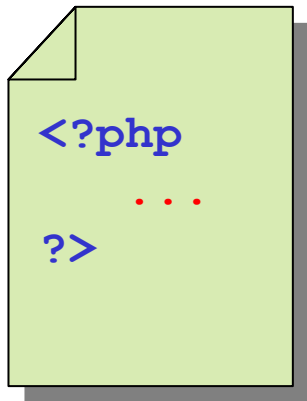


COSC 2206 Internet Tools



Introduction to PHP
Installing PHP (brief)
Language Summary

<http://localhost/php/>



Introduction to PHP

Origins of PHP
Important Features
Version Incompatibilities



PHP Timeline

- Rasmus Lerdorf's Personal Home Page tools, 1994
- Later recursively renamed to **P**HP **H**ypertext **P**rocessor
- PHP 1.0, June 1995
- PHP 2.0, April 1996
- PHP 3.0, June 1998
- We are using version 4.3.7 (June 2003)



Important PHP Features (1)

- Designed from beginning as a server-side web scripting language.
- Runs on all major operating systems.
- Built-in support for several relational databases such as MySQL
- Built-in support for session management using cookies and / or URL rewriting.
- Can also be used for command-line scripting



Important PHP Features (2)

- Interpreted language
- Loosely typed language
 - a variable can hold a value of any type at different stages in its lifetime
- Hybrid language
 - can be used in a non object-oriented style or an object-oriented style or a mixture of both.
- Currently the most popular web scripting language



Important PHP features (3)

- Can be configured as a CGI process
 - This is inefficient.
 - a separate process is needed for each connection to the web server.
- Can be configured as an Apache module
 - This is efficient.
 - one instance of the PHP interpreter remains running as long as the web server is running.

Version incompatibilities

- The latest versions 4.1 and above are not compatible with earlier versions if `register_globals` is set to `off` (now the default in the `php.ini` file).
- It is highly recommended to leave it off for security reasons. We will assume this.
- This means that values in certain global arrays are no longer mapped to variables in PHP.
- More on this when we do web apps in PHP

A red, cloud-like shape with a black outline and a drop shadow, containing the word "Warning" in white text.

Warning



COSC 2206 Internet Tools

Brief PHP Installation Instructions Windows XP



Installing PHP for Win XP

- Here we give only a brief description of installation
- More complete details are available on the course CD-ROM or online at

<http://www.cs.laurentian.ca/badams/c2206/install-notes/php/>



Getting PHP

- Go to www.php.net to get the software.
 - It's also on the CD-ROM (version 4.3.7)
- For windows it should have a name like `php-4.3.7-Win32.zip`
- Unzip it into your top level `c:\` directory
- This gives a directory called `c:\php-4.3.7-Win32`
- Rename this directory to just `c:\php`



Getting PHP documentation

- The documentation is an 1800 page manual and it is necessary to have it.
- It comes in several formats but the best for Windows is the help file format
- To get this format download the file `php_manual_chm_12.zip`
- Unzip it in your `c:\php\docs` directory and put a shortcut to it on your start menu
- It is also available on the CD



php.ini configuration file

- In the directory `c:\php` you will find a default configuration file called `php.ini-recommended`
- Copy it to your windows directory
 - `c:\windows`
- Rename the copy to just `php.ini`



Editing `php.ini` (1)

- Make the following changes to `php.ini`
 - `locate doc_root =`
 - change to `doc_root = "c:\Apache\htdocs"`
 - `locate display_errors = Off`
 - change to `display_errors = On`
(for learning PHP)
- We assume that Apache is installed directory `c:\Apache`



Editing `php.ini` (2)

- Make the following changes to `php.ini`
 - Uncomment the lines
 - `;extension=php_gd2.dll`
 - `;extension=php_pdf.dll`
- Replace the line
 - `extension_dir = "./"`
- with
 - `extension_dir = "c:\php\extensions"`



Editing `php.ini` (3)

- Make the following changes to `php.ini`
- Replace the line
 - `session_save_path = "/tmp"`
- with (make directory `sessions`)
 - `session_save_path = "c:\Apache\sessions"`
- Change the line for Windows
 - `;include_path = ".;c:\php\includes"`
- to (make directory `php-includes`)
 - `include_path = ".;c:\php;c:\Apache\php-includes"`



Configuring Apache (1)

- To configure PHP as an Apache module
locate the file `c:\php\php4ts.dll`
- Copy it to
 - `c:\windows\system32`
- Now it is necessary to configure apache as a module (built-in to Apache)



Configuring Apache (2)

