

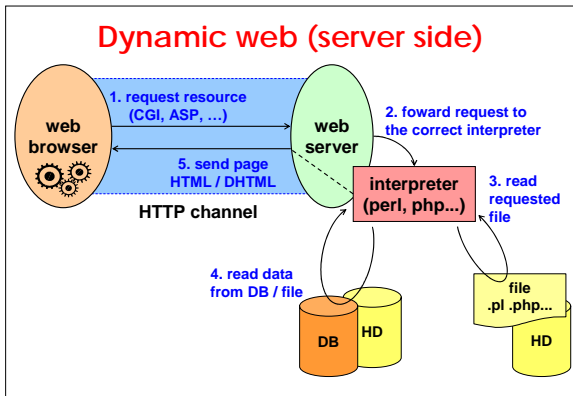
Server-side programming in PHP

Diana Berbecaru
 < diana.berbecaru@polito.it >

Politecnico di Torino
 Dip. Automatica e Informatica

Functionality of a web application

- **generally the web applications:**
 - provide an interface to the user to query information from the web server (query interface), if necessary made "active" throughout client-side processing
 - send the data inserted by the user to the web server
 - execute the processing of data on the server site (server-side processing), if necessary interacting with the information provided by a database
 - transmit the results of the query operation to the client in the form of an HTML page



Introduction to PHP

- **what is PHP?**
 - PHP stands for "PHP Hypertext Preprocessor"
 - an embedded scripting language for HTML like ASP or JSP
 - created by Rasmus Lerdorf in 1995 as a simple preprocessor of HTML for tracking access to his resume
 - originally a set of Perl scripts known as the "Personal Home Page" tools
 - rewritten in C with database functionality
 - further improved and released in 1997 (v 2.0), 1998 (v 3.0), 2000 (v 4.0), 2004 (v 5.0.0)

PHP history

- initially, PHP comprised a simple parser and a library of C functions
- the parser would scan an HTML file looking for instances of a new non-standard tag and replace the contents of these tag instances with the result of executing some functions in C
- as a result, PHP syntax looks like C but combines also elements from Perl and C++
- PHP is available on many servers today, in Windows and all types of Unix environments
- <http://www.php.net>

Why use PHP?

- **portability: available for many platforms**
 - hardware (Intel, Sparc, Mac, etc....)
 - operating system (Linux, Unix, Windows, etc...)
 - web server (Apache, IIS, iPlanet, etc...)
- **PHP interpreter is Open Source**
 - a lot of tools & support from developers, libraries
 - large user community
- **quite easy to understand**
 - similar to C
 - simpler syntax and data types
- **able to interact with various database management systems (MySQL, PostgreSQL, Oracle, ...)**

A simple PHP file

- PHP code is inserted in the HTML code
- PHP code is executed on the server and the result (HTML and script output) is sent to the browser

```

<html>
<head><title>Hello world !</title></head>
<body>
<?php
    // This is PHP code
    echo "<h1> Hello World !</h1>";
?>
</body>
</html>
    
```

helloworld.php

PHP Language Basics: Script tags

- PHP source code can be placed in any point in the HTML page
- all PHP code is contained in one of the several script tags:
 - standard tag:


```
<?php
// some PHP code
?>
```
 - short tag (not advised):


```
<?
// some PHP code here
?>
```

PHP Language Basics: Script tags

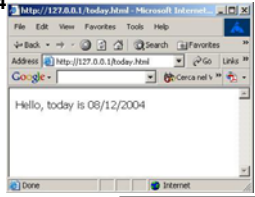
- script tags:
 - standard HTML tag for inclusion of scripting code:


```
<script language="PHP">
// some PHP code here
</script>
```
 - ASP-style tags:
 - introduced in v 3.0; may be removed in the future
 - ```
<%
// some PHP code here
%>
```

### Today is ...

- view the message:
  - "Hello, today is ..."
- statically:
 

```
<html><body>
Hello, today is 08/12/2004
</body></html>
```
- and tomorrow?:

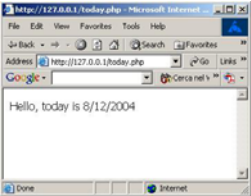


today.html

### Today is ...

- dynamically:
 

```
<html><body>Hello, today is
<?php
 // format GG/MM/AAAA
 $today = date("j/m/Y");
 echo $today;
?>
</body></html>
```
- is updated in real time



today.php

### Today is ...

- like Perl, there is more than one way to do it
  - ```
<html><body>Hello, today is
<?php
    $today = date("j/m/Y");
    echo $today;
?>
</body></html>
```

j= day of the month
 m= numeric month
 Y= year in 4 digits
 - ```
<html><body>
<?php
 $greeting="Hello, today is ";
 printf("%s", $greeting);
 echo date("j/m/Y");
?></body></html>
```

today3.php

### Today is ...

- like Perl, there is more than one way to do it (cont.)
  - `<html><body>`
  - `<?php`
  - `// format GG/MM/AAAA`
  - `$hello="Hello, ";`
  - `$todaymess="today is ";`
  - `print $hello.$todaymess;`
  - `echo date("j/m/Y");`
  - `?>`
  - `</body></html>`

today4.php

### What did we use?

- to manage dynamic data are used:
  - comments (`// format...` )
  - variables (`$today, $greeting, $hello ...`)
  - operators (`=`)
  - language constructs (`echo`)
  - functions (`date`)

