PHP

Saba sadat Faghih Imani

2023

PHP Introduction

- PHP stands for
  - Originally: “Personal Home Pages”
  - Now: “PHP: Hypertext Preprocessor”
    - Recursive acronym such as GNU ;-)
- Widely-used scripting language
- Specially suited for web development
  - Server side scripting ->Dynamic Content
  - Typically runs on a web server that takes PHP as input and gives out HTML pages as output

PHP Features

- Open source & free
- A syntax similar to C and Java
- Connects with 20+ databases
- Version 5+ supports OOP
- Multi-platform compatible
- Rich library: Over 1000 built-in functions
- Easy to learn

PHP Scripts

- Typically file ends in .php
  - Set by the web server configuration
- PHP scripts run when sent a GET/POST request to them
- PHP commands can make up an entire file, or can be contained in html
  - Server recognizes embedded script and executes
- Separated in files with the `<?php ?>` tag
  - Or `<? ?>` tag
  - Can be placed anywhere in the document
- Result passed to browser, source isn't visible

The PHP “Hello World”: Server Side

```
<!DOCTYPEhtml>
<html>
<head>
</head>
<body>
```

Sent to client in “copy mode”

```
<?php
print "<h1>Hello World</h1>";
?>
```

Parsed and output is sent to client in “interpret mode”

```
</body>
</html>
```

The PHP “Hello World”: Client Side

```
<!DOCTYPEhtml>
<html>
<head>
</head>
<body>
<h1>Hello World</h1>
</body>
</html>
```

The PHP “Hello World-2”: Server Side

```
<!DOCTYPEhtml>
<html>
<body>
```

```
<?php
$i=10;
?>
```

```
<p>Hello World</p>
```

```
<?php
print "<h1>i= ".$i."</h1>\n";
?>
```

```
</body> </html>
```

Sent to client in “copy mode”

Parsed and output is sent to client in “interpret mode”

The PHP “Hello World-2”: Client Side

```
<!DOCTYPE html>
<html>
<body>
<p>Hello World</p>
<h1>i= 10</h1>
</body> </html>
```

PHP Basic

PHP Syntax

- The syntax of PHP is very similar to C/C++/Java
- Required semicolon after each statement.
- In PHP, keywords (e.g. if, else, while, echo, etc.), classes, functions, and user-defined functions are not case-sensitive.
- However; all variable names are case-sensitive!
- Commenting
  - Single line comment: # and // Multi-line comment: /* */
- Conditional Statements: `if, if…else, if…elseif…else, switch`
- Loops: `while, do…while, for, foreach`

PHP Variables

- Rules for PHP variables:
  - A variable starts with the $ sign, followed by the name of the variable
  - A variable name must start with a letter or the underscore character
  - A variable name cannot start with a number
  - A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
  - Variable names are case-sensitive

Variable Scope

- The scope of a variable defined within a function is local to that function
- A variable defined in the main body of code has a global scope but it is NOT available inside functions!!!
- To use global variables in functions, it is referenced "**global**" keyword

```
<?php
$gvar= 10;
function f(){ global $gvar; $lvar= $gvar; }
?>
```

- Static variables are also supported by the **static** keyword
  - when a function is completed/executed sometimes we want a local variable NOT to be deleted.

PHP Constants

- A constant is an identifier (name) for a simple value. The value cannot be changed during the script.
- A valid constant name starts with a letter or underscore (no $ sign before the constant name).
- **Note:** Unlike variables, constants are automatically global across the entire script.
- To create constant:
  - `define(name, value, case-insensitive)`
  - `const` keyword

