Web Technologies

PHP

PHP

What is PHP?

- PHP is an acronym for "PHP: Hypertext Preprocessor"
- PHP is a widely-used, open source scripting language
- PHP scripts are executed on the web server
- PHP is free to download and use
- Many php frameworks

PHP

What is a PHP File?

- PHP files can contain text, HTML, CSS, JavaScript, and PHP code
- PHP code is executed on the server, and the result is returned to the browser as plain HTML
- PHP files have extension ".php"

PHP

Why PHP?

- 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

What PHP can 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

With PHP you are not limited to output HTML. You can output images or PDF files. You can also output any text, such as XHTML and XML.

Basic PHP Syntax

A PHP script can be placed anywhere in the document.

A PHP script starts with <?php and ends with ?>:

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

The default file extension for PHP files is ".php".

PHP echo/print

With PHP, there are two basic ways to get output: **echo()** and **print()**.

echo and print are more or less the same. They are both used to output data to the screen.

The differences are small: 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

PHP Comments

```php
<?php
// This is a single-line comment

# This is also a single-line comment
?>
```

```php
<?php
/*
This is a multiple-lines comment block
that spans over multiple
lines
*/
?>
```

Creating (Declaring) PHP Variables

In PHP, a variable starts with the $ sign, followed by the name of the variable:

Example

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

PHP Variables Naming Rules

- 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, $Age and $AGE are three different variables)

PHP

PHP Variables Scope

- local
- global
- static

Global:

- The global keyword is used to access a global variable from within a function.
- PHP also stores all global variables in an array called $GLOBALS[index]. The index holds the name of the variable.

Static:

- 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.
- To do this, use the static keyword when you first declare the variable.

PHP Data Types

Variables can store data of different types, and different data types can do different things.

PHP supports the following data types:

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

PHP Arrays

An array is a special variable, which can hold more than one value at a time.

In PHP, the array() function is used to create an array:

array();

In PHP, there are three types of arrays:

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

PHP Arrays

An array stores multiple values in one single variable:

Example

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

PHP Arrays

Working With Arrays

In this tutorial you will learn how to work with arrays, including:

- Create Arrays
- Access Arrays
- Update Arrays
- Add Array Items
- Remove Array Items
- Sort Arrays

PHP Arrays

Array Items

Array items can be of any data type.
The most common are strings and numbers (int, float), but array items can also be objects, functions or even arrays.
You can have different data types in the same array.

Example

Array items of four different data types:

```
$myArr = array("Volvo", 15, ["apples", "bananas"],
myFunction);
```

PHP Arrays

Array Functions

The real strength of PHP arrays are the built-in array functions, like the `count()` function for counting array items:

Example

How many items are in the `$cars` array:

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

echo count($cars);
```

PHP Arrays

PHP Indexed Arrays

In indexed arrays each item has an index number.
By default, the first item has index 0, the second item has item 1, etc.

Example

Create and display an indexed array:

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

var_dump($cars);
```

PHP Arrays

Access Indexed Arrays

To access an array item you can refer to the index number.

Example

Display the first array item:

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

echo $cars[0];
```

PHP Arrays

PHP Associative Arrays

Associative arrays are arrays that use named keys that you assign to them.

Example

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

var_dump($car);
```

PHP Arrays

Access Associative Arrays

To access an array item you can refer to the key name.

