Introduction to PHP

Fulvio Corno, Laura Farinetti, Dario Bonino, Marco Aime

Goals

- Understand server-side architectures based on PHP
- Learn the syntax and the main constructs of the PHP language
- Design simple applications for elaborating Form data
- Understand the interactions between web servers and application servers
- Design 3-tier web applications

Agenda

- PHP - language basics
- PHP - functions
- PHP - forms
- PHP - sessions
- PHP - databases

PHP a (brief) history

- Originally published in 1994, was designed to ease the development of personal web pages
  - Original meaning of the acronym
    - PHP, Personal Home Page
  - Nowadays PHP means
    - PHP: Hypertext Preprocessor
- Specifically designed for the creation of dynamic web pages
- The PHP code is embedded inside XHTML/HTML pages
- The application server interprets the PHP code and provides the result to the web server, which sends it back to the client.

Why PHP? (1/2)

- It is supported by many platforms
  - hardware (Intel, Sparc, Mac, ecc....)
  - Operating system (Linux, Unix, Windows, ecc...)
  - Web servers (Apache, IIS, iPlanet, ecc...)

The PHP code is highly portable

Why PHP? (2/2)

- PHP is open source
  - free
  - Many tools, IDEs, support, users, developers
  - Wide community
- Easy to learn
  - Similar to the C language
  - Simpler syntax and datatypes
  - Easy to extend
- Able to interact with many DBMS (MySQL, PostgreSQL, Oracle,...)

A first example

```
<html>

<head><title>Hello World !</title></head>

<body>

<?php

   // this is simple PHP

   echo "<h1> Hello World !</h1>";

?>

</body>

</html>
```

This is a simple ASCII file with a .php extension

Tag

- PHP code can be positioned everywhere inside the XHTML page
- It must be enclosed by the start and end tags
- Standard format for tags

```
<?php
  // this is a PHP program
  echo "<h1> Hello World!
</h1>";
?>
```

Tag

- Short tags (deprecated)
  - Sometimes used for the sake of comfort
  - Usually disabled

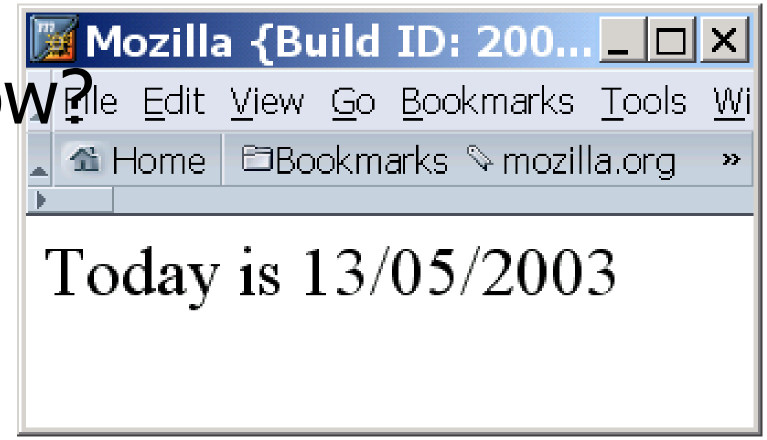
```
<?
  // this a simple PHP progam
  echo "<h1> Hello World!</h1>";
?>
```

Today is … (1/2)

- Show the current date
  - "Today is …"
- Statically:

```
<html><body>
       Today is 13/05/2003
</body></html>
```

- But what happens tomorrow?


Mozilla {Build ID: 200...
File Edit View Go Bookmarks Tools Wi
Home Bookmarks mozilla.org

Today is 13/05/2003

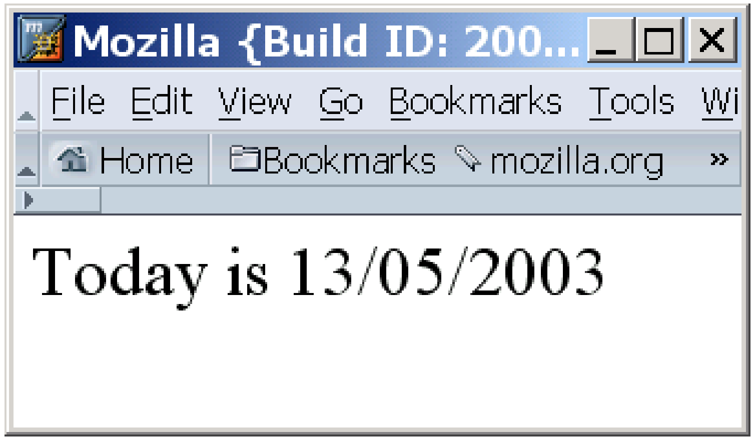
Today is … (2/2)

- Dynamically

```
<html><body>Today is
<?
  // date format GG/MM/AAAA
  $today = date("j/m/Y");
  echo $today;
?>
<body></html>
```

j= day of the month
m= numeric month
Y = year in 4 digits

- Updated in real time
- The date on the server


Mozilla {Build ID: 200...
File Edit View Go Bookmarks Tools Wi
Home Bookmarks mozilla.org

Today is 13/05/2003

What features we used?

