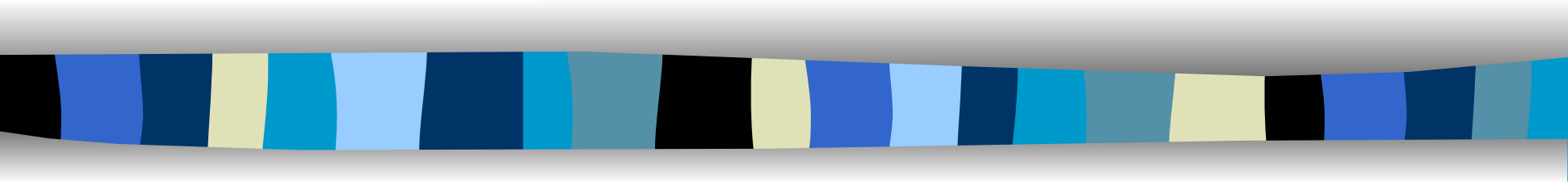
PHP: Hypertext Preprocessor

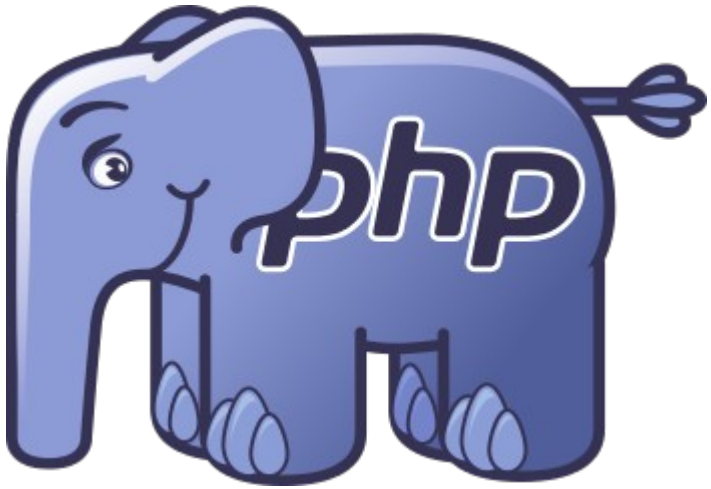


CSCI 3000
Web Programming

Dr. Luis A. Cueva-Parra

PHP: Hypertext Preprocessor

- PHP is a server-side scripting language designed for Web development, but also used as a general-purpose programming language.



- It was created by Rasmus Lerdorf in 1994.

PHP

- PHP code may be embedded into HTML code.
- PHP can be used in combination with various web template systems, web content management systems, and web frameworks.
- PHP code is usually processed by a PHP interpreter implemented as a module in the web server or as a Common Gateway Interface (CGI) executable.

PHP File

- PHP files can contain text, HTML, CSS, JavaScript, and PHP code
- PHP code are executed on the server, and the result is returned to the browser as plain HTML
- PHP files have extension ".php"

```
<!DOCTYPE html>
<html> <body>
<?php
echo "My first PHP script!";
?>
</body></html>
```

What we can do with PHP? (1/2)

- We can
  - generate dynamic page content
  - create, open, read, write, delete, and close files on the server
  - collect form data
  - send and receive cookies
  - add, delete, modify data in your database
  - control user-access
  - encrypt data

What we can do with PHP? (2/2)

- We can
  - output images, PDF files, and even Flash movies.
  - output any text, such as XHTML and XML.

Advantages of using PHP

- It is multi-platform runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)
- It is compatible with almost all servers used today (Apache, Internet Information Services (MS), etc.)
- It supports a wide range of databases.
- It is free, open source resource: See www.php.net
- It is easy to learn and runs efficiently on the server side.

PHP Installation

- Install a web server and then install PHP and MySQL.
- Easier is to install XAMPP locally.

PHP Syntax

- A PHP script can be placed anywhere in the document.
- A PHP script starts with **<?php** and ends with **?>**

```
<?php
echo "My first PHP script! <br>";
echo “<h1> My PHP heading </h1><br>”;
print “My last PHP line”;
?>
```

Comments in PHP

```
<?php
// This is a single-line comment
# This is also a single-line comment
/* This is a multiple-lines comment block
that spans over multiple lines   */
// You can also use comments to leave
out parts of a code line
$x = 5 /* + 15 */ + 5;
echo $x;
?>
```

PHP Case Sensitivity (1/2)

- PHP keywords (e.g. if, else, echo, etc.) are not case sensitive:

```
<?php
ECHO "Hello World!<br>";
echo "Hello World!<br>";
EcHo "Hello World!<br>";
?>
```

PHP Case Sensitivity (2/2)

- All PHP variable names are case-sensitive.

```
<?php
 $color = "blue";
 echo "My car is " . $color . "<br>";
 echo "My house is " . $COLOR . "<br>";
 echo "My boat is " . $coLOR . "<br>";
?>
```

PHP Variables (1/2)

- A PHP variable name starts with the **$** sign, followed by the name of the variable.
- It must start with a letter or the underscore character.
- It cannot start with a number.
- It can only contain alpha-numeric characters and underscores.
- It is case-sensitive.

