ĐẠI HỌC
BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY
OF SCIENCE AND TECHNOLOGY

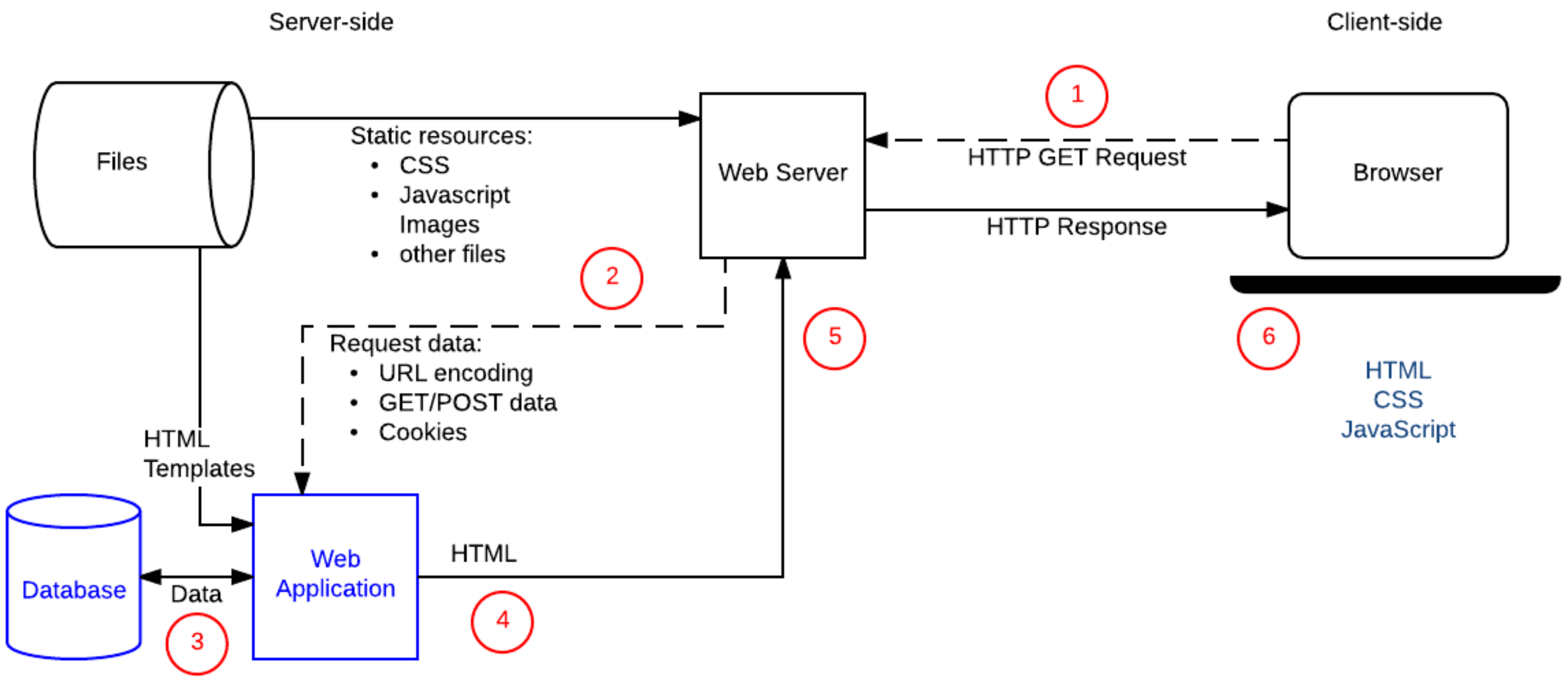
PHP INTRODUCTION

ONE LOVE. ONE FUTURE.

Content

- Introduction to PHP
- Basic PHP syntax
- Some useful PHP functions
- How to create a basic checker for user-entered data

Example of a dynamic website



src: https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Introduction

Server-side vs Client-side programming

- Different purposes
  - Client-side code: improve apperance and behavior: UI, layout, form validation
  - Server-side code: choose which content is returned to client

- Different programming languages (except JavaScript)
  - Client-side code: HTML, CSS, JavaScript
  - Server-side code: PHP, Python, Ruby, C#, JavaScript

- Different operating system enviroments
  - Client-side code: run inside a browser and has limited access to underlying operating systems
  - Server-side code: full access to server operating systems