Default is false

- ```
  <?php
  define("GREETING", "Welcome to W3Schools.com!");
  echo GREETING;
  ?>
  ```
  ```

PHP Operators

- PHP divides the operators in the following groups:
- Arithmetic operators
  - +, -, *, /, %, **
- Assignment operators
- Comparison operators
- Increment/Decrement operators
- Logical operators
- String operators
  - Concatenation: .
- Array operators
- Conditional assignment operators
  - ?: , ??

PHP Arrays

- Similar to C/C++/… the index of array can be an integer number
  - Numeric array
- Similar to JS the index of array can be everything
  - Associative array
  - Mapping between key (index) and value
- Similar to other languages array containing one or more arrays
  - Multidimensional array
- Arrays can also be created by `array` function

PHP Arrays

- Numeric arrays
  - `$cars[0]="Saab"; $cars[1]="Volvo";`
    `$cars[2]="BMW"; $cars[3]="Toyota";`
  - `$cars=array("Saab","Volvo","BMW","Toyota");`
- Associative arrays
  - `$ascii["A"]=65; $ascii["B"]=66; $ascii["C"]=67`
  - `$ascii = array("A"=>65, "B"=>66, "C"=>67);`
- Multidimensional arrays
  - `$std=array("one"=>array("Ali", 1122, 20),`
    `"two"=>array("Hossein", 1133, 15));`

PHP Superglobals

- Some predefined variables in PHP are “superglobals”
  - they are always accessible, regardless of scope - and you can access them from any function, class or file without having to do anything special.
- `$GLOBALS`: All global variables, the variable names are the keys of the array.
- `$_REQUEST`: Contains `$_GET`, `$_POST` and `$_COOKIE`
- `$_POST`: Variables passed by HTTP POST
- `$_GET`: Variables passed in URL’s query part
- `$_FILES`: Uploaded file information
- `$_COOKIE`: Cookies sent by HTTP cookie header
- `$_SESSION`: Session variables

PHP Superglobals

- $_SERVER: Information such as headers, server and client
  - Examples of the important keys:
  - `'SERVER_ADDR':` The IP address of the server
  - `'SERVER_NAME':` The name of the server host
  - `'SERVER_PORT':` The port of web server
  - `'REQUEST_METHOD':` The HTTP request method
  - `'HTTP_USER_AGENT':` Contents of the HTTP User-Agent
  - `'REMOTE_ADDR':` The IP address of client

- Complete list: php.net/manual/en/index.php

echo & print & var_dump

```
<?php
$foo= 25;                   // Numerical variable
$bar = "Hello";             // String variable
echo $bar."\n";             // Outputs Hello
echo $foo,$bar,"\n";        // Outputs 25Hello
echo "5x5=".$foo."\n";      // Outputs 5x5=25
echo "5x5=$foo\n";          // Outputs 5x5=25
echo '5x5=$foo\n’;          // Outputs 5x5=$foo\n
print "\n";// newline
print "Output is ".$foo;    // Output is 25
var_dump($foo);             // int(25)
?>
```

PHP include and require

- Complex server side processing ->lot of PHP codes
  - Avoid mixing HTML design & PHP
  - Break processing into multiple files (team working)
- Four functions to insert code from external files
  - `include`: Try to insert file, continues if cannot find it
    - `include_once 'A':` does not include 'A' if it is already included even by other included files 'B'
  - `require`: Try to insert external file, dies if cannot find it
    - `require_once:` does not include if file is already included
- The included code is interpreted & run (if is not function)
- An implementation of server side include(SSI)

```
<html> <body> <?php include 'header.php'; ?>
```

File System Operations

- PHP filesystem operations are similar to C
- `readfile()`: reads a file and writes it to the output buffer.
- File open/read:
  - `fopen()`: The file may be opened in one of the following modes
    - `r, w, a, x, r+, w+, a+, x+`
  - `fread()`: reads from an open file.
  - `fclose()`
  - `fgets()`: is used to read a single line from a file.
  - `feof()`: checks if the "end-of-file" (EOF) has been reached.
  - `fgetc()`: is used to read a single character from a file.

File System Operations

- File create/write:
- `fopen(), fwrite(), fclose();`
- `fopen` opens URL of supported protocols
  - file://, http://, ftp://, …
  - php://stdin, php://stdout, php://stderr
- To open binary files safely: b

File System Operations

- To increase security of web-servers, the `fopen` function may be disabled
  - So, none of the previous functions can be used
- Alternative functions (limited functionalities)
- `file_get_contents`: To read file content into a string
- `file_put_contents`: To write a string into a file

Simple Web Page Counter

