

Origins and Uses of PHP

- *Origins*

- Rasmus Lerdorf - 1994

- Developed to track visitors to his Web site



- PHP is an open-source product

- <http://www.php.net/>

- PHP is an acronym for Personal Home Page, or PHP: Hypertext Preprocessor

- PHP is used for form handling, file processing, and database access

Overview of PHP

- PHP is a server-side scripting language whose scripts are embedded in HTML documents 
- **.php**, .php3, .phtml
- Similar to JavaScript, but on the server side
- PHP is an alternative to
 - CGI, Active Server Pages (ASP), JSP, etc
- The PHP processor has two modes of operation: 
 - copy mode
 - : copy HTML and client-side scripts to output file
 - interpret mode
 - : interpret PHP script and send the result to output
- PHP syntax is similar to JavaScript and Perl
- PHP is dynamically typed

General Syntactic Characteristics

- PHP code can be specified in HTML document internally or externally:

Internally: `<?php ... ?>`

Externally: `include 'filename'`

- Including files saves a lot of work (standard header, footer, or menu file for all web pages)
- File can have both PHP and HTML
- Copied into the document where the call appears
- If the file has PHP, the PHP must be in `<?php .. ?>`, even if the include is already in `<?php .. ?>`
- All variable names begin with \$
- Variable names are case sensitive
 - neither reserved words nor function names are
- A named constant; `define ()` takes two arguments, the name and value of the constant
`define("VALUE", 5);`
- Comments - three different kinds (Java and Perl)
 - single line: `//` , `#`
 - multiple line: `/* ... */`
- Statements are terminated with *semicolons*
- Compound statements are formed with *braces*



Primitives, Operations, Expressions

- *Eight primitive types:*

- Four scalar types: boolean, integer, double, and string
- Two compound types: array and object
- Two special types:
 - resource (a reference to an external resource such as opened files & database connections)
 - NULL

- *Variables*

- There are no type declarations
- Unassigned (unbound) variable has the value, **NULL**
 - The **unset** function sets a variable to **NULL**
 - The **IsSet** function is used to determine whether a variable is **NULL**
 - **IsSet (\$fruit)** returns **TRUE** if it has non-NULL

- **Integer & double** are typical
- correspond to **long** & **double** types of C
- **Strings**
 - Single character: a string of length one
 - String literals use single or double quotes
 - **Single-quoted string literals** 
 - Variables are NOT interpolated
 - Escape sequences are NOT recognized
 - **Double-quoted string literals** 
 - Embedded variables ARE interpolated
 - If there is a variable name in a double-quoted string but you don't want it interpolated, it must be backslashed
 - Embedded escape sequences ARE recognized
 - For both single- and double-quoted literal strings, embedded delimiters must be backslashed

- Boolean - values are **true** and **false** (case insensitive)
- 0 and "" and "0" are **false**; others are **true**
- Arithmetic Operators and Expressions
 - If the result of integer division is not an integer, a double is returned
 - Any integer operation that results in integer overflow produces a double
 - The modulus operator (%) coerces its operands to integer

TABLE 12.2 SOME USEFUL PREDEFINED FUNCTIONS

Function	Parameter Type	Returns
floor	Double	Largest integer less than or equal to the parameter
ceil	Double	Smallest integer greater than or equal to the parameter
round	Double	Nearest integer
srand	Integer	Initializes a random number generator with the parameter
rand	Two numbers	A pseudorandom number greater than the first parameter and smaller than the second
abs	Integer or double	Absolute value of the parameter
min	One or more numbers	Smallest
max	One or more numbers	Largest

- String Operations and Functions

- The only operator is period (.), for catenation
- Indexing : braces
 - \$str = "apple", `$str{3}` is the fourth character

12.3 SOME COMMONLY USED STRING FUNCTIONS

Function	Parameter Type	Returns
strlen	A string	The number of characters in the string
strcmp	Two strings	0 if the two strings are identical, a negative number if the first string belongs before the second (in the ASCII sequence), or a positive number if the second string belongs before the first
strpos	Two strings	The character position in the first string of the first character of the second string, if it is there; false if it is not there
substr	A string and an integer	The substring of the string parameter, starting from the position indicated by the second parameter; if a third parameter is given (an integer), it specifies the length of the returned substring
chop	A string	The parameter with all whitespace characters removed from its end
trim	A string	The parameter with all whitespace characters removed from both ends
ltrim	A string	The parameter with all whitespace characters removed from its beginning
strtolower	A string	The parameter with all uppercase letters converted to lowercase
strtoupper	A string	The parameter with all lowercase letters converted to uppercase