Introduction to PHP

- Developed in 1995 by Rasmus Lerdorf (member of the Apache Group)
  - originally designed as a tool for tracking visitors at Lerdorf's Web site
  - widely-used, runs on vaious platforms (Windows, Linux, Mac OS)
  - supports a wide range of databases (MySQL, SQL Server)
- PHP is similar to JavaScript, only it’s a server-side language
  - PHP code is embedded in HTML using tags
  - the server executes the PHP code, substitutes output into the HTML page
  - the resulting page is then downloaded to the client
  - user never sees the PHP code, only the output in the page
- The acronym PHP means Hypertext Preprocessor

Example

```
<!DOCTYPE html>
<html>
<head>
   <title>Example</title>
</head>
<body>
<?php
    // start of PHP code
    echo "Hi, I'm a PHP script!";
?>
</body>
</html>
```

A PHP scripting block always starts with **<?php** and ends with **?>**. A PHP scripting block can be placed (almost) anywhere in an HTML document.

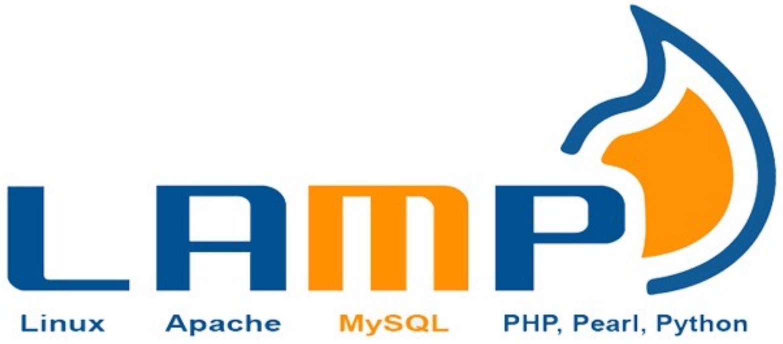
print and **echo** for output

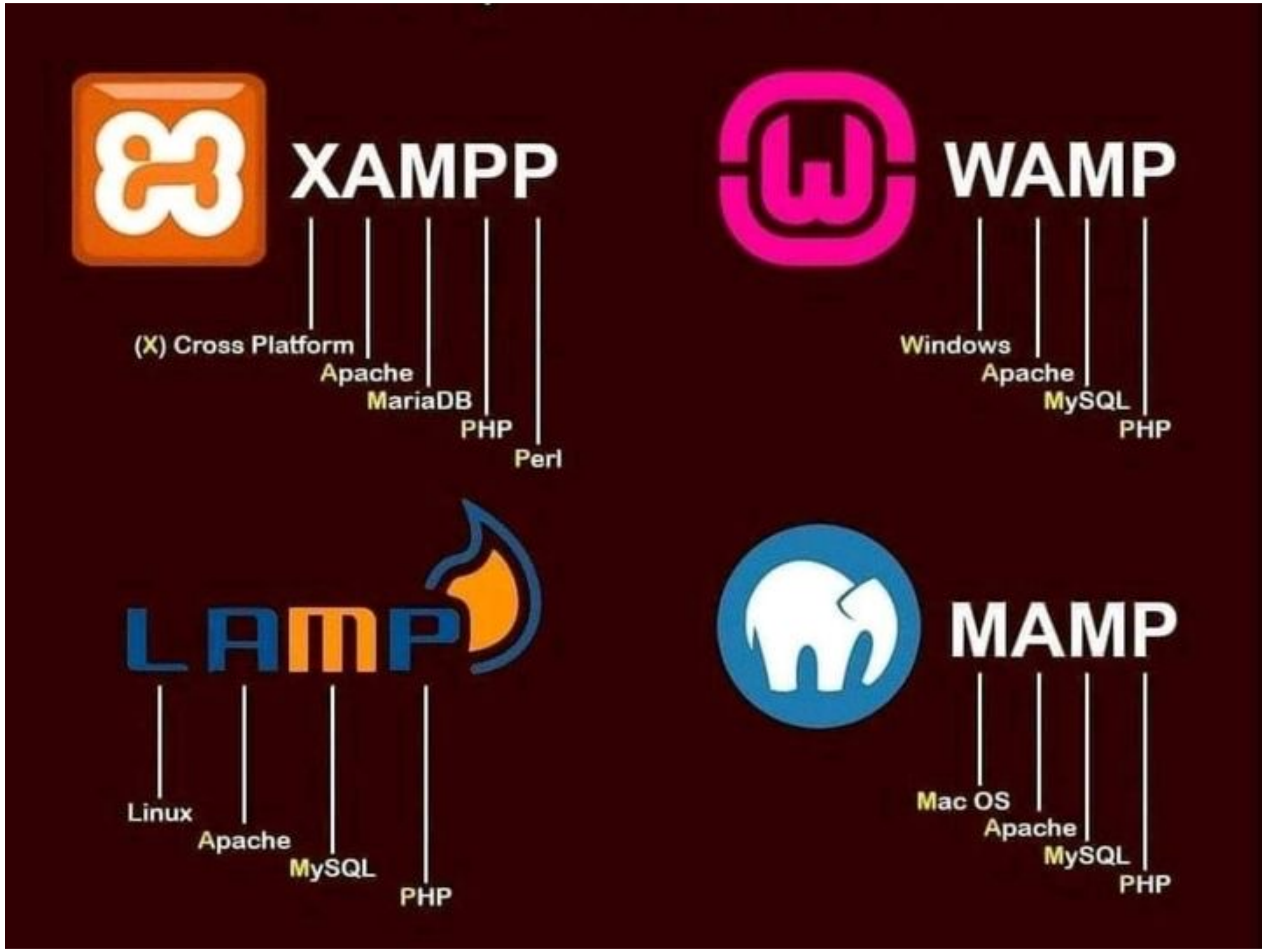
a semicolon (;)
at the end of each statement

