

# PHP Scripting language

Ludovico Antonio Muratori  
Ci S.B.i.C. snc  
Cesena, ITALY

# Introduction to PHP

*“PHP is a server-side scripting language designed specifically for the Web. Within an HTML page, you can embed PHP code that will be executed each time the page is visited. Your PHP code is interpreted at the Web server and generates HTML or other output that the visitor will see” (“PHP and MySQL Web Development”, Luke Welling and Laura Thomson, SAMS)*

# PHP History

1994: Created by Rasmus Lerdorf, software engineer (part of Apache Team)

1995: Called Personal Home Page Tool, then released as version 2 with name PHP/FI (Form Interpreter, to analyze SQL queries)

Half 1997: used by 50,000 web sites

October 1998: used by 100,000 websites

End 1999: used by 1,000,000 websites

# Alternatives to PHP

Practical extraction and Report Language (Perl)

Active Server Pages (ASP)

Java server pages (JSP)

Ruby

# (Good) Topics about PHP

Open-source

Easy to use ( C-like and Perl-like syntax)

Stable and fast

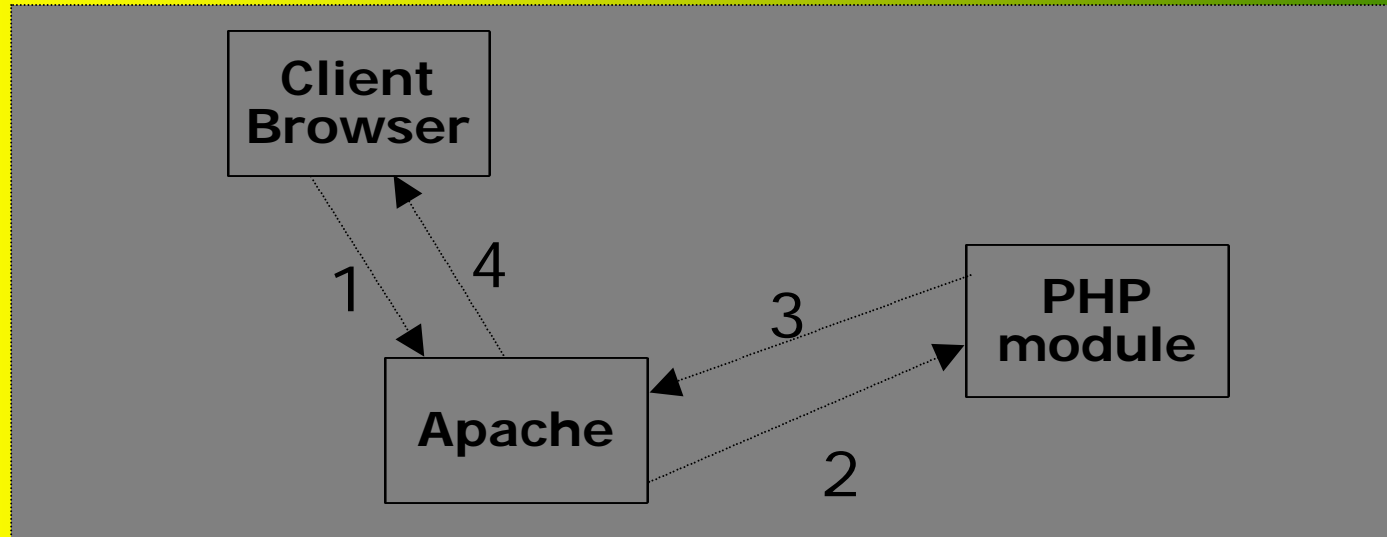
Multiplatform

Many databases support

Many common built-in libraries

Pre-installed in Linux distributions

# How PHP generates HTML/JS Web pages



- 1: Client from browser send HTTP request (with POST/GET variables)
- 2: Apache recognizes that a PHP script is requested and sends the request to PHP module
- 3: PHP interpreter executes PHP script, collects script output and sends it back
- 4: Apache replies to client using the PHP script output as HTML output

# Hello World! (web oriented)

```
<html>
<head>
  <title>My personal Hello World! PHP script</title>
</head>
<body>
  <?
    echo "Hello World!";
  ?>
</html>
```

PHP tag, allow to insert PHP code. Interpretation by PHP module will substitute the code with code output

# Variables (I)

- . To use or assign variable \$ must be present before the name of the variable
- . The assign operator is '='
- . There is no need to declare the type of the variable
- . the current stored value produces an implicit type-casting of the variable.
- . A variable can be used before to be assigned

```
$A = 1;
```

```
$B = "2";
```

```
$C = ($A + $B); // Integer sum
```

```
$D = $A . $B; // String concatenation
```

```
echo $C; // prints 3
```

```
echo $D; // prints 12
```