PHP Variables (2/2)

- Examples:

```
<?php
$greeting = “Welcome to PHP!”;
$txt = “PHP”;
$amount = 25;
$total = 23.56;
$y = 50;
$newTotal = $amount + $total;
echo “My final total is $newTotal”;
echo “I love” . $txt . “!”;
echo $amount + $y;
?>
```

echo and print statements (1/2)

- For displaying an output we can use echo or print functions.
- They are very similar.
- echo does not have return value.
- print, if successful will return 1.

echo and print statements (2/2)

```
<?php
print “Hello there!”;
print “<h1> PHP is fun!</h1>”;
$txt1 = “PHP”;
$txt2 = “Web Programming”;
$x = 3;
$y = 4;
print "<h2>" . $txt1 . "</h2>";
print "I love " . $txt1 . "! <br>";
print $x + $y;
?>
```

PHP Data Types (1/3)

- PHP supports a variety of data types.
- ***PHP String:***

```
$txt1 = “PHP”;
echo $txt1;
```

- ***PHP Integer:***

```
$x = 3456;
echo $x;
```

- ***PHP Float:***

```
$x = 13.456;
var_dump($x);
```

PHP Data Types (2/3)

- ***PHP Boolean:*** true and false values.

```
$a = true;
$b = false;
```

- ***PHP Array:***

```
$cars = array("Volvo","BMW","Toyota");
var_dump($cars);
```

- ***PHP Object:*** stores data and information for processing that data.
  - A class need to be declared.
  - Objects are created explicitly.

PHP Data Types (3/3)

- ***PHP Object:***

```
<?php
class Car {
    function Car() {
        $this->model = "VW";
    }
}
// create an object
$herbie = new Car();
// show object properties
echo $herbie->model;
?>
```

PHP Strings (1/3)

- ***Length of a String***: Use the `strlen()` function:

```
<?php
echo strlen("Hello world!"); // outputs 12
?>
```

- ***Counting words in a String***: Use the `str_word_count()` function:

```
<?php
echo str_word_count("Hello world!"); //
outputs 2
?>
```

PHP Strings (2/3)

- ***Reversing a String***: Use the `strrev()` function:

```
<?php
echo strrev("Hello world!");    // outputs !dlrow olleH
?>
```

- ***Searching specific text within a String***: Use the `strpos()` function:

```
<?php
echo strpos("Hello world!", "world"); // outputs 6
?>
```

PHP Strings (3/3)

- ***Replacing text within a String***: Use the `str_replace()` function:

```
<?php
echo str_replace("world", "Dolly", "Hello world!"); // outputs Hello Dolly!
?>
```

- There are many more PHP string functions.
- The PHP string functions are part of the PHP core. No installation is required to use these functions.

PHP Constants (1/4)

- A constant is an identifier (name) for a simple value.
- Once defined, constants cannot be changed or undefined.
- A constant name starts with a letter or underscore (no $ sign before the constant name).
- Constants are global across the entire script.

PHP Constants (2/4)

- ***Creating a PHP constant*** : Use the `define()` function:
- Syntax:

```
define(name, value, case-insensitive)
```

- Where:
  - `name` is the name of the constant
  - `value` is the value of the constant
  - `case-insensitive` specifies if the constant is case-insensitive. By default is case-sensitive.

PHP Constants (3/4)

- ***Creating a constant with a case-sensitive name:***

```
<?php
define("GREETING", "Welcome to UNG!");
echo GREETING;
?>
```

- ***Creating a constant with a case-insensitive name:***

```
<?php
define("GREETING", "Welcome to UNG!",true);
echo greetinG;
?>
```

PHP Constants (4/4)

- Constants are global.

```
<?php
define("GREETING", "Welcome to UNG!");

function myTest() {
    echo GREETING;
}

MyTest();
?>
```

PHP Arithmetic Operators

Operator	Name	Example
+	Addition	$x + $y
-	Subtraction	$x - $y
*	Multiplication	$x * $y
/	Division	$x / $y
%	Modulus	$x % $y
**	Exponentiation	$x ** $y