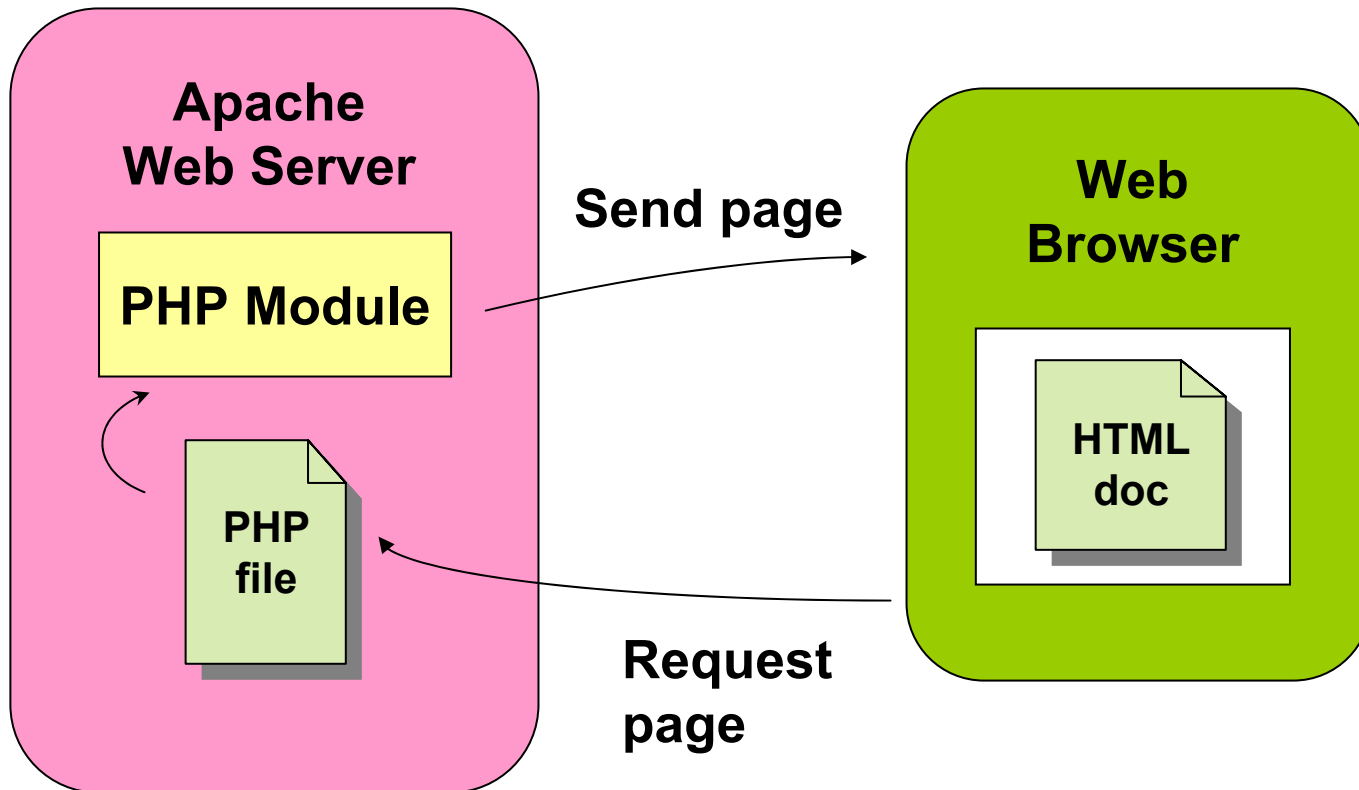
- Edit `c:\apache\conf\httpd.conf` and add the following LoadModule statement
 - `LoadModule php4_module c:/php/sapi/php4apache.dll`
- Locate the section of the file with the AddModule statements and add the two lines
 - `AddModule mod_php4.c`
 - `AddType application/x-httpd-php .php`
 - `AddType application/x-httpd-php-source .phps`
- This tells apache to treat files with the php extension as executable by PHP



Configuring Apache (3)

- Restart or start Apache so that the modifications to `httpd.conf` will be read.
- In the command window you should see
 - **Apache/1.3.29 (Win32) PHP/4.3.7 running**
- This indicates that PHP has been properly installed as an Apache module

PHP-Apache Integration





Testing PHP

- Create the following `hello.php` script

```
<html>
<head><title>Hello Script</title></head>
<body>
<?php echo "<h1>Hello PHP World!</h1>" ?>
</body>
</html>
```

- Put it in `c:\apache\htdocs\testphp`
- Try the URL
 - <http://localhost/testphp/hello.php>



Language Summary

Where does PHP code go?

Data types

Built-in functions

Control structures

Where does PHP code go?

