you to use arrays as if they were stacks (LIFO)**
- **You can also pull a value from the top of the array, thus implementing a queue (FIFO)**
  - $x = array_shift ($array)

Serializing Arrays

- **Like with objects, you can serialize arrays so that they can be conveniently stored outside your script:**
  - $s = serialize ($array);
  - $array = unserialize ($s);
  - Unserialization *will* preserve references inside an array, sometimes with odd results

Q&A Time

- Given a comma-separated list of values in a string, which function can create an array of each individual value with a single call?
- The _____ function can be used to ensure that a string always reaches a specific minimum length.
- Which function would you use to rearrange the contents of the array (‘a’, ‘b’, ‘c’, ‘d’) so that they are reversed?

Answers

- Given a comma-separated list of values in a string, which function can create an array of each individual value with a single call?

- explode()
- preg_split() would have also been acceptable

Answers

- The _____ function can be used to ensure that a string always reaches a specific minimum length.

- str_pad()

Answers

- **Which function would you use to rearrange the contents of the array ('a', 'b', 'c', 'd') so that they are reversed?**

- **rsort()**
- **array_reverse()**

PART III — User Input / Time & Dates

- **What we'll cover in this section:**
  - HTML form management
  - File uploads
  - Cookies
  - Magic Quotes
  - Sessions
  - Times and dates in PHP
  - Formatting date values
  - Locale-dependent date formatting
  - Date validation

HTML Form Management

- **HTML forms are submitted by the browser using either GET or POST**
  - GET transaction data is sent as part of the query string
  - POST data is sent as part of the HTTP transaction itself
  - POST is often considered "safer" than GET—WRONG!
- **POST data is made available as part of the $_POST superglobal array**
- **GET data is made available as part of the $_GET superglobal array**
  - Both are "superglobal"—in-context everywhere in your scripts
  - If duplicates are present, only the ones sent last end up in the appropriate superglobal

HTML Form Management

- **Element arrays can also be sending by postfixing the element names with []**
  - These are transformed into arrays by PHP
  - The brackets are discarded
  - A very common (and pernicious) type of security attack
- **You can also specify your own keys by placing them inside the brackets:**
  - <input type="hidden" name="a[ts]" value="1">
  - Will result in $a['ts'] = 1 being inserted in the appropriate superglobal

Uploading Files

- **Files are uploaded through a special type of HTML form:**
  - <form enctype="multipart/form-data" action="/upload.php" method="post">
    <input type="my_file" type="file" />
    <input type="hidden" name="MAX_FILE_SIZE" value="100000" />
    </form>
- **An arbitrary number of files can be uploaded at the same time**

Uploading Files

- **Once uploaded, file information is available through the $_FILES superglobal array**
  - [my_file] => Array
    (
    [name] => php.gif
    [type] => image/gif
    [tmp_name] => /tmp/phpMJLN2g
    [error] => 0
    [size] => 4644
)
- **Uploaded file can be moved using move_uploaded_file()**
  - You can also determine whether a file has been uploaded using is_uploaded_file()

Uploading Files

- **File uploads are controlled by several PHP.INI settings:**
  - file_uploads — whether or not uploads are enabled
  - upload_tmp_dir — where temporary uploaded files are stored
  - upload_max_filesize — the maximum size of each uploaded file
  - post_max_size — the maximum size of a POST transaction
  - max_input_time — the maximum time allowed to process a form

Cookies

- **Cookies are small text strings that are stored client-side**
- **Cookies are sent to the client as part of the HTTP response, and back as part of the HTTP headers**
- **Cookies are notoriously unreliable:**
  - Some browsers are set not to accept them
  - Some users do not accept them
  - Incorrect date/time configuration on the client's end can lead to cookies expiring *before* they are set

Cookies

- **To set a cookie:**
  - setcookie ($name, $value, $expires, $path, $domain);
  - setcookie ($name, $value); // sets a session cookie
- **Cookies are then available in the $_COOKIE superglobal array:**
  - $_COOKIE['mycookie']
  - $_COOKIE is populated at the beginning of the script. Therefore, it does **not** contain cookies you set during the script itself (unless you update it manually)
- **You cannot "delete" a cookie**
  - You can set it to Null or an empty string
    - Remember **not** to use isset()!
  - You can expire it explicitly

$_REQUEST

- **$_REQUEST is a superglobal populated from other superglobals**
  - You have no control over *how* data ends up in it
  - The variables_order PHP.INI setting controls how data is loaded into it, usually Get -> Post -> Cookie
- **Generally speaking, you're better off *not* using it, as it is a virtual security black hole.**

Magic Quotes

- By default, PHP will escape any “special” characters found inside the user’s input
- You should not rely on this setting being on (as most sysadmins turn it off anyway)
- You also (and most definitely) should not rely on it performing proper input filtering for you
- In fact, supply your own escaping and “undo” magic quotes if they are enabled!