- Scalar Type Conversions

- String to numeric – coercions

- If the string contains a period, e, or E,
 - converted to double; otherwise to integer
- If the string does not begin with a sign or a digit
 - zero is used


- Explicit conversions – casts

- e.g., `(int)$total` or `intval($total)` or `settype($total, "integer")`
- The type of a variable can be determined with `gettype` or `is_type`

`gettype($total)` - return a string or "unknown"

`is_integer($total)` – return Boolean value

Output

- Output from a PHP script: HTML
- Three ways to produce output:
 - **echo, print, printf**
 - **echo** : with or without parentheses
 - if parentheses, only a single string parameter
**echo “whatever
”, “Apples
”; # legal**
**echo(“first
”, “Apples
”) # illeal** 
 - **print** take one parameter
print "Welcome to my site!";
 - **printf** is exactly like its counterpart in C

- A PHP code block can be placed anywhere in the HTML document

- An Example: [today.php](#)

```
<html>
```

```
<head><title> Trivial php example </title>
```

```
</head>
```

```
<body>
```

```
<?php
```

```
print "<b>Welcome to my Web site<br /> <br />";
```

```
print "Today is: <b/>";
```

```
print date("l, F jS");
```

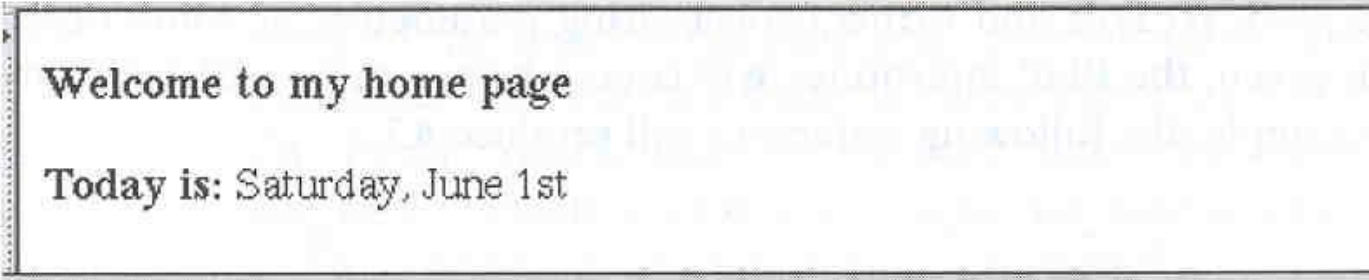
```
# l (day of the week), F(month), j(day)S(suffix for the day)
```

```
print "<br />";
```

```
?>
```

```
</body>
```

```
</html>
```



Welcome to my home page

Today is: Saturday, June 1st

FIGURE 12.1 Display of the output of `today.php`

Control Statements

- Control Expressions

- Relational operators - same as JavaScript, (including `===` [same value AND same type] and `!==`)

- Boolean operators - same as Perl (two sets, `&&` and `and`, etc.)

- Selection statements

- `if`, `if-else`, `elseif`

- `switch`

- The switch expression type must be integer, double, or string

- `while`


- `do-while`

- `for`

- `foreach` - discussed later

- **break** - in any **for**, **foreach**, **while**, **do-while**, or **switch**
 - **break *n***: break out of *n* enclosing blocks

```
<?php
$num = 0;
$file = fopen("input.txt", "r");
while (!feof($file)) {
    $line = fgets($file, 80);
    switch ($line{0}) {
        // Arbitrary code here--the details aren't important
        case 'i': // increment
            echo ++$num. " ";
            break;
        case 'f': // find factors
            for ($factor = 1; $factor <= $num; ++$factor) {
                if ($num % $factor == 0) {
                    echo "[".$factor."] ";
                    // stop processing file if 7 is a factor
                    if ($factor == 7)
                        break 3; // break out of while loop
                }
            }
        }
    }
}
fclose($file);
?>
```

- **continue** - in any loop
 - continue *n***: tells how many levels of enclosing loops it should skip 

- Alternative compound delimiters – more readability
- **endif, endswitch, endfor, endwhile**

```
if(...):  
    ...  
endif;
```

- HTML can be *intermingled* with PHP script

```
<?php  
$a = 7;  
$b = 7;  
if ($a == $b) {  
    $a = 3 * $a;  
?>
```


