PHP: Hypertext Preprocessor

- PHP is a widely-used, open source scripting language
- PHP scripts are executed on the server
- PHP is free to download and use
- PHP files can contain text, HTML, CSS, JavaScript, and PHP code

Features

- 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

- PHP runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)
- PHP is compatible with almost all servers used today (Apache, IIS, etc.)
- PHP supports a wide range of databases
- PHP is free. Download it from the official PHP resource: www.php.net
- PHP is easy to learn and runs efficiently on the server side

PHP Installation

- install a web server (WAMP, XAMPP,IIS etc.)
- install PHP
- install a database, such as MySQL

Basic Syntax

- **<?php**
  // PHP code goes here
  ?>

- A PHP file normally contains **HTML tags**, and some PHP **scripting code**.
- "echo“ is used to output the text "Hello World!" on a web page in following example:

-
  ```
  <!DOCTYPE html>
  <html>
  <body>
  <h1>My first PHP page</h1>

  <?php
  echo "Hello World!";
  ?>

  </body>
  </html>
  ```

- PHP statements end with a semicolon (;).

Comments in PHP

-
  ```
  <!DOCTYPE html>
  <html>
  <body>
  <?php
  // This is a single-line comment

  # This is also a single-line comment

  /*
  This is a multiple-lines comment block
  that spans over multiple
  lines
  */

  // You can also use comments to leave out parts of a code line
  $x = 5 /* + 15 */ + 5;
  echo $x;
  ?>

  </body>
  </html>
  ```

Case Sensitivity

- In PHP, all keywords (e.g. if, else, while, echo, etc.), classes, functions, and user-defined functions are **NOT case-sensitive.**

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

  all three echo statements are legal
- *First echo will be printed because all three $color variables are different.*

Creating (Declaring) PHP Variables

A variable starts with the $ sign, followed by the name

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

    echo $txt;
    echo "<br>";
    echo $x;
    echo "<br>";
    echo $y;
?>
```

PHP Variables

- A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume).

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** ($age and $AGE are two different variables)

PHP is a Loosely Typed Language

- *we did not have to tell PHP which data type the variable is.*

- PHP automatically converts the variable to the correct data type, depending on its value.
- In other languages such as C, C++, and Java, the programmer must declare the name and type of the variable before using it.

Variables Scope

PHP has three different variable scopes:

- local
- global
- static

Global and Local Scope

- A variable declared **outside** a function has a GLOBAL SCOPE and can only be accessed outside a function:
-
```
<!DOCTYPE html>
<html>
<body>

<?php
$x = 5; // global scope

function myTest() {
    // using x inside this function will generate an error
    echo "<p>Variable x inside function is: $x</p>";   //no value will be printed
}
myTest();

echo "<p>Variable x outside function is: $x</p>";   //print   5
?>
</body>
</html>
```

- A variable declared **within** a function has a LOCAL SCOPE and can only be accessed within that function:

-
  ```
  <?php
  function myTest() {
      $x = 5; // local scope
      echo "<p>Variable x inside function is: $x</p>";
  }
  myTest();   //print 5

  // using x outside the function will generate an error
  echo "<p>Variable x outside function is: $x</p>";
  ?>
  ```

The global Keyword

- The **global keyword** is used to access a global variable from within a function.

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

function myTest() {
   global $x, $y;
   $y = $x + $y;
}

myTest();
echo $y; // outputs 15
?>
```

- PHP also stores all global variables in an array called $GLOBALS[*index*].
- The *index* holds the name of the variable.
- This array is also accessible from within functions and can be used to update global variables directly.