Sessions

- Sessions are mechanisms that make it possible to create a per-visitor storage mechanism on your site
- Sessions we born—and remain—a hack, so you can only depend on them up to a certain point
- On the client side, sessions are just unique IDs passed back and forth between client and server
- On the server side, they can contain arbitrary informaiton

Sessions

- **In order to write to a session, you must explicitly start it**
  - session_start()
  - This is not necessary if session.auto_start is on in your PHP.INI fil
- **You can then write directly into the $_SESSION array, and the elements you create will be transparently saved into the session storage mechanism**
  - $_SESSION[‘test’] = $myValue

Sessions

- By default, session data is stored in files; however, you can specify a number of built-in filters
- You can also define your own session handlers in “userland”

Date Manipulation in PHP

- **For the most part, PHP handles dates in the UNIX timestamp format**
  - Timestamps indicate the number of seconds from the UNIX "epoch", January 1st, 1970
  - Not all platforms support negative timestamps (e.g.: Windows *prior* to PHP 5.1)
- **Timestamps are very handy because they are just large intergers**
  - This makes it easy to manipulate them, but not necessarily to represent them
  - They are also handy for time calculations
  - For more precision, you can use microtime()

Date Manipulation in PHP

- **Another way of representing dates is through *date arrays* using getdate()**
  - A date array contains separate elements for each component of a date
  - [seconds] => 15 // 0 - 59
    [minutes] => 15 // 0 - 59
    [hours] => 9 // 0 - 23
    [mday] => 4 // 1 - 31
    [wday] => 3 // 0 - 6
    [mon] => 8 // 1 - 12
    [year] => 2004 // 1970 - 2032+
    [yday] => 216 // 0 - 366
    [weekday] => Wednesday // Monday - Sunday
    [month] => August // January - December
    [0] => 1091625315 // UNIX time stamp

Time and Local Time

- **The time() function returns the timestamp for the current time**
  - time() (no parameters needed)
- **Localtime performs similarly, but returns an array**
  - [0] => 59 // seconds 0 - 59
    [1] => 19 // minutes 0 - 59
    [2] => 9 // hour 0 - 23
    [3] => 4 // day of month 1 - 31
    [4] => 7 // month of the year, starting with 0 for January
    [5] => 104 // Years since 1900
    [6] => 3 // Day of the week, starting with 0 for Sunday
    [7] => 216 // Day of the year
    [8] => 1 // Is daylight savings time in effect

More Local Time

- **Localtime() can also return an associative array:**
  - var_dump (localtime(time, 1));
  - Outputs:
    - [tm_sec] => 1 // seconds 0 - 59
      [tm_min] => 23 // minutes 0 - 59
      [tm_hour] => 9 // hour 0 - 23
      [tm_mday] => 4 // day of month 1 - 31
      [tm_mon] => 6 // month of the year, 0 for January
      [tm_year] => 104 // Years since 1900
      [tm_wday] => 0 // Day of the week, 0 for Sunday
      [tm_yday] => 185 // Day of the year
      [tm_isdst] => 1 // Is daylight savings time in effect

Formatting Dates

- **Timestamps are great for calculations, but not for human redability**
- **The date() function can be used to format a date according to an arbitrary set of rules:**
  - date ("Y-m-d H:i:s\n");
  - date ('\d\a\t\e: Y-m-d');
- **strftime() provides a printf-like, locale-dependent formatting mechanism for date/time values:**
  - strftime ("%A", time()); // Prints weekday
  - You need to use setlocale (LC_TIME, $timezone) in order to set the timezone to a particular value