The server executes the **print** and **echo** statements, substitutes output.

Set up PHP

- https://www.php.net/manual/en/install.php
  - Install a web server
  - Install PHP
  - Install a DBMS





Variable

- A variable starts with the $ sign, followed by the name of the variable
- Example

```
<?php
   $txt = "Hello world!";
   $x = 5;
   $y = 10.5;
?>
```

- Rules
  - Permissioned characters for name: A-z, 0-9, and _
  - Names must start with a letter (0-9) or the underscore character (_)
  - Names are case-sensitive ($a and $A are two different variables)

Variable Scope

- Three scopes: local, global, and static

```
<?php
  function myTest() {
    $x = 5; // local scope can only
    used inside the function
  }
  myTest();
  // using x will generate an error
  echo "<p>Variable x is: $x</p>";
?>
```

local scope

```
<?php
  $x = 5; // global scope can only be
   used outside the function

  function myTest() {
    // using x will generate an error
    echo "<p>Variable x is: $x</p>";
  }
?>
```

global scope

Variable Scope

- Three scopes: local, global, and static

```
<?php
   $x = 5; // global scope
  function myTest() {
    global $x; //global keyword
    allows to access a global variable
    echo "<p>Variable x is: $x</p>";
  }
?>
```

global keyword

```
<?php
  $x = 5;
  $y = 10;
  function myTest() {
    $GLOBALS['y'] = $GLOBALS['x’] +
                    $GLOBALS['y’];
  }
?>
```

GLOBAL array

Variable Scope

- Three scopes: local, global, and static

```
<?php
  function myTest() {
      static $x = 0;
      echo $x;
      $x++;
}
myTest();
myTest();
```

- The varible is local to the function
- The variable does not lose its value

static scope

Superglobal Variables

- Always available, can access them from any funtion, class, file
  - $GLOBALS
  - $_SERVER
  - $_REQUEST
  - $_POST
  - $_GET
  - $_FILES
  - $_ENV
  - $_COOKIE
  - $_SESSION

Superglobal Variables

```
<?php
echo $_SERVER['PHP_SELF'];
echo "<br>";
echo $_SERVER['SERVER_NAME'];
echo "<br>";
echo $_SERVER['HTTP_HOST'];
?>
```

$_SERVER

```
<?php
echo 'Username: ' .$_ENV['USER]';
?>
```

$_ENV

Superglobal Variables - COOKIE

- A small file that the server embeds on the user's computer
- Each time the same computer requests a page with a browser, it will send the cookie too

```
<?php
$cookie_name = "user";
$cookie_value = "John Doe";
setcookie($cookie_name, $cookie_value, time() + (86400 * 30), "/");
?>
<html>
<body>
<?php
if(!isset($_COOKIE[$cookie_name])) {
  echo "Cookie named '" . $cookie_name . "' is not set!";
} else {
  echo "Cookie '" . $cookie_name . "' is set!<br>";
  echo "Value is: " . $_COOKIE[$cookie_name];
}
?>
```

Superglobal Variables - COOKIE

- Delete a cookie by setting expiration data in the past

```
<?php
// set the expiration date to one hour ago
setcookie("user", "", time() - 3600);
?>
```