 At this point, \$a and \$b are equal

So, we change \$a to three times \$a

```
<?php  
}   
?>
```

```

<!DOCTYPE html PUBLIC "-//w3c//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<!--    powers.php
        An example to illustrate loops and arithmetic
    -->

<html>
<head> <title> powers.php </title>
</head>
<body>
<table border = "border">
    <caption> Powers table </caption>
    <tr>
        <th> Number </th>
        <th> Square Root </th>
        <th> Square </th>
        <th> Cube </th>
        <th> Quad </th>
    </tr>
    <?php
        for ($number = 1; $number <=10; $number++) {
            $root = sqrt($number);
            $square = pow($number, 2);
            $cube = pow($number, 3);
            $quad = pow($number, 4);
            print("<tr align = 'center'> <td> $number </td>");
            print("<td> $root </td> <td> $square </td>");
            print("<td> $cube </td> <td> $quad </td> </tr>");
        }
    ?>
</table>
</body>
</html>

```



Figure 12.2 displays the output of powers.php.

Powers table

Number	Square Root	Square	Cube	Quad
1	1	1	1	1
2	1.4142135623731	4	8	16
3	1.7320508075689	9	27	81
4	2	16	64	256
5	2.2360679774998	25	125	625
6	2.4494897427832	36	216	1296
7	2.6457513110646	49	343	2401
8	2.8284271247462	64	512	4096
9	3	81	729	6561
10	3.1622776601684	100	1000	10000

FIGURE 12.2 The output of powers.php

Arrays

- **Not like JavaScript**
- **PHP array is**
 - array of typical language + hash of Perl
 - element: key & value 
 - keys can be numbers (to get a traditional array) or strings (to get a hash) 

- Array creation: two ways

1. Assignment operation

`$list[0] = 17;`

- If a key is omitted and there have been integer keys, the default key will be the largest current key + 1

`$list[1] = "Today";`

`$list[] = 42; # key will be 2`

- If a key is omitted and there have been no integer keys, 0 is the default key

- Array creation: two ways (continued)

2. Use the `array()` construct

- key => value pairs or values as parameters

```
$list = array(1, 3, 5, 7, 9);
```

```
# same as $list = array(0=>1, 1=>3, 2=>5, 3=>7, 4=>9);
```

- keys: non-negative integer or string literals
- values can be anything

```
e.g., $list = array(0 => "apples",  
                  1 => "oranges",  
                  2 => "grapes")
```

- This is a “regular” array of strings

- Arrays can have mixed kinds of elements

- e.g.,

```
$list = array("make" => "Cessna",  
             "model" => "C210",  
             "year" => 1960,  
             3 => "sold");
```



```
$list = array(5, 3 => 7, 5 => 10,  
             "month" => "May");
```

- Accessing array elements – use brackets

```
$list[4] = 7;  
$list["day"] = "Tuesday";  
$list[] = 17;
```

- If an element with the specified key does not exist, it is created
- If the array does not exist, the array is created
- The keys or values can be extracted from an array

```
$highs = array("Mon" => 74, "Tue" => 70,  
               "Wed" => 67, "Thu" => 62,  
               "Fri" => 65);  
$days = array_keys($highs);  
$temps = array_values($highs);
```



- Dealing with Arrays

- An array can be deleted with **unset**

```
unset($list);  
unset($list[4]); # No index 4 element now
```

- Dealing with Arrays (continued)

- `is_array($list)` returns `true` if `$list` is an array
- `in_array` — checks if a value exists in an array
e.g., `in_array(17, $list)` returns `true` if 17 is an element of `$list`

- Conversion between strings and arrays

- `explode(" ", $str)` explodes a string into substrings and returns them in array

```
$str = "April in Paris, Texas is nice";
```

```
$words = explode(" ", $str)
```

- `implode(" ", $list)`

```
$words = array("April", "in", "Paris");
```

```
$str = implode(" ", $words)
```

- creates a string of the elements from `$words`, separated by a space

- Sequential access to array elements

- current and next



- current pointer is initialized to reference the first element of the array at the time the array is created

```
$colors = array("blue", "red", "green", "yellow");  
$color = current($colors);  
print("$color <br />"); # blue  
while ($color = next($colors))  
    print ("$color <br />");
```

- This does not always work – for example, if the value in the array happens to be **FALSE**

- Alternative: **each**, instead of **next**

```
while ($element = each($colors)) {   
    $name = $element["key"];  
    $color = $element["value"];  
    print ("$color <br />");  
}
```

- The **prev** function moves **current** backwards
 - **FALSE** if there are no more elements
- **reset**: set the **current** pointer to the first element
- **array_push** and **array_pop**
 - Used to implement stacks in arrays
 - Push one or more elements onto the end of array
 - Pop the element off the end of array
- **array_unshift** and **array_shift**
 - Prepend passed elements to the front of the *array*
 - Shift the first value of the *array* off and returns it
 - All numerical keys will be modified to start counting from zero while literal keys won't be touched

- **foreach** (array_name **as** scalar_variable) { ... } 

```
foreach ($colors as $color) {  
    print "Is $color your favorite?<br /> ";  
}
```

Is blue your favorite color?