Creating Dates

- **Dates can be created using mktime():**
  - mktime (hour, min, sec, mon, day, year, daylight)
- **Several date-related functions have GMT-equivalents:**
  - gmmktime()
  - gmdate()
  - gmstrftime()
- **It is also possible to change the timezone—just change the TZ environment variable:**
  - putenv ("TZ=Canada/Toronto");
  - This will be equivalent to EST or EDT

Interpreting Date Input

- **It is also possible to create a timestamp from a formatted string date using strtotime():**
  - strotime("now");
  - strtotime("+1 week");
  - strtotime("November 28, 2005");
  - strtotime("Next Monday");
- **You can also check whether a date is valid by using the checkdate() function:**
  - checkdate (month, date, year)
  - Automatically accounts for leap years
  - **Not** foolproof—incapable for example, to account for the Gregorian gap

Q&A Time

- How would you make a cookie expire in exactly one hour (assuming that the client machine on which the browser is set to the correct time and time zone—and that it resides in a time zone different from your server's)?
- What is the simplest way of transforming the output of microtime() into a single numeric value?
- If no expiration time is explicitly set for a cookie, what happens to it?

Answers

- How would you make a cookie expire in exactly one hour (assuming that the client machine on which the browser is set to the correct time and time zone—and that it resides in a time zone different from your server's)?

- Pass time() + 3600 as the expiry

Answers

- **What is the simplest way of transforming the output of microtime() into a single numeric value?**
- **array_sum (explode (‘ ‘, microtime()));**

Answers

- If no expiration time is explicitly set for a cookie, what happens to it?
- It expires at the end of the browser's session

PART IV: Files and E-mail

- **What we'll cover in this section:**
  - Opening and closing files
  - Reading from and writing to files
  - Getting information about a file
  - Copying, renaming, deleting files
  - File permissions
  - File locks
  - Sending e-mail
  - MIME
  - HTML E-mails
  - Multipart E-mails

Files — Opening and Closing

- **Files are open using the fopen() function:**
  - fopen ($filename, $mode)
  - returns a file resource (not a pointer!)
- **The $mode parameter indicates *how* the file should be open:**
  - r — read only
  - r+ — read/write
  - w — write only and create the file
  - w+ — read/write and create the file
  - a — write only and position at end of file
  - a+ — read/write and position at end of tile
  - x — write only, fail if file already exists

Files — Opening and Closing

- **If your PHP has been compiled with URL wrappers support, fopen() works both on local and "remote" files via any of the supported protocols:**
  - fopen ("http://www.phparch.com", "r");
- **Files can be closed using fclose()**
  - This is not *necessary*, because PHP closes all open handles at the end of script
  - However, it's a good idea in some cases

Files — Reading & Writing

- **Data is read from a file through a number of functions. The most common one is fread():**
  - $data = fread ($file, $qty);
  - Returns the maximum data available, up to $qty bytes
- **The fgets() function reads data one line at a time:**
  - $data = fgets ($file, $maxLen);
  - Returns data up to (and including) the next newline character or $maxLen - 1;
  - May or may not work depending on how the file has been encoded
    - auto_detect_line_endings PHP.INI setting

Files — Reading and Writing

- **Writing works in a similar way:**
  - fwrite ($file, $data)
  - Writes as much of $data as possible, returns amount written
- **You can also use fputs(), which is effectively an alias for fwrite()**

Files — File Position

- **The file position is updated as your read from or write to a file**
  - ftell ($file) — Returns the current offset (in bytes) from the beginning of the file
- **You can manually alter the current position using fseek():**
  - fseek ($file, $position, $from)
  - $from can be one of three constants:
    - SEEK_SET (beginning of file)
    - SEEK_CUR (current offset)
    - SEEK_END (end of file — $from should be < 0)

Files — File Information

- **The fstat() function returns several pieces of information about a file:**
  - var_dump (fstat ($file))
    - [dev] => 5633 // device
      [ino] => 1059816 // inode
      [mode] => 33188 // permissions
      [nlink] => 1 // number of hard links
      [uid] => 1000 // user id of owner
      [gid] => 102 // group id of owner
      [rdev] => -1 // device type
      [size] => 106 // size of file
      [atime] => 1092665414 // time of last access
      [mtime] => 1092665412 // time of last modification
      [ctime] => 1092665412 // time of last change
      [blksize] => -1 // blocksize for filesystem I/O
      [blocks] => -1 // number of blocks allocated