```
<?php
$data = file_get_contents("counter");
$data = $data + 1;
file_put_contents("counter", $data);
echo "This page has been viewed " . $data . " times ";
?>
```

PHP In Web Application

Input Data Handling

- One of the main functionalities of server side scripting is to process user input data, e.g.
  - Save data on server
  - Login & Sessions
  - Query from database server
  - …
- Input data from forms/Ajax/API/ …
  - GET method
  - POST method
    - File upload

Input Data Handling

- Major steps of input data handling:
- 1) Read the data
  - How to read the URL query part? Post data? File?
- 2) Check presence & existence
  - Is variable set? Is it empty?
- 3) Validation
  - Is data valid? Correct format?
- 4) Processing
  - Application dependent, e.g., query to DB, ….

1)Reading Submitted Data

- Main feature: data sent in “URL Query Part” or “Packet Body” are automatically available to PHP scripts by the run-time environment
  - Does not matter HTML form or Ajax or …
- The PHP pre-assigned `$_GET` and `$_POST` variables are used to retrieve the data
- The predefined `$_REQUEST` variable contains the contents of `$_GET`, `$_POST`, `$_COOKIE`
- The `$_REQUEST` variable can be used to collect form data sent with both GET and POST methods

1)Reading Submitted Data

- $_GET, $_POST, and $_REQUEST are associative arrays
  - Key is the name attribute of input element in a form
  - Value is the value of the input element in a form
- HTML

```
<form method="GET" action="index.php">
<input type="text" name="grade" value="">
</form>
```

- PHP

```
$g = $_GET["grade"];
```

GET vs. POST

- GET is used to request data from a specified resource.
- Note that the query string (name/value pairs) is sent in the URL of a GET request:

  `/test/demo_form.php?name1=value1&name2=value2`

- GET requests can be cached
- GET requests remain in the browser history
- GET requests can be bookmarked
- GET requests should never be used when dealing with sensitive data
- GET requests have length restrictions

GET vs. POST

- POST is used to send data to a server to create/update a resource.
- The data sent to the server with POST is stored in the request body of the HTTP request
- POST requests are never cached
- POST requests do not remain in the browser history
- POST requests cannot be bookmarked
- POST requests have no restrictions on data length

2) Checking Input Presence/Existence

- **`isset($var)`** is false if and only if `$var` is NULL
  - i.e., either `$var` does not exist or is never assigned a value
  - Use this function to check if a check box, radio button, or select box list has a value
- **`empty($var)`** is true if `$var` is 0, empty string, NULL, or FALSE
  - Use this function to check if a text field, password field, or text area has a value that is not an empty string
  - These input fields are always set -> `isset` does not work!

Form Processing Example

```
<form method="post" action="form.php">
Submit By Post!!
<fieldset>
<legend>University Grade</legend>
          <input type="radio" name="grade" value="BS" /> BS
          <input type="radio" name="grade" value="MS" /> MS
          <input type="radio" name="grade" value="PhD" /> PhD
</fieldset>
<fieldset>
<legend><em>Web Development Skills</em></legend>
          <input type="checkbox" name="skill_1" value="html" />HTML
          <input type="checkbox" name="skill_2" value="xhtml" />XHTML
          <input type="checkbox" name="skill_3" value="cs" />CSS
          <input type="checkbox" name="skill_4" value="js" />JavaScript
          <input type="checkbox" name="skill_5" value="aspnet" />ASP.Net
          <input type="checkbox" name="skill_6" value="php" />PHP
</fieldset>
```

Form Processing Example (cont'd)

```
Favorite Programming Language:
<select name="lang">
<option value="c">C</option>
<option value="java">Java</option>
<option value="python">Python</option>
</select>
<input type="submit" value="Submit" />
</form>