Is red your favorite color?

Is green your favorite color?

Is yellow your favorite color?

- **foreach** can iterate through both keys and values:
- **foreach** (array_name **as** key=>value) { ... }

```
$lows = array("Mon"=>23, "Tue" =>18, "Wed"=>27);  
foreach ($lows as $day=>$temp) {  
print "The low temperature on $day was $temp<br />";  
}
```


- Example: each()

```
<?php
$foo = array("bob", "fred", "jussi", "jouni", "egon", "marliese");
$bar = each($foo);
print_r($bar);
?>
```

\$bar now contains the following key/value pairs:

```
Array
(
    [1] => bob
    [value] => bob
    [0] => 0
    [key] => 0
)
```

- Example: array_push()

```
<?php
$stack = array("orange", "banana");
array_push($stack, "apple", "raspberry");
print_r($stack);
?>
```

The above example will output:

```
Array
(
    [0] => orange
    [1] => banana
    [2] => apple
    [3] => raspberry
)
```

- Example: array_unshift()

```
<?php
$queue = array("orange", "banana");
array_unshift($queue, "apple", "raspberry");
print_r($queue);
?>
```

The above example will output:

```
Array
(
    [0] => apple
    [1] => raspberry
    [2] => orange
    [3] => banana
)
```

- Example: array_shift()

```
<?php
$stack = array("orange", "banana", "apple", "raspberry");
$fruit = array_shift($stack);
print_r($stack);
?>
```

The above example will output:

```
Array
(
    [0] => banana
    [1] => apple
    [2] => raspberry
)
```

- **print_r ()**

- prints human-readable information about a variable

```
<pre>
<?php
$a = array ('a' => 'apple', 'b' => 'banana', 'c' => array ('x', 'y', 'z'));
print_r ($a);
?>
</pre>
```

- the above example will output:

```
<pre>
Array
(
    [a] => apple
    [b] => banana
    [c] => Array
        (
            [0] => x
            [1] => y
            [2] => z
        )
)
</pre>
```

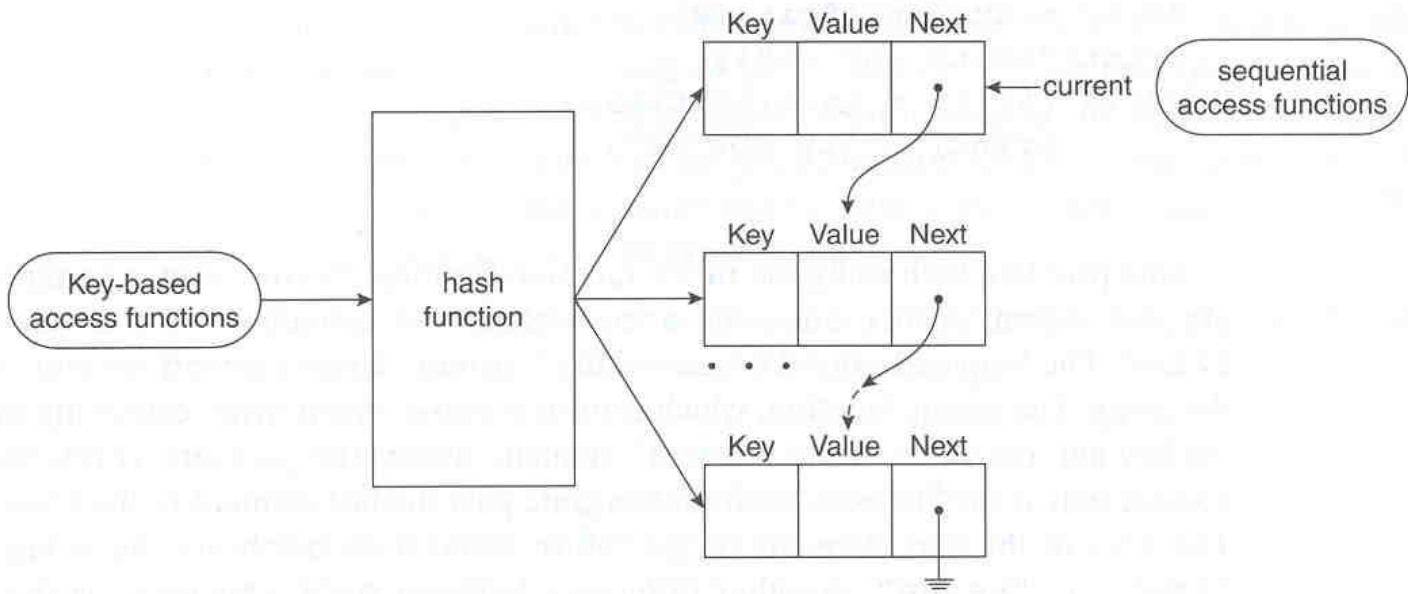



FIGURE 12.3 Logical internal structure of arrays

- sort

- sorts the values, replacing the keys (0,1,2,...)
- strings migrate to the beginning in alphabetical order
numbers follow in ascending order 

