CT214H – Web Programming Fundamentals

# Chapter 4
# PHP (Hypertext Preprocessor)

Tran Cong An
(tcan@cit.ctu.edu.vn)

# Content

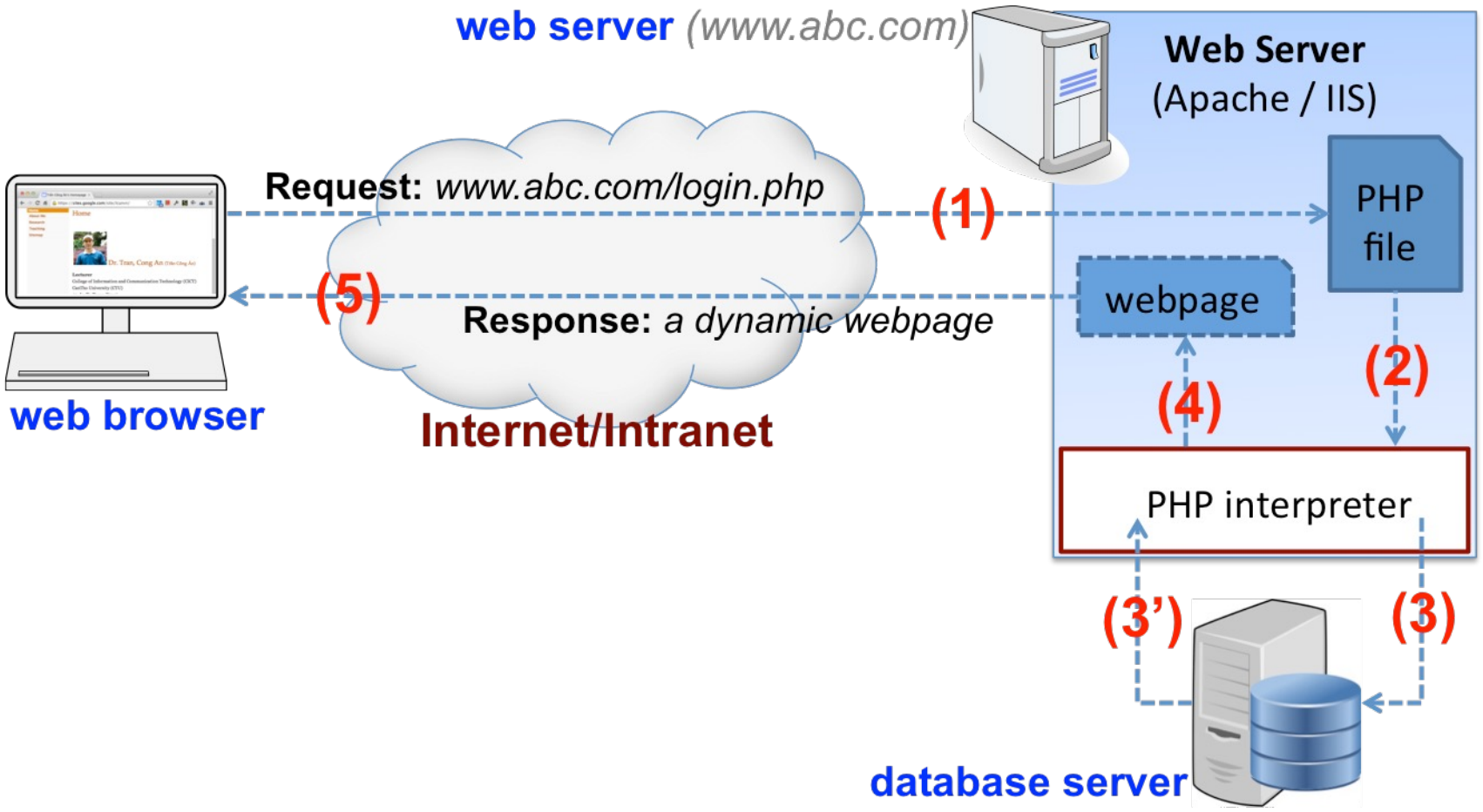
1. Introduction
2. PHP language basics
3. PHP functions
4. OOP in PHP
5. PHP and forms
6. PHP and MySQL
7. Cookies and sessions
8. Advanced PHP techniques (file upload, pagination, AJAX, etc.)
9. Appendix

# Introduction to PHP

# What is PHP?

- PHP is a server-side scripting language:
  - Scripts are embedded in HTML documents
    *(or: scripts to generate HTML)*
  - They are processed by server before returned to browser
- Basic characteristics:
  - Widely used and open source scripting language
  - Executed on the server
  - Dynamically typed and purely interpreted
  - Supported by most of popular web servers (IIS, Apache, etc.) and OS (Windows, Linux, MacOS, etc.)

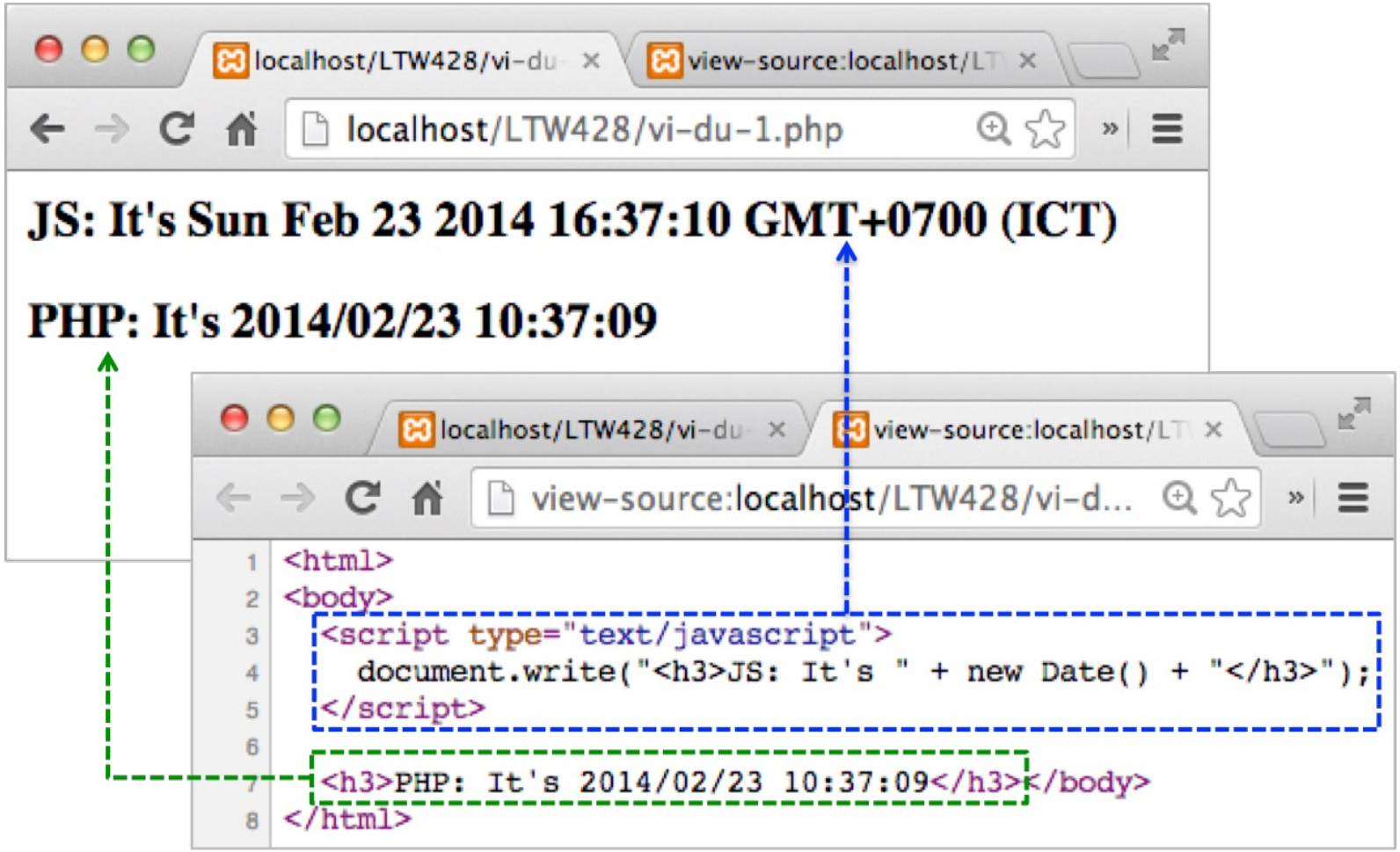
# What is PHP?


web server (www.abc.com)
Web Server
(Apache / IIS)
Request: www.abc.com/login.php
(1)
PHP file
(5)
webpage
Response: a dynamic webpage
(2)
(4)
web browser
Internet/Intranet
PHP interpreter
(3')
(3)
database server

# PHP vs. JavaScript

```
//vi-du-1.php
<html>
  <body>
    <script type="text/javascript">
      document.write("<h3>JS: It's " + new Date() + "</h3>");
    </script>
    <?php
      echo("<h3>PHP: It's " . date('Y/m/d H:i:s') . "</h3>");
    ?>
  </body>
</html>
```

# PHP vs. JavaScript



# What Can PHP Do?

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

# Why PHP?

- PHP runs on various platforms (Windows, Linux, Unix, MacOS, etc.)
- PHP is compatible with almost all servers used today (Apache, IIS, etc.)
- PHP supports a wide range of databases
- PHP is free.
- PHP is easy to learn and runs efficiently on the server side

# PHP Development Environment

- Suggested development tools:
  - Web server: Apache (http://httpd.apache.org/download.cgi)
  - PHP interpreter (http://www.php.net/downloads.php)
  - DBMS: MySQL/MariaDB (http://www.mysql.com/downloads/)

- Setting-up development environment:
  - Option 1: download and install the above tools separately and configure them to let them to be able to "talk" to each other
  - Option 2 (recommended): install a software that packages all above software (e.g. XAMPP, AMPPS, etc.)

# PHP Language Basics

# Basic Syntax

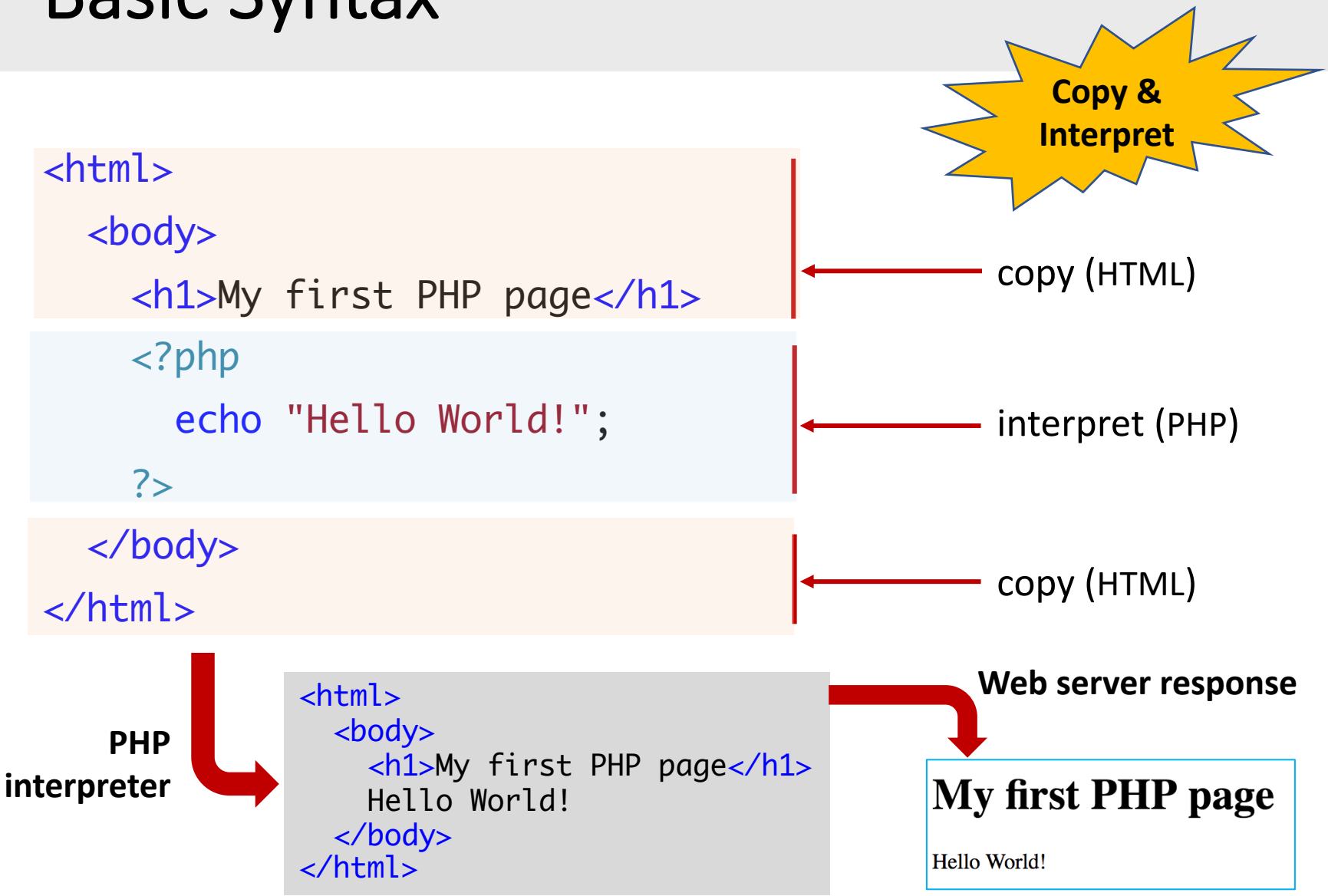
- A PHP file normally contains HTML tags, and some PHP scripting code (default PHP file extension: .php)
- PHP script can be placed anywhere in the document
- A PHP script starts with <?php and ends with ?>

```
<?php
    //php code goes here
?>
```

- PHP statements end with a semicolon ;
- Variable names are case sensitive, others are case insensitive

The PHP processor has two modes: **copy** (HTML) and **interpret** (PHP)

# Basic Syntax



# Variables

- Variable can be used without declaration (dynamically typed)
- Naming rules:
  - A variable start with a $, followed by the variable name
  - A variable name starts with a letter or the underscore
  - A variable name can contain only alpha-numeric characters and underscore