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

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

  myTest();
  echo $y; // outputs 15
  ?>
  ```

The static Keyword

- Normally, when a function is completed/executed, all of its variables are deleted. However, sometimes we want a local variable NOT to be deleted. We need it for a further job.

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

  myTest();
  echo "<br>";
  myTest();
  echo "<br>";
  myTest();
  ?>
  ```

Result: 0

1

2

Note: Each time the function is called, that variable will still have the information it contained from the last time the function was called.

echo and print (output) Statements

Two basic ways to get output:

- echo
- Print

```
<?php
print "<h2>PHP is Fun!</h2>";
print "Hello world!<br>";
print "I'm about to learn PHP!";
?>
```

Difference b/w echo and print

- echo has no return value while print has a return value of 1 so it can be used in expressions.
- echo can take multiple parameters (although such usage is rare) while print can take one argument.
- echo is marginally faster than print.

```
<?php
 $x= print(“hello”);  //return 1
echo “hello”, “Bye”; //multiple parameter

  print "<h2>PHP is Fun!</h2>";
  echo "Hello world!<br>";
  print "I'm about to learn PHP!";
  ?>
```

Data Types in PHP

Variables can store data of different types.

- String
- Integer
- Float (floating point numbers - also called double)
- Boolean
- Array
- Object
- NULL

String

- A string is a sequence of characters, like "Hello world!".
- A string can be any text inside quotes. You can use single or double quotes:

```
<?php
  $x = "Hello world!";
  $y = 'Hello world!';

  echo $x;
  echo "<br>";
  echo $y;
  ?>
```

Integer

- An integer is a whole number (without decimals). It is a number between -2,147,483,648 and +2,147,483,647.

<u>Rules for integers:</u>

- An integer must have at least one digit (0-9)
- An integer cannot contain comma or blanks
- An integer must not have a decimal point
- An integer can be either positive or negative
- Integers can be specified in three formats: decimal (10-based), hexadecimal (16-based - prefixed with 0x) or octal (8-based - prefixed with 0)

In the following example $x is an integer. The var_dump() function returns the data type and value:

```
<?php
   $x = 5985;
   var_dump($x);
   ?>
```

Result: int(5985)

Float

- A float (floating point number) is a number with a decimal point or a number in exponential form.

```
<?php
   $x = 10.365;
   var_dump($x);
   ?>
```

Boolean

A Boolean represents two possible states: TRUE or FALSE.

```
$x = true;
$y = false;
```

- Booleans are often used in conditional testing.

Array

- ***An array stores multiple values in one single variable***.
- In the following example $cars is an array. The PHP var_dump() function returns the data type and value:

```
<!DOCTYPE html>
    <html>
    <body>

    <?php
    $cars = array("Volvo","BMW","Toyota");
    var_dump($cars);
    ?>

    </body>
    </html>
```

Result: **array**(3) { [0]=> string(5) "Volvo" [1]=> string(3) "BMW" [2]=> string(6) "Toyota" }

NULL Value

- Null is a special data type which can have only one value: NULL.
- A variable of data type NULL is a variable that has no value assigned to it.
- **Tip:** If a variable is created without a value, it is automatically assigned a value of NULL.
- Variables can also be emptied by setting the value to NULL:

```
<?php
   $x = "Hello world!";
   $x = null;
   var_dump($x);
   ?>
```

PHP Operators

Operators are used to perform operations on variables and values.

- **Arithmetic operators**
- **Assignment operators**
- **Comparison operators**
- **Increment/Decrement operators**
- **Logical operators**
- **String operators**
- **Array operators**

Arithmetic Operators

Operator	Name	Example	Result
+	Addition	$x + $y	Sum of $x and $y
-	Subtraction	$x - $y	Difference of $x and $y
*	Multiplication	$x * $y	Product of $x and $y
/	Division	$x / $y	Quotient of $x and $y
%	Modulus	$x % $y	Remainder of $x divided by $y
**	Exponentiation	$x ** $y	Result of raising $x to the $y'th power (Introduced in PHP 5.6)

Assignment Operators

Assignment	Same as...	Description
x = y	x = y	The left operand gets set to the value of the expression on the right
x += y	x = x + y	Addition
x -= y	x = x - y	Subtraction
x *= y	x = x * y	Multiplication
x /= y	x = x / y	Division
x %= y	x = x % y	Modulus