e.g., `sort($list);`

- Works for both strings and numbers, even mixed strings and numbers

```
$list = ('h', 100, 'c', 20, 'a');
```

```
sort($list);
```

```
// Produces ('a', 'c', 'h', 20, 100)
```

- After PHP 4, the sort function can take a second parameter, which specifies a particular kind of sort

```
sort($list, SORT_NUMERIC);
```

- sort numeric values without sorting strings,
but string values are moved to the beginning

- **rsort**: sort the values of an array in reverse order

- **asort**

- keep the original key/value relationships
 - *intended for hashes* (associative array)

```
$fruits = array("d" => "lemon", "a" => "orange",  
               "b" => "banana", "c" => "apple");  
asort($fruits);  
foreach ($fruits as $key => $val)  
{echo "$key = $val";}
```

➔ c = apple b = banana d = lemon a = orange

- **arsort**: reverse version of asort

- **ksort**: sort the elements of an array by the keys,
maintaining the key/value relationships (mainly for
associative arrays)

```
$list("Fred" => 17,"Mary" => 21,"Bob" => 49,"Jill"=>28);  
ksort($list);  
// $list is now ("Bob" => 49,"Fred" => 17,  
//              "Jill" => 28, "Mary" => 21)
```

- **krsort**

- To sort the elements of an array by the keys
into reverse order

```
<!DOCTYPE html PUBLIC "-//w3c//DTD XHTML 1.0 Strict//EN"
```

```
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

```
<!-- sorting.php - An example to illustrate several of the  
      sorting functions -->
```

```
<html>
```

```
<head> <title> Sorting </title>
```

```
</head>
```

```
<body>
```

```
<?php
```

```
$original = array("Fred" => 31, "Al" => 27, "Gandalf" => "wizzard",  
                  "Betty" => 42, "Frodo" => "hobbit");
```

```
?>
```

```
<h4> Original Array </h4>
```

```
<?php
```

```
foreach ($original as $key => $value)
```

```
    print("[$key] => $value <br />");
```

```
$new = $original;
```

```
sort($new);
```

```
?>
```

```
<h4> Array sorted with sort </h4>
```

```
<?php
```

```
foreach ($new as $key => $value)
```

```
    print("[$key] = $value <br />");
```

```
$new = $original;
```

```
sort($new, SORT_NUMERIC);
```

```
?>
```

```
<h4> Array sorted with sort and SORT_NUMERIC </h4>
```

```
<?php
```

```
foreach ($new as $key => $value)
```

```
    print("[$key] = $value <br />");
```

```
$new = $original;
```

```
rsort($new);
```

```
?>
```

```
<h4> Array sorted with rsort </h4>
```

```
<?php
```

```
foreach ($new as $key => $value)
```

```
    print("[$key] = $value <br />");
```



```
$new = $original;
asort($new);
?>
<h4> Array sorted with asort </h4>
<?php
foreach ($new as $key => $value)
    print("[ $key] = $value <br />");

$new = $original;
arsort($new);
?>
<h4> Array sorted with arsort </h4>
<?php
foreach ($new as $key => $value)
    print("[ $key] = $value <br />");
?>
</body>
</html>
```

Figure 12.4 shows the output of `sorting.php`.

Original Array

```
[Fred] => 31  
[Al] => 27  
[Gandalf] => wizzard  
[Betty] => 42  
[Frodo] => hobbit
```

Array sorted with sort

```
[0] = hobbit  
[1] = wizzard  
[2] = 27  
[3] = 31  
[4] = 42
```

Array sorted with sort and SORT_NUMERIC

```
[0] = wizzard  
[1] = hobbit  
[2] = 27  
[3] = 31  
[4] = 42
```

Array sorted with rsort

```
[0] = 42  
[1] = 31  
[2] = 27  
[3] = wizzard  
[4] = hobbit
```

Array sorted with asort

