

PHP

What is PHP?

- acronym for "PHP: Hypertext Preprocessor"
- widely-used, efficient, open source scripting language
- free to download and use
- Server-side scripting language
- executed on the server

scripting language

- **Client-side scripting**
- **Server-side scripting**

- A script is generally a series of program or instruction, which has to be executed on other program or application.
- A **scripting language** is a **programming language** designed for integrating and communicating with other **programming languages**.
- Scripting languages, which can be embedded within HTML, commonly are used to add functionality to a Web page, such as different menu styles or graphic displays or to serve dynamic needs.
- A **script** or **scripting language** is a computer **language** with a series of commands within a file that is capable of being executed without being compiled.
- Some of the most widely used **scripting languages** are JavaScript, VBScript, PHP, Perl, Python, Ruby.
- As we know that the web works in a client-server environment. The client-side script executes the code to the client side which is visible to the users while a server-side script is executed in the server end which users cannot see.

- **Client-side scripting** is source code that is executed on the **client's** browser instead of the web-server, and allows for the creation of faster and more responsive web applications.
- Basically, these types of scripts are placed inside an HTML document.
- The client-side scripting can be used to examine the user's form for the errors before submitting it and for changing the content according to the user input.
- So it is just basically affecting the data that the end user sees in a browser window.
- For example, when a user makes a request via browser for a webpage to the server, it just sent the HTML and CSS as plain text, and the browser interprets and renders the web content in the client end.
- The **client-side scripting language** involves **languages** such as HTML, CSS and JavaScript.
- The best example of a client side scripting language is JavaScript.

- A **language** used to develop programs that are executed by the **server**.
- Server-side scripting is a technique used in web development which involves employing scripts on a web server which produce a response customized for each user's request to the website.
- in simple words any scripting or programming that can run on the web server is known as server-side scripting.
- The operations like customization of a website, dynamic change in the website content, response generation to the user's queries, accessing the database, and so on are performed at the server end.
- The server-side scripting constructs a communication link between a server and a client (user).
- Good examples of server-side scripting languages include PHP, ASP.NET, Node. js, Java, Ruby, Perl and Python

So, in general we can say that -

- The scripts can be written in two forms, at the server end (back end) or at the client end (server end).
- The main difference between server-side scripting and client-side scripting is that the server side scripting involves server for its processing.
- On the other hand, client-side scripting requires browsers to run the scripts on the client machine but does not interact with the server while processing the client-side scripts.
- **server-side code** is run on a web **server** and that its **main role** is to control what information is sent to the user (while **client-side code** mainly handles the structure and presentation of that data to the user)

| BASIS FOR COMPARISON | SERVER-SIDE SCRIPTING | CLIENT-SIDE SCRIPTING |
|----------------------|---|--|
| Basic | Works in the back end which could not be visible at the client end. | Works at the front end and script are visible among the users. |
| Processing | Requires server interaction. | Does not need interaction with the server. |
| Languages involved | PHP, ASP.net, Ruby on Rails, ColdFusion, Python, etcetera. | HTML, CSS, JavaScript, etc. |
| Affect | Could effectively customize the web pages and provide dynamic websites. | Can reduce the load to the server. |
| Security | Relatively secure. | Insecure |

Conclusion

- Client-side scripting and server-side scripting works in a coordinated manner with each other.
- However, both the scripting techniques are very different, where the client-side scripting emphasize on making the interface of the web application or website more appealing and functional.
- Conversely, server-side scripting emphasizes on the data accessing methods, error handling and fast processing etcetera.

What is PHP?

- a powerful tool for making dynamic and interactive Web pages
- PHP code is executed on the server, and the result is returned to the browser as plain HTML
- PHP files have extension ".php"

CONT...

- code is embedded into HTML
- The php code is enclosed in special start and end tags that allow you to jump into and out of “php mode”

```
<html>
  <head>
    <title> Example</title>
  </head>

  <body>
    <?php
      echo “this is a php script”;
    ?>
  </body>
</html>
```

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.
- PHP is easy to learn, use and runs efficiently on the server side

- LAMP – for linux
- WAMP – for windows
- MAMP – for MAC

- XAMPP – cross platform

A:: apache

M:: Mysql

P:: PHP

P:: Perl

Use a Web Host With PHP Support

- If your server has activated support for PHP you do not need to do anything.
- Just create some .php files, place them in your web directory, and the server will automatically parse them for you.
- You do not need to compile anything or install any extra tools.
- Because PHP is free, most web hosts offer PHP support.

