Trường Đại học Công nghệ Thông tin – Đại học Quốc gia Hồ Chí Minh
Bộ môn An toàn thông tin

PHP Basics

Ths. Trần Tuấn Dũng – dungtran@uit.edu.vn

What is PHP?

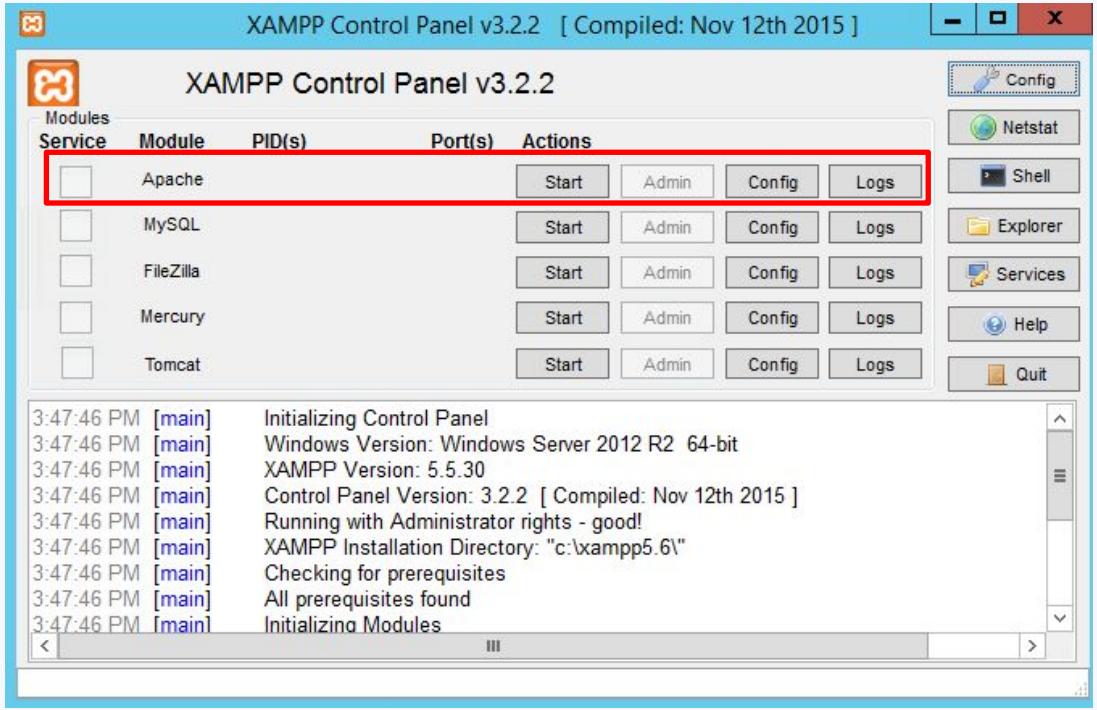
- PHP stands for "PHP Hypertext Preprocessor"
- Server-side scripting language
- Used to make web pages dynamic:
  - provide different content depending on context
  - interface with other services: database, e-mail, etc.
  - authenticate users
  - process form information
- PHP code can be embedded in HTML code

Why PHP?

- Free and open source
- Compatible
  - as of November 2006, there were more than 19 million websites (domain names) using PHP.
- Simple