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

# Variables

- Scope (the parts of script where the variable can be used):
  - **Local**:
    - Created within a function
    - Can be accessed within that functions
  - **Global**:
    - Created outside any functions
    - Can only be accessed outside a function
    - The `global` keyword is used to access global variable within a function
    - All global variables can also be accessed by the associative array `$GLOBALS`

# Variables

```php
<?php
    $x = 5;
    $y = 10;

    function myTest() {
      global $x;
      $GLOBALS['y'] = $x + $GLOBALS['y'];
    }

    myTest();
    echo $y;
?>
```

Result: 15

# Variables

- Static variables:
  - Created inside a function
  - Not deallocated/deleted when the function completed (i.e. their values are retained for further usage in the latter runs)
- Variable datatypes:
  - Automatically assigned depending on its value (loosely typed)
  - To get the datatype of a variable: `gettype(var_name)`
  - Functions to check variable type: `is_bool(), is_int(),is_float(), is_double(), is_string(), is_object(),is_array(), is_numeric(), is_resource(), is_null(),isset(), empty()`

# Variables

```
<!-- datatype.php -->
<?php
    $f = 123.4;
    $s = "Hello world";
    $i = 100;
    echo("<p>");
    echo(gettype($f) . "<br>");  //double
    echo(var_dump($s) . "<br>"); //string(11) "Hello World"
    echo(is_int($i) . "<br>");   //int
    $i = 123.45;
    echo(var_dump($i)); //float(123.45)
    echo("</p>");
?>
```

# PHP Outputs

- `echo` and `print` statement are used to output data to screen
- **`echo`**: `echo(str); or echo str [,str...];`
  - Can take multiple parameters
  - Has no return value (i.e. cannot be used in expressions)

```
echo ($grade > 5) ? "pass" : "fail";
```

  - Marginally faster than `print()`
- **`print`**: `print(str); or print str;`
  - Can take only one parameter
  - Has return value of 1 (i.e. can be used in expressions)

```
($grade > 5) ? print("pass") : print("fail");
```

# Datatypes

- Common datatypes supported by PHP:
  - String (string values are enclosed in single or double quotes)
  - Integer (32-bit or 64-bit number)
  - Float (platform-dependent but typically: 64-bit, with precision of 14 decimal digits)
  - Boolean (TRUE or FALSE values)
    - 0, 0.0, empty string , "0", NULL, empty array: FALSE
    - Others: TRUE
  - Array (can store multiple values with different datatypes)
  - Object (the class of the object must be declared first)

# Operators

- Arithmetic: +, –, *, /, %, ++, --, ** (exponentiation)
- Assignment: =, +=, -=, *=, /=
- Comparison: == (equal), === (identical), !=, <>, !==, >, <, >=, <=
- Logical: and, or, xor, &&, ||, !
- String: . (concatenation), .=
- Array: + (union), ==, ===, !=, !==, <>

# Strings

- String values are enclosed in single quotes, double quotes, or heredoc (starts with <<< followed by an identifier, ends with the identifier)
- Example:

```
$str = <<<__ABC
    This is a string
__ABC;
```

- A string value can be expanded in multiple lines
- Variables and escape sequences will not be expanded when using single quote
- Some string functions: `strlen(), strpos(), strrev(),...`

# Constants

- A constant is an identifier (name) for a simple value
- A valid constant name starts with a letter or underscore (no $ sign before the constant name)
- Automatically global across the entire script
- To create a constant, use the `define()` function

```
define(name, value [, case-insensitive = FALSE])
```

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

# Control Statements

- Condition statements:
  - if ... else
  - switch ... case
  - ?
- Loops:
  - while
  - do … while
  - for
  - foreach

# Control Statements – if ...else

```
if (condition) {
   code to be executed if this condition is true;
}
[ elseif (condition) {
   code to be executed if this condition is true;
} ... ]
[ else {
   code to be executed if all conditions are false;
} ]
```

```
date_default_timezone_set("Asia/Ho_Chi_Minh");
$t = date("H");
if ($t < "10") {
   echo "Have a good morning!";
} elseif ($t < "20") {
   echo "Have a good day!";
} else {
   echo "Have a good night!";
}
```

# Control Statements – switch … case

```
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;
}
```

```
switch ($dw) {
  case 'Saturday':
  case 'Sunday':
    echo "Nice weekend!";
    break;
  default:
    echo "Nice weekday!";
}
```

# Control Statements – while

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

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

# Control Statements – do ... while

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

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

# Control Statements – for/foreach

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

foreach ($array as $value) {  //works only on arrays
   code to be executed;
}
```

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

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

# Functions

# Declarations

- A user-defined function is declared using the keyword `function`
- Syntax:

```
function function_name() {
    //function body (statements)
}
```

- Note:
  - A function name starts with a letter or underscore
  - Function name should reflect what the function does
  - Function names are NOT case-sensitive

# Example

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

      writeMsg();
      ?>
  </body>
</html>
```

```
<!DOCTYPE html>
<html>
  <body>
    Hello world!
  </body>
</html>
```

# Function Arguments

- Declared inside the parentheses, separated by commas

```php
<?php
  function ptb1($a, $b)
  {
      if ($a == 0) {
          if ($b == 0)
              echo ”Infintely many solutions";
          else
              echo ”No solution";
      }
      else
          echo ”Solution: x = " . $a / $b;

  }

  ptb1(1, 2); //output: Solution x = 0.5
  ptb1(0, 2); //output: No solution
  ptb1(0, 0); //output: Infintely many solutions
?>
```

```
Solution: x = 0.5
No solution
Infintely many solutions
```

# Function Arguments

- May have default values (should be declared at the end)

```
<html>
  <body>
    <?php
      function createPar($noOfPar = 1, $idStart=0) {
        for ($i=0; $i<$noOfPar; $i++) {
    ?>
      <p id='<?="par" . ($i+$idStart)?>'>... </p>
    <?php
        }  //for
      }  //createPar()

      createPar();
      createPar(2, 2);
    ?>
  </body>
</html>
```

```
<html>
  <body>
    <p id='par0'>...</p>
    <p id='par2'>...</p>
    <p id='par3'>...</p>
  </body>
</html>
```

# Function Arguments

- Functions arguments are pass-by-value by default
- Pass-by-reference: prepend an & to the argument name
- Example:

```php
<?php
  function add_some_extra(&$string) {
    $string .= 'and something extra.';
  }

  $str = 'This is a string, ';
  add_some_extra($str);
  echo $str; //outputs 'This is a string, and something extra.'
?>
```

# Function Arguments

- Related built-in functions:
  - `int func_num_args(void)`: get the number of arguments
  - `mixed func_get_arg(int $arg_num)`: get the argument value

```php
<?php
  function foo() {
    $argsnum = func_num_args();
    echo "Number of arguments: $argsnum; ";
    echo "Argument values: ";
    for ($i = 0; $i<$argsnum; $i++)
      echo func_get_arg($i) . "\t";
  }
  foo();      //output: Number of arguments: 0; Argument values:
  foo(1, 2); //output: Number of arguments: 2; Argument values: 1    2
?>
```

# Argument Type Declarations

- Type declaration allows functions to require that parameters are of a certain type at call time (int, float, string, array, boolean,…)
- To enable strict requirement: `declare(strict_types=1);`

```php
<?php declare(strict_types=1); // strict requirement

  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
?>
```

```
Fatal error: Uncaught TypeError: Argument 2 passed to
addNumbers() must be of the type integer, string given,
```

# Return Values

- A function always return one and only one value
- To return a value for a function call: `return <value>;`
  - Any types may be returned
  - If no `return` statement in a function, `NULL` will be returned

```php
<?php
  function square($num) {
      return $num * $num;
  }

  echo square(4);   // outputs '16'
?>
```

# Return Type Declarations

- Add a colon ":" and the type right before the "{" in the function declaration to specify the function return type
- To enable strict requirement: `declare(strict_types=1);`

```php
<?php declare(strict_types=1); // strict requirement
  function addNumbers($a, $b) : int {
      return $a + $b;
  }
  echo addNumbers(1, 1);
  echo addNumbers(1.2, 2.0);
?>
```

```
6
Fatal error: Uncaught TypeError: Return value of addNumbers
() must be of the type integer, float returned in /Users/tc
an/Desktop/code.php:3
```

# Arrays

# Arrays

- An array stores multiple values in one single variable
- Elements of an array may have difference data types

- Types of array:
  - Indexed arrays: arrays with a numeric index
  - Associative arrays: arrays with named keys
  - Multidimensional arrays: arrays containing one or more arrays

# Indexed Arrays

- Declaration:
  - Empty array: `$arr_name = array()`
  - Array with initial values: `$arr_name = array(val1, val2,…);`
- Access (set/get) array elements: `$arr_name[index]`
- Get array length: `count($arr_name)`
- Loop through an array:
  - `for` loop (usually combined with the `count()` function)
  - `foreach ($arr_name as $var)` loop

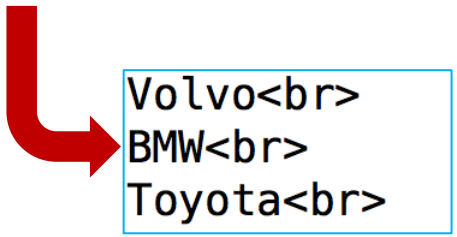
Note

✓ Array index may not be contiguous
✓ Index may be omitted, $a[]: a new available index will be used

# Indexed Arrays

```php
<?php
  $cars = array("Volvo", "BMW", "Toyota");
  $arrlength = count($cars);

  for($x = 0; $x < $arrlength; $x++) {
      echo $cars[$x];
      echo "<br>\n";
  }
?>
```



# Associative Arrays

- Two ways to create an associative array:
  - `$arr_name = array(key => value [, key => value …] );`
  - ```
    $arr_name['key1'] = value1;
    $arr_name['key2'] = value2;
    ```
- Access array elements: `$arr_name['key']`
- Get array key set: `array_keys($arr_name)`
- Loop through an associative array:
  - `foreach ($arr_name as $key => $value) { ... }`
  - `for` loop using the key set
- Remove a key/value pair: using `unset($arr[key])` function
    ```

# Associative Array Example

```php
<?php
  $age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");

  foreach ($age as $x => $x_value) {
    echo "Key=" . $x . ", Value=" . $x_value;
    echo "<br>";
  }
?>
```

```
Key=Peter, Value=35
Key=Ben, Value=37
Key=Joe, Value=43
```

```php
<?php
  //another method to loop through an associative array
  $keys = array_keys($age);
  for ($i=0; $i<count($keys); $i++) {
    echo $keys[$i] . ": " . $age[$keys[$i]] . "\n";
  }
?>
```

# Multi-dimensional Arrays

- A multi-dimensional array is an array of arrays
  - Two dimensional: an array of arrays
  - Three dimensional: an array of arrays of arrays
- Syntax:

```
$arr_name = array(
            array(…),
            array(…)
            [,…]);
```

- Arrays of an array may be different dimensions

# Multi-dimensional Arrays

```php
<?php
  $cars = array (
    array("Volvo",22,18), array("BMW",15,13),
    array("Saab",5,2), array("Land Rover",17,15));

  for ($row = 0; $row < 4; $row++) {
    echo "<p><b>Row number $row</b></p>";
    echo "<ul>";
    for ($col = 0; $col < 3; $col++) {
      echo "<li>".$cars[$row][$col]."</li>";
    }
    echo "</ul>";
  }
?>
```

**Row number 0**

- Volvo
- 22
- 18

**Row number 1**

- BMW
- 15
- 13

**Row number 2**

- Saab
- 5
- 2

**Row number 3**

- Land Rover
- 17
- 15

# Multi-dimensional Arrays

```php
<?php
  $fruits = array(
     "fruits" => array("a"=>"orange", "b"=>"banana",
                       "c"=>"apple"),
     "numbers" => array(1, 2, 3, 4));

  // Some examples to address values in the array above
  echo $fruits["fruits"]["a"]; //prints "orange"
  echo $fruits["numbers"][2];  //prints "second"

  // Add a new element to $fruits
  $fruits["apple"]["green"] = "good";
  var_dump($fruits);
?>
```

```
array(3) {
  ["fruits"]=>
  array(3) {
    ["a"]=>
    string(6) "orange"
    ["b"]=>
    string(6) "banana"
    ["c"]=>
    string(5) "apple"
  }
  ["numbers"]=>
  array(4) {
    [0]=>
    int(1)
    [1]=>
    int(2)
    [2]=>
    int(3)
    [3]=>
    int(4)
  }
  ["apple"]=>
  array(1) {
    ["green"]=>
    string(4) "good"
  }
}
```

# OOP in PHP

# Class Definition

- Syntax:

```
class SimpleClass {
    //property declaration: similar to variable declaration
    public $var = 'a default value';

    //method declaration: similar to function declaration
    public function displayVar($str) {
        echo $str . ": " . $this->var;
    }
}
```

- Access modifiers: `public`, `protected`, `private`
- `$this`: reference to the calling object

# Creating Objects

- Use the new keyword: `$var = new Classname([arg]);`
- If there is no argument for the constructor, parentheses may be omitted

```php
<?php
  $obj1 = new SimpleClass();

  //This can also be done with a variable:
  $className = 'SimpleClass';
  $obj2 = new $className(); // new SimpleClass()


  $obj->displayVar();
?>
```

# Object Assignment

- `$obj2 = $obj1`: $obj2 references to the same object as $obj1 (shadow copy)
- `$obj2 = &$obj1`: $obj2 is a reference of $obj1 (reference is an alias)
- `$obj2 = clone $obj1`: A new object is cloned from $obj1 and it is referenced by $obj2 (deep copy)

```php
<?php
  $obj1 = new SimpleClass();

  $obj2 = $obj1;
  $obj3 = clone $obj1;
  $obj4 = &$obj1;

  $obj1->var = "new val";

  $obj1->displayVar(); //new val
  $obj2->displayVar(); //new val
  $obj3->displayVar(); //default
  $obj4->displayVar(); //new val

  $obj1 = null;

  var_dump($obj1); //NULL
  var_dump($obj2); //new val
  var_dump($obj3); //default
  var_dump($obj4); //NULL
?>
```

# Static Members

- Declared using `static` keyword
- Accesses though class, not the objects of the class
  - Outside the class:
    - `classname::$property`
    - `classname::method()`
  - Inside class:
    - `self::$property`
    - `self::method()`
- Note: `$this` cannot be used inside static methods

```php
<?php
  class Foo {
     public static $static_v = 'foo';

     public function staticValue() {
         return self::$static_v;
     }
  }

  print Foo::$static_v . "\n";

