

SPERIMENTAZIONI DI TECNOLOGIE E COMUNICAZIONI MULTIMEDIALI

salvatore.iaconesi@artisopensource.net

Scripting languages

PHP

Inside the HTML page

```
<?php
```

```
... PHP code ...
```

```
?>
```

EXECUTED ON THE SERVER
BEFORE GIVING BACK THE PAGE TO THE USER

USED TO **DYNAMICALLY** PRODUCE
THE CONTENT OF THE WEB PAGE

WE WILL USE THEM TO INTEGRATE INFORMATION
COMING FROM A **DATABASE** OR FROM A
CONTENT MANAGEMENT SYSTEM (CMS) INTO
WEB AND MOBILE-WEB CONTENT

THE IDEA OF THE TEMPLATE

IN A “NORMAL” HTML DOCUMENT:

```
<H1>WELCOME Mario!</H1>
```

IN A TEMPLATE:

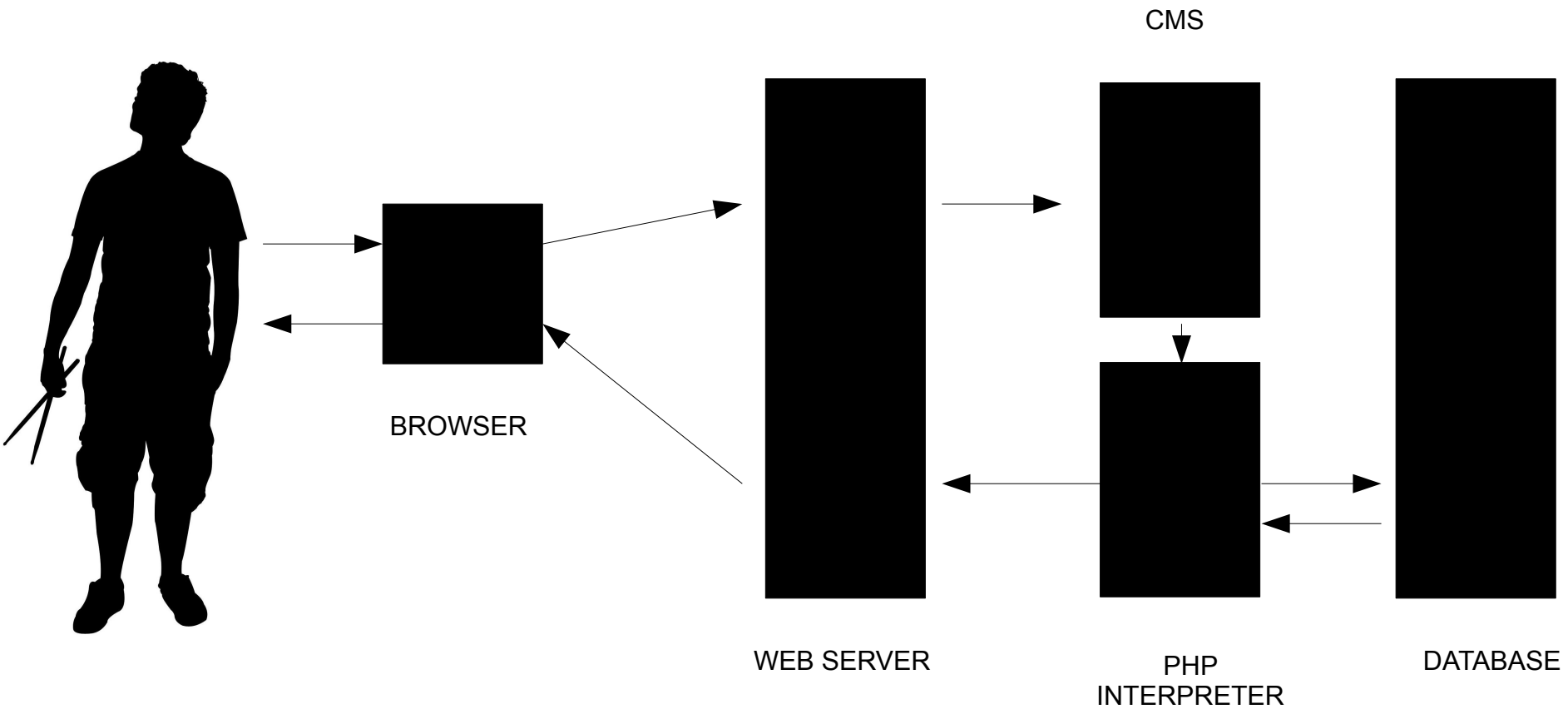
```
<H1>WELCOME <?php echo( $user_name); ?></H1>
```

```
<?php echo( $user_name ); ?>
```

IT MEANS THAT THIS PHP INSTRUCTION WILL BE EXECUTED ON THE SERVER BEFORE DELIVERING THE HTML PAGE TO THE USER

\$user_name is a variable that will contain an information (and in a bit we will see how it is possible to provide it)

The welcome greeting will thus be composed by a fixed part and by a **dynamic** one (maybe the name of the user comes from the database?)