PHP Syntax

- A PHP script is executed on the server, and the plain HTML result is sent back to the browser.
- A PHP script starts with `<?php` and ends with `?>`:

```
<?php
// PHP code goes here
?>
```
- PHP statements end with a semicolon (;)
- A PHP script can be placed anywhere in the document.
- A PHP file normally contains HTML tags, and some PHP scripting code
- PHP is Case-insensitive
 - keywords (e.g. if, else, while, echo, etc.), classes, functions, and user-defined functions. (see - `case_insensitive.php`)
 - However, all variable names are case-sensitive! (see - `case_sensitive_var.php`)

Comments

- Syntax for single-line comments:

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

```
<?php
```

```
// This is a single-line comment
```

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

```
?>
```

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


Cont ...

- Syntax for multiple-line comments:

```
<html>
```

```
<body>
```

```
<?php
```

```
/*
```

```
This is a multiple-lines comment block  
that spans over multiple
```

```
lines
```

```
*/
```

```
?>
```

```
</body>
```

```
</html>
```

PHP Variables

- Variables are "containers" for storing information
- In PHP, a variable starts with the \$ sign, followed by the name of the variable

```
<?php
```

```
$txt = "Hello world!";
```

```
$x = 5;
```

```
$y = 10.5;
```

```
?>
```

- When you assign a text value to a variable, put quotes around the value.

- Sometimes it is convenient to be able to have variable variable names.
- That is, a variable name which can be set and used dynamically.
- A normal variable is set with a statement such as:
 - `$a = 'hello';`
- A variable variable takes the value of a variable and treats that as the name of a variable.
- In the above example, *hello*, can be used as the name of a variable by using two dollar signs. i.e.
 - `$$a = 'world';`
- At this point two variables have been defined and stored in the PHP symbol tree: *\$a* with contents "hello" and *\$hello* with contents "world".
- Therefore, this statement:
 - `echo "$a ${$a}";`
- produces the exact same output as:
 - `echo "$a $hello";`
 - i.e. they both produce: hello world.

PHP is a Loosely Typed Language

- we did not have to tell PHP which data type the variable is.
- PHP automatically associates a data type to the variable, depending on its value.
- In c: `int a;` or `int a = 1;`
 `a = 1.5;` \rightarrow not possible
- In php: `$txt = 1;`
 `$txt = "I AM NOW A STRING";`

PHP Variables Scope

- variables can be declared anywhere in the script.
- The scope of a variable is the part of the script where the variable can be referenced/used.
- PHP has three different variable scopes:
 - local
 - global
 - static

Global and Local Scope

Global Scope

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

Local Scope

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

You can have local variables with the same name in different functions, because local variables are only recognized by the function in which they are declared.

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)
- OR
- **`$GLOBALS[index] → $GLOBALS[var_name]`**

\$GLOBALS

- PHP also stores all global variables in an array called `$GLOBALS[index]`.
- The index holds the name of the variable.
- This array is also accessible from within functions and can be used to update global variables directly.
- So, The `$GLOBALS` array is an associative array - which contains references to all variables which are currently defined in the global scope of the script.
 - and the variable names are the keys of the array.

static Keyword

- use the static keyword when you first declare the variable
- each time the function is called
 - that variable will still have the information it contained from the last time the function was called.

PHP Data Types

- `var_dump()` function - returns the data type and value

PHP String Functions

- `strlen()` - Return the Length of a String
- `str_word_count()` - Count Words in a String
- `strrev()` - Reverse a String
- `strpos()` - Search For a Specific Text Within a String
 - If a match is found, the function returns the character position of the first match.
 - If no match is found, it will return `FALSE`. (prints nothing)
- `str_replace()` - Replace Text Within a String

PHP Constants

- like variables – except - once defined - cannot be changed or undefined
- constants are automatically global across the entire script
- no \$ sign before the constant name
- create a constant - define() function
 - `define(name, value, case-insensitive)`
- case-insensitive : Default is false
 - case-sensitive : true

PHP Operators

- perform operations on variables and values
 - Arithmetic operators
 - Assignment operators
 - Comparison operators
 - Increment/Decrement operators
 - Logical operators
 - String operators
 - Array operators
 - Conditional assignment operators

PHP Arithmetic Operators

- used with numeric values to perform common arithmetical operations

| 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

- used with numeric values to write a value to a variable

| 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 |

Cont ...

```
<?php
```

```
$x = 20;
```

```
$x += 100;
```

```
echo $x;
```

```
?>
```