Files — File Information

- **The stat() function is a version of fstat() that does not require you to open the file**
  - var_dump (stat ($fileName))
- **Several functions provide only *portions* of the info returned by stat() and fstat()**
  - file_exists ($fileName)
  - fileatime ($fileName) — Last access time
  - fileowner ($fileName)
  - filegroup ($fileName)
- **The results of these functions are cached**
  - This can lead to confusing results if you make changes to a file in the same after you've run one of these convenience functions

Files — File Information

- **File permissions can be determined using either the bitmask from fstat() or some more convenience functions**
  - is_readable ($fileName);
  - is_writable ($fileName);
  - is_executable ($fileName);
  - is_uploaded_file ($fileName);
- **They can also be changed:**
  - chmod ($fileName, 0777);
  - Note use of octal number
- **The filesize() function returns the size of a file**
  - echo filesize ($fileName)

Copying, Renaming & Deleting

- **Files can be copied using the copy() function:**
  - copy ($sourcePath, $destPath)
- **Renaming is done through rename():**
  - rename ($sourcePath, $destPath);
  - Guaranteed to be atomic across the same partition
- **Files are deleted using unlink():**
  - unlink ($fileName);
  - **NOT** delete()!
- **Files can also be "touched":**
  - touch ($fileName);
- **All these functions report success/failure via a Boolean value**

Directories

- **Directories cannot be removed using unlink:**
  - $success = rmdir ($dirName);
  - The directory must be empty
  - This means that you must write your own code to empty the directory and any subdirectories

File Locking

- **File locking ensures ordered access to a file**
- **PHP's locking module is collaborative**
  - *Every* application that accesses the file must use it
- **Locks can be shared or exclusive**
  - $lock = ($file, $lockType, &$wouldBlock);
  - $lockType: LOCK_SH, LOCK_EX
  - To release a lock: LOCK_UN
  - To prevent blocking, OR with LOCK_NB
- **Several limitations:**
  - Doesn't work on most networked filesystems, or on FAT (Win98)
  - Sometimes implemented per-process

More File Fun

- **Some useful file functions**
- **file():**
  - Reads an entire file in memory, splits it along newlines
- **readfile():**
  - Reads an entire file, outputs it
- **fpassthru():**
  - Same as readfile(), but works on file pointer and supports partial output
- **file_get_contents():**
  - Reads entire file in memory
  - Remember that file_put_contents() is a PHP 5-only function!

PHP and E-mail

- **PHP supports sending of e-mail through the mail() function**
  - Contrary to popular belief, it's *not* always available
  - Relies on sendmail in UNIX, implements its own wrappers in Windows and Netware
  - Built-in wrappers do not support authentication
  - The from address is set automatically under Linux (php_user@serverdomain), must be set in PHP.ini under Windows

E-mail — The mail() Function

- **The mail() function accepts five parameters:**
  - mail ($to, $subject, $body, $headers, $extra)
- **mail() provides a *raw* interface to sending mail**
  - No support for attachments
  - No support for MIME
  - No support for HTML mail
- **Extra headers can be set, including overriding the default From:**
  - On UNIX machines, this may require setting -f in $extra
  - This may not work if PHP user is not “trusted” by sendmail

E-mail — MIME

- **E-mail only supports 7-bit ASCII**
  - Good for anglophones, not so good for the rest of the world
  - MIME provides support for sending arbitrary data over e-mail
  - MIME is supported by most MUAs, although often the target of spam filters
- **MIME headers also define the type of data that is being sent as part of an e-mail:**
  - For example, HTML:
    - "MIME-Version: 1.0\r\n" .
      "Content-Type: text/html; charset=\"iso-8859-1\"\r\n" .
      "Content-Transfer-Encoding: 7bit\r\n"

E-mail — MIME and Multipart

- **Multipart e-mails make it possible to send an e-mail that contains more than one "part"**
  - "MIME-Version: 1.0\r\n" .
    "Content-Type: multipart/alternative;\r\n" .
    " boundary=\"{$boundary}\"\r\n";
  - Examples:
    - HTML and Text bodies (plain-text should go first)
    - Attachments
- **Most clients support multipart—but for those who don't, you always provide a plain-text message at the beginning**
  - "If you are reading this, your client is too old!"