  $foo = new Foo();
  print $foo->staticValue() . "\n";
?>
```

# Constructors and Destructors

- Constructors: called automatically when an object is created

```
public function __construct($arg1, $arg2,...) {
   //initialization
 }
```

- Destructors: called automatically when an object is destroyed

```
public function __destruct($arg1, $arg2,...) {
   //clean-up tasks
 }
```

# Constructors and Destructors

```php
<?php
  class Person {
    public $id;  public $name;
    public $dob;
    protected static $count = 0;

    public function __construct($name) {
      $this->id = ++self::$count;
      $this->name = $name;
      $this->dob = date("d/m/Y");
    }
    public function __destruct() {
      $this->id = --self::$count;
    }
    public function displayInfo() {
      echo $this->id . " - "
           . $this->name . " - "
           . $this->dob . " ("
           . self::$count . ")";
    }
  }
?>
```

```
1 - Mr. Tom - 27/03/2019 (1)
2 - Ms. Jerry - 01/01/2002 (2)
1 - Mr. Tom - 27/03/2019 (1)
```

```php
<?php
  $tom = new Person("Mr. Tom");
  $tom->displayInfo();
  //output: 1 - Mr. Tom - 07/03/2014
  $jerry = new Person("Ms. Jerry");
  $jerry->dob = "01/01/2002";
  $jerry->displayInfo();
  //output: 2 - Ms. Jerry - 01/01/2002
  unset($jerry);
  $tom->displayInfo();
?>
```

# Inheritance

- Syntax: use the `extends` keyword

```
class classname extends parentClassname {
    //child class members
  }
```

- Child class inherits all members of the parent class (but it can access public and protected members)
- Child class can override methods of the parent class
- Access parent members from the child class:
  - `parent::property`
  - `parent::method()`

# Inheritance

```php
<?php
  class Student extends Person {
    public $enroll;

    function __construct($name) {
      parent::__construct($name);
      $this->enroll = @date("d/m/Y");
    }

    public function displayInfo() {
      parent::displayInfo();
      echo " - " . $this->enroll;
    }
  }
?>
```

```php
<?php
  $tom = new Person("Mr. Tom");
  $tom->displayInfo();
  //output: 1 - Mr. Tom - 07/03/2014
  $jerry = new Person("Ms. Jerry");
  $jerry->dob = "01/01/2002";
  $jerry->displayInfo();
  //output: 2 - Ms. Jerry - 01/01/2002
  unset($jerry);
  $tom->displayInfo();
?>
```

# Further Reading

- Abstract class (class contains some abstract methods – methods without implementation)
- Final methods (methods that cannot be overridden)
- Interfaces ("pure" abstract classes)
- `__toString()` method (object to string auto conversion )
- Using code from other .php files (`include` statement)

# PHP and Forms

# HTML Forms

- Forms are used to get user inputs
- Form data is usually sent to server (PHP) to process

```
<form action="..." method="...">
    <!-- form controls -->
</form>
```

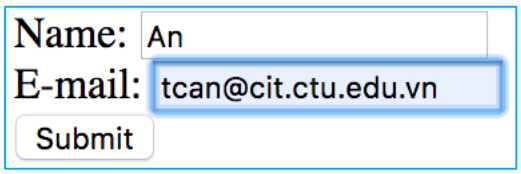
  - action: PHP page that will receive and process form data
  - method: HTTP method that is used to send data to server
    - GET: form data is sent via the URL parameters (1K-4KB)
    - POST: form data is sent inside the request package body
  - Form data is sent to server in the form of an array (key => value) where key is the name of the control and value is the input data

# Getting Form Data

- The PHP global variables $_GET and $_POST are used to collect form data (corresponding to the GET and POST method)

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

```
<!– welcome.php –>
<html>
  <body>
    Welcome <?php echo $_GET["name"]; ?><br>
    Your email is: <?php echo $_GET["email"]; ?>
  </body>
</html>
```




Welcome An
Your email is: tcan@cit.ctu.edu.vn

# Checking Data Existence

- Use the `isset()` function:

```
<body>
   <h3>
     <?php
       if (isset($_POST["uname"]) && isset($_POST["pwd"])) {
         if ($_POST["pwd"] == "abc") {
           echo "Welcome " . $_POST["uname"];
         } else {
           echo "Sorry " . $_POST['uname'] . ", wrong pass!";
         }

       } else {
         echo "<span style=\"color: red;\">"
              . "<i>uname</i> and <i>pword</i> are expected</span>";
       }
       ?>
   </h3>
 </body>
```

# PHP and MySQL

# MySQL Server Access from PHP

- Two methods in accessing MySQL DB from PHP:
  - **MySQLi** (object-oriented or procedural, only work with MySQL)
  - PDO (PHP Data Objects, can work with 12 DBMS)
- Steps in accessing MySQL databases from PHP:
  1. Create connection to MySQL server
  2. Select the database
  3. Execute SQL statement (query, insert, update, delete)
  4. Get and process the returned data
  5. Produce the output
  6. Close the connection

# Related MySQLi Classes

- `mysqli`: represents a connection between PHP and MySQL
  - `::__construct()`: creates connections
  - `::$connect_errno`: contains the last connection error code (0: no error)
  - `::$connect_error`: contains the last connection error description
  - `::select_db()`: select the default DB
  - `::query()`: perform a query
  - `::$errno`: error code for the most recent function call (0: no error)
  - `::$error`: contains the last error description
  - `::prepare()`: create a prepared statement (`mysqli_stmt` class)
  - `::close()`: close a connection

# Related MySQLi Classes

- `mysqli_stmt`: represents a prepared statement
  - `::bind_param()`: binds variables to a prepared statement as parameters
  - `::bind_result()`: binds variables to a prepared statement for result storage
  - `::execute()`: executes a prepared statement
  - `::get_result()`: returns the result set from the statement
  - `::fetch()`: fetch results from a prepared statement
  - `::$num_rows`: number of rows in the statement result set
  - `::close()`: closes a prepared statement

# Related MySQLi Classes

- `mysqli_result`: represents the result set obtained from a query
  - `::fetch_array()`: afetch a row as an associative or numeric array or both
  - `::fetch_assoc()`: fetch a row as an associative array
  - `::fetch_all()`: fetch all rows as an associative or numeric array or both
  - `::fetch_row()`: fetch a row as an numeric array
  - `::$field_count`: get the number of fields in a result
  - `::$num_rows`: get the number of rows in a result

# Creating Connections

- Use the constructor of the class `mysqli`:

```
mysqli($servername, $username [, $password, $dbname]);
```

  - servername: can be either hostname or an IP
  - username: MySQL user name
  - password: MySQL user password (not provided/NULL: no password)
  - returns an object representing the connection to MySQL server
- The connection information:
  - usually declared in a separated file
  - included into the PHP file using the `require_once()` statement
- `mysqli::$connect_error` contains error description

# Creating Connections

```php
<?php
  require_once 'connection.inc';

  //create connection
  $conn = @new mysqli($servername, $username, $password);

  //check connection
  if ($conn->connect_error) {
      die("Connection failed: " . $conn->connect_error);
  }
  echo "Connected successfully";
  ?>
```

```php
<?php  /* connection.inc */
  $servername = "localhost";
  $username = "root";
  $password = "1234";
?>
```

- `mysqli::$connect_error` – contains description of the last error, or NULL of no error occurred

# Selecting DB

- Select DB when create the connection:

```
mysqli($servername, $username, $password, $dbname);
```

- Select DB after the connection created:
  - `mysqli::select_db($dbname)`: returns TRUE on success, FALSE on failure

```
<?php
  $db = "test";
  if ($conn->select_db($db)) {  //$conn: connection object
    echo "Select DB '$db' successfully";
  }
  else {
    echo "Couldn't select the '$db' DB";
  }
?>
```

# Executing SQL Statements

- Use the `mysqli::query()` method:
  - Failure: returns `FALSE`
  - Success: `mysqli_result` object for SELECT, SHOW, DESCRIBE, EXPLAIN queries; TRUE for other queries

```php
<?php
  $sql = "SELECT id, firstname, lastname FROM students";
  $result = $conn->query($sql);

  if ($result && $result->num_rows > 0) {
     // process the received data
  } else {
     // query failed OR no data received
  }
?>
```

# Executing SQL Statements

- Prepared statements:
  - `mysqli::prepare($query)`: create prepared statement
    - Returns a `mysqli_stmt` object or FALSE if an error occurred
  - `mysqli_stmt::bind_param($type, $var [, $var…])`: binds variables to a prepared statement as parameters
    - Returns TRUE on success; FALSE on failure
    - Parameter types: i (integer), d (double), s (string), b (blob)
  - `mysqli_stmt::execute()`: execute a prepared query
    - Returns TRUE on success; FALSE on failure

Prepared statement is recommended due to its better performance, less bandwidth and useful against SQL injection

# Executing SQL Statements

```php
<?php
  // prepare and bind
  $stmt = $conn->prepare("INSERT INTO courses VALUES (?, ?, ?)");
  $stmt->bind_param("ssi", $courseId, $courseName, $credits);

  // set parameters and execute
  $courseId= "CT219H";
  $courseName = "Web Programming Fundamental";
  $credits = 3;
  $stmt->execute();     //insert the 1st record

  $courseId= "CT123H";
  $courseName = "Object-Oriented Pgrogramming";
  $credits = 3;
  $stmt->execute();     //insert the 2nd record

  $stmt->close();       //close the prepared statement
  $conn->close();       //close the connection
<?php
```

# Accessing Returned Data

- Access data in a `mysqli_result` object with `fetch_assoc()`:

```php
<?php
  //...
  $result = $conn->query($sql);

  if ($result->num_rows > 0) {
    // output data of each row
    while($row = $result->fetch_assoc()) {
      echo "ID: " . $row["id"] . " - Name: "
           . $row["firstname"]. " " . $row["lastname"]. "<br>";
    }
  } else {
      echo "No student found";
  }

  $conn->close();
?>
```

# Accessing Returned Data

- Access data in a `mysqli_result` object with `fetch_row()`:

```php
<?php
  //...
  $result = $conn->query($sql);

  if ($result->num_rows > 0) {
    // output data of each row
    while($row = $result->fetch_row()) {
      echo "ID: " . $row[0]. " - Name: "
           . $row[1]. " " . $row[2]. "<br>";
    }
  } else {
      echo "No student found";
  }

  $conn->close();
?>
```

# Accessing Returned Data

- Access data in a `mysqli_result` object with `fetch_array()`:

```php
<?php
  //...
  $result = $conn->query($sql);

  if ($result->num_rows > 0) {
    // output data of each row
    while($row = $result->fetch_array(MYSQLI_BOTH)) {
      echo "ID: " . $row[0]. " - Name: "
           . $row["firstname"]. " " . $row["lastname"]. "<br>";
    }
  } else {
      echo "No student found";
  }

  $conn->close();
?>
```

# Accessing Returned Data

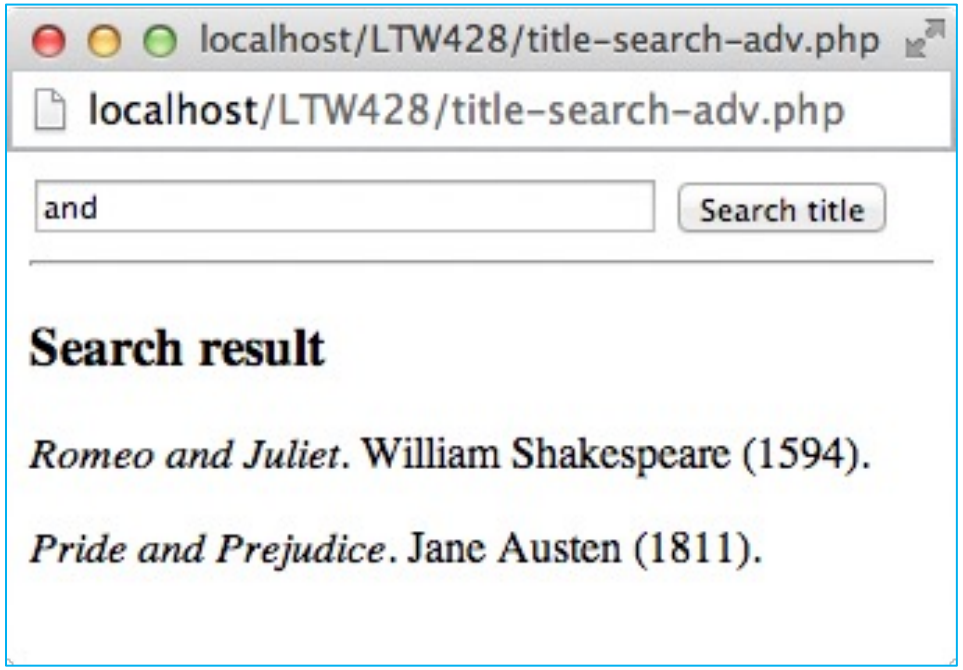
- Access data in a `mysqli_stmt` object with `bind_result()` and `fetch()`:

```
<?php
  //...
  if ($stmt = $mysqli->prepare("SELECT id, firstname, lastname
                                FROM students")) {
    $stmt->execute();

    // bind variables to prepared statement
    $stmt->bind_result($col1, $col2);

    while ($stmt->fetch()) {   //fetch values
      printf("%s %s\n", $col1, $col2);
    }
    $stmt->close();
  }
  $mysqli->close();
  ?>
```

# Template for PHP-MySQL Data Access

```php
<?php /* php-mysql-template.php */
  //connect + select DB
  $conn = @new mysqli($hostname, $username, $password, $dbname) or die("...");

  //create query string + execute query
  //OR use prepared statement for better performance
  $query = "SELECT * FROM table_name";
  $result = @$conn->query($query) or die("...");

  //process the returned data
  while ($row = $result->fetch_array/fetch_assoc/fetch_row()) {
    //...process the record...
  }

  //close the connection
  $conn->close();
?>
```

# Example – Book Search




```
MariaDB [ct219h]> describe classics;
+--------+--------------+------+-----+
| Field  | Type         | Null | Key |
+--------+--------------+------+-----+
| isbn   | char(13)     | NO   | PRI |
| author | varchar(128) | YES  | MUL |
| title  | varchar(128) | YES  |     |
| type   | varchar(16)  | YES  |     |
| year   | char(4)      | YES  |     |
| price  | float        | YES  |     |
+--------+--------------+------+-----+
```

# Example 1 – Book Search

- Webpage and form:

```
<html>  <!-- title-search-adv.php -->
  <body>
    <form action="title-search-avd.php" method="POST">
      <input type="text" size="40" name="search_kw"
        value="<? if (!empty($_POST['search_kw']))
          echo $_POST['search_kw'];?>"/>
      <input type="submit" value="Search title">
      <hr>
    </form>
    <h3>Search result</h3>
    <?php
      if (isset($_POST['search_kw'])) {
         include 'title-search-func.php';
         search($_POST['search_kw']);  //defined in title-search-func.php
      }
    ?>
  </body>
</html>
```

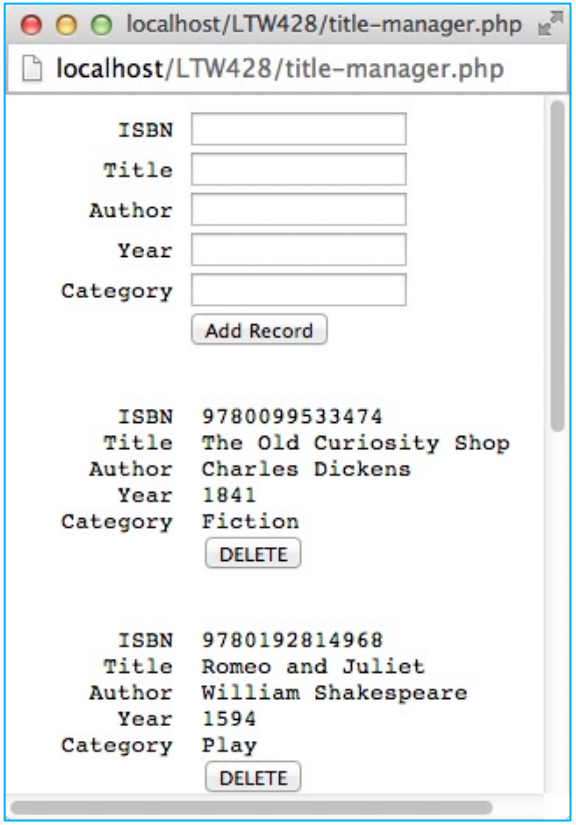
# Example 1 – Book Search

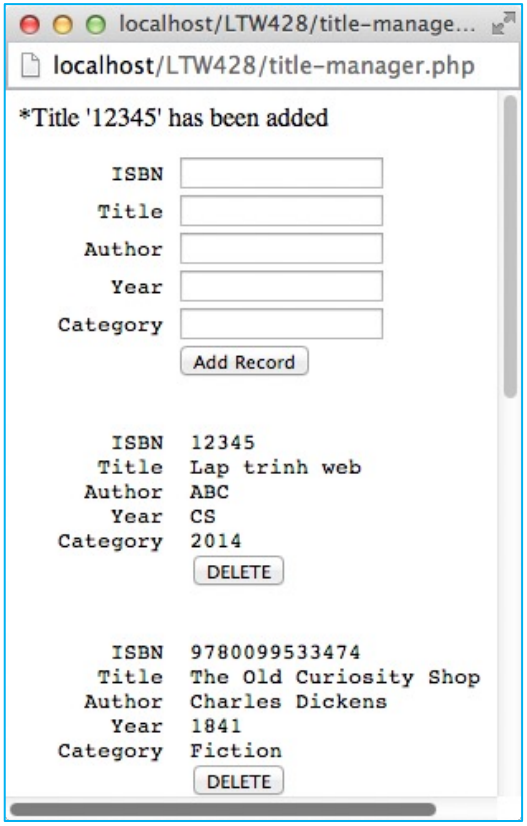
- Search function:

```
<?php    /* title-search-func.php */
  function search($keyword) {
    require "connect-select-db.php";
    $keyword = trim($keyword);
    $new_kw  = str_replace(" ", "%' OR title LIKE '%", $keyword);
    $query   = "SELECT * FROM classics WHERE title LIKE '%$new_kw%'";

    $result = $conn->query($query)
       or die("Query failed: " . $conn->error);

    if ($result->num_rows > 0) {
      while ($row = $result->fetch_assoc()) {
        echo "<p><i>$row[title]</i>. $row[author] ($row[year]).</p>\n";
      }
    }
    else
      echo "No title found";
  }
?>
```

# Example 2 – Title Management

- Application interface:



# Example 2 – Title Management

- Main page:

```php
<?php   /* title-management.php */

  require "connect-select-db.php"; //connect to mySQL and provide $conn
  require "title-delete-process.php";   //delete a title
  require "title-add-process.php";      //add new title
  require "title-add-form.php";         //create add-title form
  require "title-list-delete-form.php"; //function del_form_gen()

  //retrieve all records (to create [delete title] forms)
  $query  = "SELECT * FROM classics";
  $result = $conn->query($query)
              or die("DB Access error: " . mysql_error());

  //generate [delete title] form
  while ($row = $result->fetch_assoc()) {
    del_form_gen($row);
  }
  $conn->close(); //close connection
?>
```

# Example 2 – Title Management

- Add title form:

```
<?php    /* title-add-form.php */
  echo <<<_ADD_TITLE_FORM
  <form action="title-manager.php" method="POST">
   <pre>
      ISBN <input type="text" name="isbn"/>
     Title <input type="text" name="title"/>
    Author <input type="text" name="author"/>
      Year <input type="text" name="year"/>
  Category <input type="text" name="type"/>
           <input type="submit" value="Add Record">
    </pre>
    <input type="hidden" name="add" value="yes">
  </form>
_ADD_TITLE_FORM;
?>
```

# Example 2 – Title Management

- List and delete title form:

```
<?php    /* title-list-delete-form.php */
  function del_form_gen($row) {
     echo <<<_DEL_TITLE_FORM
     <form action="title-manager.php" method="POST" >
     <pre>
      ISBN $row[isbn]
     Title $row[title]
    Author $row[author]
      Year $row[year]
  Category $row[type]
           <input type="submit" value="DELETE">
     </pre>
     <input type="hidden" name="delete" value="yes">
     <input type="hidden" name="isbn" value="$row[isbn]">
  </form>

  _DEL_TITLE_FORM;
  } //add_form_gen()
?>
```

**Confirm before deleting:**

```
onsubmit="return confirm('Do you
really want to delete the book?');"
```

# Example 2 – Title Management

- Process the delete title function:

```php
<?php    /* title-delete-process.php */
  if (isset($_POST['delete']) && isset($_POST['isbn'])) {
    $isbn = $_POST['isbn'];
    $query = "DELETE FROM classics WHERE isbn='$isbn'";

    if (!$conn->query($query)) {
      echo "<h3> DELETE failed: " . $isbn . ". Error: "
           . $conn->error . "</h3>";
    }
    else {
      echo "*Title '$isbn' has been deleted<br>";
    }
  }
  ?>
```

**To show an alert box about the deletion:**

```php
echo <<<_DEL_OK_PROMPT
  <script>
    alert("*Title '$isbn' has been deleted");
  </script>
_DEL_OK_PROMPT;
```

# Example 2 – Title Management

- Process the add title function:

```
<?php    /* title-add-process */
  if (isset($_POST["add"])) {
     $isbn = $_POST["isbn"];
     $author = $_POST["author"];
     $title = $_POST["title"];
     $year = $_POST["year"];
     $type = $_POST["type"];
     $query = "INSERT INTO classics VALUES "
         . "('$isbn', '$author', '$title', '$year', '$type', 0)";

     if (!$conn->query($query))
        echo "<h3>INSERT failed. " . $conn->error . "</h3>";
     else
        echo "*Title '$isbn' has been added<br>";
  }
?>
```

# Exercise – Edit Book Details

- Add the “Edit Book Detail” for the Book Manager Application as follow:
  - Add an Edit button next to the Delete button
  - When the user click the Edit button, open a new page that allows user to edit the details of the selected book (update page)
  - In the update page, user can
    - Edit and Save the change
    - Go back to the home page

# Cookies and Sessions

# What is Cookie?

- Cookies are data, stored in a small text file on the browser (user computer) by the server
- This small file stores a set of `name => value` items (called cookie)
- Usually used to identify the user
- Each time the browser making a request, it will include the cookies in the request message
- To create a cookie (name => value) in PHP:
  - `setcookie(name [, value, expire …])`
  - Expiration unit is second
  - The calls to this function must appear before the `<html>` tag

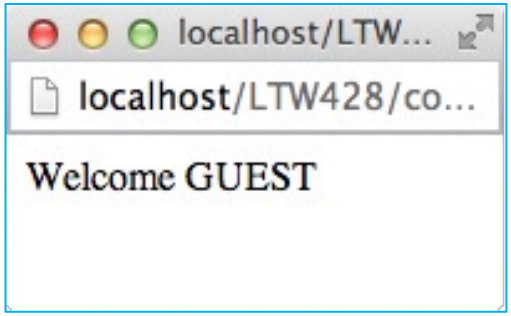
# Getting and Deleting Cookies

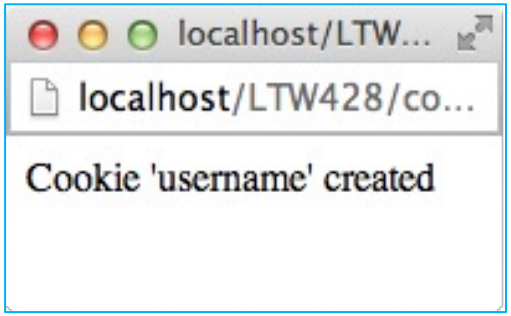
- Cookies sent to server is stored in the array variable `$_COOKIE`
- To get a cookie value, provide cookie name `$_COOKIE[name]`
- A cookie is automatically deleted when:
  - Time is expire
  - The browser closes, if no expire is set
- To delete a cookie, set expire time of the cookie with an expiration date in the past

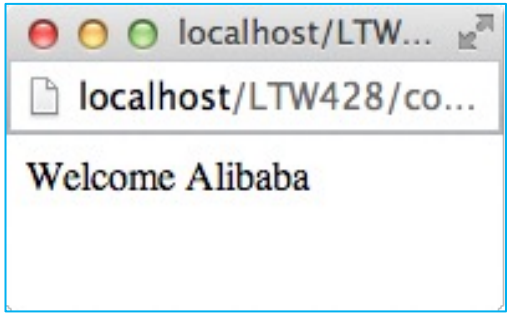
# Cookie Example

```
<html>
  <body>
    <?php
      if (isset($_COOKIE['username']))
         echo "Welcome $_COOKIE[username]";
      else
         echo "Welcome GUEST";
      ?>
  </body>
</html>
```

```
<?php
  //expiration: 60 seconds
  setcookie("username", "Alibaba",
            time() + 60 * 10);
?>
<html>
  <body>
    <p>Cookie 'username' created</p>
  </body>
</html>
```



# Sessions

- Used to store information (in variables) across multiple pages of an application (website)
- Sessions are stored on the server
- By default, session variables last until the browser closes
- To start a session: call `session_start()` (before the `<html>` tag)
- Session variables are stored in the global variable `$_SESSION`
  - Access a session: `$_SESSION[name]`
- Destroy all sessions: `session_unset()` or `session_destroy()`
- Destroy a session: `unset($_SESSION[name])`

# Session Example – Login

```php
<?php    /* login.php */
  //$login_ok = /* check the login information */;
  if ($login_ok) {
     session_start();
     $_SESSION['logged_in'] = true; //OR = username
     /* redirect to other page */
  }
?>

<html>
 <!-- login form -->
</html>
```

```php
<?php    /* logout.php */
  session_start();
  if (isset($_SESSION['logged_in']))
     unset($_SESSION['logged_in']);
?>
```

# Session Example – Login

```php
<?php    /* update-data.php */
  session_start();
  if (!isset($_SESSION['logged_in'])) {
    header('Location: login-session.php');
    die();
  }
?>

<html>
  <!-- update data form... -->
</html>
```

- The code to check login session is usually stored in a separated file and included into the webpages which need the authorization

# Advanced PHP Technique

*File upload, image saving in DB, pagination, AJAX*

# File Upload

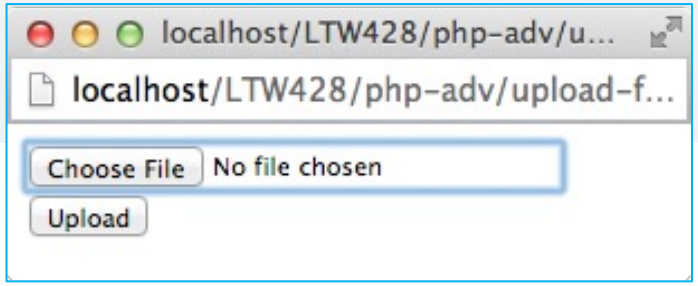
# Setting Up

- Turn on file upload option:
  - Set the `file_uploads` directive to On in file "**php.ini**"
    - XAMPP:
      - Windows: [XAMPP_HOME]\php\php.ini
- Steps to upload files by PHP:
  1. Create a form for file upload with appropriate encryption method (browser)
  2. Validate data received at server (error, format, size, etc.)
  3. Save files received to storage devices

# Upload Form (Client/Browser)

- Form attributes:
  - Method: `POST`
  - Encryption method: `enctype=multipart/form-data`
  - File select control: `<input type=file ...>`

```
<html> <!-- upload-form.html -->
  <body>
    <form method="POST" action="upload.php"
          enctype="multipart/form-data">
      <input type="file" name="up-file"><br>
      <input type="submit" value="Upload">
    </form>
  <body>
</html>
```



# File Information (Server)

- After uploading, the file will be stored temporarily on the server (configured with the `upload_tmp_dir` directive in **php.ini**)
- The global associative array `$_FILES` contain all the uploaded file information:
  - `$_FILES["filename"]["name"]`: the original name of the file
  - `$_FILES["filename"]["type"]`: the MIME type of the file
  - `$_FILES["filename"]["size"]`: the file size (in bytes)
  - `$_FILES["filename"]["tmp_name"]`: the temporary name of the file (stored temporarily in the server after uploading)
  - `$_FILES["filename"]["error"]`: error code of the upload

# Validation (Server)

```php
<?php    /* upload.php (pre-processing) */

  if (($_FILES['up-file']['error'] == 0) &&   //upload error?
      ($_FILES["up-file"]["type"] == "...") &&  //expected size?
      ($_FILES["up-file"]["size"] < ...))  //size excess?