PHP Assignment Operators

Assignment	Equivalent to
$x = $y	$x = $y
$x += $y	$x = $x + $y
$x -= $y	$x = $x - $y
$x *= $y	$x = $x * $y
$x /= $y	$x = $x / $y
$x %= $y	$x = $x % $y

PHP Comparison Operators

Operator	Name	Example
==	Equal	$x == $y
===	Identical	$x === $y
!=	Not equal	$x != $y
<>	Not equal	$x <> $y
!==	Not identical	$x !== $y
>	Greater than	$x > $y
<	Less than	$x < $y
>=	Greater than or equal to	$x >= $y
<=	Less than or equal to	$x <= $y

PHP Increment/Decrement Operators

Operator	Name	Description
++$x	Pre-increment	Increments $x by one, then returns $x
$x++	Post-increment	Returns $x, then increments $x by one
--$x	Pre-decrement	Decrements $x by one, then returns $x
$x--	Post-decrement	Returns $x, then decrements $x by one

PHP Logical Operators

Operator	Description	Example
and	True if both $x and $y are true	$x and $y
or	True if either $x or $y is true	$x or $y
xor	True if either $x or $y is true, but not both	$x xor $y
&&	True if both $x and $y are true	$x && $y
||	True if either $x or $y is true	$x || $y
!	True if $x is not true	!$x

PHP Conditional Statements (1/z)

- ***If Statement*** : executes some code if one condition is true.
- Syntax:

```
if (condition) {
 code to be executed if condition is true;
}
```

```
<?php
$t = date("H");
if ($t < "20") {
    echo "Have a good day!"; }  ?>
```

PHP Conditional Statements (2/z)

- ***If..else Statement*** : executes some code if one condition is true and another code if that condition is false.
- Syntax:

```
if (condition) {
 code to be executed if condition is true;
} else {
 code to be executed if condition is false;
}
```

PHP Conditional Statements (3/z)

- ***If..elseif..else Statement*** : executes different codes for more than two conditions.
- Syntax:

```
 if (condition1) {
code to be executed if condition1 is true;
} elseif (condition2) {
code to be executed if condition2 is true;
} else {
code to be executed if all conditions are false;  }
```

PHP Conditional Statements (4/z)

- ***`switch` Statement*** : selects one of many blocks of code to be executed
- Syntax (see next page):

PHP Conditional Statements (5/z)

```
switch (n) {
    case label1:
        code to be executed if n=label1;
        break;
    case label2:
        code to be executed if n=label2;
        break;
    ...
    default:
        code to be executed if n is
different from all labels;   }
```

PHP While Loop

- ***The `while` Statement*** : executes a block of code while the specified condition is true.
- Syntax:

```
while (condition is true) {
 code to be executed;   }
```

```
<?php
$x = 1;
while($x <= 5) {
    echo "The number is: $x <br>";
    $x++;  }  ?>
```

PHP Do-While Loop

- ***The `do-while` Statement*** : executes a block of code at least once, then while the specified condition is true.
- Syntax:

```
do {
code to be executed;
} while (condition is true);
```

```
<?php  $x = 1;
do { echo "The number is: $x <br>";
     $x++; } while ($x <= 5);   ?>
```

PHP For Loop

- ***The `for` Statement*** : executes a block of code a fix number of times.
- Syntax:

```
for (init count; test count; increment
count) { code to be executed;   }
```

```
<?php
for ($x = 0; $x <= 10; $x++) {
    echo "The number is: $x <br>";
}
?>
```

PHP Foreach Loop

- ***The `foreach` Statement*** : loops through each key/value pair in an array.
- Syntax:

```
foreach ($array as $value) {
    code to be executed;
}
```

```
<?php
$colors = array("red", "green", "blue", "yellow");
foreach ($colors as $value) {
    echo "$value <br>";  }   ?>
```

PHP User defined Functions (1/5)

- A function is a block of statements that can be used repeatedly in a program.
- A function will not execute immediately when a page loads.
- A function will be executed by a call to the function.
- Syntax:

```
function functionName() {
    code to be executed;
}
```

PHP User defined Functions (2/5)

- A function name can start with a letter or underscore (not a number).l
- Function names are NOT case-sensitive.
- Example:

```
<?php
function writeMsg() {
    echo "Hello world!";
}

writeMsg(); // call the function
?>
```

PHP User defined Functions (3/5)

- Arguments are specified after the function name, inside the parentheses.

```
<?php
function familyName($fname) {
    echo "$fname Refsnes.<br>";  }
familyName("Jani");
familyName("Hege");
familyName("Stale");
familyName("Kai Jim");
familyName("Borge");
writeMsg(); // call the function    ?>
```