E-mail — MIME and Multipart

- **The different parts are separated by a unique boundary**
  - $message .= "--" . $boundary . "\r\n" .
    "Content-Type: text/plain; charset=us-ascii\r\n" .
    “Content-Transfer-Encoding: 7bit\r\n\r\n" .
    "Plain text" .
    "\r\n\r\n--" . $boundary . "--\r\n";
  - Note the two dashes before each boundary, and *after* the last boundary
- **Binary attachments must be encoded:**
  - "Content-Transfer-Encoding: base64\r\n" .
    ‘Content-disposition: attachment; file="l.gif"\r\n\r\n"
  - base64_encode ($file);

E-mail — Getting a handle

- **It's impossible to know whether an e-mail was successfully sent**
  - mail() only returns a success/failure Boolean for **its** end of the deal
  - E-mail can get lost at pretty much *any* point in the tranmission process
  - The mail protocol does not have a thoroughly-respected feedback mechanism

Q&A Time

- Which function(s) retrieve the entire contents of a file in such a way that it can be used as part of an expression?
- What does the built-in delete function do?
- Which MIME content type would be used to send an e-mail that contains HTML, rich text, and plain text versions of the same message so that the e-mail client will choose the most appropriate version?

Answers

- **Which function(s) retrieve the entire contents of a file in such a way that it can be used as part of an expression?**

- **file_get_contents()**
- **file()**

Answers

- What does the built-in delete function do?

- It doesn't exist!
- Use unlink() instead

Answers

- **Which MIME content type would be used to send an e-mail that contains HTML, rich text, and plain text versions of the same message so that the e-mail client will choose the most appropriate version?**

- **multipart/alternative**
  - segment which contains sub-segments representing multiple versions of the same content

PART V: Databases and Networks

- **What we'll cover in this section:**
  - Databasics
  - Indices and keys
  - Table manipulation
  - Joins
  - Aggregates
  - Transactions
  - File wrappers
  - Streams

Databasics

- **The exam covers databases at an abstract level**
  - No specific implementation
  - SQL-92 standards only
- **Only the basics of database design and programming are actually required**
  - Table creation/population/manipulation
  - Data extraction
  - Reference integrity
  - Joins / Grouping / Aggregates

Databasics

- **Relational databases**
  - Called because the relationship among different entities is its foundation
- **Schemas/databases**
- **Tables**
- **Rows**
  - Data types
    - Int
    - Float
    - Char/varchar
    - BlOBs

Indices

- **Indices organize data**
  - Useful to enforce integrity
  - Essential to performance
- **Indices can be created on one or more columns**
  - More rows == bigger index
  - Columns that are part of indices are called **keys**
- **Indices can be of two types: unique or not unique**
  - Unique indices make it possible to ensure that no two combination of the same keys exist in the table
  - Non-unique indices simply speed up the retrieval of information

Creating Schemas and Tables

- **Schemas are created with CREATE DATABASE:**
  - CREATE DATABASE *database_name*
- **Tables are created with CREATE TABLE:**
  - CREATE TABLE *table_name* (
    column1 column1_type,
    ...)
- **Table names are unique**
  - This is true on a per-schema basis
- **Each table must contain at least one column**
  - Most DBMSs implement some sort of limits to the size of a row, but that is not part of the standard

Creating Indices

- **Indices are created using CREATE INDEX:**
  - CREATE [UNIQUE] INDEX *index_name* (
    column1,
    ...)
- **Index names must be unique**
  - On a per-schema basis
- **Primary keys are special unique indices that indicate the "primary" method of accessing a table**
  - There can only be one primary key per table
  - Generally, the primary key indicates the way the data is physically sorted in storage

Creating Good Indices

- **A good index provides maximum performance at minimum cost**
  - Create only indices that reflect database usage
  - Implement the minimum number of columns per index
  - Create as few indices as possible
- **Many DBMSs can only use one index per query**
  - Make sure you understand how your DBMS uses indices
  - Analyze, analyze, analyze
  - Continue analyzing once you're done!