Data types

- Built-in types
  - null
  - Scalar types
    - bool
    - int
    - float
    - string
  - array
  - object
  - resource
  - never //functions never return a value, always throw an exception or teminate
  - void //allows calling return without an explicit value
- User-defined types
  - Interfaces
  - Classes
  - Enumerations

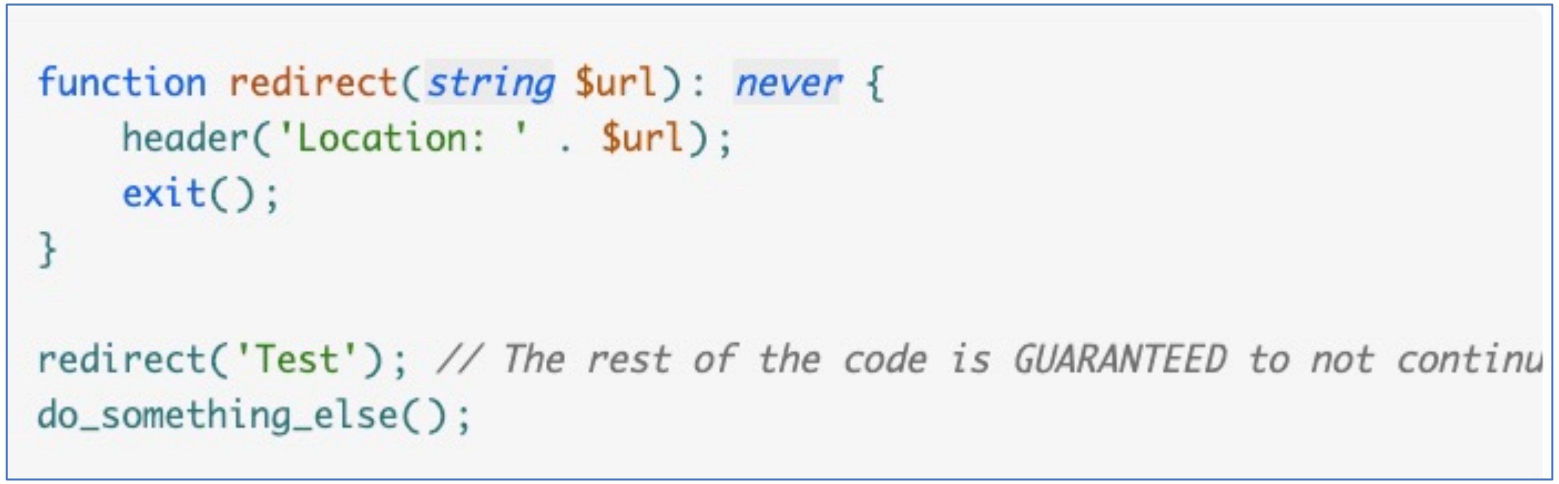
String

```
<?php
   echo ”Hi\n"; // Hi
   echo ‘Hi’;      // Hi
 ?>
```

```
<?php
   echo "This’s \"Peter\". \n"; //This’s "Peter".
  echo ‘This\'s \"Peter\".’;     //This's \"Peter\"
?>
```

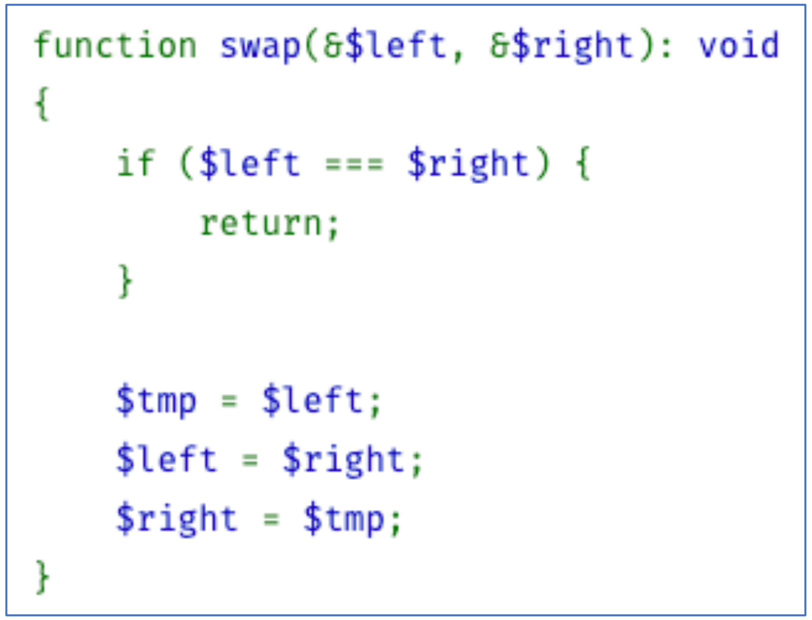
```
<?php
   $name = "Bond";
   echo "The name is $name. \n"; //The name is Bond.
   echo 'The name is $name.’;     //The name is $name.
?>
```