Example

Display the model of the car:

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

echo $car["model"];
```

PHP Arrays

PHP - Multidimensional Arrays

A multidimensional array is an array containing one or more arrays.
PHP supports multidimensional arrays that are two, three, four, five, or more levels deep. However, arrays more than three levels deep are hard to manage for most people.

The dimension of an array indicates the number of indices you need to select an element.

- For a two-dimensional array you need two indices to select an element
- For a three-dimensional array you need three indices to select an element

PHP Arrays

First, take a look at the following table:

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

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

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

PHP Operators

- Arithmetic operators
- Assignment operators
- Comparison operators
- Increment/Decrement operators
- Logical operators
- String operators
- Array operators
- Conditional assignment operators

PHP 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

PHP 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

PHP

PHP 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

PHP 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

PHP String Operators

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

PHP Array Operators

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 Conditional Assignment Operators

Operator	Name	Example	Result
?:	Ternary	$x = *expr1* ? *expr2* : *expr 3*	Returns the value of $x. The value of $x is *expr2* if *expr1* = TRUE. The value of $x is *expr3* if *expr1* = FALSE
??	Null coalescing	$x = *expr1* ?? *expr2*	Returns the value of $x. The value of $x is *expr1* if *expr1* exists, and is not NULL. If *expr1* does not exist, or is NULL, the value of $x is *expr2*. Introduced in PHP 7

PHP Conditional Statements

`if` statement - executes some code if one condition is true

`if...else` statement - executes some code if a condition is true and another code if that condition is false

`if...elseif...else` statement - executes different codes for more than two conditions

`switch` statement - selects one of many blocks of code to be executed

PHP 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

PHP Loops

The PHP foreach Loop on Arrays. The most common use of the foreach loop, is to loop through the items of an array.

Example Loop through the items of an indexed array:

```
$colors = array("red", "green", "blue", "yellow");

foreach ($colors as $x) {

  echo "$x <br>";

}
```

PHP Function

PHP Built-in Functions

PHP has over 1000 built-in functions that can be called directly, from within a script, to perform a specific task.

PHP User Defined Functions

Besides the built-in PHP functions, it is possible to create your own functions.

- A function is a block of statements that can be used repeatedly in a program.
- A function will not execute automatically when a page loads.
- A function will be executed by a call to the function.

PHP Function

PHP Functions Argument Types

- Positional Argument
- Default Argument
- Named Argument

PHP Variable Scope

Global and Local Scope

A variable declared **outside** a function has a GLOBAL SCOPE and can only be accessed outside a function:

```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>";
}
myTest();

echo "<p>Variable x outside function is: $x</p>";
```

PHP Variable Scope

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

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

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

PHP Variable Scope

PHP The global Keyword

The `global` keyword is used to access a global variable from within a function.
To do this, use the `global` keyword before the variables (inside the function):

```
$x = 5;
$y = 10;

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

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

PHP

PHP Variable Scope

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
```

PHP Variable Scope

PHP 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.
To do this, use the `static` keyword when you first declare the variable:

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

myTest();
myTest();
myTest();
```

OOP in PHP

What Is OOP in PHP?

Object-Oriented Programming (OOP) is a programming style where you model your program as a collection of **objects**, each representing something real, with **properties** (data) and **methods** (behavior).

PHP supports full OOP features, like **classes**, **inheritance**, **interfaces**, and **polymorphism**.

OOP in PHP

Class and Object

A **class** is a blueprint; an **object** is a specific instance of that class.

```
<?php
class Car {
    public $brand;
    public $color;

    public function startEngine() {
        echo "The engine is starting...\n";
    }
}

// Create an object
$myCar = new Car();
$myCar->brand = "Tesla";
$myCar->color = "Red";

echo "Brand: " . $myCar->brand . "\n";
$myCar->startEngine();
?>
```

PHP

OOP in PHP

Constructor and Destructor

Constructors initialize an object automatically.

Destructors run when an object is destroyed or the script ends.

```
<?php
class Car {
   public $brand;

   public function __construct($brand) {
      $this->brand = $brand;
      echo "Car created: $this->brand\n";
   }

   public function __destruct() {
      echo "Car object for {$this->brand} destroyed.\n";
   }
}

$car1 = new Car("BMW");
?>
```

PHP

OOP in PHP

Constructor and Destructor

Constructors initialize an object automatically.