Comparison Operators

Operator	Name	Example	Result
==	Equal	$x == $y	Returns true if $x is equal to $y
===	Identical	$x === $y	Returns true if $x is equal to $y, and they are of the same type
!=	Not equal	$x != $y	Returns true if $x is not equal to $y
<>	Not equal	$x <> $y	Returns true if $x is not equal to $y
!==	Not identical	$x !== $y	Returns true if $x is not equal to $y, or they are not of the same type
>	Greater than	$x > $y	Returns true if $x is greater than $y
<	Less than	$x < $y	Returns true if $x is less than $y
>=	Greater than or equal to	$x >= $y	Returns true if $x is greater than or equal to $y
<=	Less than or equal to	$x <= $y	Returns true if $x is less than or equal to $y

Increment / Decrement Operators

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

Logical Operators

Operator	**Name**	**Example**	**Result**
and	And	$x and $y	True if both $x and $y are true
Or	Or	$x or $y	True if either $x or $y is true
xor	Xor	$x xor $y	True if either $x or $y is true, but not both
&&	And	$x && $y	True if both $x and $y are true
||	Or	$x || $y	True if either $x or $y is true
!	Not	!$x	True if $x is not true

String Operators

Operator	Name	Example	Result
.	Concatenation	$txt1 . $txt2	Concatenation of $txt1 and $txt2
.=	Concatenation assignment	$txt1 .= $txt2	Appends $txt2 to $txt1

Array Operators

The PHP array operators are used to compare arrays.

Operator	Name	Example	Result
+	Union	$x + $y	Union of $x and $y
==	Equality	$x == $y	Returns true if $x and $y have the same key/value pairs
===	Identity	$x === $y	Returns true if $x and $y have the same key/value pairs in the same order and of the same types
!=	Inequality	$x != $y	Returns true if $x is not equal to $y
<>	Inequality	$x <> $y	Returns true if $x is not equal to $y
!==	Non-identity	$x !== $y	Returns true if $x is not identical to $y

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).
- **To create a constant, use the define() function.**

e.g. define(*name*, *value*, *case-insensitive*)

Parameters:

- *name*: Specifies the name of the constant
- *value*: Specifies the value of the constant
- *case-insensitive*: Specifies whether the constant name should be case-insensitive. Default is false
- The example below creates a constant with a **case-sensitive** name:

```
<?php
    define("GREETING", "Welcome to
    W3Schools.com!“, true);
    echo GREETING;
echo greeting; // result swill display value because
    case-insensitivity is true
    ?>
```

Constants are Global

- Constants are automatically global and can be used across the entire script.
- The example below uses a constant inside a function, even if it is defined outside the function:

```
<?php
    define("GREETING", "Welcome to W3Schools.com!");
    function myTest() {
        echo GREETING;
    }
     myTest();
    ?>
```

Loops

- **while** - loops through a block of code as long as the specified condition is true
- **do...while** - loops through a block of code once, and then repeats the loop as long as the specified condition is true
- **for** - loops through a block of code a specified number of times
- **foreach** - loops through a block of code for each element in an array

While………Loop

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

do...while Loop

- do {
  code to be executed;
  }

for Loop

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

foreach Loop

- The foreach loop works only on arrays, and is used to loop through each key/value pair in an array.

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

- For every loop iteration, the value of the current array element is assigned to $value and the array pointer is moved by one, until it reaches the last array element.

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

  ```
  red
  green
  blue
  yellow
  ```

Array?