- To handle dynamic data we need
  - Comments (`// date format...`)
  - Variables (`$today`)
  - Operators (`=`)
  - Language constructs (`echo`)
  - Functions (`date`)

```
// formato GG/MM/AAAA
$today = date("j/m/Y");
echo  $today;
```

Agenda

- PHP - language basics
- PHP - functions
- PHP - forms
- PHP - sessions
- PHP - databases

Comments

```php
<?php
// this is a single-line comment
/* This is a comment spanning
Multiple lines */
# this is also a single-line comment
?>
```

Variables (1/2)

- A **variable** is
  - "A symbol or a name that represents a value"
  - Always preceded by the $ sign
- A **variable** can represent different value types
  - Integer numbers, real numbers, characters, …
- Variable declaration is not mandatory
  - The variable name is automatically declared at the first usage
  - The variable data type can change during the program execution

Variables (2/2)

- When a program is executed, variables are substituted by real values
  - Therefore the same program can work on different data
- Together with variables there may also be **constants**
  - A constant represents a value that never changes during the program execution

PHP datatypes (1/2)

- Simple datatypes
  - numbers
    - `$today = 12 (integer)`
    - `$today = 3.14 (float)`
  - String or characters
    - `$today = "today"` (string)
    - `$today = 'a'` (character)
  - Boolean – truth value
    - `$today = TRUE; $today = FALSE`

PHP datatypes (2/2)

- Complex datatypes
  - Array
    - `$today = array("lab",3)`
  - Objects

Array

- An array is
  - An ordered data set
  - Every data element stored into an array is called "array element"
- Ex: 2-elements array
  - `$friends = array("Luke", "Louis");`
- Non-homogeneous arrays
  - `$mix = array("Luke", TRUE, 14);`
  - `$mix_mix = array("Luke", array(TRUE, 14));`

How to access array elements

- echo $mix[0];
  - //the “Luke” string
- echo $mixmix[1][1];
  - // the integer 14
- **Hint**: array indexes start from 0!

Assignment

- The assignment operator allows to associate a value to a variable
  - variable = value
    - `$today = “today”;`
  - variable = variable
    - `$today = $yesterday;`
  - variable = expression
    - `$today = $today + 3;`

Expression

- In computer programs an expression is
  - "Whatever combination of legal symbols that leads to a value"
- Consists of
  - At least one operand
    - An operand is a value
  - One or more operators
    - $today = 5 + 3;
    - $today = $yesterday + 1;
    - $today = $yesterday++;

Datatype conversions in PHP

```
$a = 10 ; //$a it is an integer variable with a value of 10

$b = $a * 1.5 ; //$b it is a float variable with a value of 15.0

$c = "5+$b" ; //$c it is a string variable with the value “5+15”

$d = 10.8 ; //$d it is a float variable with value 10.8

$d = (int)$d ; //now $d is an integer variable with value 10
```

Flow control

- PHP constructs for controlling the program flow:
  - if..else
  - while
  - for
- Allow the conditional execution of some program subset
- Allow the iterative execution of some program part
- Evaluate some condition

Conditions

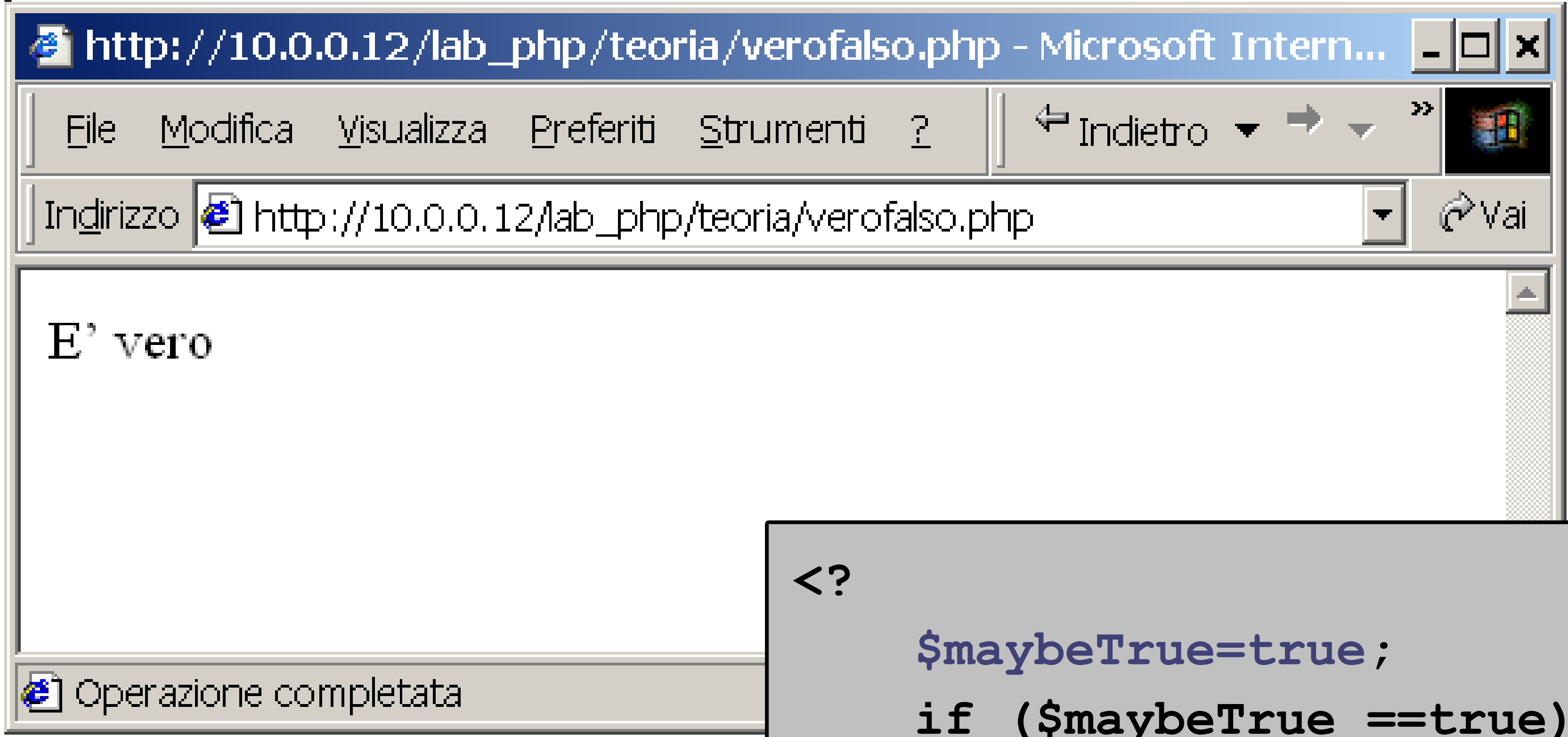
- A condition is
  - An expression yielding to a boolean value, either true or false
- Many values are equivalent to a boolean true:
  - Integers and floats with values different from zero
  - Non-empty strings
  - Non-empty arrays