Destructors run when an object is destroyed or the script ends.

```php
<?php
class myclass{
	public function __construct(){
		echo "Constructor is invoked!<br/>";
	}

	public function display(){
		echo "Display function invoked.<br/>";
	}

	public function __destruct(){
		echo "Destructor is invoked!<br>";
	}
}

$obj = new myclass();
$obj->display();
```

OOP in PHP

Constructor example

```php
class myclass{

        private $x;

        public function __construct($y){
                $this->x = $y;
        }

        public function display(){
                echo "X = $this->x<br/>";
        }

}

$obj = new myclass(67);
$obj->display();
```

OOP in PHP

Inheritance

A **child class** can inherit properties and methods from a **parent class**.

```
<?php
class Vehicle {
    public function move() {
        echo "Vehicle is moving\n";
    }
}

class Car extends Vehicle {
    public function honk() {
        echo "Beep beep!\n";
    }
}

$car = new Car();
$car->move(); // Inherited method
$car->honk(); // Child method
?>
```

PHP

OOP in PHP

Inheritance example

demo.php

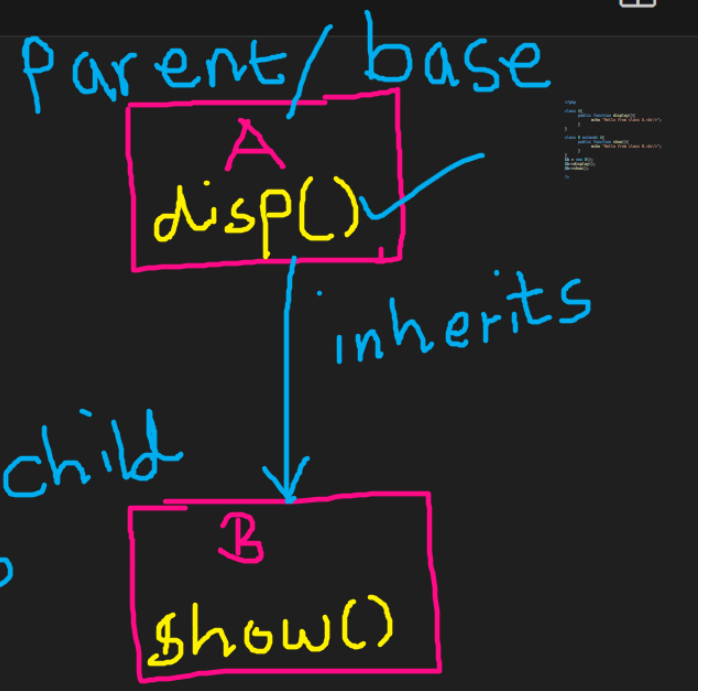
D: > xampp > htdocs > mca1b > demo.php

```php
<?php

class A{
	public function display(){
		echo "Hello from class A.<br/>";
	}
}

class B extends A{
	public function show(){
		echo "Hello from class B.<br/>";
	}
}
$b = new B();
$b->display();
$b->show();

```



PHP

OOP in PHP

Access Modifiers: `public`, `protected`, `private`

They define visibility:

- **public**: accessible from anywhere
- **protected**: accessible in the class and subclasses
- **private**: accessible only in the class itself

```php
<?php
class BankAccount {
    private $balance = 0;

    public function deposit($amount) {
        $this->balance += $amount;
    }

    public function getBalance() {
        return $this->balance;
    }
}

$acc = new BankAccount();
$acc->deposit(100);
echo $acc->getBalance(); // 100
?>
```

PHP

OOP in PHP

Private member example with constructor

```php
<?php

class myclass{
        private $x;

        public function __construct($y){
                $this->x = $y;
        }

        public function display(){
                echo "x = $this->x <br/>";
        }
}
$obj = new myclass(10);
$obj->display();

?>
```

OOP in PHP

Polymorphism (Method Overriding)