<form method="get" action="form.php">
          <fieldset>
          <legend> Submit by GET </legend>
          Title: <input type="text" length="20" name="title" />
          Name: <input type="text" length="20" name="name" />
          Family: <input type="text" length="20" name="fam" />
          <input type="submit" value="Submit" />
          </fieldset>
</form>
```

Form Processing Example (cont’d)

```
$grade = $_POST["grade"];
$lang= $_POST["lang"];
if(isset($grade))
         echo "You are ". $grade;
else
         echo "I don't know your grade";
echo "<br/>";

echo "You are master in ";
for($i= 1; $i< 7; $i++)
         if(isset($_POST["skill_".$i]))
                  echo$_POST["skill_".$i]. " ";
echo "<br/>";
echo "You love ". $lang;
```

Form Processing Example (cont’d)

```
$name = $_GET["name"];
$fam= $_GET["fam"];
$title = $_GET["title"];

if((! empty($name)) && (! empty($fam)) && (! empty($title))){
         echo "A message by GET <br/>";
         echo "<h2> Welcome " . $title ." ". $name ." ". $fam." </h2>";
}
```

3) Input Data Validation

- Be very very careful about input data
  - Maybe they are coming from bad guys
- There is a HTML form corresponding to PHP
  - On client side, we (developers) try to validate input data by JavaScript
    - We cannot fully & completely validate the data
  - What happen if attacker want to inject code/data
    - He does not use our forms
    - No data validation on client side
- Server side data validation is required

PHP Filters

- PHP filters to make data filtering easier
- Two kinds of filters:
  - Validating filters:
    - Are used to validate user input
    - Strict format rules (like URL or E-Mail validating)
    - Returns the expected type on success or FALSE on failure
  - Sanitizing filters:
    - To allow or disallow specified characters in a string
    - Remove the invalid characters
    - Always return a valid output

PHP Filters

- Filters are applied by these functions:
- `filter_var(`variable, filter`)`: Filters a single variable.
- `filter_var_array(`array of variables, array of filters`)`: Filter several variables with a set of filters
- `filter_input(`type, variable, filter`)`: Get one input variable from given type and filter it, e.g. INPUT_GET, INPUT_POST, …
- `filter_input_array(`type, filters`)`: Get several input variables and filter them with specified filters

PHP Filters

- Each filter has a unique id (integer number)
  - `FILTER_VALIDATE_INT` ->257
  - `FILTER_VALIDATE_FLOAT` ->259
  - Filtering functions decide based on the value
- A filter can have options and flags
  - E.g., for `FILTER_VALIDATE_INT`
    - Option: `max_range` and `min_range`
    - Flag: `FILTER_FLAG_ALLOW_OCTAL`
- Flag and options are passed using <u>associative arrays with keys "options"& "flags"</u>

PHP Filters: Filtering a Variable

```
$i= 10;
$j = filter_var($i, FILTER_VALIDATE_INT);
if($j)
       echo "1-j = ". $j . "\n";
else
       echo "1-Data is not valid\n";
$fdata= array("options"=>array("min_range"=>15, "max_range"=>50));
$j = filter_var($i, FILTER_VALIDATE_INT, $fdata);
if($j)
       echo "2-j = ". $j . "\n";
else
       echo "2-Data is not valid\n";
```

PHP Filters: Filtering an Array of Variables

```
$data = array("int"=>10, "float"=>30.1);

$filter = array("int"=>array("filter"=>FILTER_VALIDATE_INT, "options"=>array("min_range"=>0)),
         "float"=>array("filter"=>FILTER_VALIDATE_FLOAT));

$valid = filter_var_array($data, $filter);

var_dump($valid);


$data = array("int"=>"a1z0", "float"=>30.1);

$valid = filter_var_array($data, $filter);

var_dump($valid);


$filter2 = array("int2"=>array("filter"=>FILTER_VALIDATE_INT, "options"=>array("min_range"=>0)),
         "float"=>array("filter"=>FILTER_VALIDATE_FLOAT));