Never vs Void



```
function redirect(string $url): never {
    header('Location: ' . $url);
    exit();
}

redirect('Test'); // The rest of the code is GUARANTEED to not continu
do_something_else();
```

A function with the never return type *must* prevent the rest of the code in scope from being executed



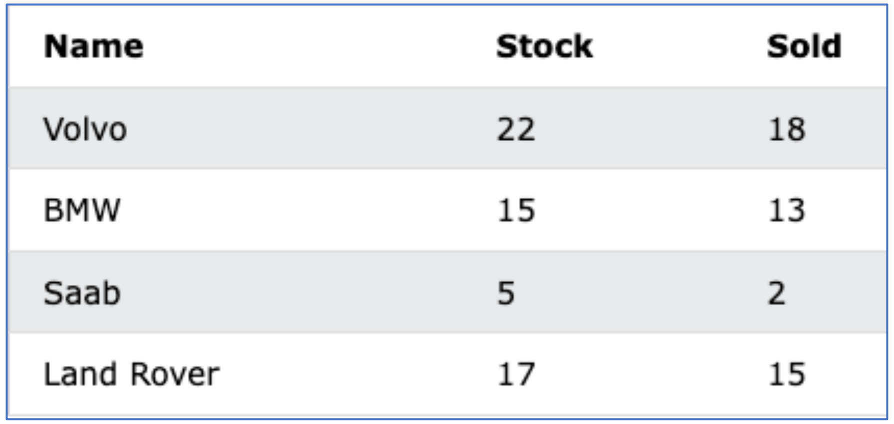
```
function swap(&$left, &$right): void
{
    if ($left === $right) {
        return;
    }

    $tmp = $left;
    $left = $right;
    $right = $tmp;
}
```

Functions must either omit their return statement altogether, or use an empty return statement

Array

- Use array() function
- Indexed array: two ways for declaration
  - `$cars = array("Volvo", "BMW");`
  - `$cars[0] = "Volvo";`
    `$cars[1] = "BMW";` As
- Asociative array: uses key-value pairs
  - `$age = array("Peter"=>"35", "Ben"=>"37");`
  - `$age['Peter'] = "35";`
    `$age['Ben'] = "37";`
- Multidimensional array

  -
    ```
    $cars = array (
       array("Volvo",22,18),
       array("BMW",15,13)
    );
    ```



Name	Stock	Sold
Volvo	22	18
BMW	15	13
Saab	5	2
Land Rover	17	15

Array: sorting

- Can sort in alphabetical or numerical order, descending or ascending
- Functions
  - sort() - sort arrays in ascending order
  - rsort() - sort arrays in descending order
  - asort() - sort associative arrays in ascending order, according to the value
  - ksort() - sort associative arrays in ascending order, according to the key
  - arsort() - sort associative arrays in descending order, according to the value
  - krsort() - sort associative arrays in descending order, according to the key

Operators

- **Arithmetic Operators:** +, -, *,/ , %, ++, --
- **Assignment Operators:** =, +=, -=, *=, /=, %=

Example	Is the same as
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

- **Comparison Operators:** ==, !=, >, <, >=, <=
- **Logical Operators:** &&, ||, !
- **String Operators:** . and .= (for string concatenation)

```
$a = "Hello ";
$b = $a . "World!"; // now $b contains "Hello World!"

$a = "Hello ";
$a .= "World!";
```

Control Statements: if else

```
<?php
  $d=date("D");
  echo $d, "<br/>";
  if ($d=="Fri")
   echo "Have a nice weekend! <br/>";
  else
   echo "Have a nice day! <br/>";

?>
```

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

date() is a built-in PHP function that can be called with many different parameters to return the date in various formats

In this case we get a three letter string for the day of the week.

Control Statement: switch

```php
<?php
$x = rand(1,5);  // random integer
echo "x = $x <br/><br/>";
switch ($x)
{
case 1:
  echo "Number 1";
  break;
case 2:
  echo "Number 2";
  break;
case 3:
  echo "Number 3";
  break;
default:
  echo "No number between 1 and 3";
  break;
}
?>
```