        //code for saving file

  }
     else {
        //error handler ...
        echo "Upload error: " . $_FILES['up-file']['error'];
  }
?>
```

# File Storage (Server)

- Related functions:
  - `move_uploaded_file(tmp_file, persistent_file)`: moves an uploaded file to a new location (returns TRUE on success)
  - `file_exists(filename)`: check whether a file or directory exists (returns TRUE on success)

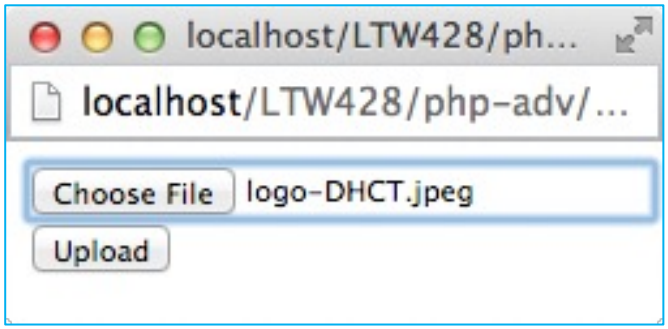
```php
<?php

  // verify file upload conditions (pre-processing) ...

  // save file to folder “uploads”
  move_uploaded_file($_FILES["up-file"]["tmp_name"],
           "uploads/" . $_FILES["up-file"]["name"]);

 // post processing ...

?>
```

# Complete Image Upload Example

1. Check for upload error and file type (jpeg)
2. Save image file to “uploads” folder
3. Display uploaded image on upload page

```
<?php    /* upload.php */
  if (($_FILES['up-file']['error'] == 0)) {

        move_uploaded_file($_FILES["up-file"]["tmp_name"],
           "uploads/" . $_FILES["up-file"]["name"]);

        echo "File uploaded: " . $_FILES["up-file"]["name"] . "<br>";
        echo "Type: " . $_FILES['up-file']['type'] . "<br>";
        echo "Size: " . $_FILES['up-file']['size'] . "b<br>";
        echo '<img src="uploads/' . $_FILES["up-file"]["name"] . '"/>';
  }
  else
     echo "Upload error: " . $_FILES['up-file']['error'];
?>
```

# Complete Image Upload Example




**upload-form.html**


**upload.php**

# Saving Images to a Database

- Datatype to store image: `blob`
- Steps to save image to database:
    1. Validate file uploaded: upload status, type, size, etc. (using the global variable `$_FILES` and function `getimagesize()`)
    2. Get the image file content which is saved temporarily on the server to a variable
    3. Add backslashes ( \ ) before predefined character, using the `addslashes()` function
    4. Save image into the table, using the image variable create in step 2

# Saving Images to a Database

- Given a table with the following structure:

```
MariaDB [ct219h]> describe images;
+---------+-----------+------+-----+---------+
| Field   | Type      | Null | Key | Default |
+---------+-----------+------+-----+---------+
| imgname | char(100) | NO   | PRI | NULL    |
| imgdata | longblob  | NO   |     | NULL    |
+---------+-----------+------+-----+---------+
```

- Upload form:

```
<form method="POST" action="upload-img-db.php"
      enctype="multipart/form-data">

  <input type="file" name="img_file">
  <input type="submit" value="Upload">
</form>
```

# Saving Images to a Database

```php
<?php /* upload-img-db.php */

  //connecte to DBMS + select DB... ($conn: connection object)

  if (isset($_FILES["img_file"]["name"]) &&
      getimagesize($_FILES['img_file']['tmp_name']) != false) {

    //CHECK other CONTRAINTS if any ...
    $image = file_get_contents($_FILES['img_file']['tmp_name']);
    $image = addslashes($image);
    $imgname = $_FILES["img_file"]["name"];
    $conn->query("INSERT INTO images VALUES ('$imgname', '$image')")
       or die("Cannot insert image into DB: " . $conn->connect_error);

    echo "Uploaded image: <br><img src='get_img.php?name=$imgname'>";
  }
  else
    echo "No image has been uploaded";
?>
```

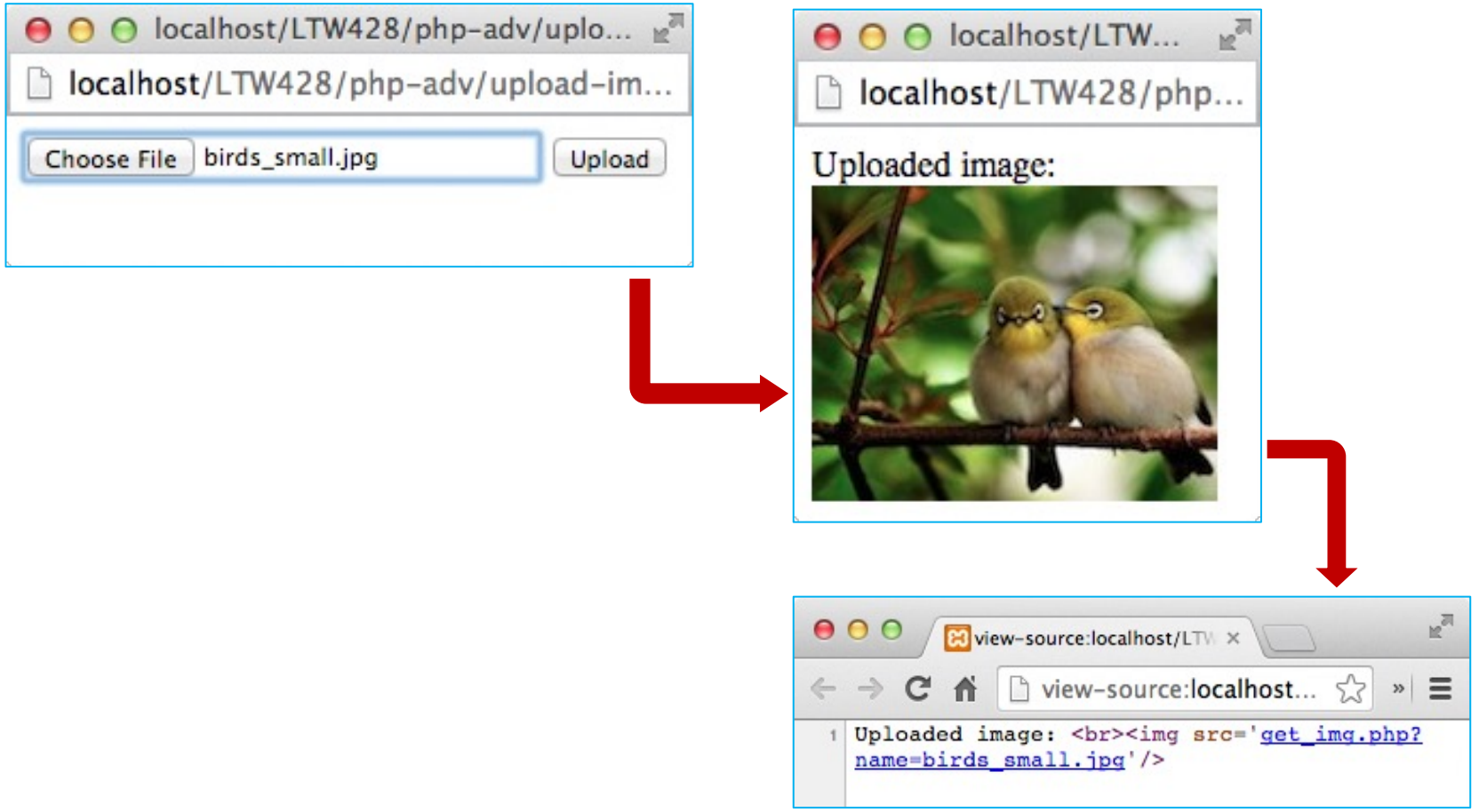
# Displaying Images Saved in a Database

- To display an image in DB: `<img src=get_img?id=img_id>`

  (***get_img.php*** *is script that read the image with the* ***id*** *is* ***img_id*** *from the DB* )

```php
<?php    /* get_img.php */
  if (isset($_REQUEST['name'])) {
    // connect to DBMS and select the DB... ($conn: connection object)
    $query = "SELECT * FROM images WHERE imgname='$_REQUEST[name]'";
    $result = $conn->query($query)
        or die("Data retrieval failed" . $conn->connect_error);
     $row = $result->fetch_assoc();
     if ($row) {
        header("Content-type: image/jpeg");
        echo $row['imgdata'];
     }
     else
        echo "Image '$_REQUEST[name]' is not found";
  }
  else
     echo "Image name is required"
?>
```

# Displaying Images Saved in a Database



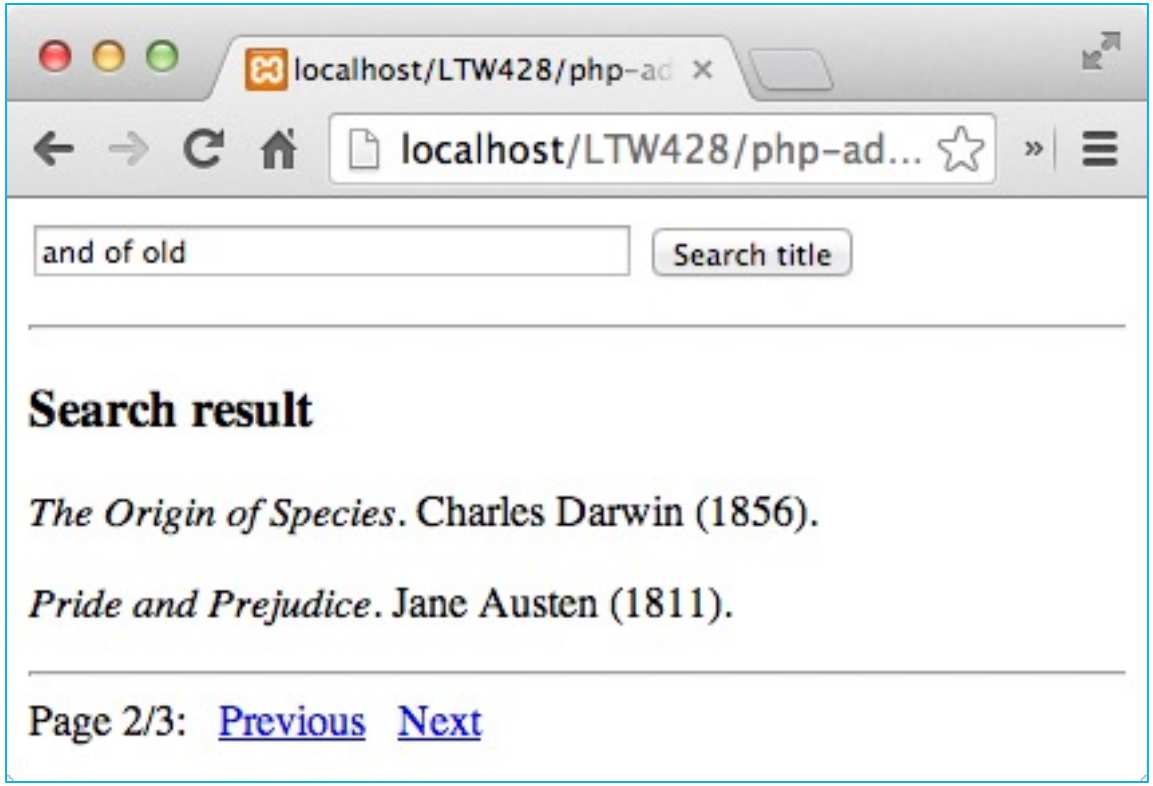
# Pagination

# Steps in Creating Pagination

1. Calculate the total number of records: `$total_records`
2. Identify the number of records displayed per page: `$records_per_page`
3. Store the current page number (use hidden variable or send directly to server by GET method)
4. Query data for the requested page
5. Create Next page and Previous page links

# Case Study

- Build a book search page with pagination (2 results per page)



# Structure of the Application

- Main page:
  - Show the form to get search keywords + Search button
  - Call `search()` function to search for books and get the pagination information
  - Call `page_nav_link()` function to create links to Prev/Next page
- PHP functions:
  - `compute_paging(search_kw)`: computer pagination parameters
  - `search(kw)`: search and return search result and pagination info
  - `page_nav_links(paging_info, search_kw)`: create links to Prev/Next page

# Main page

```
<html>
  <body>
    <!-- title-search-paging.php -->
    <form action="title-search-paging.php" method="GET">
      <input type="text" size="40" name="search_kw"
        value="<? empty($_REQUEST['search_kw']) ||
          print $_REQUEST['search_kw'];?>"/>
      <input type="submit" value="Search title">
    </form>
    <hr>
    <h3>Search result</h3>
    <?php
      if (isset($_REQUEST['search_kw'])) {
        include 'title-search-paging-func.php';
        $paging = search($_REQUEST['search_kw']);
        echo "<br><hr>";    //links to next/prior page
        page_nav_links($paging, $_REQUEST['search_kw']);
      }
      ?>
  </body>
</html>
```

# Function compute_paging()

```php
<?php   /* title-search-paging-func.php */
  $record_ppage = 2;

  function compute_paging($search_kw) {
    global $record_ppage;
    $query = "SELECT count(*) FROM classics "
                . "WHERE title LIKE '%$search_kw%'";
    $result = $conn->query($query);
    $row = $result->fetch_row();
    $p_total = ceil($row[0]/$record_ppage);
    $page = (isset($_REQUEST["page"]))? $_REQUEST["page"] : 1;
    $start = ($page - 1) * $record_ppage;
    $p_next = ($page > 1)? $page - 1 : 0;
    $p_pre = ($page < $p_total)? $page + 1 : 0;
    return array("p_total"=>$p_total, "p_no"=>$page,
                 "p_start"=>$start, "p_prev"=>$p_prev,
                 "p_next"=>$p_next, "total"=>$row[0]);
  } //compute_paging()
```

**p_total**: total no of pages;
**p_no**: current page;
**p_start**: index of the first record in the page;
**total**: total number of records

# Function search()

```php
<?php /* title-search-paging-func.php */
  function search($keyword) {
    global $record_ppage;
    $search_kw = str_replace(" ", "%' OR title LIKE '%", trim($keyword));

    $paging = compute_paging($search_kw);

    $query = "SELECT * FROM classics WHERE title LIKE '%$search_kw%'" .
             . " LIMIT $paging[p_start], $record_ppage";