The user asks to see a certain web page (using the browser)

The request goes to the web server, that forwards it to the CMS

The CMS interprets the request as the need to **aggregate a series of resources** (texts, images, videos, sounds, information, animations...) in order to produce the content that was requested by the server, and **selects a template** that is fit to composing these resources together.

The CMS uses a PHP program (or any other scripting language) to compose the resources into the template

Templates are regular HTML documents but in some parts of them Information is replaced by **placeholders**, by **PHP instructions**, by **loops** and by **conditions**.

PLACEHOLDER

A placeholder will be replaced by relevant information.

For example in the Wordpress CMS there are various types of placeholders. One of them are **shortcodes**

Inserting the shortcode

[gallery]

means that it will be automatically replaced in the HTML by a browsable gallery of all the images contained in the content that the user has requested.

Pretty powerful, isn't it?

PHP INSTRUCTIONS

They will be executed to produce the content of the page.

For example, in an e-commerce site you might have a shopping cart used by the user to buy some products.

In the Template you will have variables that contain the unit price (for example **\$unit_price**) and the quantity (**\$quantity**) of a certain product that the user wishes to buy.

You could use PHP instructions to show the total cost to the user:

<h3>Your total price is: <?php echo(\$unit_price * \$quantity); ?></h3>

The echo command outputs information to the HTML document.
The * means multiply

LOOPS

Many informations have the same structure. You can use loops to **repeat the same task multiple times**.

For example you could print out all the information of the articles on a certain page by making a loop:

```
<?php
    for($i = 0; $i < 10 ; $i++){
        ?>
            <div id="article<?php echo($i) ?>" class="article-box">
                <p><?php echo( $articles[ $i ] ); </p>
            </div>
        <?php
    }
?>
```

The \$articles variable is an array. Arrays contain more than one value. You access them by using square brackets. The first value is [0], the second is [1] and so on. You know how many values an array contains by using the **count** instruction. The array \$articles will contain values for \$articles[0], \$articles[1] and up to (but not including) the number shown by **count(\$articles)**, meaning **\$articles[count(\$articles) - 1]**. In the loop, **\$i++** means "add 1 to \$i" and it is the same as writing **\$i = \$i + 1;**

The FOR LOOP

The FOR loop seen in the previous slide has a very useful structure:

```
for ( initialization ; condition ; increment ){  
    ... instructions to be repeated ...  
}
```

INITIALIZATION gets executed only once at the beginning of the loop, and it describes **where we want to start from** (let's say that if we want to count from 0 to 10, we will start at 0 --> \$i = 0; in the previous example)

CONDITION tells us **when we need to stop**: whenever the condition is verified the loop goes on with another cycle. In the previous example \$i<10

INCREMENT tells us **how to proceed**, go ahead, in the loop. The increment is executed after each cycle of the loop, and then the condition is checked, to see if there will be another cycle or if the loop has finished looping. In the previous example, increment is \$i++ to move forward with our count.

CONDITIONS

Conditions help you create flexible templates that assume different forms according to different scenarios.

For example we could use this condition in our template:

```
<?php
    if($age<18){
        echo( "you are under-age!" );
    } else if ($age<90){
        echo( "Ok, this content is perfect for you!" );
    } else {
        echo("You're too old for this!");
    }
?>
```

The IF statement is very powerful it can combine multiple IF, IF ELSE, IF ELSE IF ELSE statements and by reading it in plain english it is pretty obvious in what it does. It verifies conditions, one by one (or just one, if there is only a plain IF statement with no ELSE IF or ELSE statements to follow) and as soon as it verifies that one is TRUE (like in $3 < 10$) it executes the code found in the curly brackets `}` right after it. In PHP curly brackets mark **blocks** of code: these can be used to specify code that should be executed together. Blocks of code can be nested one into each other.

GETTING VALUES INTO PHP

We can get the values that are used in PHP in several ways:

- Get them from another page, using the GET and POST methods
 - For example by creating a form
- Get them from a database (next lessons)
- Synthesize them (by creating them according to some process)

The MOST fundamental is the use of GET and POST mechanisms to propagate information from one page to the other.

GET

As you might have seen, internet addresses are often in a form like this:

<http://www.aninternetdomain.com/index.php?name1=value1&name2=value2>

This internet address is passing values to the “index.php” web page:

```
name1=value1  
name2=value2
```

This method of passing values is called GET and is easily accessible from PHP. For this we can use the **\$_GET** predefined variable (meaning that the system creates it for us):

```
$my_variable = $_GET[“name1”];
```

In the example, **\$my_variable** will, thus, contain the value “**value1**”

GET is very easy to use, but it has one major limitation: the string of characters after the “?” in the internet address must not be longer than 1024 characters, or it will be cut off. So GET is perfect for every case in which you must not send too much data, and when data can be represented as a string (not to send an image, for example).