PHP Comparison Operators

- used to compare two values (number or string)

| 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 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 |

Contd ...

```
<?php
```

```
$x = 100;
```

```
$y = "100";
```

```
var_dump($x == $y);    // returns true because values are equal
```

```
Echo "<BR>";
```

```
echo ($x == $y);
```

```
?>
```

```
bool(true)
```

```
1
```

Contd ...

```
<?php
```

```
$x = 100;
```

```
$y = "100";
```

```
var_dump($x === $y); // returns false because types are not equal
```

```
?>
```

```
bool(false)
```

Cont...

```
<?php
```

```
$x = 100;
```

```
$y = "100";
```

```
var_dump($x != $y); // returns false because values are equal
```

```
?>
```

bool(false)

Cont...

```
<?php
```

```
$x = 100;
```

```
$y = "100";
```

```
var_dump($x !== $y); // returns true because types are not equal
```

```
?>
```

bool(true)

PHP Increment / Decrement Operators

- used to increment/decrement a variable's value.

| 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 |

Cont ...

```
<?php
$x = 10;
echo ++$x, "\t";
echo $x++, "\t";
echo --$x, "\t";
echo $x--, "\t";
echo $x, "\t";
?>
```

11 11 11 11 10

PHP Logical Operators

- used to combine conditional statements

| 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 |

Cont...

```
<?php
```

```
$x = "100;
```

```
$y = 50;
```

```
if ($x == 100 and $y == 50) {
```

```
    echo "Hello world!";
```

```
}
```

```
?>
```

PHP String Operators

- PHP has two operators that are specially designed for strings

| Operator | Name | Example | Result |
|-----------------|-----------------------------|-------------------------------|--|
| . | Concatenation | <code>\$txt1 . \$txt2</code> | Concatenation of <code>\$txt1</code> and <code>\$txt2</code> |
| <code>.=</code> | Concatenation assignment | <code>\$txt1 .= \$txt2</code> | Appends <code>\$txt2</code> to <code>\$txt1</code> |

Cont...

```
<?php
$txt1 = "Hello";
$txt2 = " world!";
echo $txt1 . $txt2;
?>
```

Hello world!

```
<?php
$txt1 = "Hello";
$txt2 = " world!";
$txt1 .= $txt2;
echo $txt1;
?>
```

Hello world!

PHP Conditional Assignment Operators

- used to set a value depending on conditions

| Operator | Name | Example | Result |
|-----------|----------------|---|---|
| ?: | Ternary | $\\$x = \textit{expr1} ? \textit{expr2} : \textit{expr3}$ | Returns the value of $\$x$. The value of $\$x$ is <i>expr2</i> if <i>expr1</i> = TRUE The value of $\$x$ is <i>expr3</i> if <i>expr1</i> = FALSE |

PHP Conditional Statements

- used to perform different actions based on different conditions
- following conditional statements:

| Statement | Execution |
|---------------------------|--|
| if | executes some code if one condition is true |
| if...else | executes some code if a condition is true and another code if that condition is false |
| if...elseif...else | executes different codes for more than two conditions |
| switch | selects one of many blocks of code to be executed |

PHP - The if Statement

- Syntax:

```
if (condition) {  
    code to be executed if condition is true;  
}
```

- Example-

```
<?php  
    $t = 10;  
    if ($t < 20) {  
        echo "Have a good day!";  
    }  
?>
```

PHP - The if...else Statement

- Syntax:

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

- Example-

```
<?php  
    $t = 100;  
    if ($t < 20) {  
        echo "Have a good day!";  
    } else {  
        echo "Bye";  
    }  
?>
```

PHP - The if...elseif...else Statement

- Syntax:

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


Cont ...

- Example –

```
<?php
    $d = date("D");
    echo $d; //Wed

    if ($d == "Wed")
        echo "Have a nice wednesday!";

    elseif ($d == "Sun")
        echo "Have a nice Sunday!";

    else
        echo "Have a nice day!";

?>
```

```
<?php
    $d = date("D");
    echo $d;

    if ($d == "Wed")
    {
        echo "Today is wednesday";
        echo "Have a nice wednesday!";
    }

    elseif ($d == "Sun") {
        echo "Have a nice Sunday!";
    } else
        echo "Have a nice day!";

?>
```

PHP Switch Statement

- used to perform different actions based on different conditions
- Use the switch statement to select one of many blocks of code to be executed.