Different classes can define methods with the same name but different behavior.

```
<?php
class Animal {
    public function makeSound() {
        echo "Some generic sound\n";
    }
}

class Dog extends Animal {
    public function makeSound() {
        echo "Bark!\n";
    }
}

class Cat extends Animal {
    public function makeSound() {
        echo "Meow!\n";
    }
}

$animals = [new Dog(), new Cat(), new Animal()];
foreach ($animals as $a) {
    $a->makeSound();
}
?>
```

PHP Form Handling

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

PHP

PHP Form Handling

$_GET example.

```html
<html>
    <body>
        <form method="GET" action="get_data.php">
            <label>User Name:</label>
            <input type="text" name="txtname">
            <br/>
            <label>User City:</label>
            <input type="text" name="txtcity">
            <br/>
            <input type="submit" value="Submit">
        </form>
    </body>
</html>
```

```php
<?php

$name = $_GET["txtname"];
$city = $_GET["txtcity"];

echo "Name = $name<br/>";
echo "City = $city<br/>";

?>
```

PHP

PHP Form Handling

$_POST example.

```html
<html>
    <body>
        <form method="POST" action="get_data.php">
            <label>User Name:</label>
            <input type="text" name="txtname">
            <br/>
            <label>User City:</label>
            <input type="text" name="txtcity">
            <br/>
            <input type="submit" value="Submit">
        </form>
    </body>
</html>
```

```php
<?php

$name = $_POST["txtname"];
$city = $_POST["txtcity"];

echo "Name = $name<br/>";
echo "City = $city<br/>";



?>
```

PHP & MySql

With PHP, you can connect to and manipulate databases.

MySQL is the most popular database system used with PHP.

PHP 5 and later can work with a MySQL database using:

- MySQLi extension (the "i" stands for improved)
- PDO (PHP Data Objects)

Earlier versions of PHP used the MySQL extension. However, this extension was deprecated in 2012.

PHP & MySql

MySQLi or PDO?

Both MySQLi and PDO have their advantages.

PDO will work on 12 different database systems, whereas MySQLi will only work with MySQL databases.

So, if you have to switch your project to use another database, PDO makes the process easy. You only have to change the connection string and a few queries. With MySQLi, you will need to rewrite the entire code - queries included.

PHP & MySql

- MySQLi (procedural)
- MySQLi (object-oriented)
- PDO

PHP & MySql

Example (MySQLi Procedural)

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";

// Create connection
$conn = mysqli_connect($servername, $username, $password);

// Check connection
if (!$conn) {
  die("Connection failed: " . mysqli_connect_error());
}
echo "Connected successfully";
?>
```

PHP

PHP & MySql

Example (MySQLi Procedural)

Student Table

	#	Name	Type
☐	1	regno	int(10)
☐	2	name	varchar(200)

```html
<html>
    <body>
        <form method="POST" action="get_data.php">
            <label>Reg. No.:</label>
            <input type="text" name="txtreg">
            <br/>
            <label>Name:</label>
            <input type="text" name="txtname">
            <br/>
            <input type="submit" value="Submit">
        </form>
    </body>
</html>
```

```php
<?php

$regno = $_POST["txtreg"];
$name = $_POST["txtname"];

$conn = mysqli_connect("localhost", "root", "", "mcaa");

$sql = "INSERT INTO student VALUES($regno, '$name')";

if(mysqli_query($conn, $sql)){
    echo "Record Inserted";
}

mysqli_close($conn);

?>
```

PHP & MySql

Example (MySQLi Procedural) fetch data example

```php
<?php

$conn = mysqli_connect("localhost", "root", "", "mcab");

$sql = "SELECT * FROM student";

$result = mysqli_query($conn, $sql);

while($row = mysqli_fetch_array($result)){
    echo $row[0] . " " . $row[1] . "<br/>";
}

mysqli_close($conn);