PHP User defined Functions (4/5)

- Example: Function with 2 arguments.

```
<?php
function familyName($fname, $year) {
    echo "$fname Refsnes. Born in $year
<br>";
}

familyName("Hege", "1975");
familyName("Stale", "1978");
familyName("Kai Jim", "1983");
?>
```

PHP User defined Functions (5/5)

- If we call a function without arguments it will take the default value as argument:

```
<?php
function setHeight($minheight = 50) {
    echo "The height is : $minheight <br>";
}
setHeight(350);
setHeight(); //will use default value of 50
setHeight(135);
setHeight(80);
?>
```

PHP – Form Handling (1/4)

- PHP has two superglobal variables to collect data from HTML forms: `$_GET` and `$_POST`
- Simple HTML form: `myForm.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>
```

PHP – Form Handling (2/4)

- PHP file that process the data (sent using the HTTP POST method): `welcome.php`

```
<html>
<body>
 <H2>GREETING</H2><BR>
 Welcome <?php echo $_POST["name"]; ?><br>
 Your email address is: <?php echo
 $_POST["email"]; ?>
</body>
</html>
```

PHP – Form Handling (3/4)

- Using the method `GET` modify your files accordingly.
- Simple HTML form: `myForm_get.html`

```
<html>
<body>
<form action="welcome_get.php" method="get">
Name: <input type="text" name="name"><br>
E-mail: <input type="text" name="email"><br>
<input type="submit">
</form>
</body>
</html>
```

PHP – Form Handling (4/4)

- PHP file that process the data (sent via URL parameters): welcome_get.php

```
<html>
<body>
 <H2>GREETING</H2><BR>
 Welcome <?php echo $_GET["name"]; ?><br>
 Your email address is: <?php echo
 $_GET["email"]; ?>
</body>
</html>
```

PHP GET and POST

- Both `GET` and `POST` create an array of pair of values (keys and values), where keys are the names of the fields in the form and the values are the input data.
- Both `$_GET` and `$_POST` are global variables accessible from everywhere.
- Use `GET` when sending limited and non-sensitive data.
- Use `POST` when sending unlimited and critical data.

PHP Form Validation (1/5)

- Input data retrieved using HTML forms need to be validated for security and consistency purposes.
- Data can be required or optional.
- For data validation we impose rules

PHP Form Validation (2/5)

- Basic validation rules:
  - Avoid user to enter scripts in the form fields: Use the htmlspecialchars() function.
  - Strip unnecessary characters (extra space, tab, newline) from the user input data (with the PHP trim() function)
  - Remove backslashes (\) from the user input data (with the PHP stripslashes() function)

PHP Form Validation (3/5)

```
<?php
$name=$email=$gender=$comment=$website = "";
if ($_SERVER["REQUEST_METHOD"] == "POST") {
  $name = test_input($_POST["name"]);
  $email = test_input($_POST["email"]);
  $website = test_input($_POST["website"]);
  $comment = test_input($_POST["comment"]);
  $gender = test_input($_POST["gender"]); }
function test_input($data) {
  $data = trim($data);
  $data = stripslashes($data);
  $data = htmlspecialchars($data);
  return $data; } ?>
```

PHP Form Validation (4/5)

```
<h2>PHP Form Validation Example</h2>

<form method="post" action="<?php echo
htmlspecialchars($_SERVER["PHP_SELF"]);?>">

Name: <input type="text" name="name"> <br><br>

E-mail: <input type="text" name="email"> <br><br>

Website: <input type="text" name="website"> <br><br>

Comment: <textarea name="comment" rows="5"
cols="40"> </textarea> <br><br>

Gender: <input type="radio" name="gender"
value="female"> Female

<input type="radio" name="gender" value="male"> Male

<input type="radio" name="gender"
value="other">Other  <br><br>

<input type="submit" name="submit" value="Submit">

</form>
```

PHP Form Validation (5/5)

```
<?php
echo "<h2>Your Input:</h2>";
echo $name;
echo "<br>";
echo $email;
echo "<br>";
echo $website;
echo "<br>";
echo $comment;
echo "<br>";
echo $gender;
?>
```

PHP Required Fields (1/6)

- Validation rules:
  - **Name**: Required. + Must only contain letters and whitespace
  - **E-mail**: Required. + Must contain a valid email address (with @ and .)
  - **Website**: Optional. If present, it must contain a valid URL
  - **Comment**: Optional. Multi-line input field (textarea)
  - **Gender**: Required. Must select one

PHP Required Fields (2/6)