```
[Frodo] = hobbit  
[Gandalf] = wizzard  
[Al] = 27  
[Fred] = 31  
[Betty] = 42
```

Array sorted with arsort

```
[Betty] = 42  
[Fred] = 31  
[Al] = 27  
[Gandalf] = wizzard  
[Frodo] = hobbit
```

FIGURE 12.4 The output of `sorting.php`

User-Defined Functions

- Syntactic form:

```
function function_name(formal_parameters) {  
    ...  
}
```

- General Characteristics

- Functions need not be defined before they are called (in PHP 3, they must)
- Function overloading is not supported
- If you try to redefine a function, it is an error
- Function definitions can be nested
- Function names are NOT case sensitive
- The **return** function is used to return a value;
If there is no return, there is no returned value

- Parameters

- number of actual parameters,
 - too few actual parameters: unbound variables
 - too many actual parameters: ignored
- default parameter passing method is
 - pass by value (one-way communication)
- To specify pass-by-reference, prepend an ampersand to the formal parameter

```
function addOne(&$param) {  
    $param++;  
}
```

```
$it = 16;  
addOne($it); // $it is now 17
```

- If the function does not specify its parameter to be pass by reference, you can prepend an ampersand to the actual parameter and still get pass-by-reference semantics

```
function subOne($param) { $param--; }  
$it = 16;  
subOne(&$it); // $it is now 15
```

- Scope of Variables

- Default scope of a variable used in a function: local
- In PHP global variables must be declared **global** inside a function if they are going to be used in that function

```
<?php
```

```
$a = 1; /* global scope */
```

```
function test()
```

```
{ echo $a; /* reference to local scope variable */
```

```
test();
```

```
?>
```

➔ This script will not produce any output

(different from C in that global variables in C are automatically available to functions unless specifically overridden by a local definition)

- To access a nonlocal variable, it must be declared to be **global**, as in

```
$a = 1; /* global scope */
```

```
function test()
```

```
{ global $a; 
```

```
...
```

- Lifetime of Variables

- Normally, the lifetime of a variable in a function is from its first appearance to the end of the function's execution
- To support history sensitivity, static local variable
 - end when the script execution ends
(when the browser leaves the document)

static sum = 0; # sum is a static variable

Pattern Matching

- PHP has two kinds of string pattern matching:
- POSIX regular expression: `ereg`



Warning

This function has been **DEPRECATED** as of PHP 5.3.0 and **REMOVED** as of PHP 6.0.0. Relying on this feature is highly discouraged.

- Perl regular expression: `preg_match`

`preg_match(regex, str [,array])`

- The optional array is where to put the matches
- Note: `preg_match()` uses a Perl-compatible regular expression syntax, is often faster than `ereg()`

```
if (preg_match("/^PHP/", $str)
```

```
    print ("\"$str begins with PHP <br />");
```

```
else
```

```
    print ("\"$str does not begins with PHP <br />");
```



Tip

Do not use `preg_match()` if you only want to check if one string is contained in another string. Use `strpos()` or `strstr()` instead as they will be faster.

- Metacharacters:

`\ | () [] { } ^ $ * + ? .`

- Metacharacters can match themselves if they are backslashed

- Meta-characters outside square brackets

`\` escape character

`^` assert start of line

`$` assert end of line

`.` match any character except newline

`/a.b/` matches "aab", "abb", "acb", ...

`[` start *character class*

`]` end *character class*

`/` start of alternative branch

`(` start subpattern

`)` end subpattern

`?` extends the meaning of `(`, also 0 or 1 quantifier

`*` 0 or more quantifier

`+` 1 or more quantifier

`{` start min/max quantifier

`}` end min/max quantifier

Meta-characters inside square brackets (character class)

`\` escape character

`^` negate the class, but only if the first character

`-` indicates character range

- A character class is a string in brackets

`[abc]` means a | b | c

- A dash can be used to specify a range of characters

`[A-Za-z]`

- If a character class begins with a circumflex, it means the opposite

`[^A-Z]` matches any character except an uppercase letter

- Predefined character classes:

<i>Name</i>	<i>Equivalent Pattern</i>	<i>Matches</i>
-------------	---------------------------	----------------

<code>\d</code>	<code>[0-9]</code>	a digit
<code>\D</code>	<code>[^0-9]</code>	not a digit
<code>\w</code>	<code>[A-Za-z_0-9]</code>	a word character
<code>\W</code>	<code>[^A-Za-z_0-9]</code>	not a word character
<code>\s</code>	<code>[\r\t\n\f]</code>	a whitespace character
<code>\S</code>	<code>[^ \r\t\n\f]</code>	not a whitespace character

- Pattern Quantifiers (Repetition)

- pattern{n} means repeat the pattern n times

`/a{5}bc{5}/`

- pattern* means repeat the pattern zero or more times

`/a*bc*/`

- pattern+ means repeat the pattern 1 or more times

- pattern? means zero or one match

`/\d*b?c+/`

- Subpatterns are delimited by parentheses

- localizes a set of alternatives

`cat(aract|erpillar|)` matches

"cat", "cataract", or "caterpillar"

Without the parentheses, matches

"cataract", "erpillar" or the empty string

- if the string "the red king" is matched against the pattern `((red|white) (king|queen))`, the captured substrings are

"red king", "red", and "king"

- If an opening parenthesis is followed by "?:", the subpattern does not do any capturing if "the white queen" is matched against

`((?:red|white) (king|queen))`, the captured substrings are "white queen" and "queen"

- Example: find the string of text "php"