?>
```

OUTPUT (All data from Student table)

1001 abc
1002 xyz
1004 sdsfdfdsf

PHP & MySql

Example (MySQLi Procedural) fetch data example 2 (HTML TABLE FORM)

```php
<?php

$conn = mysqli_connect("localhost", "root", "", "mcab");

$sql = "SELECT * FROM student";

$result = mysqli_query($conn, $sql);

echo "<table border=1>";
echo "<tr><th>Reg. No.</th><th>Student Name</th>";

while($row = mysqli_fetch_array($result)){
    echo "<tr>";
    echo "<td>" . $row[0] . "</td><td>" . $row[1] . "</td>";
    echo "</tr>";
}
echo "</table>";
mysqli_close($conn);

?>
```

OUTPUT (All data from Student table in HTML Table form)

Reg. No.	Student Name
1001	abc
1002	xyz
1004	sdsfdfdsf

PHP & MySql

Example (MySQLi Object-Oriented)

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";

// 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 & MySql

Example (MySQLi Object-Oriented) insert example

```php
<?php

$regno = $_POST["txtreg"];
$name = $_POST["txtname"];

$conn = new mysqli("localhost", "root", "", "mcaa");

$sql = "INSERT INTO student VALUES($regno, '$name')";

if($conn->query($sql)){
    echo "Record Inserted";
}

$conn->close();

?>
```

PHP & MySql

Example (PDO)

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";

try {
  $conn = new PDO("mysql:host=$servername;dbname=myDB", $username, $password);
  // set the PDO error mode to exception
  $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  echo "Connected successfully";
} catch(PDOException $e) {
  echo "Connection failed: " . $e->getMessage();
}
?>
```

PHP & MySql

Example (PDO) insert example

```php
<?php

$regno = $_POST["txtreg"];
$name = $_POST["txtname"];

$conn = new PDO("mysql:host=localhost;dbname=mcaa", "root", "");

$sql = "INSERT INTO student VALUES($regno, '$name')";

if($conn->exec($sql)){
    echo "Record Inserted";
}
$conn = NULL;

?>
```

Cookies

What is a Cookie?

A cookie is often used to identify a user. A cookie is a small file that the server embeds on the user's computer. Each time the same computer requests a page with a browser, it will send the cookie too. With PHP, you can both create and retrieve cookie values.

Create Cookies With PHP

A cookie is created with the `setcookie()` function.

Syntax

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

Cookies

PHP Create/Retrieve a Cookie

The example creates a cookie named "user" with the value "John Doe". The cookie will expire after 30 days (86400 * 30). The "/" means that the cookie is available in entire website (otherwise, select the directory you prefer).

We then retrieve the value of the cookie "user" (using the global variable $_COOKIE). We also use the `isset()` function to find out if the cookie is set:

PHP

Cookies

```php
<?php
$cookie_name = "user";
$cookie_value = "John Doe";
setcookie($cookie_name, $cookie_value, time() + (86400 * 30), "/"); // 86400 = 1
day
?>
<html>
<body>
<?php
if(!isset($_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>
```

Cookies

Cookie example: Sharing data across multiple pages

set_cookie.php

```php
<?php

    setcookie("uname", "donald duck");

    if(isset($_COOKIE["uname"])) {
    echo $_COOKIE["uname"];
    }

?>
```

get_cookie.php

```php
<?php

    echo "The value of cookie is: ";
    echo $_COOKIE["uname"];


?>
```

Session

What is a PHP Session?

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.

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.

Session

Start a PHP Session

A session is started with the `session_start()` function.
Session variables are set with the PHP global variable: $_SESSION.

PHP

Session

```php
<?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>
```

```php
<?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>
```

PHP

Session

Session Example

set_session.php

```php
<?php

$_SESSION["user_name"] = "Donald Duck";


?>
```

get_session.php

```php
<?php

    echo "Accessing session data.<br/>";
    echo $_SESSION["user_name"];

?>
```

THANK YOU