How to use PHP

XAMPP Installers and Downloads for Apache Friends



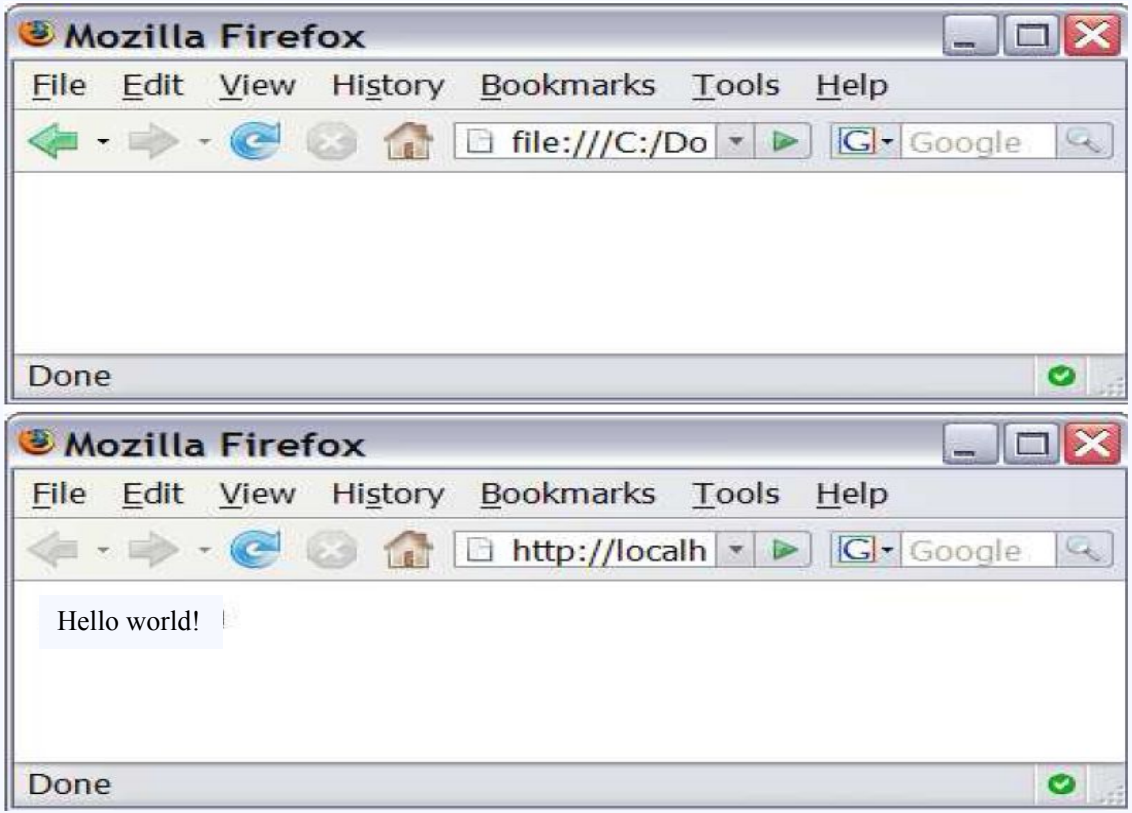
Hello World!

Go to your Xampp installation folder => htdocs => modify the **index.php** file

```
<?php
print "Hello, world!";
?>
```

Hello world!

Viewing PHP output



PHP syntax template

```
HTML content
<?php
PHP code
?>
HTML content
<?php
PHP code
?>
HTML content ...
```

- Contents of a .php file between <?php and ?> are executed as PHP code
- All other contents are output as pure HTML
- We can switch back and forth between HTML and PHP "modes"

Console output: print

```
print "text";
```

```
print "Hello, World!\n";
print "Escape \"chars\" are the SAME as in Java!\n";
print "You can have
line breaks in a string.";
print 'A string can use "single-quotes". It\'s cool!';
```

Hello world! Escape "chars" are the SAME as in Java! You can have line breaks in a string. A string can use "single-quotes". It's cool!

Variables

```
$name = expression;
```

```
$user_name = "mundruid78";
$age = 16;
$drinking_age = $age + 5;
$this_class_rocks = TRUE;
```

- names are case sensitive
- names always begin with $, on both declaration and usage
- always implicitly declared by assignment (type is not written)
- a loosely typed language

Variables

- basic types: *int, float, boolean, string, array, object, NULL*
  - test type of variable with is_type functions, e.g. is_string
  - gettype function returns a variable's type as a string
- PHP *converts between types automatically* in many cases:
  - string → int auto-conversion on +
  - int → float auto-conversion on /
- type-cast with **(type):**
  - $age = **(int)** "21";

Arithmetic operators

. + - * / % . ++ --

. = += -= *= /= %= .=

Many operators auto-convert types: 5 + "7" is 12

Comments

```
# single-line comment
// single-line comment
/*
multi-line comment
*/
```

String Type

```
$favorite_food = "Ethiopian";
print $favorite_food[2];
$favorite_food = $favorite_food . " cuisine";
print $favorite_food;
```

- zero-based indexing using bracket notation
- there is no char type; each letter is itself a String
- string concatenation operator is . (period), not +
  - 5 + "2 turtle doves" === 7
  - 5 . "2 turtle doves" === "52 turtle doves"
- can be specified with "" or ''

String Functions

```
# index 0123456789012345
$name = "Stefanie Hatcher";
$length = strlen($name);
$cmp = strcmp($name, "Brian Le");
$index = strpos($name, "e");
$first = substr($name, 9, 5);
$name = strtoupper($name);
```

String Functions (cont.)