```
<?php
// The "i" after the pattern delimiter indicates a case-insensitive search
if (preg_match("/php/i", "PHP is the web scripting language of choice.")) {
    echo "A match was found.";
} else {
    echo "A match was not found.";
}
?>
```

- Example: find the word "web"

```
<?php
/* The \b in the pattern indicates a word boundary, so only the distinct
 * word "web" is matched, and not a word partial like "webbing" or "cobweb" */
if (preg_match("/\bweb\b/i", "PHP is the web scripting language of choice.")) {
    echo "A match was found.";
} else {
    echo "A match was not found.";
}

if (preg_match("/\bweb\b/i", "PHP is the website scripting language of choice.")) {
    echo "A match was found.";
} else {
    echo "A match was not found.";
}
?>
```

- Example: using named subpattern

```
<?php
$str = 'foobar: 2008';

// Works in PHP 5.2.2 and later.
preg_match('/(?<name>\w+): (?<digit>\d+)/', $str, $matches);

// Before PHP 5.2.2, use this:
// preg_match('/(?P<name>\w+): (?P<digit>\d+)/', $str, $matches)

print_r($matches);

?>
```

- The above example will output:

```
Array
(
    [0] => foobar: 2008
    [name] => foobar
    [1] => foobar
    [digit] => 2008
    [2] => 2008
)
```

Form Handling

- Simpler than either CGI or servlets
- PHP superglobals
- **\$_GET** and **\$_POST** are used to collect form-data
 - example

```
<!DOCTYPE HTML>
<html>
<body>
<form action="welcome.php" method="post">
  Name: <input type="text" name="name"><br>
  E-
  mail: <input type="text" name="email"><br>
  <input type="submit">
</form>
</body>
</html>
```
 - welcome.php

```
<html>
<body>
Welcome <?php echo $_POST["name"]; ?><br>
Your email address is: <?php echo
$_POST["email"]; ?>
</body>
</html>
```

- **PHP superglobals**
 - mean that they are always accessible, regardless of scope
- **Provide information about a script's environment**
 - Type of Web browser
 - Type of server
 - Details of HTTP connection
- **E.g., PHP stores the client's environment variables in the `$_ENV` array**

Variable name	Description
<code>\$_SERVER</code>	Data about the currently running server.
<code>\$_ENV</code>	Data about the client's environment.
<code>\$_GET</code>	Data posted to the server by the get method.
<code>\$_POST</code>	Data posted to the server by the post method.
<code>\$_COOKIE</code>	Data contained in cookies on the client's computer.
<code>\$GLOBALS</code>	Array containing all global variables.
Some useful superglobals	

Superglobals

`$_GET`: Variables provided to the script via HTTP GET.

Analogous to the old `$HTTP_GET_VARS` array (still available, but deprecated).

`$_POST` : Variables provided to the script via HTTP POST. Analogous to the old `$HTTP_POST_VARS` array (still available, but deprecated).

`$_COOKIE` : Variables provided to the script via HTTP cookies. Analogous to the old `$HTTP_COOKIE_VARS` array (still available, but deprecated)

- e.g., **`extract($_POST);`** //Creates variables corresponding to each key-value pair in array

`$_POST`

- Even though both the superglobal and `HTTP_*_VARS` can exist at the same time, they are not identical, so modifying one will not change the other

- Examples

// A common function

```
function test_input($data) {
```

```
    $data = trim($data);
```

//trim(): strips unnecessary characters (extra space, tab, newline) from the user input data

```
    $data = stripslashes($data);
```

//stripslashes(): remove backslashes

```
    $data = htmlspecialchars($data);
```

//htmlspecialchars(): converts special characters to HTML entities