# Variables (II)

- . Function isset tests if a variable is assigned or not

```
$A = 1;  
if (isset($A))  
    print "A isset"  
if (!isset($B))  
    print "B is NOT set";
```

- . Using \$\$

```
$help = "hiddenVar";  
$$help = "hidden Value";  
echo $$help; // prints hidden Value  
$help = 10;  
$help = $$help * $$help;  
echo $help; // print 100
```

# Strings (I)

- . A string is a sequence of chars

```
$stringTest = "this is a sequence of chars";
```

```
echo $stringTest[0]; output: t
```

```
echo $stringTest; output: this is a sequence of chars
```

- . A single quoted strings is displayed "as-is"

```
$age = 37;
```

```
$stringTest = 'I am $age years old'; // output: I am $age years old
```

```
$stringTest = "I am $age years old"; // output: I am 37 years old
```

- . Concatenation

```
$conc = "is "."a "."composed "."string";
```

```
echo $conc; // output: is a composed string
```

```
$newConc = 'Also $conc '.$conc;
```

```
echo $newConc; // output: Also $conc is a composed string
```

# Strings (II)

- Explode function

```
$sequence = "A,B,C,D,E,F,G";  
$elements = explode(",", $sequence);  
// Now elements is an array with all substrings between "," char  
echo $elements[0]; // output: A;  
echo $elements[1]; // output: B;  
echo $elements[2]; // output: C;  
echo $elements[3]; // output: D;  
echo $elements[4]; // output: E;  
echo $elements[5]; // output: F;  
echo $elements[6]; // output: G;
```

# Arrays (I)

- . Groups a set of variables, every element stored into an array as an associated key (index to retrieve the element)

```
$books = array( 0=>"php manual",1=>"perl manual",2=>"C manual");
```

```
$books = array( 0=>"php manual","perl manual","C manual");
```

```
$books = array ("php manual","perl manual","C manual");
```

```
echo $books[2]; output: C manual
```

- . Arrays with PHP are associative

```
$books = array( "php manual"=>1,"perl manual"=>1,"C manual"=>1); // HASH
```

```
echo $books["perl manual"]; output: 1
```

```
$books["lisp manual"] = 1; // Add a new element
```

# Arrays (II)

- . Working on an arrays

```
$books = array( "php manual","perl manual","C manual");
```

- . Common loop

```
for ($i=0; $i < count($books); $i++)
```

```
    print ($i+1)."-st book of my library: $books[$i]";
```

- . each

```
$books = array( "php manual"=>1,"perl manual"=>2,"C manual"=>3);
```

```
while ($item = each( $books )) // Retrieve items one by one
```

```
    print $item["value"]."-st book of my library: ".$item["key"];
```

```
// each retrieve an array of two elements with key and value of current element
```

- . each and list

```
while ( list($value,$key) = each( $books ))
```

```
    print "$value-st book of my library: $key";
```

```
// list collect the two element retrieved by each and store them in two  
different // variables
```

# Arrays (III)

- Multidimensional arrays

```
$books = array( array("title"=>"php manual","editor"=>"X","author"=>"A"),  
                array("title"=>"perl manual","editor"=>"Y","author"=>"B"),  
                array("title"=>"C manual","editor"=>"Z","author"=>"C"));
```

- Common loop

```
for ($i=0; $i < count($books); $i++ )  
    print "$i-st book, title: ".$books[$i]["title"]." author: ".$books[$i]["author"].  
        " editor: ".$books[$i]["editor"];  
  
// Add ".$n" for text new page or "<BR>" for HTML new page;
```

- Use list and each

```
for ($i=0; $i < count($books); $i++)  
{  
    print "$i-st book is: ";  
    while ( list($key,$value) = each( $books[$i] ))  
        print "$key: $value ";  
    print "<BR>"; // or "\n"  
}
```

# Case study (small database I)

- . You need to build one or more web pages to manage your library, but:
  - ***“You have no time or no knowledge on how to plan and design database”***
  - ***or “You have no time or knowledge on how to install a free database”***
  - ***And “The database to implement is small” (about few thousands entries, but depends on server configuration)***

# Case study (small database II)

```
#cat /usr/local/myDatabaseDirectory/library.txt
```

```
php manual  X   A   330
```

```
perl manual Y   B   540
```

```
C manual      Z   C   480
```

(fields separated by tabs: 'php manual<tab>X<tab>A', new line at the end of each entry)

```
<? // script to show all book in my library
```

```
$books = file("/usr/local/myDatabaseDirectory/library.txt"); // retrieve library "database"
```

```
for ($i=0; $i<count($books), $i++ )
```

```
    $books_array[$i] = explode( "\t", $books[$i]); // Extract elements from line
```

```
...
```

```
for ($i=0; $i<count($books_array), $i++ )
```

```
    print "$i-st book, title: ".$books_array[$i]["title"]." author: ".$books_array[$i]["author"].
```

```
        " editor: ".$books_array[$i]["editor"]." <BR>";
```