### PHP Language Basics: Comments

- styles:
  - shell-style (`#`)
  - C++ style (`//`)
  - C style (`/* ... */`)

```

<?php
// This is a comment on one line
/* This is a
comment on more than one line */
This is also a comment on one line
?>
```

### PHP Language Basics: Variables

- a variable is:
  - "a symbol or a name that represents a value"
  - denoted by a dollar sign (`$`) followed by the variable name
  - case-sensitive: `$some_stuff`, `$Some_stuff`, `$some_stuff`, `$SOME_STUFF` are different
- a variable can represent different types of values
  - integer, real, character
- it is not necessary to declare the variable
  - the name is declared automatically at its first use
  - the data type can change during the execution of the program

### Automatic variables in PHP

- one of the main benefits of PHP is that it provides lots of variables automatically

```

<html>
<head>
<title> Your browser </title>
</head>
<body>
<p> You are using <?php echo
$_ENV["HTTP_USER_AGENT"]; ?> to view this page.
</p>
</body>
</html>
```

automaticvariables.php  
phpinfo.php

### PHP Language Basics: Constants

- define a string or a numeric value
- do not begin with a `$` sign
- examples:
  - `define("COMPANY", "Politecnico di Torino");`
  - `define("PI", 3.14);`
  - `define("NL", "<br>\n");`
- using a constant
  - `print("Company name: ". COMPANY . NL);`

## PHP Language Basics: Data types

- **PHP supports eight primitive data types**
- there are four scalar types:
  - **boolean**
    - ex: \$today = TRUE; \$today = FALSE
  - **integer**
    - ex: \$today = 12
  - **floating-point number**
    - ex: \$today = 3.14
  - **string**
    - ex: \$today = "oggi" (string)
    - ex: \$today = 'a' (character)

## PHP Language Basics: Data types

- there are two structured types:
  - **array**
    - ex1: \$today = array("lab", 3)
  - **object**
    - supports OOP (subject too big to cover here)

```
<?php
class thingAlice
{
 function say_hello()
 {echo "Hello, World!";}
}
$thing1 = new thingAlice;
$thing1->say_hello();
?>
```

## PHP Language Basics: Data types

- there are two special data types:
  - **resource**
    - used for maintaining links to external resources, e.g. databases
  - **NULL**
    - contains only one value: NULL
    - returned when some expression has no value
- the programmer does not specify the type of variable
  - a variable's type is determined from the context of use

## Type conversion in PHP

```
$a = 10 ; // $a is an integer variable and has value 10
$b = $a * 1.52 ; // $b is a float and has value 15.2
$c = "5+$b" ; // $c is a string and has value "5+15"

$d = 10.8 ; // $d is a float and has value 10.8
$d = (int)$d ; // now $d is an integer and has value 10
$e = (int)$c ; // now $e is an integer and has the value 5
```

cast.php

## More on arrays

- a structure which maps keys to values
- **format of usage**
  - array([key =>] value, ...)
- a key is either a string or a non-negative value
- a value can be anything
- **other examples:**
  - ex2: \$mix = array("Luca", TRUE, 14);
    - echo \$mix[0];
    - // is the string "Luca"

## More on arrays

- **other examples:**
  - ex3: \$mixmix = array("Luca", array(TRUE, 14));
    - echo \$mixmix[1][1];
    - // is the integer value 14
    - Note: numbering starts with 0 if key is omitted!
  - ex4: \$capital = array('Italy'=>'Rome', 'France'=>'Paris');
    - echo \$capital['Italy'];
    - // is the string "Rome"

### Instruction control flow

- **constructs for the control flow:**
  - if, if/elseif
  - switch/case
  - while, do/while
  - for
  - foreach
  - exit and return
  - include and require statements for code reuse
- **allow the conditional execution of parts of the program**
- **allow the iterative execution of parts of the program**

### Conditions

- **a condition is**
  - an expression that generates a boolean value (true or false)
- **are equivalent to the boolean value TRUE**
  - integer or float numbers different from 0 and 0.0
  - non-empty strings (the string "" is empty)
  - array with at least one element
- **are equivalent to the boolean value FALSE**
  - integer or float number 0 and 0.0
  - empty string ("") and the string "0"
  - array with zero elements
  - value NULL

### if ... else

```

if ($a > $b)
 {print "a is bigger than b";}
else {print "a is NOT bigger than b";}

```

```

if (condition)
 { action1; }
else
 { action2; }

```

### while, do/while

- **just like their counterparts in C**
- **repeat action1 until condition is TRUE**
- **in general action1 modifies the parameters involved in condition**

```

$i = 1;
while ($i <= 10)
 {echo $i++;}

```

```

do
 { action1; }
while (condition)