    $result = $conn->query($query)
    or die ("DB accessed failed: " . $conn->error);

    while ($row = $result->fetch_assoc())
       echo "<p><i>$row[title]</i>. $row[author] ($row[year]).";

    if ($result->num_rows == 0)
       echo "No title found";
    return $paging;
  } //search()
?>
```

**$paging**

**p_total**: total no of pages;
**p_no**: current page;
**p_start**: index of the first record in the page;
**total**: total number of records

# Function search()

```php
<?php    /* title-search-paging-func.php */
  function page_nav_links($paging, $search_kw) {
    echo "Page $paging[p_no]/$paging[p_total]:   ";

    if ($paging['p_prev'] > 0) {    //previous
      echo "<a href='title-search-paging.php?search_kw=$search_kw" .
        "&page=" . $paging['p_prev'] ."'>Previous</a>   ";
    }

    if ($paging['p_next'] > 0) {    //next
      echo "<a href='title-search-paging.php?search_kw=$search_kw " .
        "&page=" . $paging['p_next'] ."'>Next</a>";
    }
  }  //page_nav_links()
?>
```

# AJAX

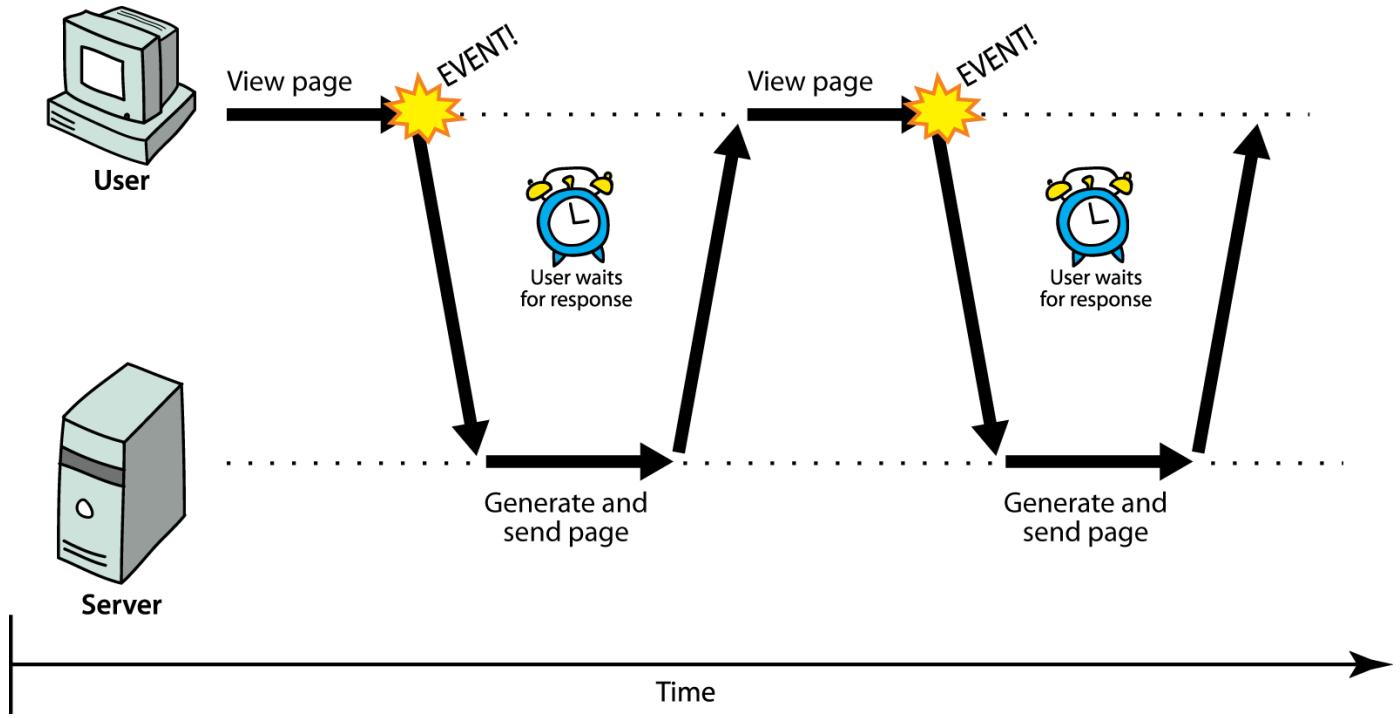
*Asynchronous JavaScript And XML*

# What is AJAX?

- A technology that allow asynchronous communication between web browsers and web servers
- Not a programming language, just a combination of:
  - A browser built-in `XMLHttpRequest` object (to send request to server)
  - JavaScript and HTML DOM (to display or use response data)
- With AJAX, we can:
  - Read data from a web server, after the page has loaded
  - Update a (part of) web page without reloading the page
  - Send data to a web server in the background (i.e. update parts of a web page, without reloading the whole page)

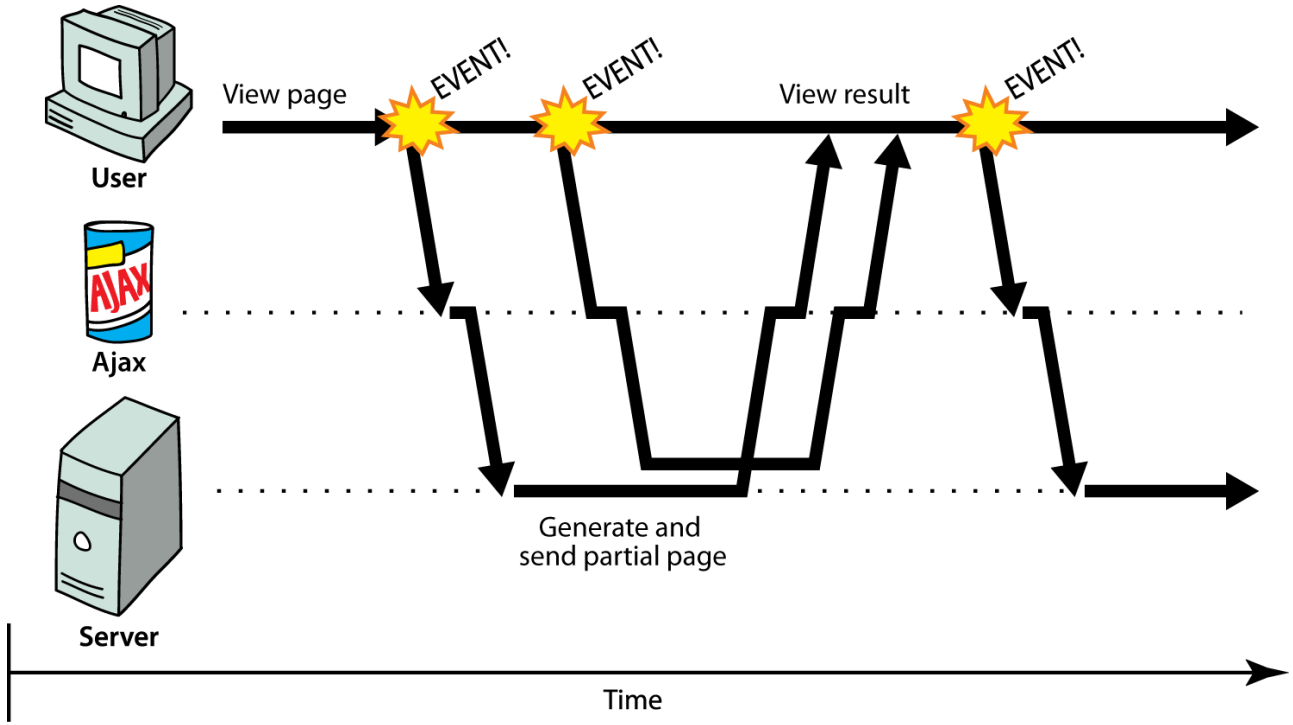
# Synchronous Communication

- User have to wait while browser communicates with the web server

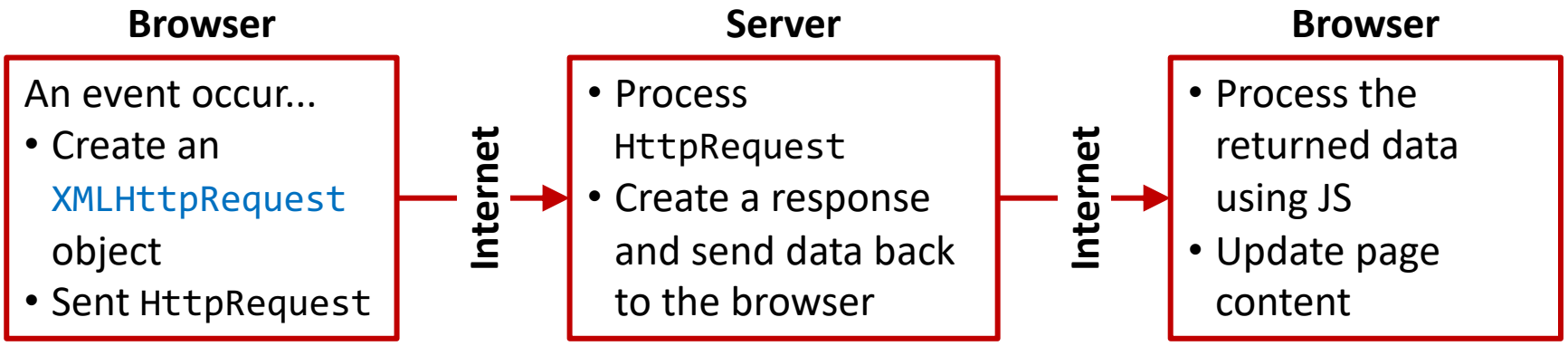


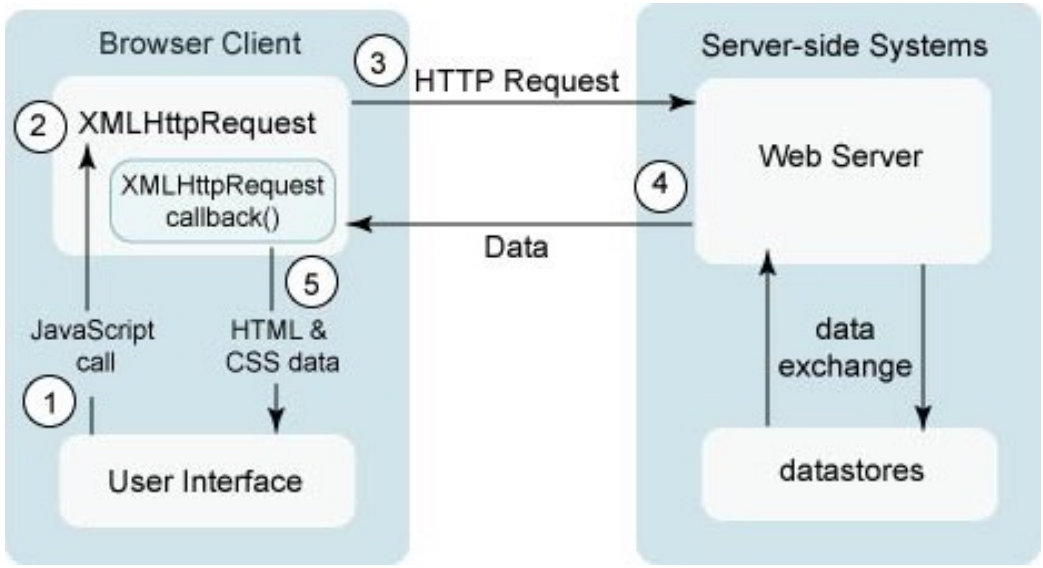
# Asynchronous Communication

- Users can interact with the web page while the browser is communicating with web server



# How AJAX Works?



# `XMLHttpRequest` Object

- Used to exchange data with web server behind the scenes
- Send request to server:
  - `open(method, url, async)`: specify the type of the request
    - method: GET/POST
    - url: server/file location
    - async: true (asynchronous) or false (synchronous)
  - `send()`: send the request to the server (used for GET)
  - `send(string)`: send the request to the server (used for POST)
  - `setRequestHeader(header, value)`: add HTTP header to the request

# XMLHttpRequest Object

- Request examples:
  - GET request:

```
var xhttp = new XMLHttpRequest();
xhttp.open("GET", "demo_get.php", true);
xhttp.send();
```

  - To avoid cached result:

```
xhttp.open("GET", "demo_get.php?t=" + Math.random(), true);
```

  - POST request:

```
xhttp.open("POST", "demo_post2.asp", true);
xhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
xhttp.send("fname=Henry&lname=Ford");
```

# XMLHttpRequest Object

- Request examples:
  - POST with JSON data:

```
var req = new XMLHttpRequest();
req.open("POST", "demo_post3", true);
req.setRequestHeader("Content-Type",
                     "application/json; charset=UTF-8");
var jsonBody = {
  "name": "An Tran",
  "url": "http://sites.google.com/site/tcanvn"
};

req.send(jsonBody);
```

# XMLHttpRequest Object

- Server response:
  - `readyState`: the status of the `XMLHttpRequest` object

    (0: request not initialized, 1: server connection established, 2: request received, 3: processing request, **4: response is ready**)
  - `onreadystatechange`: a function to be execute when the `readyState` changes

```
xhttp.onreadystatechange = function() {
  if (this.readyState == 4 && this.status == 200) {
    aCallbackFunction(this);
  }
};
```

# XMLHttpRequest Object

- Server response:
  - `status`, `statusText`: status of the XMLHttpRequest object (**200: "OK"**, 403: "Forbidden", 404: "Page not found")
  - `responseText`: contains the response data as a string
  - `responseXML`: contains the response data as XML data
  - `getResponseHeader(header)`: returns specific header information from the server
  - `getAllResponseHeaders()`: Returns all the header information from the server

# loadDoc() Function

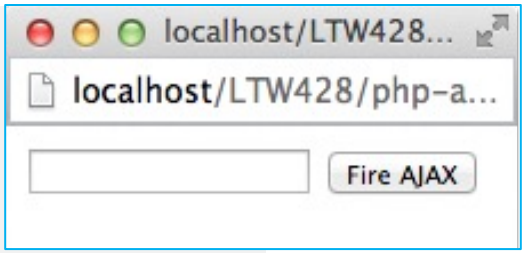
- If there are more than one AJAX tasks, we should create:
  - A loadDoc() function for executing the XMLHttpRequest with two parameters: the requested URL and the callback function
  - A callback function for each AJAX task with

```
loadDoc("url1", callbackFunction1);  //AJAX task 1
loadDoc("url2", callbackFunction2);  //AJAX task 2...