Foreign Keys

- **A foreign key establishes a relationship between two tables:**
  - CREATE TABLE A (ID INT NOT NULL PRIMARY KEY)
  - CREATE TABLE B (A_ID INT NOT NULL REFERENCES A(ID))
- **Foreign keys enforce referential integrity**
  - They ensure that you cannot add rows to table B with values for A_ID that do not exist in table A
  - It also ensures that you cannot delete from table A if there are TABLE B rows that still reference it
- **Some DBMSs do not support foreign keys**
  - Notably, MySQL until version 5.0

Inserting, Updating and Deleting

- **Rows are inserted in a table using the INSERT INTO statement:**
  - INSERT INTO TABLE A (ID) VALUES (123)
  - INSERT INTO TABLE A VALUES (123)
- **Updates are performed using UPDATE:**
  - UPDATE A SET ID = 124
- **Deletions are performed using DELETE:**
  - DELETE FROM A
- **Both additions and deletion can be limited by a WHERE clause:**
  - UPDATE A SET ID = 124 WHERE ID = 123

Retrieving Data

- **Data is retrieved using the SELECT FROM statement:**
  - SELECT * FROM A
  - SELECT ID FROM A
- **SELECT statements can also be limited by a WHERE clause**
  - SELECT * FROM A WHERE ID = 123
  - SELECT ID FROM A WHERE ID = 123
  - Where clauses are what makes indices so important

Joins

- **A join makes it possible to... join together the results from two tables:**
  - SELECT * FROM A INNER JOIN B ON A.ID = B.A_ID
- **Inner Joins require that both tables return rows for a particular set of keys**
- **Outer Joins require that either table return rows for a particular set of keys**
  - SELECT * FROM A LEFT JOIN B
    ON A.ID = B.A_ID
  - SELECT A.ID, B.* FROM A RIGHT JOIN B
    ON A.ID = B.A_ID

Joins

- **Joins don't always work the way you expect them to**
  - SELECT * FROM A INNER JOIN B
    WHERE A.ID <> B.A_ID
  - This **won't** return a list of the rows that A and B do not have in common
  - It **will** return a list of all the rows that each row of A does not have in common with B!
- **Joins also rely on indices**
- **Joins can be stacked, and they are executed from left to right**

Grouping and Aggregates

- **The GROUP BY clause can be used to group return sets according to one or more columns:**
  - SELECT A_ID FROM B GROUP BY A_ID
- **Grouped result sets can then be used with aggregates to perform statistical analysis on data:**
  - SELECT A_ID, COUNT(A_ID) FROM B GROUP BY A_ID
- **When using GROUP BY, only aggregates and columns that appear in the GROUP BY clause can be extracted**
  - This is the standard, but it's not always respect (notably by MySQL)

Aggregates

- **Sum of all rows**
  - SUM(column_name)
- **Count of rows returned**
  - COUNT(column_name)
  - COUNT(*)
- **Arithmetic average:**
  - AVG(column_name)
- **Maximum / minimum**
  - MAX (column_name)
  - MIN (column_name)
- **Not all aggregates can be sped up by proper indexing**

Sorting

- **Result sets can be sorted using the ORDER BY clause**
  - SELECT * FROM A ORDER BY ID
    - This is superfluous — ID is the primary key!
  - SELECT * FROM A ORDER BY ID DESC
  - SELECT * FROM B ORDER BY A_ID DESC, ID
- **Sorting performance is affected by indexing**

Transactions

- **Transaction create atomic sets of operations that can be committed or rolled back without any chaange to the underlying data**
  - BEGIN TRANSACTION
    DELETE FROM A
    DELETE FROM B
    ROLLBACK TRANSACTION
  - BEGIN TRANSACTION
    UPDATE A SET ID = 124 WHERE ID = 123
    UPDATE B SET A_ID = 124 WHERE ID = 123
    COMMIT TRANSACTION
- **Not all DBMSs support transactions**
  - For example, MySQL only supports them with InnoDB

SQL and Dates

- **Most DBMSs can handle dates much better than PHP**
  - Extended range
  - Higher resolution
- **Therefore, you should keep all date operations *within* your DBMS for as long as possible**

File Wrappers

- **File wrappers extend PHP's file handling**
  - use fopen(), fread() and all other file functions with something *other* than files
  - For example, access HTTP, FTP, ZLIB and so on
- **Built-in wrappers, or your own**
  - Simply define your own wrapper class:

```
class wrap {
function stream_open($path, $mode, $options, &$opened_path) {}
function stream_read($count) {}
function stream_write($data) {}
function stream_tell() {}
function stream_eof() {}
function stream_seek($offset, $whence) {}
}
stream_wrapper_register("wrap", "wrap"); // register wrapper
$fp = fopen("wrap://some_file", "r+"); // open file via new wrapper
```