if … else (1/2)

```
<?
    if ($maybeTrue == true)
            { echo “it's true”;}
    else
             { echo “it's false”;}
?>
```

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

if ... else (2/2)


http://10.0.0.12/lab_php/teoria/verofalso.php - Microsoft Intern...
File Modifica Visualizza Preferiti Strumenti ?
Indietro
Indirizzo http://10.0.0.12/lab_php/teoria/verofalso.php
Vai
Operazione completata

```
E' vero
```

```
<?
    $maybeTrue=true;
    if ($maybeTrue ==true)
      { echo "true";}
    else
      { echo "false";}
?>
```

06/11/08

While

- Repeats *action1* until *condition* is true
- Usually *action1* changes the parameters evaluated in the *condition*

```
while (condition)
  { action1; }
```

```
while ($vero==true)
{
    $vero=false;
    $message="done";
    echo $message;
}
```

How many times will the cycle be executed ?

For (1/2)

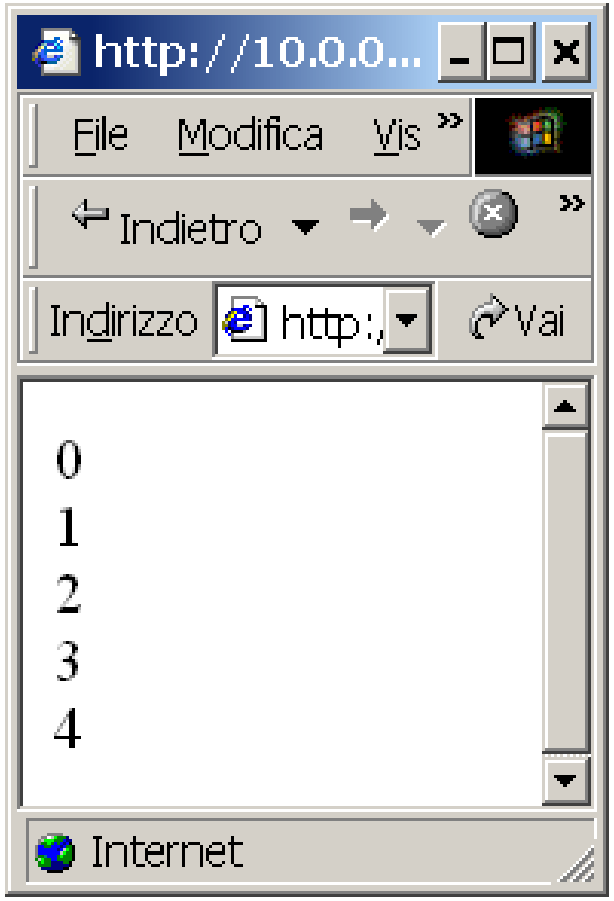
- init
  - Expression that is executed only one time, at the beginning of the cycle. Usually adopted for initializing the cycle counter
- condition
  - If true, *action1* is executed
- inc
  - Expression evaluated at the end of every cycle execution

```
for (init; condition; inc)
  { azione1; }
```

For (2/2)

```
<?
   for ($i=0; $i<5; $i++)
      { echo $i."<br>"; }
?>
```

String concatenation Operator



```
0
1
2
3
4
```

Nesting

- Control structures can be nested

```
if (condition1)
{
   if (condition2)
         { action1; }
   else
         { action2; }
}
```

```
for (init;cond1;inc)
{
  if (condition2)
  {
    for (init3;cond3;inc3)
     {  action1; }
  }
   else
  {  action2; }
}
```

Agenda

- PHP - language basics
- PHP - functions
- PHP - forms
- PHP - sessions
- PHP - databases

Functions (1/2)

- A function is
  - “a portion of code within a larger program, which performs a specific task and can be relatively independent of the remaining code”
- Modern programming languages offer several standard functions grouped in libraries
  - Math libraries, File libraries, etc.
- It's possible to define custom functions
  - Functions are the basis of every well structured program

Functions (2/2)

- Definition
  - Where the function code is defined

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

- Call
  - Where the function code is executed

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

Function arguments

- An argument is a value passed to a function
- PHP functions can accept 0 or more parameters (arguments)
- PHP functions can return 0 or 1 value

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

Variable scope (1/4)

- Inside functions new variables can be declared
- The scope of a variable is
  - The section of the program where the variable can be safely used
    - Whole program (global)
    - Function
    - Blocks of instructions { }

Variable scope (2/4)