- PHP code can be standalone (like Perl) or embedded in HTML code (like ASP)
- PHP script tags are used to distinguish PHP code from HTML code:

`<?php`

Start PHP mode

PHP code goes here

`?>`

End PHP mode

- These tags are XML compliant and should always be used for fully portable PHP code



8 Data Types

- 4 are scalar types
 - integer, floating point (double),
 - string,
 - boolean
- 2 are structured types
 - array and object
- 2 are special types
 - resource, NULL

Declaring Variables

- Variable types are not declared
- Type is type of value currently assigned
- All variables begin with a dollar sign
 - next character is a letter or underscore
 - remaining are letters, underscore or digits

```
$count = 0;  
$name = "Fred";  
$price = 45.50;  
$big_num = 1.2345E23;  
$success = TRUE;
```

case
insensitive



Naming conventions

- The convention in PHP (coming from C) is that the underscore character is used to simulate a space in variable names:
- Example:
 - `$number_of_files = 3;`
- This is different from the uppercase convention that is used in Java
 - `numberOfFiles = 3;`



Constants

- Constants are defined using the **define** function and do not begin with a dollar sign
- The convention is to use all uppercase letters and _ for names of constants
- Examples:

```
define ('COURSE', "Internet Tools");  
define ('CM_TO_INCH', 2.54);
```
- Now use **COURSE** and **CM_TO_INCH** to refer to these constants



Case sensitivity

- PHP is a weird mixture of case sensitivity and case insensitivity
- variable and constant names are case sensitive
- boolean values TRUE and FALSE are not case sensitive.
- function names are not case sensitive
- Maybe they will fix this in a future version



Three kinds of comments

- C, Java style multi-line comments

```
/* This is a multi-line  
   comment  
*/
```

- C++, Java single line comments

```
// a single line comment
```

- Unix shell script and Perl comments

```
# a single line comment
```



Literals and Strings

- Integer literals
 - -1, 1, 2, 3
- Floating point literals
 - 1.234, -4.56, 1.34E-12
- There are two kinds of string literal
 - single quotes: 'hello'
 - double quotes: "hello"
- There are two boolean literals
 - **TRUE, FALSE**



Variable Interpolation (1)

- Interpolation takes place inside doubly quoted strings.
- This means that variables are replaced by their values and special control sequences such as `\n` take effect.
- Interpolation does not take place inside singly quoted strings so `\n` is just two characters
- This is similar to interpolation in Perl.



Variable Interpolation (2)

- If `$name` has the value `"Fred"` then
- `"Hello $name"` has the value
`Hello Fred`
- `'Hello $name'` has the value
`Hello $name`
- `"\ $name = $name"` has the value
`$name = Fred`
- In the last case the escape character `\` is used to specify that the first `$` is a literal character.



Displaying strings (1)

- The **print** function displays text on the standard output.
 - **print** ("Hello \$name\n") ;
- The **echo** function is similar but parentheses are not needed and several arguments can be supplied
 - **echo** "Hello ", \$name, "\n" ;
- There is also a C-style **printf** function that produces formatted output.



Displaying strings (2)

- **`print_r(expression)`**
 - function which displays a string representation of a variable such as an array.
- **`var_dump(expression)`**
 - similar output as `print_r`
- **`var_export(expression)`**
 - returns a string representation of a variable. Can be used in a print statement (like `toString` in Java)



Type casting

- The type of a variable can be changed using the typecasting operators
 - `(int)` or `(integer)`
 - `(float)` or `(double)` or `(real)`
 - `(bool)` or `(boolean)`
 - `(string)` or `" "`
 - `(array)`
 - `(object)`



Arithmetic

- The usual operators are `+`, `-`, `*`, and `/`
- Note that `/` is always a floating point divide
- For integer divide use `(int) ($a/$b)`
- The remainder (mod) operator is `%` so if `$a` and `$b` have integer values then `$a % $b` is the remainder when `$a` is divided by `$b`.
- The `++` and `--` operators are available
- Also we have `+=`, `-=`, `*=` and so on



What is false in PHP?

- In the conversion to boolean only the following are considered to be false.
 - the boolean value **FALSE**
 - the integer 0
 - the floating point number 0.0
 - the empty string "" and the string "0"
 - an array with no elements (like **array()**)
 - an object with no elements
 - the special value **NULL**, and an unset variable



Relational Operators

- PHP has the usual C and Java style operators
 - `||` for logical or
 - `&&` for logical and
 - `!` for negation
- There are also different precedence versions
 - `or` for logical or
 - `and` for logical and



Comparison operators

- PHP has the usual C and Java style operators
 - `<`, `<=`, `==`, `!=`, `>`, `>=`
- These operators perform type conversions if necessary before the comparison
- There are also the operators
 - `===`, and `!==`
- These operators compare value and type so no type conversion is done
- These operators are also used for strings



Comparison of == and ===

- Suppose `$a` has the value `"0"` as a string
- Suppose `$b` has the value `0` as an integer
- Then
 - `$a == $b` is true since both `"0"` and `0` evaluate to false but
 - `$a === $b` is false since `$a` and `$b` have different types



Conditional statements

- The if statement is similar to C and Java except **elseif** can be used instead of **else if >>**

```
if ( booleanExpression1 )
{
    ...
}

elseif ( booleanExpression2 )
{
    ...
}

else
{
    ...
}
```




Testing variables

- **empty(variable)**

- returns true if variable is set and has an empty value (for example, 0, ' ').