```
<?php
$nameErr= $emailErr = $genderErr = $websiteErr = "";
$name = $email = $gender = $comment = $website = "";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
  if (empty($_POST["name"])) {
    $nameErr = "Name is required";
  } else {
    $name = test_input($_POST["name"]);
  }

```

PHP Required Fields (3/6)

```
if (empty($_POST["email"])) {
  $emailErr = "Email is required";
} else {
  $email = test_input($_POST["email"]);
}

if (empty($_POST["website"])) {
  $website = "";
} else {
  $website = test_input($_POST["website"]);
}
```

PHP Required Fields (4/6)

```
  if (empty($_POST["comment"])) {
    $comment = "";
  } else {
    $comment = test_input($_POST["comment"]);
  }

  if (empty($_POST["gender"])) {
    $genderErr = "Gender is required";
  } else {
    $gender = test_input($_POST["gender"]);
  }
}
?>
```

PHP Required Fields (5/6)

- Adding error messages if required fields are empty:

```
<form method="post" action="<?php echo
htmlspecialchars($_SERVER["PHP_SELF"]);?>">
Name: <input type="text" name="name">
<span class="error">* <?php echo $nameErr;?
></span> <br><br>
E-mail: <input type="text" name="email">
<span class="error">* <?php echo
$emailErr;?></span> <br><br>
```

PHP Required Fields (6/6)

```
Website: <input type="text" name="website">
<span class="error"><?php echo
$websiteErr;?></span> <br><br>
Comment: <textarea name="comment" rows="5"
cols="40"></textarea> <br><br>
Gender: <input type="radio" name="gender"
value="female">Female
<input type="radio" name="gender"
value="male">Male
<input type="radio" name="gender"
value="other">Other
<span class="error">* <?php echo
$genderErr;?></span> <br><br>
<input type="submit" name="submit" value="Submit">
</form>
```

PHP Validate Name

- Check if the name field only contains letters and whitespace. If the value of the name field is not valid, then store an error message:

```
$name = test_input($_POST["name"]);
if (!preg_match("/^[a-zA-Z ]*$/",$name)) {
  $nameErr = "Only letters and white space allowed";
}
```

PHP Validate E-mail

- Check whether an email address is well-formed. Use PHP's filter_var() function.

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

PHP Validate URL

- Check if a URL address syntax is valid (allowing dashes in the URL). If the URL address syntax is not valid, then store an error message:

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

PHP Keeping Form Values (1/3)

- For keeping the values in the input fields when the user hits the submit button:
- Add a small PHP script inside the value attribute of the following input fields: name, email, and website.
- In the comment textarea field, put the script between the <textarea> and </textarea> tags.
- These scripts output the value of the $name, $email, $website, and $comment variables.

PHP Keeping Form Values (2/3)

```
Name: <input type="text" name="name"
value="<?php echo $name;?>">


E-mail: <input type="text" name="email"
value="<?php echo $email;?>">


Website: <input type="text" name="website"
value="<?php echo $website;?>">


Comment: <textarea name="comment" rows="5"
cols="40"><?php echo $comment;?></textarea>
```

PHP Keeping Form Values (3/3)

```
Gender: <input type="radio" name="gender"
<?php if (isset($gender) &&
$gender=="female") echo "checked";?>
value="female">Female
<input type="radio" name="gender"
<?php if (isset($gender) &&
$gender=="male") echo "checked";?>
value="male">Male
<input type="radio" name="gender"
<?php if (isset($gender) &&
$gender=="other") echo "checked";?>
value="other">Other
```

PHP and MySQL Databases (1/x)

- PHP is the programming language that can connect to and manipulate databases.
- MySQL is the most popular database system used with PHP. With XAMPP it is included MariaDB.
- MariaDB is a binary drop in replacement of the same MySQL version (e.g. MySQL 5.7 is compatible with MariaDB 10.2
- MySQL and MariaDB use the same SQL (Structured Query Language)

PHP and MySQL Databases (2/x)

- MySQL is a database system that runs on a server.
- MySQL is ideal for both small and large applications.
- MySQL is very fast, reliable, and easy to use.
- MySQL is multi-platform.

PHP and MySQL Databases (3/x)

- MySQL is free to download and use.
- MySQL is developed, distributed, and supported by Oracle Corporation.
- PHP combined with MySQL are cross-platform (you can develop in Windows and serve on a Unix platform)

PHP and MySQL Databases (4/x)

- Database queries: A query is a question or a request.
- We can query a database for specific information and have a recordset returned.
- Look at the following query (using standard SQL):

```
SELECT LastName FROM Employees
```

- This query selects all the data in the "LastName" column from the "Employees" table.