Syntax:

```
switch (n) {  
    case label1:  
        code to be executed if n=label1;  
        break;  
    case label2:  
        code to be executed if n=label2;  
        break;  
    case label3:  
        code to be executed if n=label3;  
        break;  
    ...  
    default:  
        code to be executed if n is different from all labels;  
}
```

Cont ...

- Example-

```
<?php
    $favcolor = "red";
    switch ($favcolor) {
        case "red":
            echo "Your favorite color is red!";
            break;
        case "blue":
            echo "Your favorite color is blue!";
            break;
        case "green":
            echo "Your favorite color is green!";
            break;
        default:
            echo "Your favorite color is neither red, blue, nor green!";
    }
    echo $favcolor;
?>
```

PHP Loops

- used to execute the same block of code again and again
- as long as a certain condition is true.

| Loop | Execution |
|-------------------|--|
| 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 while Loop

- Syntax:

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

- Example-

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

The number is: 1
The number is: 2
The number is: 3
The number is: 4
The number is: 5

PHP do while Loop

- Syntax:

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

- Example-

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

```
The number is: 1  
The number is: 2  
The number is: 3  
The number is: 4  
The number is: 5
```

PHP for Loop

- Syntax:

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

CONT...

- The for loop is used when you know in advance how many times the script should run.
- Parameters:
 - init counter: Initialize the loop counter value
 - test counter: Evaluated for each loop iteration. If it evaluates to TRUE, the loop continues. If it evaluates to FALSE, the loop ends.
 - increment counter: Increases the loop counter value

Cont ...

- displays the numbers from 0 to 5

```
<?php
for ($x = 0; $x <= 5; $x++)
{
    echo "The number is: $x <br>";
}
?>
```

The number is: 0
The number is: 1
The number is: 2
The number is: 3
The number is: 4
The number is: 5

Cont ...

- counts to 100 by 20s

```
<?php
for ($x = 0; $x <= 100; $x+=20)
{
    echo "The number is: $x <br>";
}
?>
```

The number is: 0
The number is: 20
The number is: 40
The number is: 60
The number is: 80
The number is: 100

PHP foreach Loop

- works only on arrays
- used to loop through each key/value pair in an array
- Syntax:

```
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.

Cont ...

- Example - print the values of the given array (\$colors)

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

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

red
green
blue
yellow

Cont ...

- Example - print both the keys and the values of the given array (\$age)

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

    foreach ($age as $x => $val)
    {
        echo "$x = $val<br>";
    }
?>
```

Peter = 35
Ben = 37
Joe = 43

PHP Functions

- Syntax:

(A user-defined function declaration)

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

- 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.

Cont ...

- Example-

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

```
writeMsg();    // call the function
?>
```

Hello world!

Cont...

- Example – Function with Arguments

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

```
familyName("aish");
familyName("babita");
familyName("dina");
familyName("Krish");
?>
```

aish sahni.
babita sahni.
dina sahni.
krish sahni.

Cont...

- Example – Function with Two Arguments

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

```
familyName("aish","1975");
familyName("babita","1978");
familyName("krish","1983");
?>
```

aish sahni. Born in 1975
babita sahni. Born in 1978
krish sahni. Born in 1983

Cont...

- Example – Function with Default Argument Value

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

```
setHeight(350);
setHeight();
setHeight(135);
setHeight(80);
?>
```

The height is : 350

The height is : 50

The height is : 135

The height is : 80

Cont...

- Example – Function with Return value

```
<?php
```

```
function sum (int $x, int $y)
```

```
{
```

```
    $z = $x + $y;
```

```
    return $z;
```

```
}
```

```
echo "5 + 10 = " . sum(5,10) . "<br>";
```

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

```
echo "2 + 4 = " . sum(2,4);
```

```
?>
```

5 + 10 = 15

7 + 13 = 20

2 + 4 = 6

PHP Arrays

- stores multiple values in one single variable
- access the values by referring to an index number
- `array();` \rightarrow to create array

```
<?php
```

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

```
    echo "I like " . $cars[0] . ", " . $cars[1] . " and " . $cars[2] . " .";
```

```
?>
```

I like Volvo, BMW and Toyota.

Cont ...

- **count() function** - used to return the length (the number of elements) of an array

```
<?php  
$cars = array("Volvo", "BMW", "Toyota");  
echo count($cars);           //3  
?>
```

Cont ...