- **isset(variable)**

- returns true if the variable exists
- also see **unset**

- **is_null(variable)**

- returns true if an existing variable has the null value



empty / isset (from manual)

```
<?php
$var = 0;

// Evaluates to true because $var is empty

if (empty($var))
{
    echo '$var is either 0, empty, or not set at all';
}

// Evaluates as true because $var is set

if (isset($var))
{
    echo '$var is set even though it is empty';
}
?>
```



Math functions (1)

- There are many math constants in PHP. For example π is `M_PI` and $\sqrt{2}$ is `M_SQRT2`
- Example:

```
$radius = 1.0;  
$circ = 2.0 * M_PI * $radius;  
$area = M_PI * $radius * $radius;  
echo "Circumference is $circ\n";  
echo "Area is $area\n";
```



Math functions (2)

- There are many math functions in PHP:
 - `abs` `sqrt`
 - `sin` `cos` `tan`
 - `asin` `acos` `atan` `atan2`
 - `exp` `expm1` `log` `logp1`
 - `rand` `round`
 - `min` `max`
 - `pow`
- and many more



COSC 2206 Internet Tools

Strings and String Functions in PHP



String interpolation

- single or double quotes as delimiters
- Variables are not interpolated in singly quoted strings.
- To include a single quote it is necessary to use `\'` and to include a backslash use `\\`
- Variables are interpolated in doubly quoted strings and escape sequences such as `\"`, `\n`, `\\`, `\$`, `\t`, `\{`, `\}`, `\[`, `\]` can be used.



String operations (1)

- The dot operator is used to concatenate strings
 - `"Hello " . "World"` is the same as `"Hello World"`
- `echo`, `print`, `printf`, `print_r`, `var_dump` functions display strings on standard output
- The length of a string `$s` is `strlen($s)`
- The `$i`-th character of string `$s` is `$s[$i]`



String operations (2)

- Removing whitespace from strings
- Trim whitespace at left end
 - `$trimmed_string = ltrim($s) ;`
- Trim whitespace at right end
 - `$trimmed_string = rtrim($s) ;`
- Trim whitespace at both ends
 - `$trimmed_string = trim($s) ;`



String operations (3)

- Converting `$s` to upper case or lower case
 - `$upper = strtoupper($s);`
 - `$lower = strtolower($s);`
- Converting first character of `$s` to uppercase
 - `$upper_first = ucfirst($s);`
- Converting first character of every word in `$s` to uppercase
 - `$upper_word = ucwords($s);`



HTML encoding

- Special HTML characters such as `<`, `>` and `&` have special entities (`<`, `>`, `&`) so they can be used in HTML documents as ordinary characters without special meaning.
- The `htmlspecialchars` function can be used to do this. For example
 - `$s1 = "<p>";`
 - `$s2 = htmlspecialchars($s1);`
 - `$s2` has the value `"<p>"`



Database encoding

- SQL query strings and database fields need to be encoded. The encodings can depend on the particular database.
- The standard encoding is to precede single quotes, double quotes, NUL bytes and backslashes by a backslash
- The **addslashes** function performs the encoding and the **stripslashes** function performs the decoding.



String comparison (1)

- Can use the operators
 - <, <=, >, >=, ===, !==
- Don't use == and !=. Unexpected results for mixed comparisons of numbers and strings.
- Can use === and !== to compare 2 strings

```
if ($str1 === $str2)
{
    echo "Strings are equal";
}
```



String comparisons (2)

- To compare two strings as strings its best to use the **strcmp** function (similar to C, Java)
 - **strcmp**(\$s1, \$s2) returns
 - negative integer if \$s1 precedes \$s2
 - zero if \$s1 and \$s2 are the same
 - positive integer if \$s1 follows \$s2
- There is also **strcasecmp** to do case insensitive comparison
- There are other comparison functions.



Substrings, replacement

- **substr**(string, start)
 - return substring of string beginning at index start and continuing to end of the string
- **substr**(string, start, length)
 - return substring of string beginning at index start and consisting of length characters
- There are also functions **substr_replace**, **strrev**, **str_repeat**, **str_pad**



Exploding a string

- **explode**(separator, string)
 - extracts an array of substrings using the characters in separator as field separators

- Example

```
$account = "123:Jack Sprat:45.50";  
$fields = explode(":", $account);
```

- **\$fields** is an array of the three strings "123", "Jack Sprat", and "45.50"



Imploding a string

- **implode**(separator, array)
 - inverse of explode: makes a single string from the strings in an array using separator string between each string in the array

- Example:

```
$fields = array("123", "Jack Sprat", 45.50;  
$account = implode(":", $fields);
```

- **\$account** now contains the string
"123:Jack Sprat:45.50"



Searching strings (1)

■ **strpos**(string, target)

- return position of first occurrence of target string in string. If target not found false is returned but if target is found at position 0 then 0 is returned and this would also be false, YUK. Therefore use `===` to check if string wasn't found.

```
$pos = strpos("string to search", "str");  
if ($pos === false) {  
    // string wasn't found  
} else {  
    // string was found at position $pos  
}
```



Searching strings (2)

- Most other string searching functions are bizarre and probably should never be used.
- For example **strrpos** finds the last occurrence of a character but won't find the last occurrence of a multi-character string. To find last occurrence use **strrev** and **strpos**.
- Forget these functions and use regular expressions.