function loadDoc(url, callbackFunction) {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      callbackFunction(this);
    }
  };
  xhttp.open("GET", url, true);
  xhttp.send();
} //loadDoc()
```

```
function callbackFunctionX(xhttp) {
   // action goes here
}
```

# AJAX Simple Example

```
<html> <!-- simple-ajax.html -->
  <head>
    <script src="simple-ajax.js"></script>
  </head>

  <body>
    <form name="myform" action="">
      <input type="text" name="myname">
      <input type="button" name="mybutton"
             value="Fire AJAX"
             onclick="loadDoc('hello-name.php?name=' +
                      myform.myname.value, showHello);">
    </form>
    <p id="msg"></p>
  </body>
</html>
```



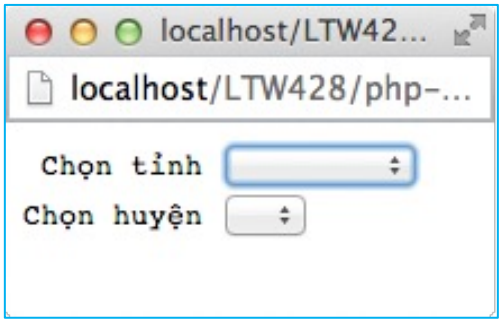

# AJAX Simple Example

```
/* simple-ajax.js */
function loadDoc(url, callback) {
  var xmlhttp = new XMLHttpRequest();
  xmlhttp.onreadystatechange = function() {
    if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
      callback(this);
    }
  }
  xmlhttp.open("GET", url, true);
  xmlhttp.send();
}

function showHello(xmlhttp) {
  document.getElementById("msg").innerHTML = xmlhttp.responseText;
}
```

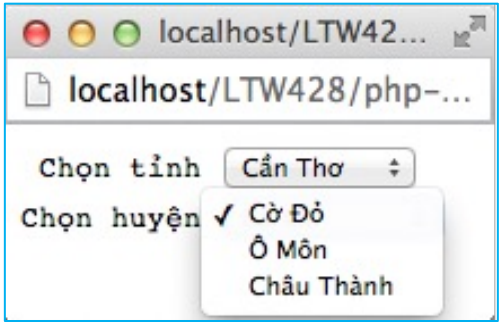
```
<?php   /* hello-name.php */

  if (isset($_GET['name'])) {
    echo "Hello <b>"
         . $_GET['name']
         . "</b>";
  }
?>
```

# AJAX Example – Dropdown List

| matinh | tentinh |
|---|---|
| 1 | Cần Thơ |
| 2 | Sóc Trăng |
| 3 | Bạc Liêu |

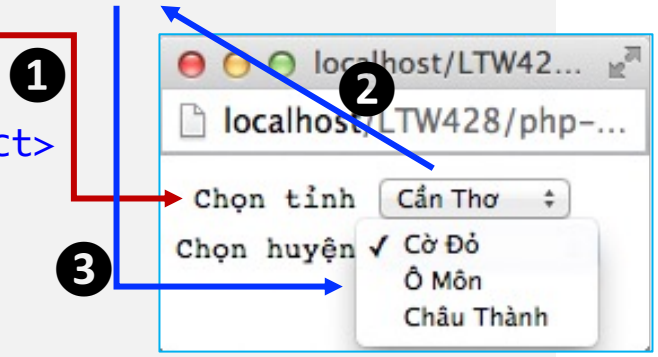



| mahuyen | tenhuyen | matinh |
|---|---|---|
| 1 | Cờ Đỏ | 1 |
| 2 | Ô Môn | 1 |
| 3 | Châu Thành | 1 |
| 4 | Long Phú | 2 |
| 5 | Thạnh Trị | 2 |
| 6 | Ngã Năm | 2 |
| 7 | Trần Đề | 2 |
| 8 | Vĩnh Châu | 3 |
| 9 | Giá Rai | 3 |



# AJAX Example – Dropdown List

```
<html>   <!-- dropdown-ajax.php -->
  <head>
    <meta charset="utf-8"/>
    <script src="dropdown-ajax-functions.js"></script>
  </head>
  <body>
    <?php require_once "dropdown-php-function.php"; ?>
    <form name="f_profile" action="" method="GET">
      <pre>
      Chọn tỉnh <select name="s_tinh" onchange="chonTinh(this.value)">
                  <? gen_tinh_options(); ?>
                </select>
      Chọn huyện <select name="s_huyen"> </select>
      </pre>
    </form>
  </body>
</html>
```



# AJAX Example – Dropdown List

```php
<?php   /* dropdown-php-function.php */
  function gen_tinh_options() {
    require_once "connect-select-db.php";
    $query = "SELECT * FROM tinh";
    $result = $conn->query($query)
      or die("SELECT failed: " . $conn->error);
    echo "<option value=''></option>";
    while ($row = $result->fetch_assoc()) {
      echo "<option value='" . $row['matinh'] . "'>"
           . $row['tentinh'] . "</option>";
    }
  }
?>
```

# AJAX Example – Dropdown List

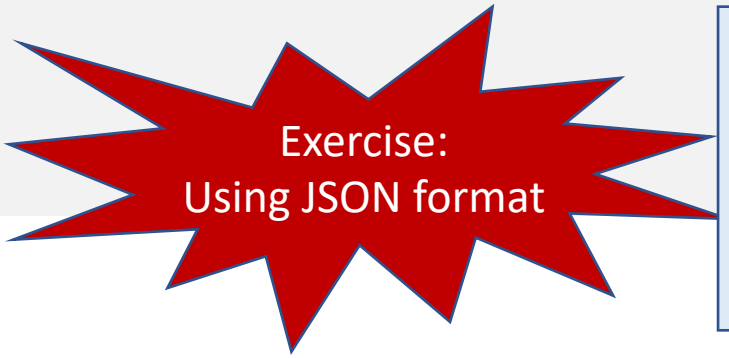
```
/* dropdown-ajax-function.js */
function chonTinh(matinh) {
  xmlhttp = new XMLHttpRequest();
  xmlhttp.onreadystatechange = function() {
    if (xmlhttp.readyState==4 && xmlhttp.status==200) {
      refreshHuyen(f_profile.s_huyen, xmlhttp.responseText);
    }
  }
  xmlhttp.open("GET", "get-huyen.php?matinh=" + matinh, true);
  xmlhttp.send();
}
```

# AJAX Example – Dropdown List

```
/* dropdown-ajax-function.js */
function refreshHuyen(selectHuyen, data) {
  //data: maHuyen&&tenHuyen;;maHuyen&&tenHuyen;;...
  //remove all items
  var length = selectHuyen.options.length;
  for (i=0; i<length; i++)
    selectHuyen.remove(0);
  //add new data
  data = data.split(";;");
  for (i=0; i<data.length-1; i++) {
    var huyen = document.createElement("option");
    var itemData = data[i].split("&&");
    huyen.value = itemData[0]; huyen.text = itemData[1];
    selectHuyen.add(huyen);
  }
}
```

# AJAX Example – Dropdown List

```php
<?php     /* get-huyen.php */

  if (isset($_GET['matinh'])) {
    require_once "connect-select-db.php";
    $matinh = $_GET['matinh'];
    $result = $conn->query("SELECT * FROM huyen WHERE matinh=$matinh")
      or die("Retrieve data failed: " . $conn->error);