- **Indexed arrays** - Arrays with a numeric index
- **Associative arrays** - Arrays with named keys
- **Multidimensional arrays** - Arrays containing one or more arrays

Indexed Arrays

- There are two ways to create indexed arrays:

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

   Here, index can be assigned automatically (index always starts at 0)

2.
   ```
   $cars[0] = "Volvo";
   $cars[1] = "BMW";
   $cars[2] = "Toyota";
   ```

Here, index can be assigned manually

How to access Array element

-
  ```
  <?php
  $cars = array("Volvo", "BMW", "Toyota");
  $arrlength = count($cars);
  for($x = 0; $x < $arrlength; $x++) {
      echo $cars[$x];
      echo "<br>";
  }
  ?>
  ```

Associative Arrays

- Associative arrays are arrays that use named keys that you assign to them.
- There are two ways to create an associative array:

1. $age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
2. $age['Peter'] = "35";
   $age['Ben'] = "37";
   $age['Joe'] = "43";

-
  ```
  <?php
  $age = array("Peter"=>"35", "Ben"=>"37",
  "Joe"=>"43");
  foreach($age as $x => $x_value) {
     echo "Key=" . $x . ", Value=" . $x_value;
     echo "<br>";
  }
  ?>
  ```

Multidimensional Arrays

- A multidimensional array is an array containing one or more arrays.

Two-dimensional Arrays

- A two-dimensional array is an array of arrays

Name	**Stock**	**Sold**
Volvo	22	18
BMW	15	13
Saab	5	2

- We can store the data from the table above in a two-dimensional array,

```
<?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; $row++)
 {
         for ($col = 0; $col <  3; $col++)
{
        echo $cars[$row][$col];
      }


    }
    ?>
```

Functions

PHP has more than 1000 built-in functions.

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

```
<!DOCTYPE html>
<html>
<body>

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

writeMsg();
?>

</body>
</html>
```

Function Arguments

-
  ```
  <!DOCTYPE html>
  <html>
  <body>

  <?php
  function familyName($fname) {
      echo "$fname Refsnes.<br>";
  }

  familyName("Jani");
  familyName("Hege");
  familyName("Stale");
  familyName("Kai Jim");
  familyName("Borge");
  ?>

  </body>
  </html>
  ```

Pass multiple arguments

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

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

Default Argument

- If we call the any function without arguments it takes the default value as argument:
- E.g. setHeight() in following example

```
<?php
  function setHeight($minheight = 50) {
     echo "The height is : $minheight <br>";
  }

  setHeight(350);
  setHeight(); // will use the default value of 50
  setHeight(135);
  setHeight(80);
  ?>
```

Functions - Returning values

-
  ```
  <?php
  function sum($x, $y) {
      $z = $x + $y;
      return $z;
  }

  echo "7 + 13 = " . sum(7, 13) . "<br>";
  echo "2 + 4 = " . sum(2, 4);
  ?>
  ```

Form handling in PHP

PHP Superglobals

- The PHP superglobal variables are:
- $GLOBALS
- $_SERVER
- $_REQUEST
- $_POST
- $_GET
- $_FILES
- $_ENV
- $_COOKIE
- $_SESSION

PHP are "superglobals", which means that they are always accessible, regardless of scope - and you can access them from any function, class or file without having to do anything special.

PHP $_SERVER

- $_SERVER is a PHP super global variable which holds information about headers, paths, and script locations.

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

PHP $_REQUEST

- PHP $_REQUEST is used to collect data after submitting an HTML form.
-
```
<html>
<body>

<form method="post" action="<?php echo $_SERVER['PHP_SELF'];?>">
  Name: <input type="text" name="fname">
  <input type="submit">
</form>

<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // collect value of input field
    $name = $_REQUEST['fname'];
    if (empty($name)) {
        echo "Name is empty";
    } else {
        echo $name;
    }
}
?>

</body>
</html>
```

PHP $_POST

- PHP $_POST is widely used to collect form data after submitting an HTML form with method="post". $_POST is also widely used to pass variables.

-
  ```
  <html>
  <body>

  <form method="post" action="<?php echo $_SERVER['PHP_SELF'];?>">
    Name: <input type="text" name="fname">
    <input type="submit">
  </form>

  <?php
  if ($_SERVER["REQUEST_METHOD"] == "POST") {
      // collect value of input field
      $name = $_POST['fname'];
      if (empty($name)) {
          echo "Name is empty";
      } else {
          echo $name;
      }
  }
  ?>

  </body>
  </html>
  ```

PHP $_GET

- PHP $_GET can also be used to collect form data after submitting an HTML form with method="get".
- $_GET can also collect data sent in the URL.