COSC 2206 Internet Tools

Regular Expressions in PHP



Regular expressions (1)

- Regular expressions provide a powerful way to do pattern matching: finding one pattern in another.
- The simplest patterns are fixed strings like those in the searching function **strpos**.
- However patterns can represent complex classes of strings.
- Useful for validating strings



Regular expressions (2)

- **ereg**(pat, str)

- returns true if match for pattern was found in string.
- **ergi** is the case insensitive version

- **ereg_replace**(pat, replace, str)

- Each occurrence of pattern match in string is replaced by the replace string. Result is returned as a new string
- **ergi_replace** is the case insensitive version



Regular expressions (3)

- **split**(`pattern`, `string`)

- splits string into substrings using pattern as the delimiter. The substrings are returned as an array.



Simple Reg Exp Examples (1)

- Matching strings containing only digits
 - `$regexp = "[0-9]+$";`
 - `ereg($regexp, "2342123")` returns `true`
 - `ereg($regexp, "2124sd")` returns `false`
- `^` matches the beginning of the string
- `$` matches the end of the string
- `[0-9]` specifies a range for a character
- `+` means 1 or more occurrences



Simple Reg Exp Examples (2)

- Matching phone numbers of the form
"ddd-ddd-dddd"
- `$regex` =
"`^[0-9]{3}-[0-9]{3}-[0-9]{4}$`";
- Here the hyphen is a literal character and `{3}` indicates exactly three occurrences of the preceding character.



Simple Reg Exp examples (3)

- Match file names with php extension in directories simple or forms, for example **simple/test.php** or **forms/temp.php**

```
$regexp = "(simple|forms)/[0-9a-zA-Z_]+\\.php$";  
$filename = "simple/test.php";  
$valid = ereg($regexp, $filename);
```

- . represents any character so to get a literal . it is necessary to use \.
- (...|...) is alternation: simple 'or' test in this case



Perl Regular Expressions

- There are PCRE (Perl Compatible Regular Expressions) versions of these functions
 - `preg_match(pattern, str)`
 - `preg_replace(pattern, replace, str)`
 - `preg_split(pattern, str)`
- Delimiters are needed in the pattern
 - e.g. `/.../`
- And a few variations:
 - `preg_match_all`, `preg_grep`



Perl Reg Exp Example

- Match names containing letters, hyphens and apostrophe's
 - `$regexp = '/^[a-zA-Z\\-\\']+$/' ;`
- Note the delimiters `/ ... /` needed for Perl
- Also note that `-` and `'` are escaped since they are special
- Now to validate a name use
 - `preg_match($regexp, $name)`



COSC 2206 Internet Tools

Arrays in PHP



Two types of arrays

- Indexed arrays
 - the indices are 0, 1, 2,
 - these are like arrays in C, Java, or Perl
- Associative arrays
 - indices are strings (keys)
 - these are similar to hashes in Perl or HashMap's in Java
- Any values can be stored in an array



Creating indexed arrays

- Creating and initializing an array
 - `$a = array(10, 20, "Help");`
 - `$a[0]` is 10, `$a[1]` is 20, `$a[2]` is "Help"
- Extending the array dynamically
 - `$a[] = 30; // this is $a[3]`
- Creating an array by autovivification
 - `$b[0] = 10; $b[1] = 20;`
 - `$b[] = "Help"; // next element`
 - `$b[] = 30; // next element`



Indexed arrays from ranges

- `$digits = range(0, 9);`
 - `$digits[0]` is 0,
 - `$digits[1]` is 1, ...
- `$letters = range('a', 'z');`
 - `$letters[0]` is 'a',
 - `$letters[1]` is 'b', ...



Indexed array list operation

- The `list` operation is quite useful for making multiple assignments from an indexed array

```
$product = array(123, 'No-name PC', 450);  
list($id, $description, $price) = $product  
echo "<p>id = $id</p>";  
echo "<p>description = $description</p>";  
echo "<p>price = $price</p>";
```

- This only works for indexed arrays



Indexed array slices

- **array_slice**(array, offset, length)
 - returns a subarray with length elements of array beginning at offset

```
$letters = range('a', 'z');  
$slice = array_slice($letters, 5, 10);
```

- Extracts `$letters[5]` to `$letters[14]` as `$slice[0]` to `$slice[9]`

Associative arrays

- An associative array is a table of key-value pairs. Here the names are the keys and the values are the ages.

```
$age = array( 'Fred' => 37, 'Gord' => 23,  
             'Alice' => 17, 'Bob' => 23 );  
  
echo "Fred's age is ", $age['Fred'];  
  
$age['Fred'] = 65; // Fred is now a senior  
  
echo "<br>Fred's age is {$age['Fred']}";
```