POST

POST method of sending information bypasses the problems of the GET method (content can be as long as you want, and can transfer binary content) but it cannot be easily experimented: to try it out we must create a FORM (next slide)

We access POST sent information in much the same way as GET: the system automatically provides us with the **\$_POST** variable that can be used to access information sent through POST methods:

```
$my_variable = $_POST["name-of-a-very-long-information"];
```

FORMS

Forms are HTML elements used to send information to another page:

```
<FORM ACTION="..." METHOD="..." >
```

```
<!-- a series of input fields -->
```

```
<!-- ... and then a SUBMIT button to send the info -->
```

```
<INPUT TYPE="SUBMIT" VALUE="SEND" />
```

```
</FORM>
```

The ACTION parameter specifies who is the information is to be sent to (ex.: if we want to send the info to the “store.php” web page, we will write its name into the ACTION parameter)

The METHOD parameter allows us to choose among GET/POST methods of sending info to a web page.

FORMS example - first step

In one page (let's call it "a.php") I could have this form:

```
<form method="POST" action="b.php">
```

```
  What is your name?<br />
```

```
  <input type="text" name="user_name" />
```

```
  <br />
```

```
  <br />
```

```
  <input type="SUMBIT" value="SEND" />
```

```
</form>
```

FORMS example - second step

In the other page (let's call it "b.php") I could have this PHP code:

```
<?php
```

```
    $user_name = "unknown user";  
    if( isset( $_POST["user_name"] ){  
        $user_name = $_POST["user_name"];  
    }
```

```
    echo( "Welcome to this webpage " . $user_name . " !" );
```

```
?>
```

The `isset()` command verifies that a certain variable actually exists. For example here we use it to verify that someone is actually sending something from the other page. If this is not the case, the default value we setup at the beginning will be used instead.

Strings are joined using a dot. In the `echo` command, we use the dot `.` symbol to attach the predefined parts of the welcome message to the variable.

EXERCISE 1

Write two web pages:

- the first one, called “a.php” contains a form, asking for the user's name, and a submit button. The FORM goes to page “b.php”
- the second one, called “b.php” takes the input from the first one and prints it out 20 times

EXERCISE 2

Write two web pages:

- the first one, called “a.php” contains a form, asking for the user's name, a number, and a submit button. The FORM goes to page “b.php”
- the second one, called “b.php” takes the input from the first one and prints out the user name a number of times that is equal to the one specified by the user in the second input field. (if the user writes “Mario” and “10”, the name “Mario” will be printed out 10 times)

EXERCISE 3

Write two web pages:

- the first one, called “a.php” contains a form, asking for the user's name, a text using a TEXTAREA input component, and a submit button. The FORM goes to page “b.php”
- the second one, called “b.php” takes the input from the first one and prints it out using the user name for a welcome message and printing out the text the user wrote in the TEXTAREA really big, by adding it to a DIV element in the b.php page whose class attribute points out to a CSS rule with an enormous font.

EXERCISE 4

Write two web pages:

- the first one, called “a.php” contains a form, asking for the user's name, a select box with a couple of options (you choose them), and a submit button. The FORM goes to page “b.php”
- the second one, called “b.php” takes the input from the first one and prints out the user name and, depending on which selection the user made in the select box, print out a custom message (ex.: if the choices were “chocolate” and “vanilla” the messages printed out could be, respectively, “Perfect! Your chocolate ice cream is coming up!” and “Oh, sorry: We're out of vanilla...”)

EXERCISE 5

Write two web pages:

- the first one, called “a.php” contains a form, asking for
 - an “article-type”, through a drop-down menu (select) with values 1,2 and 3
 - an article-title
 - an article-text
 - and a submit button to send the info
- the second one, called “b.php”:
 - takes the “article-type” parameter and creates a <div> with a different class according to it (article-type==1 --> class=”article-type-1” and so on)
 - composes the rest of the article using article-title and article-text
 - try using CSS rules so that the three article types correspond to radically different renderings on the web page

RESOURCES

XAMPP

To use PHP on your computer you need to install it and a web server.
Download and install XAMPP at:

- <http://www.apachefriends.org/it/xampp.html>

PHP TUTORIALS

- <http://www.php.net>
- <http://php.net/manual/en/tutorial.php>
- <http://www.w3schools.com/PHP/>
- <http://www.homeandlearn.co.uk/php/php.html>

FORM TUTORIALS

- http://www.w3schools.com/html/html_forms.asp
- <http://www.htmlgoodies.com/tutorials/forms>
- <http://www.phpf1.com/tutorial/php-form.html>

NEXT STEPS

In the next lesson we will use PHP to build our first Wordpress theme.

It is really important that you follow the proposed tutorials and try out the examples.

See you next time :)