Name	Java Equivalent
strlen	length
strpos	indexOf
substr	substring
strtolower, strtoupper	toLowerCase, toUpperCase
trim	trim
explode, implode	split, join
strcmp	compareTo

Interpreted Strings

```
$age = 16;
print "You are " . $age . " years old.\n";
print "You are $age years old.\n"; # You are 16 years old.
```

- strings inside " " (double quote) are interpreted
  - variables that appear inside them will have their values inserted into the string
- strings inside ' ` (single quote) are not interpreted:

```
print 'You are $age years old.\n'; # You are $age years old. \n
```

Interpreted Strings (cont.)

```
print "Today is your $ageth birthday.\n"; # $ageth not found
print "Today is your {$age}th birthday.\n";
```

Avoid ambiguity by enclosing variable in {}

Interpreted Strings (cont.)

```
$name = "Xenia";
$name = NULL;
if (isset($name)) {
print "This line isn't going to be reached.\n";
}
```

- A variable is **NULL** if
  - it has not been set to any value (undefined variables)
  - it has been assigned the constant NULL
  - it has been deleted using the unset function
- Can test if a variable is NULL using the isset function
- **NULL** prints as an empty string (no output)

for loop

```
for (initialization; condition; update) {
    statements;
}
```

```
for ($i = 0; $i < 10; $i++) {
    print "$i squared is " . $i * $i . ".\n";
}
```

bool (Boolean) type

```
$feels_like_summer = FALSE;
$php_is_great = TRUE;
$student_count = 7;
$nonzero = (bool) $student_count; # TRUE
```

- the following values are considered to be FALSE (all others are TRUE):
  - 0 and 0.0 (but NOT 0.00 or 0.000)
  - "", "0", and NULL (includes unset variables)
  - arrays with 0 elements
- FALSE prints as an empty string (no output); TRUE prints as a 1

if/else

```
if (condition) {
    statements;
} elseif (condition) {
    statements;
} else {
    statements;
}
```

Ternary syntax

```
$var = condition ? <value if true> : <value if false>
```

```
$var = condition ? : <value if false> // value if false
is default value
```

```
$var == 1 ? "One" : "No one"
```

```
$var == 1 ? : "Hello"
```

while loop

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

```
do {
    statements;
} while (condition);
```

Math operations

```
$a = 3;
$b = 4;
$c = sqrt(pow($a, 2) + pow($b, 2));
```

math functions

abs	ceil	cos	floor	log	log10	max
min	pow	rand	round	sin	sqrt	tan

math constants

M_PI	M_E	M_LN2

Int and Float Types

```
$a = 7 / 2; # float: 3.5
$b = (int) $a; # int: 3
$c = round($a); # float: 4.0
$d = "123"; # string: "123"
$e = (int) $d; # int: 123
```

- int for integers and float for reals
- division between two int values can produce a float

Associative Arrays

```
$car = array("brand"=>"Ford", "model"=>"Mustang", "year"=>1964);
```

```
echo $car["model"]; //Mustang
$car["year"] = 2024; // will change year key
```

Arrays

```
$name = array();          # create
$name = array(value0, value1, ..., valueN);
$name[index]          # get element value
$name[index] = value;     # set element value
$name[] = value;          # append
```

```
$a = array();    # empty array (length 0)
$a[0] = 23;      # stores 23 at index 0 (length 1)
$a2 = array("some", "strings", "in", "an",
"array");
$a2[] = "Ooh!";    # add string to end (at index 5)
```

- Append: use bracket notation without specifying an index
- Element type is not specified; can mix types

Array functions

function name(s)	description
count	number of elements in the array
print_r	print array's contents
array_pop, array_push, array_shift, array_unshift	using array as a stack/queue
in_array, array_search, array_reverse, sort, rsort, shuffle	searching and reordering
array_fill, array_merge, array_intersect, array_diff, array_slice, range	creating, filling, filtering
array_sum, array_product, array_unique, array_filter, array_reduce	processing elements

Array function

```
$tas = array("MD", "BH", "KK", "HM", "JP");
for ($i = 0; $i < count($tas); $i++) {
    $tas[$i] = strtolower($tas[$i]);
}
$morgan = array_shift($tas);
array_pop($tas);
array_push($tas, "ms");
array_reverse($tas);
sort($tas);
$best = array_slice($tas, 1, 2);
```