Note braces
needed for
interpolation



Associative arrays as records

- An associative array can represent a record in a database

```
$account = array( 'number' => 123,  
                  'name'   => 'Fred',  
                  'balance' => 450.50 );  
  
echo "Account balance is ", $account['balance'];  
  
$account['balance'] += 100; // deposit $100  
  
echo "<br>New balance is {$account['balance']}";
```



Associative arrays of arrays

- The values can be anything

```
$colors = array( 'red' => array(255,0,0) ,  
                'green' => array(0,255,0) ,  
                'blue' => array(0,0,255) ,  
                'yellow' => array(255,255,0) ,  
                'brown' => array(128,64,0) );  
$yellow = $colors['yellow']; // yellow array  
$red = $colors['yellow'][0]; // red part of yellow
```

- The values here are arrays each containing the red, green blue components of a color



Associative array keys

- **array_keys** (array)

- extracts the keys of the given associative array into an indexed array

```
$colors = ...  
$color_names = array_keys($colors);
```

- Then `$color_names[0]` is 'red', ..., `$color_names[4]` is 'brown'



Checking for key existence (1)

- **array_key_exists**(key, array)
 - returns true if the given key exists in the array.
If the key exists it may or may not be null.

```
$product = array('id' => 1, 'desc' => null,  
                'price' => 2.50);  
  
if (array_key_exists('desc', $product))  
{ echo 'key exists'; }  
else  
{ echo 'key does not exist'; }
```

- Here "key exists" is displayed (also see is_null)



Checking for key existence (2)

- **isset**(array['key'])

- returns true if the given key exists in the array and is not null

```
$product = array('id' => 1, 'desc' => null,  
                 'price' => 2.50);  
  
if (isset($product['desc']))  
{ echo 'key exists and is not null'; }  
else  
{ echo 'key does not exist or is null'; }
```

- Here "key does not exist or is null" is displayed



array --> variables (extract)

- The **extract** function converts an array to variables

```
$account = array('number' => 123, 'name' => 'Fred',  
                 'balance' => 45.50);  
extract($account, EXTR_PREFIX_SAME, "my");
```

- This creates the variables **my_number**, **my_name**, **my_balance** with the values 123, "Fred", and 45.50
- There is an inverse function called **compact**



Array copy function

- In PHP array assignment is a copy operation
- This is very different than Java where array assignment simply makes a new reference to the same array. To test this try

```
$age_copy = $age;  
$age_copy['Gord'] = 55;  
var_dump($age);  
var_dump($age_copy);
```

- Gord's age is still 23 in the `$age` array.



searching arrays

- **in_array**(element, array)
in_array(element, array, TRUE)
 - returns true if the given element is found in the given array (TRUE option for same types)
- **array_search**(element, array)
array_search(element, array, TRUE)
 - returns the key of the element if found, else returns false (TRUE option for same types)



Sorting arrays (1)

- **sort(array), rsort(array)**
 - sort array in ascending alphabetical or numerical order or reverse alphabetical or numerical order

```
$names = array('Fred', 'Ted', 'Barney', 'Gord');  
sort($names); // 'Barney', 'Fred', 'Gord', 'Ted'  
print_r($names);  
rsort($names); // 'Ted', 'Gord', 'Fred', 'Barney'  
print_r($names);
```



Sorting arrays (2)

- **asort(array), arsort(array)**

- sort associative array by values in ascending or descending order

```
$ages = array('Fred' => 34, 'Ted' => 45,  
              'Barney' => 23, 'Gord' => 15);  
asort($ages); // 'Gord', 'Barney', 'Fred', 'Ted'  
print_r($ages);
```

- The order is increasing order of age



Sorting arrays (3)

- **ksort(array), krsort(array)**

- sort associative array by keys in ascending or descending order

```
$ages = array('Fred' => 34, 'Ted' => 45,  
              'Barney' => 23, 'Gord' => 15);  
ksort($ages); // 'Barney', 'Fred', 'Gord', 'Ted'  
print_r($ages);
```

- The order is ascending alphabetical order of the names



Other array functions

- There are many other array functions
 - reversing arrays
 - sorting multiple arrays
 - user-defined sorting
 - **array_walk**, and **array_reduce**
 - merging two arrays
 - array filtering
 - set operations (**union**, **merge**, **unique**)
 - stack operations (**push**, **pop**, **shift**, **unshift**)



The for loop

- Similar to C and Java:

```
for ($count = 1; $count <= 10; $count++)  
{  
    echo "$count ";  
}
```

```
$ages = array(34, 45, 56, 65);  
for ($k = 0; $k < count($ages); $k++)  
{  
    echo $age[$k], ' ';  
}
```



The foreach loop (1)

- Useful for indexed arrays when the loop index is not needed as in preceding example:

```
$ages = array(34, 45, 56, 65);  
foreach ($ages as $age)  
{  
    echo $age, ' ';  
}
```

- Here `$age` successively takes on the array values
- **INEFFICIENT**: `foreach` makes a copy of array



The foreach loop (2)

- Useful for processing associative arrays

```
$product = array('id' => 23,  
    'desc' => 'No-name PC', 'price' => 549.99);  
  
foreach ($product as $key => $value)  
{  
    echo "Key = $key, Value = $value<br>";  
}
```

- `$key` and `$value` are successively the keys and values in the `$product` array



The while loop

- Similar to C and Java.