Form Handling

-
  ```
  <html>
  <body>

  <form action="“a.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>
  <body>

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

  </body>
  </html>
  ```

-
  ```
  <?php

  $name = $email = $gender = $comment = $website = "";

  if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $name = test_input($_POST["name"]);
    $email = test_input($_POST["email"]);
    $website = test_input($_POST["website"]);
    $comment = test_input($_POST["comment"]);
    $gender = test_input($_POST["gender"]);
  }

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

```
• <h2>PHP Form Validation Example</h2>
  <form method="post" action="<?php echohtmlspecialchars($_SERV
  ER["PHP_SELF"]);?>">
    Name: <input type="text" name="name">
    <br><br>
    E-mail: <input type="text" name="email">
    <br><br>
    Website: <input type="text" name="website">
    <br><br>
    Comment: <textarea name="comment" rows="5" cols="40"></tex
  tarea>
    <br><br>
    Gender:
    <input type="radio" name="gender" value="female">Female
    <input type="radio" name="gender" value="male">Male
    <input type="radio" name="gender" value="other">Other
    <br><br>
    <input type="submit" name="submit" value="Submit">
  </form>
```

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

  </body>
  </html>
  ```

Required Field

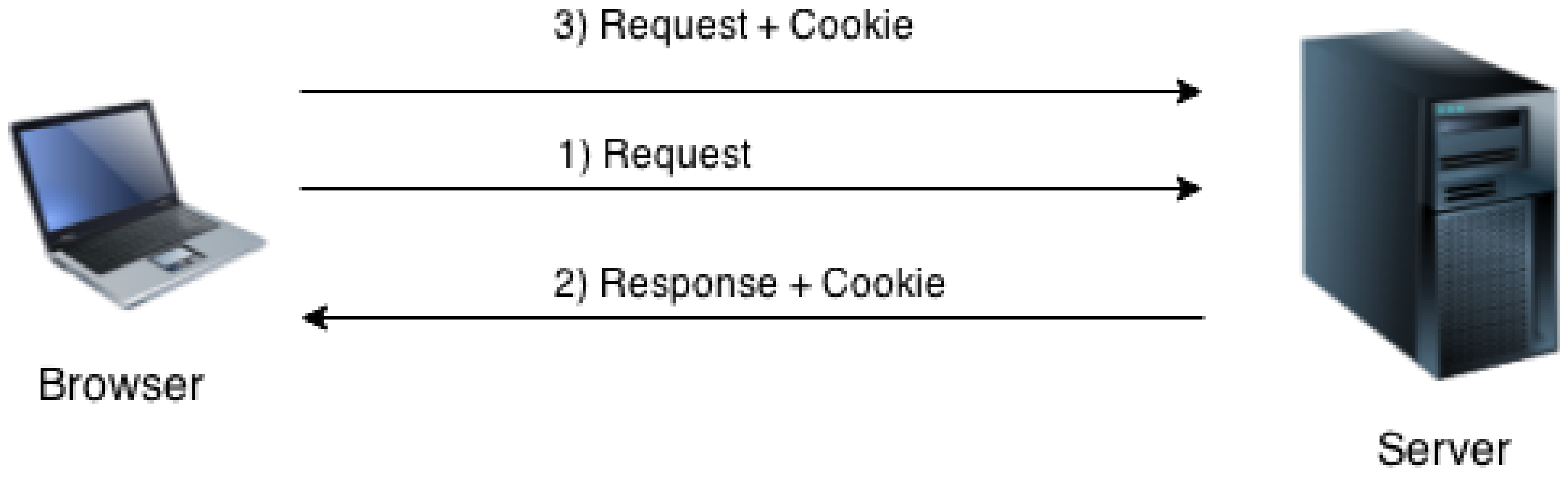
-
  ```
  <?php
  // define variables and set to empty values
  $nameErr = $emailErr = $genderErr = $websiteErr = "";
  $name = $email = $gender = $comment = $website = "";

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

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

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

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

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

Cookie in PHP

Cookie

- Cookie is a small piece of information which is stored at client browser. It is used to recognize the user.
- Cookie is created at server side and saved to client browser.
- Each time when client sends request to the server, cookie is embedded with request. Such way, cookie can be received at the server side


3) Request + Cookie
1) Request
2) Response + Cookie
Browser
Server

How to Create Cookies

- setcooke() function is used

<u>Syntax</u>

- setcookie(*name, value, expire, path, domain, secure*);

- **Name** − This sets the name of the cookie and is stored in an environment variable called HTTP_COOKIE_VARS. This variable is used while accessing cookies.

- **Value** − This sets the value of the named variable and is the content that you actually want to store.

- **Expiry** − This specify a future time in seconds since 00:00:00 GMT on 1st Jan 1970. After this time cookie will become inaccessible. If this parameter is not set then cookie will automatically expire when the Web Browser is closed.

- **Path** − This specifies the directories for which the cookie is valid. A single forward slash character permits the cookie to be valid for all directories.

- **Domain** − This can be used to specify the domain name in very large domains and must contain at least two periods to be valid. All cookies are only valid for the host and domain which created them.

- **Security** − This can be set to 1 to specify that the cookie should only be sent by secure transmission using HTTPS otherwise set to 0 which mean cookie can be sent by regular HTTP.