```

```

while (condition)
 { action1; }

```

### for

- **just like their counterparts in C**
- **init**
  - expression executed only once (initial assigning of counter)
- **condition**
  - if TRUE execute action1
- **inc**
  - expression executed at the end of each iteration

```

for ($i=1; $i <=10; $i++)
 {print $i;}

```

```

for (init; condition; inc)
 { action1; }

```

### foreach

- **allows to iterate the elements of an array**

```

$addresses=array('stud@polito.it', 'spam@polito.it');
foreach ($addresses as $valuecurrent){
 echo "Process $valuecurrent\n";}

```

```

$country=array('name'=>'Italy', 'capital'=>'Rome');
foreach ($country as $k=>$v){
 echo "Italy's $k is $v\n";}

```

```

foreach ($array as $key=>$value)
{
 //...
 foreach ($array as $current)
 {
 //...
 }
}

```

foreachexample1.php  
foreachexample2.php

### Operators

- **all C operators are valid**
  - arithmetic (+, -, \*, /, %)
  - assign (=, +=, -=, /=, \*=, %=, ^=, &=, |=, .=)
  - increment and decrement (++, --)
  - comparison (==, !=, <>, >, <, <=, >=)
    - new:
      - identical (===), when both operands have the same value and type
      - non identical (!==), when both operands doesn't have the same value or are of the same type
  - logical (!, &&, ||, xor)

### Operators

- bitwise (~, &, |, ^, <<, >>)
- concatenation of strings (.)
- type conversion: (int), (float), (string), (bool), (array), (object)
- other:
  - error suppression (@)
    - \$value=@(2/0);
  - execution (...)`
    - \$listing=`ls -al /home`;
    - echo \$listing;
  - ternary (?)
    - <? echo '<td>'.(\$active ? 'yes':'no').'</td>' ?>

### Functions

- **definition**
  - create the new function
- **call**
  - execute the function

```

/*definition*/
function sum($a,$b)
{
 $sum = ($a+$b);
 return $sum;
}

//call
$c = sum(2,3);

```

### Functions and variables

- **inside a function we can declare new variables**
- **the visibility (or scope) of the variable is**
  - the section of the program where the variable can be used and modified
  - the entire program
  - function
  - {block of instructions}

### Visibility of variables

- **in PHP, a variable declared inside a function is visible:**
  - in the function where it is declared
  - it is NOT visible outside the function or in other functions

```

function account()
{
 $pieces = 10;
 return $pieces;
}

echo $pieces;
//$pieces is an empty string !!

```

### Visibility of variables

- **in PHP, a variable declared in the main program is visible:**
  - in the main program
  - it is NOT visible inside the functions

```

//global variables
$pieces=10;

function account() {
 return $pieces;
}

echo "There are ".account()." pieces";
// prints "There are pieces"

```

### Visibility of variables

<pre>//global variable \$pieces=10; function account() {     global \$pieces;     return \$pieces; } echo "There are ".account()." pieces"; // prints "There are 10 //pieces"</pre>	<pre>//global variable \$pieces=10; function count_pieces() {     return \$GLOBALS["pieces"]; } echo "There are ".count_pieces()." pieces"; // prints "There are 10 //pieces"</pre>
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

- putting `global` at the beginning of the program does not make the variable visible inside the function – see `global.php`

### Interactive web pages

- executing a PHP page produces the same result
- not very useful
- pass data from client to the server side script
  - use HTML forms
  - the Form action is to execute the PHP script
  - script needs to know the form's structure
- pass output from script to client
  - data in the response is wrapped up in the HTML

### HTML Form - example

- text box, with name "colour"
- redirection to form.php after submit

```
<form action = "form.php" method=GET>
Your favourite colour:
<input type=text name="colour" value="">
<input type=submit name="confirm">
</form>
```

### Form – server side

- file form.php:
  - view the data inserted by the user:
 

```
<input type=text name="colour" value="">
```
  - uses the `$_GET` global array to access data from HTTP request

```
<?php
if (!empty($_GET["colour"])) {
echo "Your favourite colour is " . $_GET["colour"];
}
?>
```

### GET and POST

- method used for a particular form is specified in the form tag
  - GET
    - retrieve data from a web server (html, image)
  - POST
    - post information to the server (form data)
- PHP creates six global arrays to access data from HTTP requests/responses
  - `$_GET`, `$_POST`, `$_COOKIE`, `$_FILES`, `$_SERVER`, `$_ENV`

### GET: How parameters are passed in

- through the passage of parameters in the url
  - ?
    - separates url from parameters
  - &
    - separates a parameter from another
  - variable\_name = value
    - assigns "value" to the variable "\$variable\_name"

```
http://myhost/myscript.php?today=oggi&utente=marco
```

### Form and PHP – server site

- **\$\_GET['name']** or **\$HTTP\_GET\_VARS[ ]**
  - for forms which have been transmitted using the HTTP GET request
  - returns the value of the form element 'name'
- **\$\_POST['name']** or **\$HTTP\_POST\_VARS[ ]**
  - for forms which have been transmitted using the HTTP POST request
  - returns the value of the form element 'name'

formexample2.php

formexample.html

formexample.php

formexample.html