Can also be used with iterator functions:

```
$product = array('id' => 23,  
    'desc' => 'No-name PC', 'price' => 549.99);  
  
while (list($key,$value)= each($product))  
{  
    echo "Key = $key, Value = $value<br>";  
}
```

- More efficient than **foreach** since a copy of the array is not made (see next slide)



How **each** works

- For the array

```
$product = array('id' => 23,  
                 'desc' => 'No-name PC', 'price' => 549.99) ;
```

each(\$product) initially returns the associative array

- **array**(0 => 'id', 1 => 23,
 'key' => 'id', 'value' => 23) ;

- This array can be used either as an indexed or an associative array.



Loop iterator functions

- The loop iterator functions for arrays
 - **current** (array)
 - **reset** (array)
 - **next** (array)
 - **prev** (array)
 - **end** (array)
 - **each** (array)
 - **key** (array)



do while loop

- Similar to C and Java

```
do
{
    Statements
} while (condition)
```



User-defined functions

- User defined functions have the form

```
function name(arg_list)
{
    Statements
}
```



Variable scope

- Variable scoping is very simple in PHP
- Variables defined inside functions are local
- Variables defined outside functions are global variables not available inside a function unless declared as global inside a function using the global statement
- There are a few predefined superglobal variables available anywhere

Variable scope example

- Assume that `$name` is defined outside any function.

```
function one()  
{  
    // global $name not available here  
}
```

```
function two()  
{  
    global $name;  
    // global $name is available here  
}
```

TIP
Don't Use
Global
Variables



static variables

- Static variables are local to a function but initialized only once, when the function is called the first time

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

```
echo counter(); // displays 0  
echo counter(); // displays 1
```



A max function

- Return the max of two numeric values

```
function max2($a, $b)
{
    if ($a > $b) return $a;
    return $b;
}
```

```
echo "max of 2 and 3 is ", max2(2, 3) ;
```

Pass by value

- Multiply each element of an array by 2

```
function times2($a)
{
    for ($k = 0; $k < count($a); $k++)
    {
        $a[$k] = 2 * $a[$k];
    }
    return $a;
}
```

This
version
makes a
copy of
the array

```
$a = array(1,2,3,4,5);
$b = times2($a);
echo var_export($a), "<br/>"; // 1,2,3,4,5
echo var_export($b), "<br/>"; // 2,4,6,8,10
```

Pass by reference

- Multiply each element of an array by 2

```
function times2_ref(&$a)
{
    for ($k = 0; $k < count($a); $k++)
    {
        $a[$k] = 2 * $a[$k];
    }
}
```

Pass
by
reference

```
$a = array(1,2,3,4,5);
echo var_export($a), "<br/>"; // 1,2,3,4,5
times2_ref($a);
echo var_export($a), "<br/>"; // 2,4,6,8,10
```



Variable number of args

- Find max of several numbers

```
function maxn()  
{  
    $max_value = func_get_arg(0);  
    for ($k = 1; $k < func_num_args(); $k++)  
    {  
        if (func_get_arg($k) > $max_value)  
        {  
            $max_value = func_get_arg($k);  
        }  
    }  
    return $max_value;  
}
```



Including files (1)

- A file can be included in another file using include and require
- **include, include_once**
 - inherits the scope of the include point
 - parsing is in HTML mode so code must use the php tags `<?php ... ?>`
- **require, require_once**
 - like include but causes fatal error if file doesn't exist



Including files (2)

- PHP searches for include files in the paths specified in the `php.ini` file
- For example in windows you could use
 - `include_path =`
`". ;c:\php;c:\Apache\php-includes"`
- Then the search is in the current directory (the dot) and if not found there then in the directories `c:\php` and `c:\Apache\php-includes`



Classes and objects

- PHP has object oriented features:
 - classes
 - objects
 - constructors
 - methods
 - inheritance
- Does not support data encapsulation
however since data fields are always public