- 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 Indexed Arrays

Two ways to create :

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

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

```
<?php  
$cars = array("Volvo", "BMW", "Toyota");  
echo "I like " . $cars[0] . ", " . $cars[1] . " and " . $cars[2] . ".";  
?>
```

Loop Through an Indexed Array

- print all the values of an indexed array

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

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

Volvo
BMW
Toyota

PHP Associative Arrays

- arrays that use named keys that you assign to them
- Two ways to create:

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

OR

```
$age['Peter'] = "35";  
$age['Ben'] = "37";  
$age['Joe'] = "43";
```

Cont ...

```
<?php
    $age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
    echo "Peter is " . $age['Peter'] . " years old.";
?>
```

Peter is 35 years old.

Loop Through an Associative Array

```
<?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 Multidimensional Arrays

- array containing one or more arrays
- PHP - Two-dimensional Arrays
 - A two-dimensional array is an array of arrays

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

Cont ...

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

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

the two-dimensional \$cars array contains four arrays, and it has two indices: row and column.

Cont ...

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

```
echo $cars[0][0].": In stock: ".$cars[0][1].", sold: ".$cars[0][2]."<br>";
echo $cars[1][0].": In stock: ".$cars[1][1].", sold: ".$cars[1][2]."<br>";
echo $cars[2][0].": In stock: ".$cars[2][1].", sold: ".$cars[2][2]."<br>";
echo $cars[3][0].": In stock: ".$cars[3][1].", sold: ".$cars[3][2]."<br>";
```

```
?>
```

| Name | Stock | Sold |
|-----------------------------|---------------------|---------------------|
| [0][0] Volvo | [0][1] 22 | [0][2] 18 |
| [1][0] BMW | [1][1] 15 | [1][2] 13 |
| [2][0] Saab | [2][1] 5 | [2][2] 2 |
| [3][0] Land Rover | [3][1] 17 | [3][2] 15 |

Cont ...

```
<?php
```

```
    $cars = array
```

```
    (
```

```
        array("Volvo",22,18),
```

```
        array("BMW",15,13),
```

```
        array("Saab",5,2),
```

```
        array("Land Rover",17,15)
```

```
    );
```

```
    echo $cars[0][0].": In stock: ".$cars[0][1].", sold: ".$cars[0][2]."<br>";
```

```
    echo $cars[1][0].": In stock: ".$cars[1][1].", sold: ".$cars[1][2]."<br>";
```

```
    echo $cars[2][0].": In stock: ".$cars[2][1].", sold: ".$cars[2][2]."<br>";
```

```
    echo $cars[3][0].": In stock: ".$cars[3][1].", sold: ".$cars[3][2]."<br>";
```

```
?>
```

Volvo: In stock: 22, sold: 18.

BMW: In stock: 15, sold: 13.

Saab: In stock: 5, sold: 2.

Land Rover: In stock: 17, sold: 15.

```

<?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>";
    }
?>

```

| Name | Stock | Sold |
|-----------------------------|---------------------|---------------------|
| [0][0] Volvo | [0][1] 22 | [0][2] 18 |
| [1][0] BMW | [1][1] 15 | [1][2] 13 |
| [2][0] Saab | [2][1] 5 | [2][2] 2 |
| [3][0] Land Rover | [3][1] 17 | [3][2] 15 |


```

<?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

Sort Functions For Arrays

- **sort()** - sort the elements of the array in ascending order
- **rsort()** - sort the elements of the array in descending order

```
<?php
```

```
$cars = array("Volvo", "BMW", "Toyota");    //array(4, 6, 2, 22, 11);  
sort($cars);    //rsort($cars);
```

```
$clength = count($cars);                                // to print array  
for($x = 0; $x < $clength; $x++) {  
    echo $cars[$x];  
    echo "<br>";  
}  
?>
```

Cont ...

- `asort()` - sort associative arrays in ascending order, according to the value
- `ksort()` - sort associative arrays in ascending order, according to the key
- `arsort()` - sort associative arrays in descending order, according to the value
- `krsort()` - sort associative arrays in descending order, according to the key

Cont ...

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

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

\$_GET

- \$_GET - super global variable
- Data send in GET is visible in URL
 - **visible to everyone** (all variable names and values are displayed in the URL)
- Not preferable for send sensitive data. (Password)
- has limits on the amount of information to send
- In GET you send information to server in two way :
 - Through URL
 - Through from

Cont ...