- In PHP a variable declared inside a function is visible:
  - In the same function in which it's declared
  - It's not visible outside of the function (even in the same program)

```
function count()
{
  $pieces = 10;
  return $pieces;
}
echo $pieces;
//$pieces is an empty string!!
```

Variable scope (3/4)

- In PHP a variable declared in the main body of the program is visible
  - In the main body
  - Is not visible inside functions

```
//variable defined in the main
// body
$pieces=10;
function count() {
  return $pieces;
}
echo "there are ".count()."
pieces";
// prints out "there are pieces"
```

Variable scope (4/4)

```
//variabile globale
$pieces=10;


function count() {
  global $pieces;
  return $pieces;
}
echo "there are ".count()." pieces";
// prints out "there are 10 pieces"
```

Exercise

- age.php
  - Assign to $age your age
  - Create a function for calculating how many leap years there have been in your life
    - Assume 1 leap year every 4 years
  - Call the function and show the result

Solution

```
<?php
  global $age = 31;
  function getLeapYear()
  {
      return ($age/4);
  }
  echo "In my life there have been" . getLeapYear() . "leap 
years.";
?>
```

Agenda

- PHP - language basics
- PHP - functions
- PHP - forms
- PHP - sessions
- PHP - databases

Form – example

- Textfield named =“color”
- Form action pointing at form.php

```
<form action =“form.php” method=GET>
Your preferred color:
<input type=”text” name=“color” value=“”>
<input type=”submit” name=“submit”>
</form>
```

Form – server side elaboration

- File form.php:
  - Simply show the data entered by the user inside the color textfield
    - <input type=text name="color" value="">
  - The name attribute of the textfield shall be unique inside the form

```
<?
echo “Your preferred color is ” . $_GET["color"];
echo “Your preferred color is ” . $_REQUEST["color"];
?>
```

Form – server side elaboration

- The destination page (form.php) can access the values entered by the user
  - Using the special arrays
    - $_GET //data passed using the GET method
    - $_POST //data passed using the POST method
    - $_REQUEST //data passed either by using GET or POST
  - Every value attribute of a field becomes an array element accessible by using a string index with a value corresponding to the name attribute associated to the same field

Example form (1/2)

File index.html

```
Vote your preferred site: <BR>

<FORM ACTION=“do.php” METHOD=“GET”>

<INPUT TYPE=“TEXT” NAME=“site”><BR>

<INPUT TYPE=“SUBMIT” VALUE=“Vote”>

</FORM>
```

Example Form (2/2)

File do.php

```
The selected site is:<BR>

<?php

  echo $_GET["site"];

?>
```

Exercise

- Design a simple web calculator accepting only 2 operands and one operation chosen between
  - +
  - -
  - *
  - /
- Design the PHP page that computes the calculation result

Agenda

- PHP - language basics
- PHP - functions
- PHP - forms
- PHP - sessions
- PHP - databases

Web Sessions

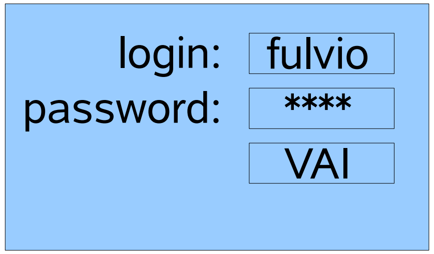
- The HTTP protocol is stateless (without memory)
  - Every request is independent from the previous one(s), and cannot “remember” the previous path of a given user
  - Input variables in each page may only depend on the FORM in the immediately preceding page
- Users, on the other hand, need to feel the impression of a “continuous” navigation, where a choice made in a page is “remembered” for all successive pages.

Requirement

- We must build an **abstraction** of a "navigation session"
- Session = sequence of pages, visited on the same site by the same user with the same browser, in the same navigation stance, having a logic sequentiality

The problem: example

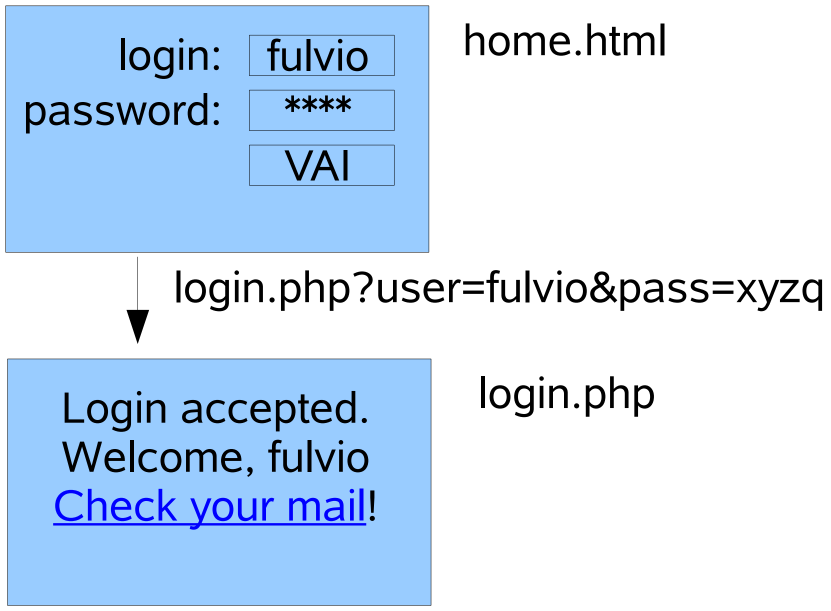
- The user enters username & password on the first page he visits
- The next page will be a dynamic page "validating" the login
- All successive pages should know the user name...