```
<?php
setcookie("name", "John Watkin", time()+3600,
    "/","", 0);
setcookie("age", "36", time()+3600, "/", "", 0); ?>
 <html>
 <head>
 <title>Setting Cookies with PHP</title>
</head> <body>
<?php echo "Set Cookies"?>
</body>
</html>
```

Accessing Cookies

PHP provides many ways to access cookies.

- $_COOKIE
- $HTTP_COOKIE_VARS variables.

```
<html>
 <head>
 <title>Accessing Cookies with PHP</title>
 </head>
 <body>
 <?php
echo $_COOKIE["name"]. "<br />";
echo $HTTP_COOKIE_VARS["name"]. "<br />";
 echo $_COOKIE["age"] . "<br />";
echo $HTTP_COOKIE_VARS["age"] . "<br />";
 ?>
 </body>
 </html>
```

isset() function

```
<html>
 <head>
 <title>Accessing Cookies with PHP</title>
</head>
 <body>
<?php
if( isset($_COOKIE["name"]))
 echo "Welcome " . $_COOKIE["name"] . "<br />";
else
echo "Sorry... Not recognized" . "<br />"; ?>
</body>
</html>
```

Modify a Cookie Value

- To modify a cookie, just set (again) the cookie using the setcookie() function:

```
<?php
    $cookie_name = "user";
    $cookie_value = "Alex Porter";
    setcookie($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];
    }
    ?>

    </body>
    </html>
```

Delete Cookie

```
<?php

setcookie("user", "", time() - 3600);


?>
    <html>
    <body>

    <?php
    echo "Cookie 'user' is deleted.";
    ?>

    </body>
    </html>
```

Sessions Management

- A session is a way to store information (in variables) to be used across multiple pages.

- Unlike a cookie, the information is not stored on the users computer.

What is a PHP Session?

- When you work with an application, you open it, do some changes, and then you close it. This is much like a Session.
- The computer knows who you are. It knows when you start the application and when you end.
- But on the internet there is one problem: the web server does not know who you are or what you do, because the HTTP address doesn't maintain state.

- Session variables solve this problem by storing user information to be used across multiple pages (e.g. username, favorite color, etc).
- By default, session variables last until the user closes the browser.
- Session variables hold information about one single user, and are available to all pages in one application.

Start a PHP Session

- session_start() function is used
- Session variables are set with the PHP global variable: $_SESSION