$valid = filter_var_array($data, $filter2);

var_dump($valid);
```

Filtering Input Data

- Types:
  - `INPUT_GET`, `INPUT_POST`, `INPUT_COOKIE`, …
- To (optionally) apply a filter F on an input with name N with type T and get valid data

  `filter_input`(T, N, F)
- To (optionally) apply filter F on array of inputs with type T

  `filter_input_array`(T, F)
- Output specified by the keys in the filter

Filtering Input Data Example

- Assume: URL:/filter.php?ip=192.168.0.1&address=http://wwww.abc.com

```
$valid_address= filter_input(INPUT_GET, "address", FILTER_VALIDATE_URL);

$filter = array("address"=>array("filter"=>FILTER_VALIDATE_URL),
"ip"=>array("filter"=>FILTER_VALIDATE_IP));

$valid_get= filter_input_array(INPUT_GET, $filter);
```

Extracting Valid Data

- Sanitize filters generate valid data from input
- FILTER_SANITIZE_EMAIL
- FILTER_SANITIZE_NUMBER_FLOAT
- FILTER_SANITIZE_NUMBER_INT
- FILTER_SANITIZE_URL
- …

```
echo filter_var("a b c", FILTER_SANITIZE_ENCODED);
```

- a%20b%20c

```
echo filter_var("ab123ca", FILTER_SANITIZE_NUMBER_INT);
```

- 123

Implementing Custom Filter

- Filter type FILTER_CALLBACK is used to register a custom filter

```
function convertSpace($string){
      return str_replace("_", " ", $string);
}

$string = "PHP_Scripting_is_fun!";
echo filter_var($string, FILTER_CALLBACK,
array("options"=>"convertSpace"));
```

Database

- Many databases; here, MySQL
- Fast review of databases' basics
  - Database, Table, Row, …
  - Database creation
  - Database modification
  - Database query
- MySQL in PHP
  - Database connection
  - Modification & Query

Database Basic Concept

- Relational database
  - Database server contains multiple databases
  - Each database consists of multiple tables
  - Each table is defined by its columns (& types)
  - Each row is a data record
  - A column is the primary key
    - A unique identifier for each record
- We use Structured Query Language (SQL) for database management
  - A famous SQL based database => MySQL
    - Free, Open source, and multiplatform

SQL Commands: Create Table

```
CREATE TABLE students(
      name VARCHAR(55), num INT(3),
      grade DECIMAL(20,2),
      primary key(num)
);
```

- Types
  - **TEXT**, **CHAR**(size), **VARCHAR**(maxsize), **INT**(maxsize), **DECIMAL**(maxsize, precision) , **DATE**(), **YEAR**(),….
- For primary key
  - `id INT AUTO_INCREMENT, primary key(id)`

SQL Commands (cont'd)

- Inserting data
  - **INSERT INTO** tabelname(column1, …) **VALUES**(val1, …);
  - **INSERT INTO** students(name,grade,num)
    VALUES ("Ali", 15.23, 1122);
- Querying data
  - **SELECT** columnname **FROM** table **WHERE** condition
    - **SELECT*** **FROM** students **WHERE** grade=20
- Conditions by comparison & logical
  - =, !=, <, <=, >, >=, …
  - AND, OR

SQL Commands (cont'd)

- Updating records
  - `UPDATE tablename SET col1=val1, col2=val2, … WHERE condition`
    - `UPDATE student SET grade=20 WHERE name='Ali';`

- Deleting a record from a table
  - `DELETE FROM tablename WHERE condition;`
  - E.g. clear the students table
    - `DELETE FROM students;`

- Deleting a table
  - `DROP TABLE tablename;`

MySQL in PHP

- There are three interfaces (API) to access MySQL in PHP
- The Old API
  - Functions start by **`mysql_`**
  - Now deprecated, will be removed
  - However, very popular, lot of web applications based on
- The New Improved Extension
  - Available in two modes
  - Procedural mode: functions start by **`mysqli_`**
    - Very similar to the old API, with minor differences & new features
  - Object oriented mode
    - The same functions but as a method of objects
- PDO (PHP Data Objects)

MySQL in PHP (cont'd)

- In general, all APIs follow the same concept to work with MySQL DB
  - Functions & Parameters are different
- The steps of the follow
  - Connect to the database server
  - Select the database in the server
  - Send SQL queries to the tables of the database
  - Process the result (typically as an array)
  - Close the connection

MySQL in PHP: Connecting & Selecting

- The first step to work with MySQL
  - 1) Connecting to the MySQL server
  - 2) Selecting database
  - Required for all operations on database