home.html

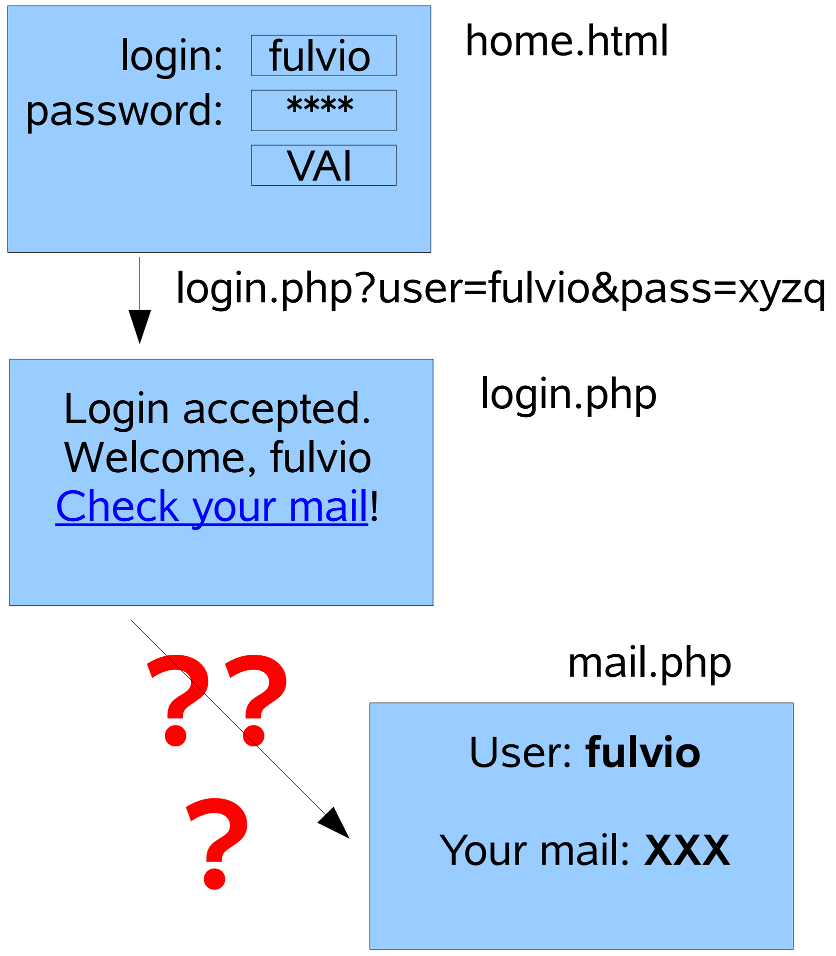
The problem: example

- The user enters username & password on the first page he visits
- The next page will be a dynamic page "validating" the login
- All successive pages should know the user name...



The problem: example

- The user enters username & password on the first page he visits
- The next page will be a dynamic page "validating" the login
- All successive pages should know the user name...

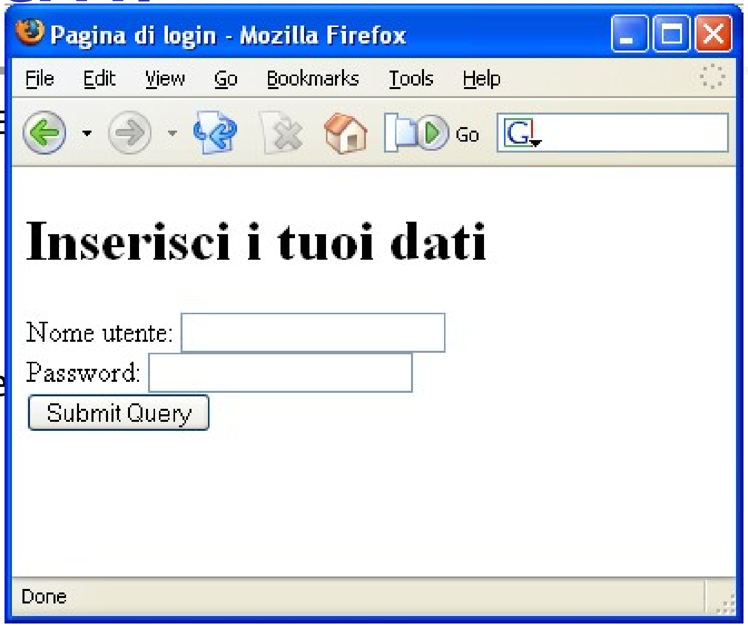


Example: home.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//E
strict.dtd">
<html>
  <head>
    <title>Pagina di login</title>
    <meta http-equiv="content-type" content="te
  </head>
  <body>

    <h1>Insert your data</h1>
    <form method="GET" action="login.php">
      Username: <input type="text" name="user" /> <br />
      Password: <input type="password" name="pass" /> <br />
      <input type="submit" />