- the array in PHP replaces many other data structures
  - list, stack, queue, set, map, ...

foreach loop

```
foreach ($array as $variableName) {
    ...
}
```

```
$fellowship = array("Frodo", "Sam", "Gandalf",
"Strider", "Gimli", "Legolas", "Boromir");
print "The fellowship of the ring members are: \n";
for ($i = 0; $i < count($fellowship); $i++) {
    print "{$fellowship[$i]}\n";
}
print "The fellowship of the ring members are: \n";

foreach ($fellowship as $fellow) {
    print "$fellow\n";
}
```

foreach loop

```
$car = array("brand"=>"Ford", "model"=>"Mustang", "year"=>1964);

foreach ($car as $key => $value) {
  echo "$key: $value <br>";
}
```

brand: Ford
model: Mustang
year: 1964

In PHP, a best practice is that we should always prioritize foreach loop over for($i)

Multidimensional Arrays

```
<?php $AmazonProducts = array( array("BOOK", "Books", 50),
                    array("DVDs", "Movies", 15),
                    array("CDs", "Music", 20)
                   );
for ($row = 0; $row < 3; $row++) {
    for ($column = 0; $column < 3; $column++) { ?>
        <p> | <?= $AmazonProducts[$row][$column] ?>
    <?php } ?>
      </p>
<?php } ?>
```

Printing HTML tags in PHP = bad style

```
<?php
print "<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML 1.1//EN\"\n";
print " \"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd\">\n";
print "<html xmlns=\"http://www.w3.org/1999/xhtml\">\n";
print " <head>\n";
print " <title>Geneva's web page</title>\n";
...
for ($i = 1; $i <= 10; $i++) {
print "<p> I can count to $i! </p>\n";
}
?>
```

- best PHP style is to minimize print/echo statements in embedded PHP code
- but without **print**, how do we insert dynamic content into the page?

PHP expression blocks

```
<?= expression ?> // one line
```

```
<?php print expression; ?> // more than one line
```

```
<h2> The answer is <?= 6 * 7 ?> </h2>
```

The answer is 42

- PHP expression block: a small piece of PHP that evaluates and embeds an expression's value into HTML

Expression block example

- We can break PHP code into multiple **<?php ?>** block and insert HTML between them
- The HTML Block inside for loop will be print out

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>CSE 190 M: Embedded PHP</title></head>
<body>
<?php
for ($i = 99; $i >= 1; $i--) {
?>
<p> <?= $i ?> bottles of beer on the wall, <br />
<?= $i ?> bottles of beer. <br />
Take one down, pass it around, <br />
<?= $i - 1 ?> bottles of beer on the wall. </p>
<?php
}
?>
</body>
</html>
```

Functions

```
function name(parameterName, ..., parameterName) {
statements;
}
```

```
function quadratic($a, $b, $c) {
    return -$b + sqrt($b * $b - 4 * $a * $c) / (2 *
$a);
}
```

- parameter types and return types are not written
- a function with no return statements implicitly returns **NULL**

Default Parameter Values

```
function print_separated($str,  $separator = ", ") {
    if (strlen($str) > 0) {
        print $str[0];
        for ($i = 1; $i < strlen($str); $i++) {
            print $separator . $str[$i];
        }
    }
}
```

```
print_separated("hello"); # h, e, l, l, o
print_separated("hello", "-"); # h-e-l-l-o
```

PHP Include File

- Insert the content of one PHP file into another PHP file before the server executes it
- Use the
  - `include()` generates a warning, but the script will continue execution
  - `require()` generates a fatal error, and the script will stop
  - `include_once()` same as `include()` but will check file already included, if not include it
  - `require_once()` same as `require()` but will check file already required, if not require it

include() example

```
<a href="/default.php">Home</a>
<a href="/tutorials.php">Tutorials</a>
<a href="/references.php">References</a>
<a href="/examples.php">Examples</a>
<a href="/contact.php">Contact Us</a>
```

```
<html>
<body>

<div class="leftmenu">
<?php include("menu.php"); ?>
</div>

<h1>Welcome to my home page.</h1>
<p>I have a great menu here.</p>

</body>
</html>
```

PHP file I/O functions

function name(s)	category
file, file_get_contents, file_put_contents	reading/writing entire files
basename, file_exists, filesize, fileperms, filemtime, is_dir, is_readable, is_writable, disk_free_space	asking for information
copy, rename, unlink, chmod, chgrp, chown, mkdir, rmdir	manipulating files and directories
glob, scandir	reading directories

Unpacking an array: list