```
    return $data;
```

```
}
```

- check if the name field only contains letters/whitespace

```
$name = test_input($_POST["name"]);
```

```
if (!preg_match("/^[a-zA-Z ]*$/", $name)) {
```

```
    $nameErr = "Only letters and white space allowed";
```

```
}
```


- check whether an email address is well-formed

```
$email = test_input($_POST["email"]);  
if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {  
    $emailErr = "Invalid email format";  
}
```

- check if a URL address syntax is valid (this regular expression also allows dashes in the URL)

```
$website = test_input($_POST["website"]);  
if (!preg_match("/^b(?:(:https?|ftp):VV|www\.)[-a-z0-9+&@#V%?~=~_]|!:,.;)*[-a-z0-9+&@#V%?~=~_]/i",$website)) {  
    $websiteErr = "Invalid URL";  
}
```

Files

- PHP can:
 - Deal with any files on the server
 - Deal with any files on the Internet, using either http or ftp
- PHP associates a variable with a file, called the file variable

`$file_var = fopen(filename, use_indicator)`

- has a file pointer (where to read or write)
- Use indicators:
 - `r` read only, from the beginning
 - `r+` read and write, from the beginning
 - `w` write only, from the beginning
(also creates the file, if necessary)
 - `w+` read and write, from the beginning
(also creates the file, if necessary)
 - `a` write only, at the end, if it exists
(creates the file, if necessary)
 - `a+` read and write, read at the beginning, write at the end
- Because `fopen` could fail, use it with `die`
`$file_var = fopen("test.dat", "r") or
die ("Error – test.dat cannot be opened");`

- Use **file_exists(filename)** to determine whether file exists before trying to open it
- Use **fclose(file_var)** to close a file
- Reading files
 1. Read all or part of the file into a string variable
\$str = fread(\$file_var, # of bytes);
 - To read the whole file, use **filesize(file_name)** as the second parameter
\$str = fread(\$file_var, filesize("test.dat"));
 2. Read array of all the lines of the file
\$file_lines = file(file_name)
ex) **\$file_lines = file("test.dat")**
 - Need not open or close the file
 3. Read a single line from the file
\$line = fgets(file_var, #bytes)
 - Reads characters until eoln, eof, or #bytes characters have been read
\$line = fgets(\$file_var, 100)

- Reading files (continued)

4. Read one character at a time

```
$ch = fgetc(file_var)
```

- **eof** detection using **feof**
 - TRUE for **eof**; FALSE otherwise

```
while(!feof($file_var)) {  
    $ch = fgetc($file_var);  
}
```

- Writing to files

```
$bytes_written = fwrite(file_var, string)
```

- returns the number of bytes it wrote
- Files can be locked (to avoid interference from concurrent accesses) with **flock** (just like Perl)
 - two parameters:
 - **file_variable**
 - **integer** that specifies the needed operation
 - **LOCK_SH**(1): specifies that file can be read by others while the lock is set
 - **LOCK_EX**(2): allows no other access
 - **LOCK_UN**(3): unlock the file

Cookies

- Create a cookie with **setcookie**

setcookie(cookie_name, cookie_value, lifetime)

e.g.,

```
setcookie("voted", "true", time() + 86400);  
/* expire in 1 day */
```

- Cookies must be created before any other HTML is created by the script
- Cookies are obtained in a script the same way form values are gotten
- There could be conflicts between GET, POST, and cookie variables
 - PHP puts all POST form variables in their own array (hash) **\$_POST**
 - **\$HTTP_POST_VARS**: deprecated
 - Ditto for GET form variables (**\$_GET**)
 - all cookies in **\$_COOKIE**

XML Parsing

- Two major types of XML parsers:
 - Tree-based parsers
 - holds the entire document in memory (DOM)
 - A better option for smaller XML documents, but not for large document
 - Example of tree-based parsers:
 - SimpleXML
 - DOM
 - Event-based parsers
 - Do not hold the entire document in memory, instead, read in one node at a time and allow you to interact with in real time
 - Parses faster and consumes less memory
 - Example of event-based parsers:
 - XMLReader
 - XML Expat Parser

- SimpleXML Parser

- Part of the PHP core. No installation is required

- Example: Reading an xml file (note.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

- Use the [simplexml_load_file\(\)](#) function to read XML data from a file

```
<?php
$xml=simplexml_load_file("note.xml") or die("Error
: Cannot create object");
print_r($xml);
?>
```

- Get the node values

```
<?php
$xml=simplexml_load_file("note.xml") or die("Error
: Cannot create object");
echo $xml->to . "<br>";
echo $xml->from . "<br>";
echo $xml->heading . "<br>";
echo $xml->body;
?>
```