    </form>
  </body>
</html>
```


Pagina di login - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
Go
Submit Query
Done

Inserisci i tuoi dati

Nome utente:
Password:

Possible solutions

- URL rewriting
- Hidden FORM variables
- Cookies
- Cookie-based session management

URL rewriting

- We may modify the text of *all links* in the pages, to pass every time, with the GET method, all desired variables.
  - <a href="mail.php?user= <?php echo "\"" . $_GET["user"] . "\"" ?> >
- We must modify *each and every* link with this strategy

Hidden variables

- We may add some hidden variables into a FORM
  - <input type="hidden" name="user" value="fulvio" />
- Invisible to the user, cannot be modified by the user
- Accessible to the server page ($_REQUEST["user"]) and can be modified by the server (value="xxxx")

Hidden variables: drawbacks

- We must repeat *all* variables in *all* pages
- Only FORM-based navigation
  - No <a href="..."> links!

Cookies

- Cookies are “small” parcels of information that the server may store on the browser
  - In the http response, the server adds a “set cookie” command
  - The browser promises to send back all cookies to the server at each successive http request
- A cookie is a 4-tuple composed of
  - Name
  - Value
  - Expiration (date/time)
  - Domain

Cookie-based solution

- When login.php validates the password, it sets a cookie on the browser
- The page mail.php may read the cookie that will be sent by the browser at the next request

Cookies in PHP

- Writing: setcookie("name", "value")
  - *bool setcookie (string name [, string value [, int expire [, string path [, string domain [, int secure]]]]])*
  - Must be called **before any** HTML contents (including spaces!)
- Reading: $_COOKIE["name"]
- Checking: if(isset($_COOKIE["name"])) ...
- Deleting: setcookie("name", "")

Cookies: pros

- Only one set-cookie is needed, at the first page receiving the information
- Many cookies may be set, one for each variable
- No need to modify links nor FORMs

Cookies: cons

- Impossible to store “large” information
- Storing a lot of cookies is “impolite”
- Trusting data coming from the browser may be unsafe
- The browser might (legitimately) refuse to send the cookies back

Session management

- We may store many different variables by using *only one cookie*
- The cookie is used to store a "magic number", different for each new session
  - name="session" value="123456"
- The server stores (somewhere) all session variables associated to that number
  - Session=123456 ⇒ { "user" = "fulvio", "language" = "IT" }

Session variables

- A set of (name, value) couples
- May be stored
  - In the web server memory
  - In a file private to the web server (ex. session.123456.vars)
  - In a database table (Session, Name, Value)
- On every new page, the received cookie will “unlock” the associated variables, and make them available to the server page.

Sessions in PHP (1/3)

- Session activation: session_start()
  - No parameter. Must be called *before any other action*
  - Creates a new session, or reconnects to the existing one
  - Creates a cookie with a name like PHPSESSID123456

Sessions in PHP (2/3)

- Session variables usage
  - Special vector $_SESSION will store all variables associated to the session
  - $_SESSION["user"] = "fulvio"
  - echo $_SESSION["user"]
  - if (isset($_SESSION["user"])) ...

Sessions in PHP (3/3)

- Session closing:
  - Expires automatically after 20-30 minutes
  - Expires automatically when the browser is closed
  - May be “reset” with session_destroy()

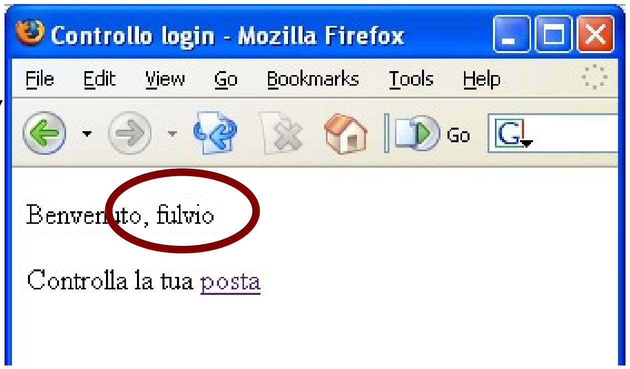
Example: login.php

```
<?php

    session_start() ;

    $user = $_REQUEST["user"] ;
    $pass = $_REQUEST["pass"] ;

    if($user == "fulvio" && $pass == "xyz")
    {
        // login OK
        $_SESSION["user"] = $user ;
        $login = 1 ;
    }
    else
    {
        //
        $l
    }
?>
```


Controllo login - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
Go

Benvenuto, fulvio

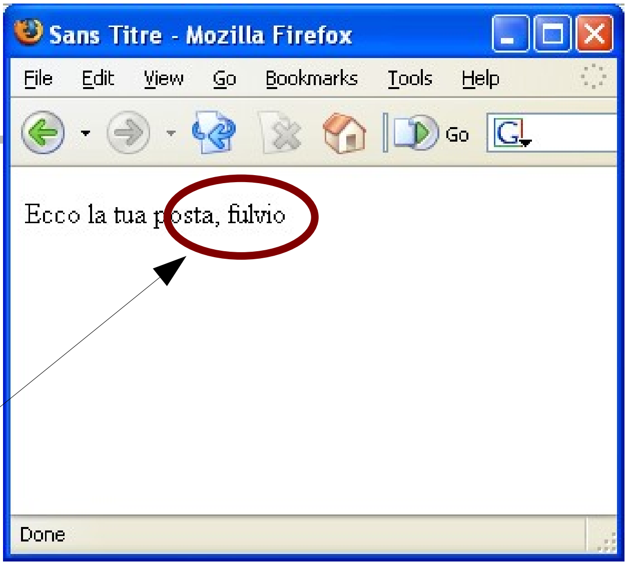
Controlla la tua posta

```
<html>
<head>
<title>Check login</title>
</head>
<body>
<?php

if($login == 1)
{
  echo "<p>Welcome, $user</p>\n";
  echo "<p>Check your <a 
href=\"mail.php\">mail</a></p>\n" ;
}
else
{
  echo "<p>ERROR: invalid login, <a 
href=\"home.html\">retry</a></p>\n" ;
}
//echo "<p>Password: $pass</p>\n" ;
?>
</body>
</html>
```

Example: mail.php