# Case study

## A way to reuse code (I)

Using functions is possible to write more general code, to allow us to reuse it to add feature:

- For the same project (always try to write reusable code, also you will work for a short time on a project)
- For new projects

<? // config.php, is a good idea to use configuration files

```
$tableFiles = array ( "books" => "/usr/local/myDatabaseDirectory/books.txt");
```

```
$bookTableFields = array ("title","author","editor","pages");
```

```
// future development of the library project (add new tables)
```

```
$tableFiles = array ( "users" => "/usr/local/myDatabaseDirectory/users.txt");
```

```
$userTableFields = array ("code","firstName","lastName","age","institute");
```

```
?>
```

# Case study

## A way to reuse code (II)

```
<? // script to show all book in my library
$books = file("/usr/local/myDatabaseDirectory/library.txt");
// retrieve library "database"
for ($i=0; $i<count($books), $i++ )
    $books_array[$i] = explode( "\t", $books[$i]); // Extract elements from line
...
for ($i=0; $i<count($books_array), $i++ )
    print "$i-st book, title: ".$books_array[$i]["title"]." author:
    ".$books_array[$i]["author"].
        " editor: ".$books_array[$i]["editor"]." <BR>";
```

# Functions in details (I)

The syntax to implement a user-defined function is :

```
function function_name([parameters-list]opt)
```

```
{ .....implementation code..... }
```

parameters-list is a sequence of variables separated by “,”

- it's not allowed to overload the name of an existing function;
- Function names aren't case-sensitive;
- To each parameter can be assigned a default value;
- arguments can be passed by value or by reference
- It's possible using a variable number of parameters

# Object Oriented PHP

Encapsulation

Polymorphism

Inheritance

Multiple Inheritance: actually unsupported

# Encapsulation

<?

```
class dayOfWeek {  
    var $day,$month,$year;  
  
    function dayOfWeek($day,$month,$year) {  
        $this->day = $day;  
        $this->month = $month;  
        $this->year = $year;  
    }  
  
    function calculate(){  
        if ($this->month==1){  
            $monthTmp=13;  
            $yearTmp = $this->year - 1;  
        }  
        if ($this->month == 2){  
            $monthTmp = 14;  
            $yearTmp = $this->year - 1;  
        }  
    }  
}
```

```
$val4 = (($month+1)*3)/5;  
$val5 = $year/4;  
$val6 = $year/100;  
$val7 = $year/400;  
$val8 = $day+($month*2)+$val4+$val3+$val5-  
        $val6+$val7+2;  
$val9 = $val8/7;  
$val0 = $val8-($val9*7);  
return $val0;  
}  
}  
  
// Main  
$instance =  
    new dayOfWeek($_GET["day"],$_GET["week"],$_GET["  
        month"]);  
  
print "You born on ".$instance->calculate()."\n";  
  
?>
```

# Inheritance

*Allow the creation of a hierarchy of classes*

```
Class reuseMe {
```

```
    function  
    reuseMe(){...}
```

```
    function  
    doTask1(){...}
```

```
    function  
    doTask2(){...}
```

```
    function  
    doTask3(){...}  
}
```

```
Class extends reuseMe {
```

```
    function example(){  
        ... // local initializations  
        // call super constructor  
        reuseMe::reuseMe();  
    }
```

```
    function doTask4(){...}
```

```
    function doTask5(){...}
```

```
    function doTask6(){...}  
}
```

# Polymorphism

A member function can override superclass implementation. Allow each subclass to reimplement a common interfaces.

```
class reuseMe {
```

```
    function  
    reuseMe(){...}
```

```
    function  
    doTask1(){...}
```

```
    function  
    doTask2(){...}
```

```
    function  
    doTask3(){...}  
}
```

```
Class extends reuseMe {
```

```
    function example(){  
        ... // local initializations  
        // call super constructor  
        reuseMe::reuseMe();  
    }
```

```
    function doTask4(){...}
```

```
    function doTask5(){...}
```

```
    function doTask6(){...}
```

```
    function doTask3(){...}  
}
```



# Multiple Inheritance not actually supported by PHP

```
class reuseMe1 {  
function reuseMe1(){...}  
function doTask1(){...}  
function doTask2(){...}  
function doTask3(){...}  
}
```

```
class reuseMe2 {  
function reuseMe2(){...}  
function doTask3(){...}  
function doTask4(){...}  
function doTask5(){...}  
}
```

```
class extends reuseMe1,reuseMe2 {...}
```



# Bibliography

- [1] *“PHP and MySQL Web Development”, Luke Welling and Laura Thomson, SA*