File Wrappers

- **Not all file wrappers support all operations**
  - For example, HTTP is read-only
- **Remote file access may be turned off**
  - Use the allow_furl_open PHP.INI directive
- **Some wrappers are write-only**
  - For example: php://stdout and php://stderr
- **Some wrappers do not support appending**
  - For example ftp://
- **Only the "file://" wrapper allows simultaneous read and write operations**

File Wrappers

- **File wrappers support information retrieval via stat() and fstat()**
  - This is only implemented for file://
  - Remember, however, that SMB and NFS files are "local" as far as the operating system is concerned
- **Deleting and renaming is also supported**
  - Renaming only supported for local file (but see above)
  - Both require write access
- **You can also access and manipulate directories**
  - Supported only for local files
- **Remember to close unused wrapper instance**
  - Not necessary, but often a good idea

Streams

- **Streams represent access to network services**
  - File wrapper
  - One or two pipelines
  - Context
  - Metadata
- **Pipelines**
  - Established to allow for the actual streaming of data
  - Can be one only (read *or* write) or two (read *and* write)
- **Context**
  - Provides access to advanced options
    - For example, under HTTP you can set additional headers

Streams

- **Metadata**
  - Contains “out-of-band” information provided by the stream
    - print_r(stream_get_meta_data(fopen("http://www.php.net", "r")));

```
/* Array (
    [wrapper_data] => Array (
        [0] => HTTP/1.1 200 OK
        [1] => Date: Wed, 25 Aug 2004 22:19:57 GMT
        [2] => Server: Apache/1.3.26 (Unix) mod_gzip/1.3.26.1a PHP/4.3.3-dev
        [3] => X-Powered-By: PHP/4.3.3-dev
        [4] => Last-Modified: Wed, 25 Aug 2004 21:12:17 GMT
        [5] => Content-language: en
        [8] => Content-Type: text/html;charset=ISO-8859-1
    )
    [wrapper_type] => HTTP
    [stream_type] => socket
    [unread_bytes] => 1067
    [timed_out] =>
    [blocked] => 1
    [eof] =>
```

Sockets

- **Sockets provide the lowest-level form of network communication**
  - Because of this, you should use them only when strictly necessary
- **Several transports are supported:**
  - TCP/UPD
  - SSL
  - TLS
  - UNIX
  - UDG
- **You can't switch transports mid-stream**
  - Sometimes problematic for TLS

Sockets

- **Opening:**
  - $fp = fsockopen ($location, $port, &$errno, &$errstr)
  - You can then use fwrite, fread(), fgets(), etc.
- **Opening persistend sockets:**
  - $fp = pfsockopen ($location, $port, &$errno, &$errstr)
  - Persistent sockets will only work for persistent APIs, like mod_php on Apache and FastCGI
  - Connections can also be terminated from the remote host because of lack of network activity
  - Use with care—lots of potential pitfalls!

Socket Timeout

- **An optional fifth parameter to fsockopen() indicates timeout**
  - $fp = fsockopen("www.php.net", 80, $errno, $errstr, 30);
  - Timeout is in seconds
  - Default is stored in default_socket_timeout PHP.INI setting
- **Timeout must be set separately for network activity:**
  - socket_set_timeout ($socket, $timeout)
- **Sockets can be blocking or non-blocking**
  - stream_set_blocking ($socket, FALSE);
  - This needs a pre-existing socket!

Q&A Time

- What does an “inner join” construct do?
- What function would you use to open a socket connection manually with the purpose of communicating with a server not supported by a file wrapper?
- When dealing with timeout values in sockets, the connection timeout can be changed independently of the read/write time out. Which function must be used for this purpose?

Answers

- What does an “inner join” construct do?
- It creates a result set based on the rows in common between two tables

Answers

- What function would you use to open a socket connection manually with the purpose of communicating with a server not supported by a file wrapper?

- fsockopen()
- pfsockopen() for persistent connections

Answers

- When dealing with timeout values in sockets, the connection timeout can be changed independently of the read/write time out. Which function must be used for this purpose?