```
<?php
session_start() ;
?>
<html>
<head>
<title>La tua posta</title>
</head>
<body>
<?
  if(isset($_SESSION["user"]))
  {
        $user = $_SESSION["user"] ;
        echo "<p>Ecco la tua posta, $user</p>\n" ;
        /*  stampa la posta... */
  }
  else
  {
        echo "<p>ERRORE: devi prima fare il <a
href=\"home.html\">login</a></p>\n" ;
  }
?>
</body>
</html>
```


Sans Titre - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
Go
Done

Ecco la tua posta, fulvio

Agenda

- PHP - language basics
- PHP - functions
- PHP - forms
- PHP - sessions
- PHP - databases

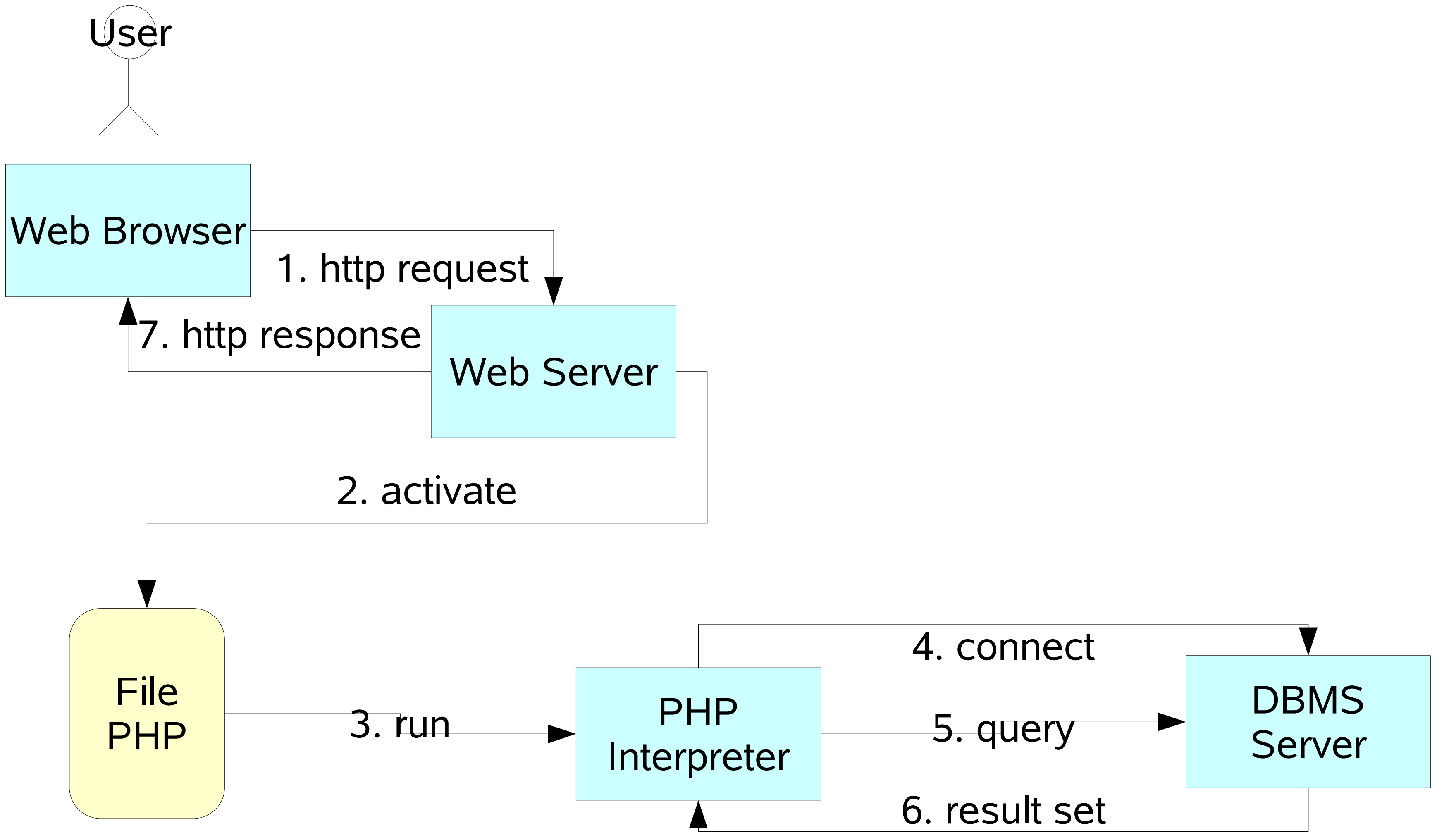
DB connections

- PHP supports many different DBMS:
  - dbm, dBase, FrontBase, filePro, Informix, Interbase, Ingres II, Microsoft SQL Server, mSQL, MySQL, MySQLi, ODBC,Oracle 8, Ovrimos, PostgreSQL, SESAM, SQLite, Sybase
- Each database interface has a set of dedicated functions, usually different between different DBs (and not compatible)
- Some high-level libraries offer a unified interface for accessing DBMS (PEAR DB, ADODB, ...)

Basic terminology

- **Connection** = open link between the PHP interpreter and a given DBMS
- **Query** = SQL command issued by the PHP interpreter to the DBMS by using an open connection
- **Result set** = table (virtual) of query results, as computed by the DBMS. Result sets are not automatically transfered to the PHP interpreter and there are no guarantees about their storing inside the DBMS (VOLATILE)
- **Row** = A single row of a Result set.

Architecture



Basic Structure

```
<?php
/* Connecting, selecting database */
```

Connect

```
$link = mysql_connect("mysql_host", "mysql_user", "mysql_password")
    or die("Could not connect : " . mysql_error());
```

DB selection