```
<?php

// Start the session
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
// Set session variables

$_SESSION["favcolor"] = "green";
$_SESSION["favanimal"] = "cat";
echo "Session variables are set.";
?>

</body>
</html>
```

Access session values

-
  ```
  <?php
  session_start();
  ?>
  <!DOCTYPE html>
  <html>
  <body>

  <?php

  echo "Favorite color is " . $_SESSION["favcolor"] . ".<br>";
  echo "Favorite animal is " . $_SESSION["favanimal"] . ".";
  ?>

  </body>
  </html>
  ```

- Another way to show all the session variable values are:

```
<?php
session_start();
    ?>
    <!DOCTYPE html>
    <html>
    <body>

    <?php
    print_r($_SESSION);
    ?>

    </body>
    </html>
```

Modify a Session Variable

Just overwrite it

```
<?php
session_start();
    ?>
    <!DOCTYPE html>
    <html>
    <body>

    <?php
    $_SESSION["favcolor"] = "yellow";
    print_r($_SESSION);
    ?>

    </body>
    </html>
```

Destroy a PHP Session

```
<?php
    session_start();
    ?>
    <!DOCTYPE html>
    <html>
    <body>

    <?php
    // remove all session variables
    session_unset();

    // destroy the session
    session_destroy();
    ?>

    </body>
    </html>
```

<u>**Sessions**</u>

A session is a way to store information (in variables) to be used across multiple pages.

Unlike a cookie, the information is not stored on the users computer.

<u>**PHP Session?**</u>

When you work with an application, you open it, do some changes, and then you close it. This is much like a Session. The computer knows who you are. It knows when you start the application and when you end. But on the internet there is one problem: the web server does not know who you are or what you do, because the HTTP address doesn't maintain state.

Session variables solve this problem by storing user information to be used across multiple pages (e.g. username, favorite color, etc). By default, session variables last until the user closes the browser.

So; Session variables hold information about one single user, and are available to all pages in one application.

<u>**Start a PHP Session**</u>

A session is started with the session_start() function.

Session variables are set with the PHP global variable: $_SESSION.

Now, let's create a new page called "demo_session1.php". In this page, we start a new PHP session and set some session variables:

```
<?php
// Start the session
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
// Set session variables
$_SESSION["favcolor"] = "green";
$_SESSION["favanimal"] = "cat";
echo "Session variables are set.";
?>

</body>
</html>
```

Get PHP Session Variable Values

Next, we create another page called "demo_session2.php". From this page, we will access the session information we set on the first page ("demo_session1.php").

Notice that session variables are not passed individually to each new page, instead they are retrieved from the session we open at the beginning of each page (session_start()).

Also notice that all session variable values are stored in the global $_SESSION variable:

```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
// Echo session variables that were set on previous page
echo "Favorite color is " . $_SESSION["favcolor"] . ".<br>";
echo "Favorite animal is " . $_SESSION["favanimal"] . ".";
?>

</body>
</html>
```

Another way to show all the session variable values for a user session is to run the following code:

```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
print_r($_SESSION);
?>

</body>
</html>
```

Modify a PHP Session Variable

To change a session variable, just overwrite it:

```
<?php
session_start();
?>
<!DOCTYPE html>
```

```
<html>
<body>

<?php
// to change a session variable, just overwrite it
$_SESSION["favcolor"] = "yellow";
print_r($_SESSION);
?>

</body>
</html>
```

Destroy a PHP Session

To remove all global session variables and destroy the session, use session_unset() and session_destroy():

```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
// remove all session variables
session_unset();

// destroy the session
session_destroy();
?>

</body>
</html>
```