- stream_set_timeout()

PART VI: Secure, Optimize, Debug

- **What we'll cover in this section:**
  - Data filtering
  - SQL injection
  - Command injection
  - XSS
  - Safe mode
  - Coding Standards
  - Error logging
  - Debugging and optimization

Data Filtering

- **Users are evil**
  - And sometimes they don't even know it
- **You should always "taint" and filter data**
  - PHP provides lots of functions that can help here
  - Never rely on register_globals
    - In fact, if you're writing for redistribution, undo its effects if it is on
- **Data filtering depends on what you need to do with it**
  - You will rarely need "raw" data
  - Most of the time, it needs to be escaped to do something or other—e.g.: display, insert into db, and so on

SQL Injection

- **SQL injection occurs when improperly filtered data ends up in a database query**
  - "SELECT * FROM USER WHERE ID = $id"
  - $id = "1; DELETE FROM USER;"
- **Most DBMS modules have their own escaping mechanisms**
  - mysql_real_escape_string()
  - addslashes() — The swiss army knife approach

Command Injection

- **Command injection takes place when improperly filtered input ends up in a shell command**
- **Both commands and parameters should be escaped:**
  - escapeshellcmd ($cmd)
  - escapeshellarg ($arg)
  - shell_exec ($cmd . ‘ ‘ . $arg)

Cross-site Scripting

- **XSS happens when improperly escaped input is outputted to the client**
  - XSS can be used for all sorts of nasty purposes
  - Often underrated, it is an extremely serious security problem
  - It's often easy to implement on the attacker's side
- **User input should be properly escaped before being outputted back to the browser**
  - htmlspecialchars()
  - htmlentities()
  - strip_tags()

Safe Mode

- **Safe mode implements certain restrictions to help prevent security problems**
  - UID matching
  - open_basedir restrictions
- **Safe mode and open_basedir have several drawbacks**
  - PHP is not the right place for implementing security at this level
  - Files created in safe_mode may not be readable by your scripts!
  - Add noticeable overhead to the system

Coding Standards

- **Coding standards help writing good code**
  - There is no "official" standard connected with the exam
- **A few ideas:**
  - Flattening if statements
  - Splitting long statements across multiple lines
  - Using substitution instead of concatenation
    - Watch out for performance hits
  - Comparison vs. Assignment
    - Reverse comparisons
  - Use type-sensitive comparisons when possible
  - Validate resources

Error Management

- PHP has an impressive array of error management facilities—use them!
- Report all errors during development
- Keep error reporting on in production, but shift to logging
- Implement your own error handlers

Debugging

- **Debugging can be very difficult**
- **"Echo" debugging is the simplest form**
  - Output status throughout the script's execution
- **Complex logic is better handled through external debuggers**
  - Lots available—from open source (Xdebug) to commercial (e.g.: Zend Studio IDE)
  - IDEs support both local and remote debugging

Optimization

- **Optimization can be as simple as installing a bytecode cache**
  - No changes to codebase
  - Immediate (but limited) benefits
- **Proper optimization requires good analysis**
  - Finding bottlenecks
- **Optimization can take place on multiple levels:**
  - Write faster code
  - Remove external bottlenecks
  - Use caching for internal bottlenecks
  - Improve web server configuration

Q&A Time

- Although the best practice is to disable register_globals entirely, if it must be enabled, what should your scripts do to prevent malicious users from compromising their security?
- When uploading a file, is there a way to ensure that the client browser will disallow sending a document larger than a certain size?
- Can you turn off all error reporting from within a script with a single PHP function call?

Answers

- Although the best practice is to disable register_globals entirely, if it must be enabled, what should your scripts do to prevent malicious users from compromising their security?

- Filter all data
- Initialize all variables

Answers

- **When uploading a file, is there a way to ensure that the client browser will disallow sending a document larger than a certain size?**

- **No.**
  - You can check a file size after it's been uploaded
  - The server can ignore files above a certain size
  - But you can't prevent the user from *trying* to send the data across the network

Answers

- **Can you turn off all error reporting from within a script with a single PHP function call?**
- **No.**
  - error_reporting() will not silence parse errors

Conclusion

- **A few quick words about the exam**
- **Pay close attention to the code**
  - Pay close attention to the code
  - Are you paying close attention yet???
- **You have 90 minutes—use them all**
- **Use the booklet to mark your questions before you transfer them over to the answer sheet**
- **Remember that you're working with PHP 4, not PHP 5—and 4.3, not 4.4!**
- **Don't forget to sign up for your exam at the registration desk**