```
echo "Connected successfully";
mysql_select_db("my_database") or die("Could not select database");
```

Query

Preparation

```
/* Performing SQL query */
$query = "SELECT * FROM my_table";
```

Execution

```
$result = mysql_query($query) or die("Query failed : " . mysql_error())
```

Row extraction

```
/* Printing results in HTML */
echo "<table>\n";
while ($line = mysql_fetch_array($result, MYSQL_ASSOC)) {
    echo "\t<tr>\n";
    foreach ($line as $col_value) {
        echo "\t\t<td>$col_value</td>\n";
    }
    echo "\t</tr>\n";
}
echo "</table>\n";
```

End of the query

```
/* Free resultset */
mysql_free_result($result);
```

Disconnect

```
/* Closing connection */
mysql_close($link);
?>
```

Connection

- The DBMS connection must be opened in every page using information from the DB:
  - `$link = mysql_connect("mysql_host", "mysql_user", "mysql_password")    or die("Could not connect : " . mysql_error());`
- For XAMPP, the parameters are:
  - `$link = mysql_connect("localhost", "root", "") or die("Could not connect : " . mysql_error());`
- Immediately after, a specific DB must be selected:
  - `mysql_select_db("my_database") or die("Could not select database");`

Query

- 4 phases:
  - **Preparation**: Builds the SQL query string
  - **Execution**: The SQL query is issued to the DBMS and the corresponding Result set is gathered
  - **Extraction**: reading of data in the result set
  - **Data Formatting**: (X)HTML code for rendering results
- Extraction and Data Formatting are usually performed inside a while (or a for) cycle
- Data formatting may be optional

Preparation

- Fixed SQL command:
  - `$query = "SELECT login FROM user" ;`
  - `$query = "INSERT INTO user (login, password) VALUES ('goofy', 'xyz')" ;`
- SQL command with variables:
  - `$query = "SELECT login FROM user WHERE password = '" . $password . "'" ;`
  - The string concatenation operator is used to insert the value of PHP variables inside the SQL query. Remember single quotes (') inside SQL queries for literal values

Execution

- Once the query is ready, it must be sent to the DBMS using an open connection

  -
    ```
    $result = mysql_query($query) or
    die("Query failed : " . mysql_error());
    ```
- For INSERT, DELETE or UPDATE queries no other work is needed
- SELECT queries, instead, need post-processing of the returned result set
- `$result` contains a reference to the query result set, by properly using `$result` required data can be retrieved

Extraction

- Based on
  - `$row = mysql_fetch_array($result, MYSQL_ASSOC) ;`
  - `$result` is the Result set given by `mysql_query`
  - `$row` is an associative array containing the query results
  - The `mysql_fetch_array` function returns the rows of the result set, one per each call
  - When the end of the result set is reached, the function returns `false`

Example

name	surname
Fulvio	Corno
Dario	Bonino

- `$query = "SELECT name,surname FROM user"`
- `$result = mysql_query($query) ;`
- `$row = mysql_fetch_array($result, MYSQL_ASSOC) ;`
- `$row["name"]` → `"Fulvio"`, `$row["surname"]` → `"Corno"`
- `$row = mysql_fetch_array($result, MYSQL_ASSOC) ;`
- `$row["name"]` → `"Dario"`, `$row["surname"]` → `"Bonino"`
- `$row = mysql_fetch_array($result, MYSQL_ASSOC) ;`
- `$row` → `0`

Extraction and data formatting

- The extraction and data formatting phases are usually collapsed into a single cycle

```
$result = mysql_query("SELECT id, name FROM mytable");
while ($row = mysql_fetch_array($result, MYSQL_ASSOC))
{
    printf("<p>ID: %s  Name: %s</p>\n",
       $row["id"], $row["name"]);
}
```

Alternative extraction

- The `MYSQL_ASSOC` parameter requires the extraction function to return `$row` in form of an associative array, where data is accessible by name:
  - `$row["name"]`, `$row["surname"]`, etc.
- `MYSQL_NUM`, instead, forces the extraction function to return a simple array, with numeric indexes
  - In this case the first column has index 0
    - `$row[0]`, `$row[1]`, etc.

Freeing results

- At the end of results extraction the memory associated to the result set shall be freed (both at the PHP interpreter and at the DBMS)
  - `mysql_free_result($result);`
- Freeing the result set leaves the DB connection open allowing to issue new queries without performing a new connection.

Connection closing

- At the end of the elaboration (usually at the end of the PHP page) the DB connection shall be closed
  - `mysql_close($link);`

Useful DB functions

- `mysql_num_rows($result)`
  - Returns the number of rows contained in the result set
- `mysql_error()`
  - Gets the error message associated to the last operation on the DB

And… for other DBMS?

- Same concepts, different syntax!
- Ex: PostgreSQL

```
<?php
$conn = pg_connect("dbname=publisher");
if (!$conn) die ("An error occured.\n");

$result = pg_query($conn, "SELECT id, author, email FROM authors");
if (!$result) die("An error occured.\n");

while ($row = pg_fetch_assoc($result)) {
     echo $row['id'];
     echo $row['author'];
     echo $row['email'];
}
?>
```

- Ex2: Oracle 8

```
<?php
$connection = oci_connect("user", "password");

$query = "SELECT id, name, lob_field FROM fruits";

$statement = oci_parse ($connection, $query);
oci_execute ($statement);

while ($row = oci_fetch_array ($statement, OCI_ASSOC)) {
     echo $row['ID']."<br>";
     echo $row['NAME']."<br>";
}
?>
```