```
list($var1, ..., $varN) = array;
```

```
$values = array("mundruid", "18", "f", "96");
...
list($username, $age, $gender, $iq) = $values;
```

- the list function accepts a comma-separated list of variable names as parameters
- use this to quickly "unpack" an array's contents into several variables

Splitting/joining strings

```
$array = explode(delimiter, string);
$string = implode(delimiter, array);
```

```
$class = "CS 380 01";
$class1 = explode(" ", $s); # ("CS", "380", "01")
$class2 = implode("...", $a); # "CSE...380...01"
```

- explode and implode convert between strings and arrays

Example explode

```
Harry Potter, J.K. Rowling
The Lord of the Rings, J.R.R. Tolkien
Dune, Frank Herbert
```

```
<?php foreach (file("books.txt") as $book) {
    list($title, $author) = explode(",", $book);
    ?>
    <p> Book title: <?= $title ?>, Author: <?=
$author ?> </p>
<?php
}
?>
```

Reading directories

function	description
scandir	returns an array of all file names in a given directory (returns just the file names, such as "`myfile.txt`")
glob	returns an array of all file names that match a given pattern (returns a file path and name, such as "`foo/bar/myfile.txt`")

Example for glob

```php
# reverse all poems in the poetry directory
$poems = glob("poetry/poem*.dat");
foreach ($poems as $poemfile) {
    $text = file_get_contents($poemfile);
    file_put_contents($poemfile, strrev($text));
    print "I just reversed " . basename($poemfile);
}
```

- glob can match a "wildcard" path with the * character
- the basename function strips any leading directory from a file path

Example for scandir

```php
<ul>
<?php
$folder = "taxes/old";
foreach (scandir($folder) as $filename) {
    ?>
    <li> <?= $filename ?> </li>
<?php
}
?>
</ul>
```

PHP

- .
- ..
- 2009_w2.pdf
- 2007_1099.doc

output

Exception

```php
<?php
//create function with an exception
function checkStr($str)
  {
  if(strcmp($str, "correct")!= 0)
    {
    throw new Exception("String is not correct!");
    }
  return true;
  }

//trigger exception
checkStr("wrong");
?>
```

PHP

Try catch

```
<?php
//create function with an exception
function checkStr($str)
  {
  …
  }

//trigger exception in a "try" block
try
  {
  checkStr(“wrong”);
  //If the exception is thrown, this text will not be shown
  echo 'If you see this, the string is correct';
  }

//catch exception
catch(Exception $e)
  {
  echo 'Message: ' .$e->getMessage();
  }
?>
```

Superglobals

- $GLOBALS: Access to most outer scope variables
- $_SERVER: Access related server information
- $_REQUEST: Access request information
- $_POST
- $_GET
- $_FILES
- $_ENV
- $_COOKIE: Access cookies key value
- $_SESSION: Access session key value

Cookies

- Problem: HTTP is stateless
- What is a cookie?
  - tiny bits of information that a web site could store on the client's machine
  - they are sent back to the web site each time a new page is requested by this client.
- Urban myth: tracking, violate privacy
- Reality:
  - cookies are relatively harmless
  - can only store a small amount of information

Sessions

- What is a session?
  - a combination of a server-side cookie and a client-side cookie,
  - the client-side cookie contains only a reference to the correct data on the server.
- when the user visits the site:
  - their browser sends the reference code to the server
  - the server loads the corresponding data.

Cookies vs Sessions

- Cookies can be set to a long lifespan
- Cookies work smoothly when you have a cluster of web servers
- Sessions are stored on the server, i.e. clients do not have access to the information you store about
- Session data does not need to be transmitted with each page; clients just need to send an ID and the data is loaded from the local file.
- Sessions can be any size you want because they are held on your server

Create a cookie

```
setcookie(name, value, expire, path, domain);
```

```
<?php
setcookie("user", "Harry Poter", time()+3600);
?>

<html>
.....
```

Work with a Cookie Value

```
<?php
// Print a cookie
echo $_COOKIE["user"];

// A way to view all cookies
print_r($_COOKIE);
?>
```

Start/end a session

- All your session data is stored in the session superglobal array, `$_SESSION`

```
bool session_start ( void )
bool session_destroy ( void )
```

```
$_SESSION['var'] = $val;
$_SESSION['FirstName'] = "Jim";
```