```
$mysqli= mysqli_connect("server address", "username","password",
"DB name") or die(mysqli_connect_error());
```

- We don’t want to continue if it fails

MySQL in PHP: SQL Commands

- SQL commands are send by

```
mysqli_query($mysqli,"SQL Command")
```

  - Syntax is the SQL
- E.g., Create table in the selected database

```
mysqli_query($mysqli, "CREATE TABLE students(
id INTAUTO_INCREMENT,
primary key(id),
name VARCHAR(50),
stdnum INT(8))");
```

MySQL in PHP: Query & Closing

- Query result is processed by **mysqli_fetch_***
- E.g., **mysqli_fetch_assoc()**

```
$result = mysqli_query($db, "SELECT ...");
while($row = mysqli_fetch_assoc($result)){
      $std_name= $row['name’];
      $std_grade= $row['grade'];
}
mysqli_free_result($result);
```

- Close database connection:

```
mysqli_close($mysqli)
```

Example

- Database: students
- Table: CE
  - (name, fam, grade, num)
- datainput.html: HTML form to insert data
- dbinsert.php: Insert data to DB
- datasearch.html: HTML form to query
- dbsearch.php: Run the query and show result

Example: datainput.html

```
<html>
<head>
</head>
<body>
<form action="dbinsert.php" method="GET">
        Name: <input type="text" name="n" /><br/>
        Family: <input type="text" name="f" /><br/>
        Std #: <input type="text" name="i" /><br/>
        Grade: <input type="text" name="g" /><br/>
        <input type="submit" value="Insert Data" />
</form>
</body>
</html>
```

Example: dbinsert.php

```
<?php
$name = $_REQUEST["n"]; $famanme= $_REQUEST["f"];
$grade = $_REQUEST["g"]; $num = $_REQUEST["i"];

if((strlen($num) > 0) && (strlen($famanme) > 0) && (strlen($grade) > 0) && (strlen($num) > 0)){
        $db = mysqli_connect("127.0.0.1", "root", "12345678", "students") or die(mysqli_connect_error());

        $result = mysqli_query($db, "INSERT INTO CE(name, fam, num, grade) VALUES('$name’, '$famanme’, '$num’,
                  '$grade');") or die(mysqli_error($db));

        mysqli_close($db);
        echo "Data has been inserted successfully <br/>";
}
else{
        echo "Wrong Input";
}
?>
```

Example: datasearch.html

```
<html>
<head>
</head>
<body>
        <form action="dbsearch.php" method="GET">
        Parameter:
        <select name="col">
                <option value="name">Name</option>
                <option value="fam">Family</option>
                <option value="grade">Grade</option>
                <option value="num">Student #</option>
        </select>
        <input type="text" name="query" /> <br/>
        <input type="submit" value="Search" />
        </form>
</body>
</html>
```

Example: dbsearch.php

```
<?php
$column = $_REQUEST["col"]; $value = $_REQUEST["query"];


if((strlen($column) > 0) && (strlen($value) > 0)){
        $db = mysqli_connect("127.0.0.1", "root", "12345678", "students") or die(mysqli_connect_error());


        $result = mysqli_query($db, "SELECT name,fam,num,grade FROM CE WHERE $column='$value' ORDER BY grade DESC") or
        die(mysqli_error($db));


        while($row = mysqli_fetch_assoc($result))
                echo "Name: ", $row["name"], ", Family: ", $row["fam"], ", Std #: ", $row["num"], ", Grade: ",
                $row["grade"], "<br/>";
        mysqli_free_result($result);
        mysqli_close($db);
}
else{
        echo "Wrong Input";
}
?>
```