Writing lines to a file

- First open the file for writing:
 - `$out_file = fopen("test.dat", "w");`
- Write some data to the file
 - `fwrite($out_file, "Line 1\n");`
 - `fwrite($out_file, "Line 2\n");`
- Close the file when finished
 - `fclose($out_file);`
- For appending use `"a"` instead of `"w"`

also fputs

Reading lines from a file

- First open the file for reading:
 - `$in_file = fopen("test.dat", "r");`
- Use a loop to read and display lines
 - ```
while (! feof($in_file))
{
 $line = fgets($in_file, 100);
 echo $line, "
";
}
```
- Close the file when finished
  - `fclose($in_file);`

At most 99  
characters  
per line



# Other file operations (1)

---

- `fgetss(fp, length)`
  - like `fgets` but strips out PHP and HTML tags
- `fgetcsv(fp, length, delimiter)`
  - like `fgets` but returns array of strings using delimiter as separation character
- `fread(fp, length)`
  - read length bytes using file pointer fp
- `fgetc(fp)`
  - reads a single character and returns it



# Other file operations (2)

---

- `readfile(filename)`
  - reads an entire file and echoes it to standard output (browser). returns number of bytes read
- `file(fp)`
  - reads entire file as lines into an array which is returned
- `file_exists(filename)`
  - returns true if the specified file exists



# Other file operations (3)

---

- **unlink(filename)**
  - delete the specified file. Returns true if the file was deleted.
- **filesize(filename)**
  - returns length in bytes of specified file
- There are file operations such as rewind, fseek, and ftell to support direct access
- file locking is supported with flock



# File Based Page Hit Counter

---

- It is easy in PHP to make a simple page hit counter.
- Use a file that contains one number, the number of times the page has been accessed.
- Update this count and display it each time the page is accessed
- [view script lang/counter test.php](#)
- <http://localhost/php/lang/counter test.php>



# counter\_test.php

```
<?php include ("counter.php") ; ?>
<html>
<head>
<title>Testing the counter</title>
</head>
<body>
<h1>Testing the counter</h1>
This page has been visited
<?php echo counter ("counter") ?> times.
</body>
</html>
```



# counter.php (1)

```
<?php
// Function returns counter value with given count
// file name (without extension)
function counter($file_name)
{
 $count = 0;
 if (file_exists($file_name . ".dat")) {
 $counter_file = fopen($file_name . ".dat", "r");
 $count = fgets($counter_file, 100);
 fclose($counter_file);
 }
 $count++;
}
```





# counter.php (2)

```
// write the new value to the file using exclusive
// lock

$fp = get_lock($file_name);
$counter_file = fopen($file_name . ".dat", "w");
fputs($counter_file, $count);
fclose($counter_file);
release_lock($fp);
return $count;
}
```



# counter.php (3)

```
function get_lock($semaphore_file_name)
{
 $fp = fopen($semaphore_file_name . ".sem", "w");
 flock($fp, 2);
 return $fp;
}

function release_lock($fp)
{
 flock($fp, 3);
 fclose($fp);
}
```

[view script simple/counter.php](#)



# Example scripts

---

- The following example scripts illustrate the PHP language
- A better way to show them is to use the link <http://localhost/php/> which shows both the source and the output in side by side frames



# Example scripts (1)

---

- <http://localhost/php/lang/scalars.php>
  - [view source](#)
- <http://localhost/php/lang/display.php>
  - [view source](#)
- <http://localhost/php/lang/strings1.php>
  - [view source](#)
- <http://localhost/php/lang/strings2.php>
  - [view source](#)



# Example scripts (2)

---

- <http://localhost/php/lang/strings3.php>
  - [view source](#)
- <http://localhost/php/lang/arithmetic.php>
  - [view source](#)
- <http://localhost/php/lang/math.php>
  - [view source](#)
- <http://localhost/php/lang/arrays1.php>
  - [view source](#)



# Example scripts (3)

---

- <http://localhost/php/lang/arrays2.php>
  - [view source](#)
- <http://localhost/php/lang/arrays3.php>
  - [view source](#)
- <http://localhost/php/lang/arrays4.php>
  - [view source](#)
- <http://localhost/php/lang/sort.php>
  - [view source](#)



# Example scripts (4)

---

- <http://localhost/php/lang/testing.php>
  - [view source](#)
- <http://localhost/php/lang/comparison.php>
  - [view source](#)
- <http://localhost/php/lang/regexp1.php>
  - [view source](#)
- <http://localhost/php/lang/scoping.php>
  - [view source](#)



# Example scripts (5)

---

- <http://localhost/php/lang/loops.php>
  - [view source](#)
- <http://localhost/php/lang/functions1.php>
  - [view source](#)
- <http://localhost/php/lang/trigtable.php>
  - [view source](#)
- <http://localhost/php/lang/write.php>
  - [view source](#)





# Example scripts (6)

---

- <http://localhost/php/lang/read.php>
  - [view source](#)
- <http://localhost/php/lang/append.php>
  - [view source](#)
- [http://localhost/php/lang/counter\\_test.php](http://localhost/php/lang/counter_test.php)
  - [view source \(counter\\_test.php\)](#)
  - [view source \(counter.php\)](#)