```
switch (expression)
{
case label1:
  code for expression = label1;
  break;
case label2:
  code for expression = label2;
  break;
default:
  code for expression is different
  from both label1 and label2;
  break;
}
```

Loops

Loops: while, do while, for, foreach

```
<?php
$x = 1;

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

while

```
<?php

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

?>
```

for

```
<?php
$x = 1;

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

do while

```
<?php
 $colors = array("red", "green", "blue");

  foreach ($colors as $value) {
      echo "$value <br>";
 }
?>
```

for each

User Defined Functions

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

- A user-defined function declaration starts with the word function
- Function names are NOT case-sensitive
- PHP is a loosely typed language

```
<?php
function addNumbers(int $a, int $b) {
  return $a + $b;
}
echo addNumbers(5, "5 days");
// since strict is NOT enabled "5 days" is
changed to int(5), and it will return 10
?>
```

```
<?php declare(strict_types=1); // strict req
function addNumbers(int $a, int $b) {
  return $a + $b;
}
echo addNumbers(5, "5 days");
// since strict is enabled and "5 days" is not an
integer, an error will be thrown
?>
```

User Defined Functions

- Arguments are usually passed by value i.e., variable's value cannot be changed
- To pass by reference, we use the & operator

```php
<?php
function add_five($value) {
  $value += 5;
}
$num = 2;
add_five($num);
echo $num; //2
```

Pass by value

```php
<?php
function add_five(&$value) {
  $value += 5;
}
$num = 2;
add_five($num);
echo $num; //7
```

Pass by reference

File

- require vs include
  - Use require when the file is required by the application.
  - Use include when the file is not required and application should continue when file is not found.
- *include 'filename';* or *require 'filename';*

```
<?php
    $color='red';
    $car='BMW';
?>
```

vars.php

```
<!DOCTYPE html>
<html>
<body>
 <?php include 'vars.php';
   echo "I have a $color $car.";
?>
</body>
</html>
```

File Open - fopen(), fread(), fclose()

```
AJAX = Asynchronous
JavaScript and XML
CSS = Cascading Style Sheets
HTML = Hyper Text Markup
Language
```

webdictionary.txt

r	Read only	r+	Read/Write
w	Write only	w+	Read/Write
a	Append	a+	Read/Append
x	Create & open for write	x+	Create&open for read/write

modes

```php
<?php
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open
file!"); //open a file
echo fread($myfile,filesize("webdictionary.txt")); //read a file
fclose($myfile);//close an open file
?>
```

read file

File Open - fgets() and feof()

```php
<?php
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");
echo fgets($myfile); //read each line
fclose($myfile);
?>
```

fgets() function

```php
<?php
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");
// Output one line until end-of-file
while(!feof($myfile)) {
  echo fgets($myfile) . "<br>";
}
fclose($myfile);
?>
```

check end-of-file

File Open - fgetc()

```php
<?php
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");
// Output one line until end-of-file
while(!feof($myfile)) {
  echo fgetc($myfile); //read each character
}
fclose($myfile);
?>
```

check end-of-file

File Write - fopen(), fwrite()

```php
<?php
$myfile = fopen("newfile.txt", "w") or die("Unable to open file!");
$txt = "John Doe\n";
fwrite($myfile, $txt);
$txt = "Jane Doe\n";
fwrite($myfile, $txt);
fclose($myfile);
?>
```

fwrite()

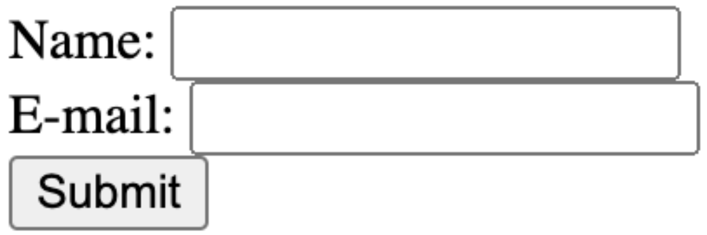
```
John Doe
Jane Doe
```

newfile.txt

Form Handling

PHP superglobals $_GET and $_POST are used to collect form-data

```html
<!DOCTYPE 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>
```



```html
<html>
<body>

Welcome <?php echo $_POST["name"]; ?><br>
Your email address is: <?php echo $_POST["email"]; ?>

</body>
</html>
```

`$_POST`
contains all POST data.

`$_GET`
contains all GET data.

PHP_SELF

$_SERVER["PHP_SELF"] returns the filename of the currently executing script => sends the submitted data to the page itself

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

- Cross-side scripting (XSS): attackers can inject commands to execute scripts