    while ($row = $result->fetch_assoc()) {
      echo "$row[mahuyen]&&$row[tenhuyen];;";
    }
  }
?>
```




**get-huyen.php:**
- set content-type to JSON (header)
- use json_encode() to encode data

**refresh-huyen.php:**
- use JSON.parse() to decode data

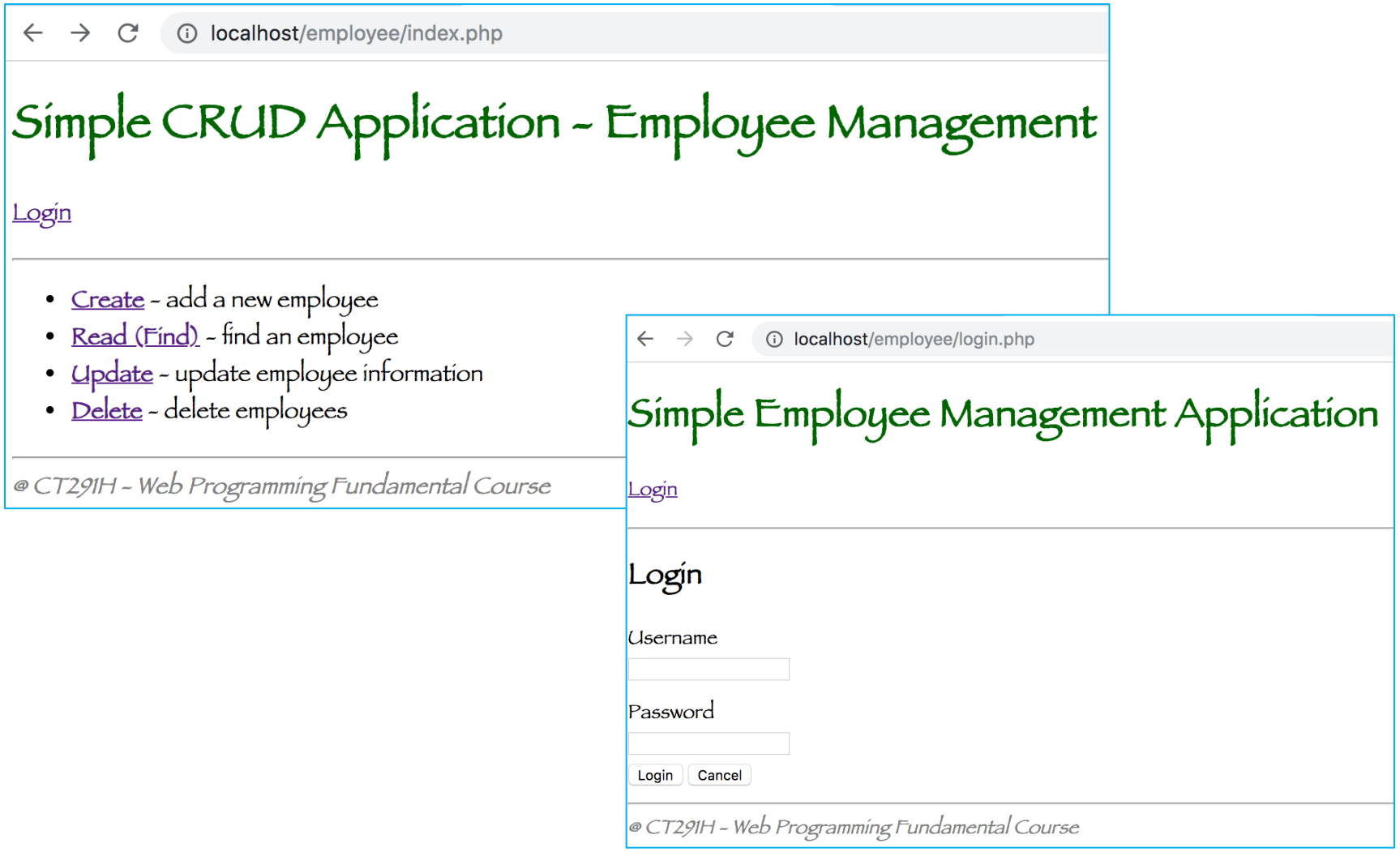
# Question?

Chapter 4 – PHP

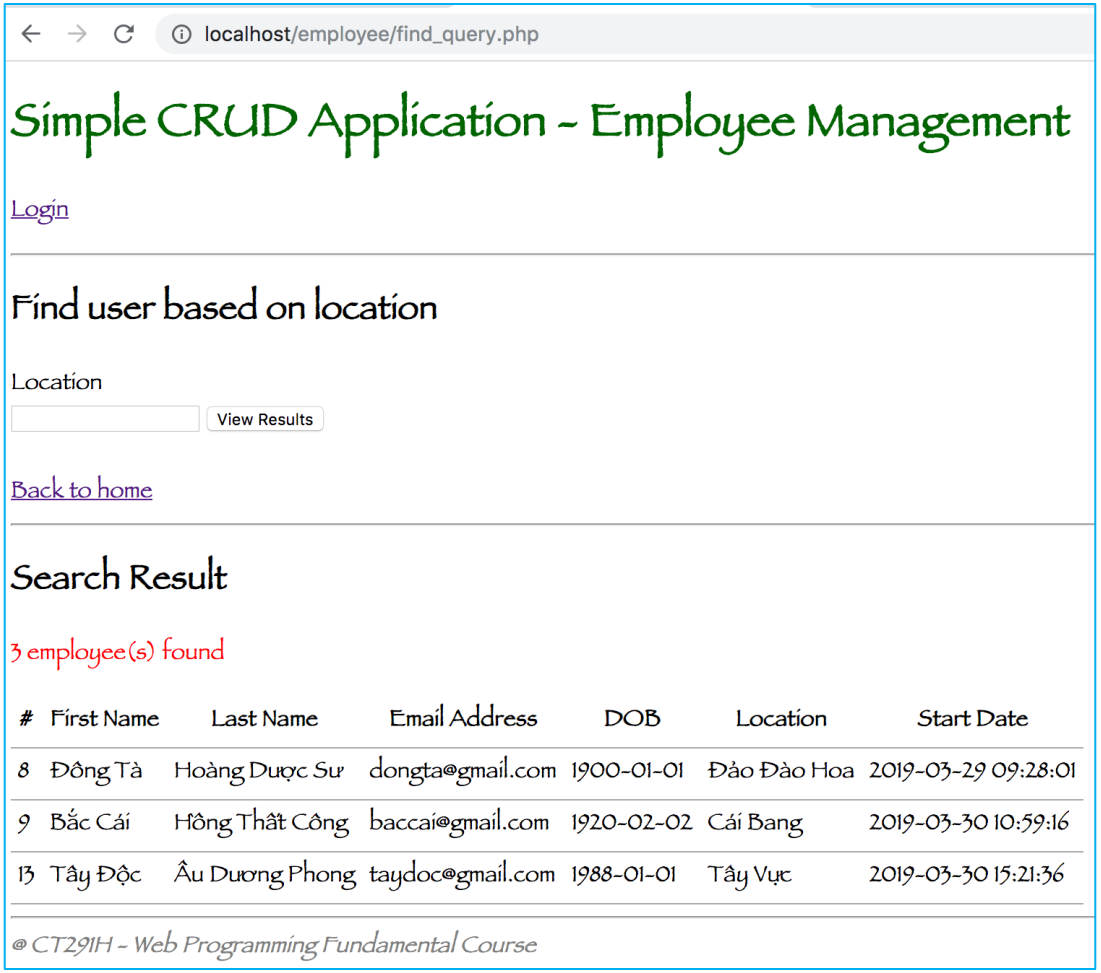
# Appendix

*A simple CRUD "Employee Manager Application"*

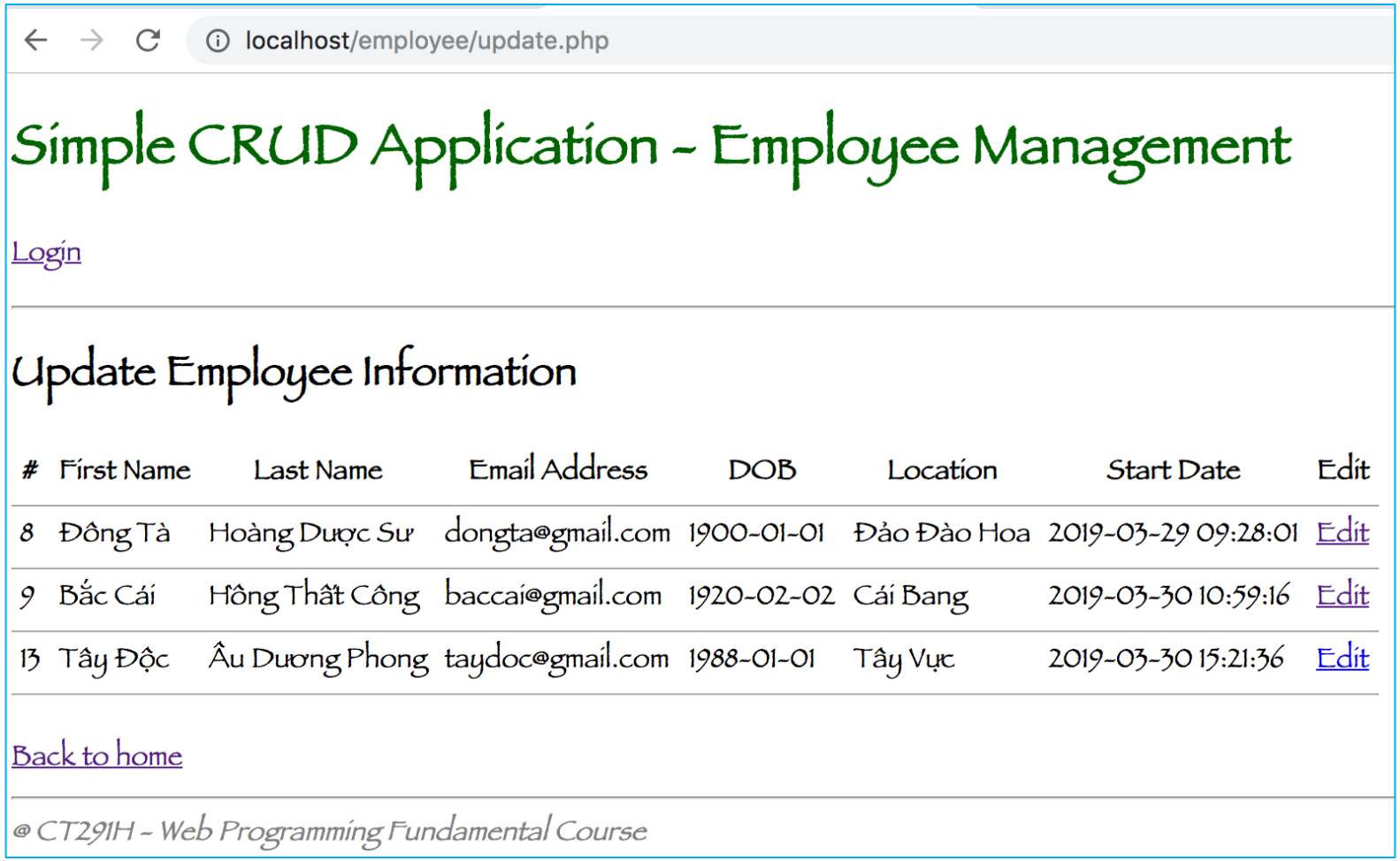
# Application GUI



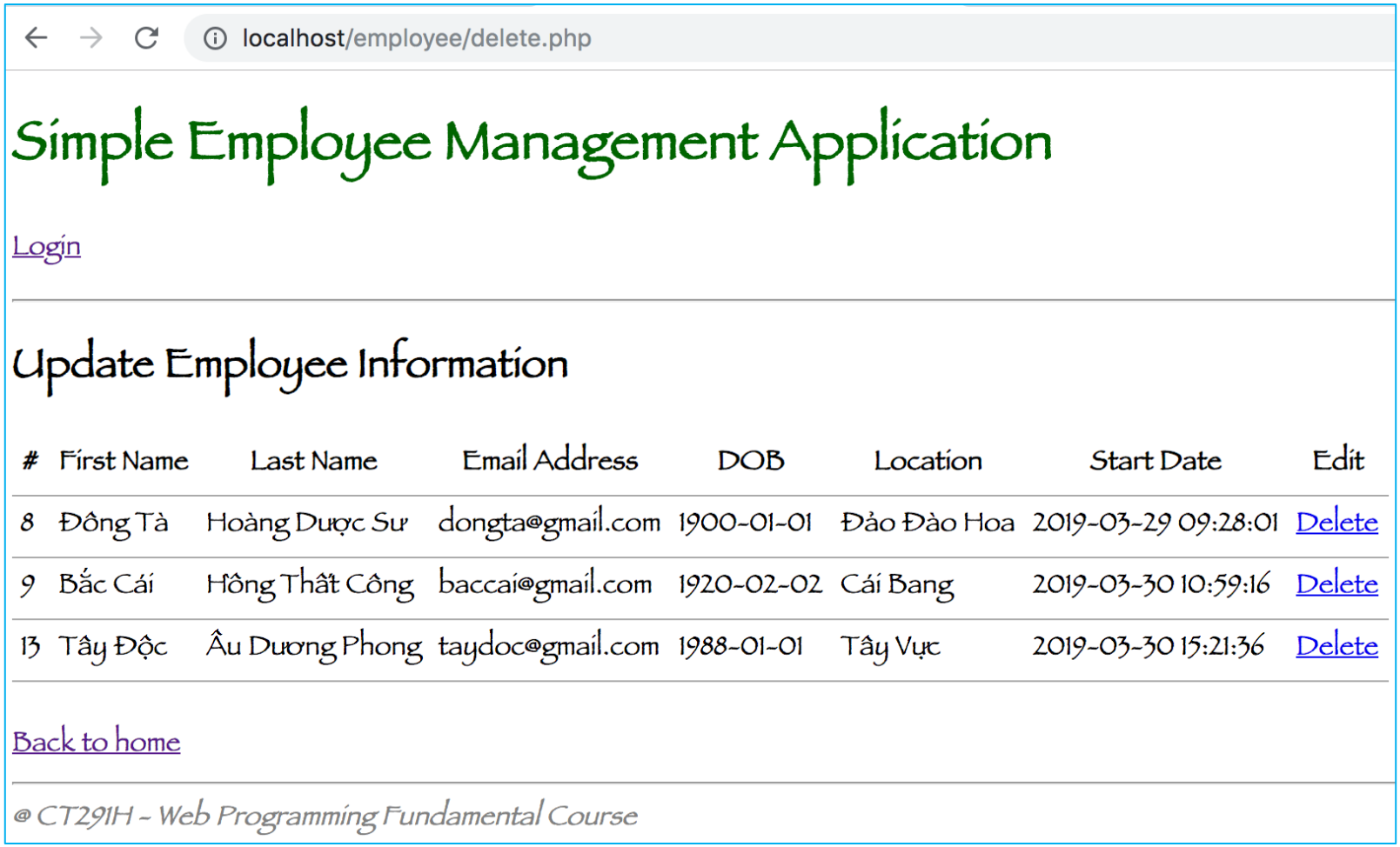
# Application GUI



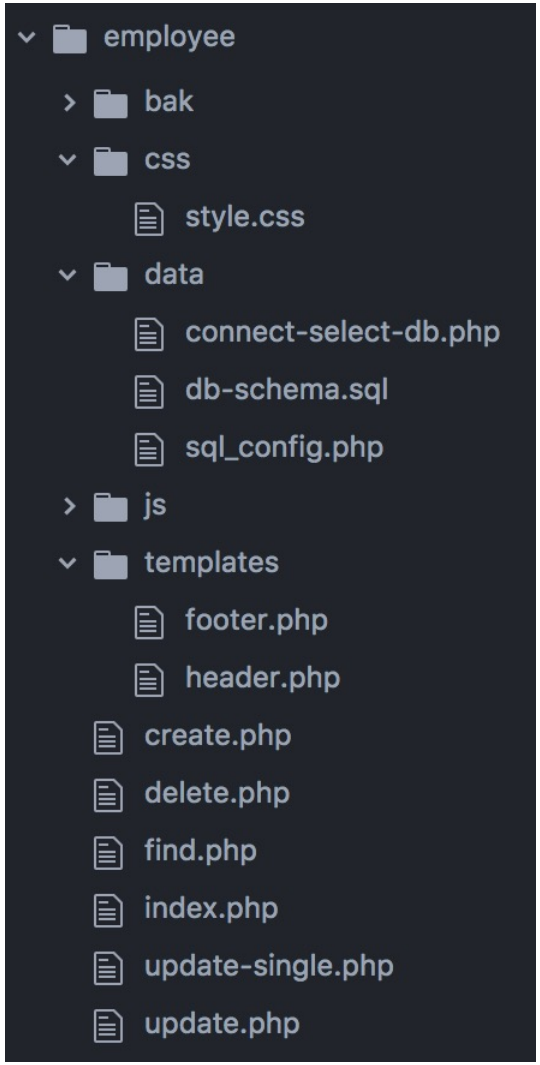
# Application GUI



# Application GUI



# Application and DB Structure




```sql
CREATE TABLE employee (
    id INT(11) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    firstname VARCHAR(30) NOT NULL,
    lastname VARCHAR(30) NOT NULL,
    email VARCHAR(50) NOT NULL,
    dob DATE,
    location VARCHAR(50),
    startdate TIMESTAMP
);
```

style.css

```css
body {
  font-family: fantasy;
}

.page-title {
  color: darkgreen;
}

address {
  color: gray;
}

label {
```

sql_config.php

```php
<?php
  $host       = "127.0.0.1";
  $username   = "root";
  $password   = NULL;
  $dbname     = "employee";
  $charset    = "utf8";
?>
```

# Connect to MySQL

connect-select-db.php

```php
<?php
  require "sql_config.php";

  //create a connection to mySQL and select the 'employee' database
  $conn = new mysqli($host, $username, $password, $dbname)
    OR die("Couldn't connect to the '$dbname' DB: " . $conn.connect_error);

  $conn->query("SET NAMES '$charset'")
    OR die("Couldn't load chaset utf8: " . $conn->error);
?>
```

# Header and Footer

header.php

```php
<?php
  session_start();
?>
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title><?php echo $page_title; ?></title>
  <link rel="stylesheet" href="css/style.css?v=<?php echo time()?>" />
</head>
<body>
<h1 class='page-title'><?php echo $page_title; ?></h1>
<hr>
```

footer.php

```php
<hr>
<address>@ CT291H – Web Programming Fundamental Course</address>
</body>
</html>
```

# Index Page

index.php

```php
<?php
$page_title = "Simple CRUD Application - Employee Management";
include "templates/header.php";
?>

<ul>
  <li>
    <a href="create.php"><strong>Create</strong></a> - add a new employee
  </li>
  <li>
    <a href="find_query.php"><strong>Read (Find)</strong></a> - find an employee
  </li>
  <li>
    <a href="update.php"><strong>Update</strong></a> - update employee information
  </li>
  <li>
    <a href="delete.php"><strong>Delete</strong></a> - delete employees
  </li>
</ul>

<?php
include "templates/footer.php";
?>
```

# Adding New Employees (Sample)

create.php

```php
<?php
  $page_title = "Simple CRUD Application - Employee Management";
  include "templates/header.php";
?>
<h2>Add Employees</h2>
<?php
  if (isset($_POST['submit'])) {
      //get form data and insert new employee into DB...
  }
?>
<form method="post">
  <label for="firstname">First Name</label>
  <input type="text" name="firstname" id="firstname">
  <label for="lastname">Last Name</label>
  <input type="text" name="lastname" id="lastname">
  <!-- ... -->
  <input type="submit" name="submit" value="Submit">
</form>
<br> <a href="index.php">Back to home</a>

<?php
include "templates/footer.php";
?>
```

# Update Form Generation (update.php)

```
<?php
foreach ($result as $row) {
?>
  <tr>
    <td><?php echo ($row["id"]); ?></td>
    <td><?php echo ($row["firstname"]); ?></td>
    <td><?php echo ($row["lastname"]); ?></td>
    <td><?php echo ($row["email"]); ?></td>
    <td><?php echo ($row["dob"]); ?></td>
    <td><?php echo ($row["location"]); ?></td>
    <td><?php echo ($row["startdate"]); ?> </td>
    <td><a href="update-single.php?id=<?php echo $row['id']; ?>">Edit</a></td>
    </tr>
<?php
} //foreach
?>
```

# Update Employee Information

update-single_bak.php

```php
<?php
  $page_title = "Simple CRUD Application - Employee Management";
  include "templates/header.php";
?>
<h2>Update Employee Information</h2>
<?php
  if (isset($_POST['update'])) {    //POST: update data
    require 'data/connect-select-db.php';
    //get form data (using $_POST), store in $emp variable and update data...
  }
?>
<?php
  if (isset($_GET['id'])) {         //GET: show data for update
    require_once('data/connect-select-db.php');
    //get id of the employee being updated (using $_GET) into $emp variable
  }
?>
<form method="post" action="<?php echo $_SERVER["PHP_SELF"];?>">
    <label for="id">ID</label>
    <input type="text" name="id" id="id" value="<?=$emp['id']?>" readonly>
    <label for="firstname">First Name</label>
    <input type="text" name="firstname" id="firstname" value="<?=$emp['firstname']?>">
    <!-- ... other form controls -->
    <input type="submit" name="update" value="Update">
</form>
<br><a href="index.php">Back to home</a>
```

# Login, Logout and Permission Checking

Session handling

# The End!