```
http://www.example.com/test_form.php/%22%3E%3Cscript%3Ealert('hacked')%3C/script%3E
```

$\downarrow$

```
http://www.example.com/test_form.php/"><script>alert('hacked')</script>
```

$\downarrow$

```
<form method="post" action="test_form.php/"><script>alert('hacked')</script>
```

htmlspecialchars()

- Converts the predefined characters to HTML entitiles
  - <   becomes &lt;   > becomes &gt;
  - &   becomes &amp;   " (double quotes) becomes &quot;
  - ' (single quotes) becomes &#039;
- Use htmlspecialchars() to avoid XSS

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

```
http://www.example.com/test_form.php/%22%3E%3Cscript%3Ealert('hacked')%3C/script%3E
```

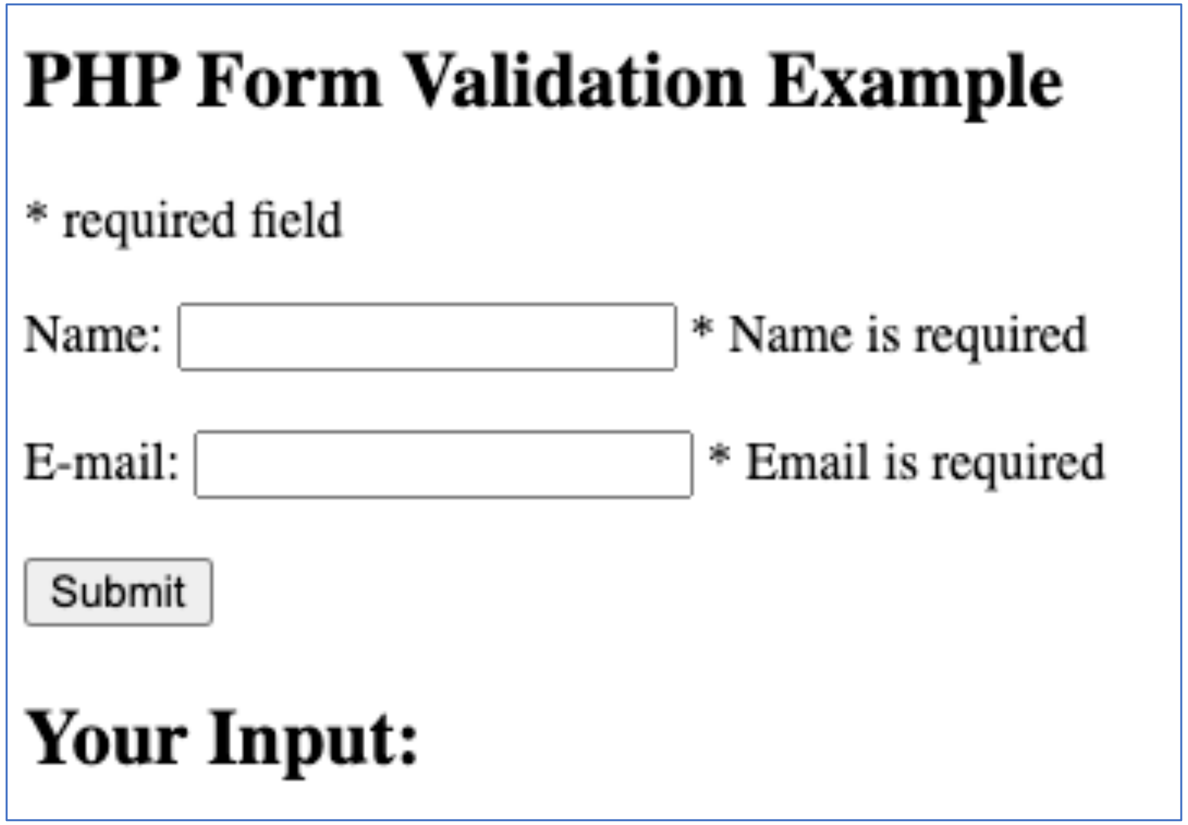
⇓

```
http://www.example.com/test_form.php/"><script>alert('hacked')</script>
```

⇓

```
<form method="post" action="test_form.php/&quot;&gt;&lt;script&gt;alert('hacked'
)&lt;/script&gt;">
```

Form Validation



PHP Form Validation Example

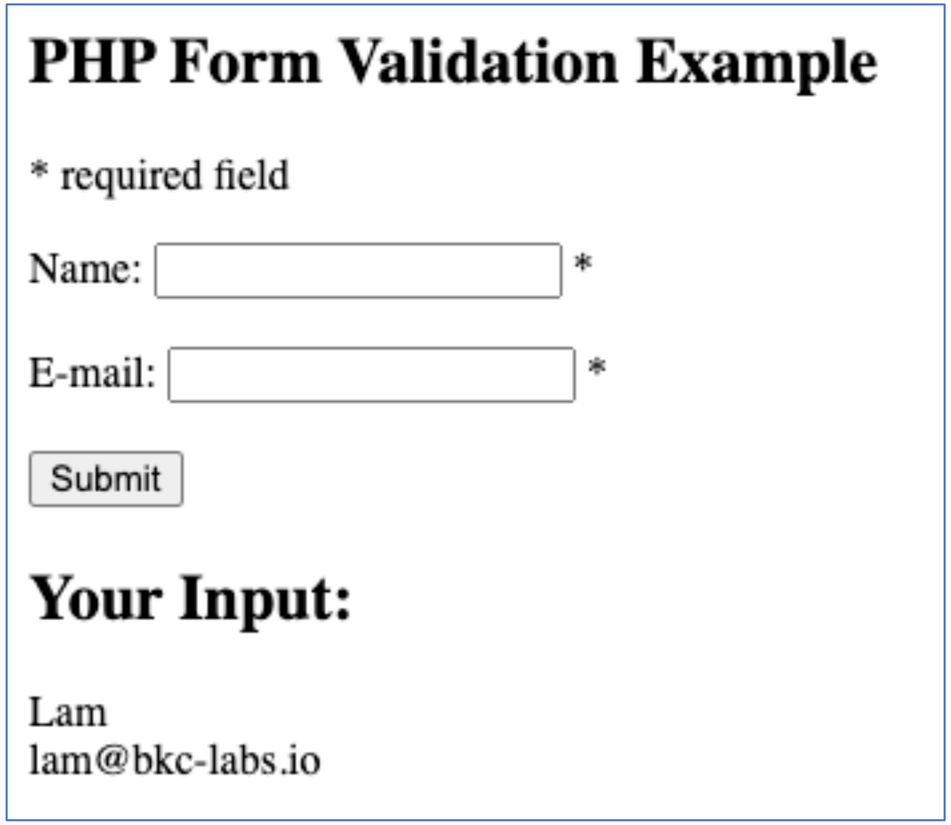
* required field

Name: * Name is required

E-mail: * Email is required

Submit

Your Input:



PHP Form Validation Example

* required field

Name: *

E-mail: *

Submit

Your Input:

Lam
lam@bkc-labs.io

Form Validation

```
<?php
// define variables and set to empty values
 $nameErr = $emailErr = "";
 $name = $email = "";

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

```
function test_input($data) {
  $data = trim($data);
  $data = stripslashes($data);
  $data = htmlspecialchars($data);
  return $data;
}
?>
```

Form Validation

```
<h2>PHP Form Validation Example</h2>
<p><span>* required field</span></p>

<form method="post" action="<?php echo
htmlspecialchars($_SERVER["PHP_SELF"]);?>">
  Name: <input type="text" name="name">
  <span>* <?php echo $nameErr;?></span>
  <br><br>

  E-mail: <input type="text" name="email">
  <span>* <?php echo $emailErr;?></span>
  <br><br>

  <input type="submit" name="submit”
  value="Submit">
</form>
```

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

Callback Functions

- a function which is passed as an argument into another function

```
<?php
function myfunction($v)
{
  return($v*$v);
}

$a=array(1,2,3,4,5);
print_r(array_map("myfunction",$a));
?>
```

array_map (*myfunction*, *array1*, *array2*, *array3*, ...) sends each value of an array to a user-defined function

Exceptions

```
try {
  code that can throw exceptions
} catch(Exception $e) {
  code that runs when an exception is caught
} finally {
  code that always runs regardless of whether an exception
was caught
}
```

Exceptions

```php
<?php
function divide($dividend, $divisor) {
  if($divisor == 0) {
    throw new Exception("Division by zero");
  }
  return $dividend / $divisor;
}
try {
  echo divide(5, 0);
} catch(Exception $e) {
  echo "Unable to divide. ";
} finally {
  echo "Process complete.";
}
?>
```

Exception Object contains information about the error

```
new Exception(message,
code, previous)
```

HUST

hust.edu.vn fb.com/dhbkhn

THANK YOU !