- **Through URL**

```
<html>
<body>
<a href="action.php?name=john&age=24">Send</a>
</body>
</html>
```

action.php

```
<html>
<body>
<?php
echo "Name" . $_GET['name'] . " age " . $_GET['age'];
?>
</body>
</html>
```

Cont ...

- **Through From**

```
<html>
<body>
<form action="action_page.php" method="get">
First name:<br>
<input type="text" name="firstname"><br>
Last name:<br>
<input type="text" name="lastname"><br><br>
<input type="submit" value="Submit">
</form>
</body>
</html>
```

\$_POST

- \$_POST - super global variable
- Data send in POST is not visible in URL
 - **invisible to others**
- Preferable for send sensitive data (Password)
- has **no limits** on the amount of information to send.
- In POST you send information to server in one way - form
- **Developers prefer POST for sending form data**

Cont ...

```
<html>
<body>
<form action="action_page.php" method="post">
First name:<br>
<input type="text" name="firstname"><br>
Last name:<br>
<input type="text" name="lastname"><br><br>
<input type="submit" value="Submit">
</form>
</body>
</html>
```

In action_page.php we gather the information of the input field using \$_POST super global variable.

GET vs. POST

- two ways to send information to server : GET and POST
- Both GET and POST create an array (e.g. array(key1 => value1, key2 => value2, key3 => value3, ...)).
- treated as `$_GET` and `$_POST` – superglobals
- Super global variables are built-in variables that are always available in all scopes.
- `$_GET` and `$_POST` are always accessible, regardless of scope

`$_SERVER`

- `$_SERVER` - holds information about headers, paths, and script locations
- `$_SERVER['PHP_SELF']` - Returns the filename of the currently executing script
- `$_SERVER['SERVER_NAME']` - Returns the name of the host server
- `$_SERVER['HTTP_HOST']` - Returns the Host header from the current request
- `$_SERVER['HTTP_REFERER']` - Returns the complete URL of the current page
- `$_SERVER['SCRIPT_NAME']` - Returns the path of the current script

`$_REQUEST`

- used to collect data after submitting an HTML form

PHP - File Inclusion

- Include the content of a PHP file into another PHP file before the server executes it

- Two PHP functions :
 - The include() Function
 - The require() Function

Syntax

```
include 'filename';
```

or

```
require 'filename';
```

- The include (or require) statement takes all the text/code/markup that exists in the specified file and copies it into the file that uses the include statement.

CONT...

- Including files is very useful when you want to include the same PHP, HTML, or text on multiple pages of a website.
- These functions helps in –
 - creating functions, headers, footers, or elements that can be reused on multiple pages.
 - This will help developers to make it easy to change the layout of complete website with minimal effort.
 - If there is any change required then instead of changing thousand of files just change included file.
- Thus, including files saves a lot of work.

CONT...

- The include and require statements are identical, except upon failure:
 - require will produce a fatal error (E_COMPILE_ERROR) and stop the script
 - include will only produce a warning (E_WARNING) and the script will continue
- So, if you want the execution to go on and show users the output, even if the include file is missing, use the include statement.
- Otherwise, in case of a complex PHP application coding, always use the require statement to include a key file to the flow of execution.
 - This will help avoid compromising your application's security and integrity, just in-case one key file is accidentally missing.

The include() Function

- takes all the text in a specified file and copies it into the file that uses the include function.
- If there is any problem in loading a file then the **include()** function generates a warning but the script will continue execution.
- Refer example – (include folder)
 - Assume you want to create a common menu for your website
 - see - menu.php
 - U can create as many pages as you like for your website and can include “menu.php” file to create same header for all those pages
 - For example – see - food.php file
 - In which menu.php file is included
 - U can find more examples in the same folder
 - Also, try them using require function - it will generate same result.

The require() Function

- is also used to include a file into the PHP code
- If there is any problem in loading a file then the **require()** function generates a fatal error and halt the execution of the script.
- So there is no difference in `require()` and `include()` except they handle error conditions.
- It is recommended to use the `require()` function instead of `include()`, because scripts should not continue executing if files are missing or misnamed.
- Refer example –
 - (in include folder) `noFileExists_include.php` (using `include` function)
 - If we do the same example using the `require` statement, the `echo` statement will not be executed because the script execution dies after the `require` statement returned a fatal error.
 - See - (in `require` folder) `noFileExists_require.php` (using `require` function)

CONT...

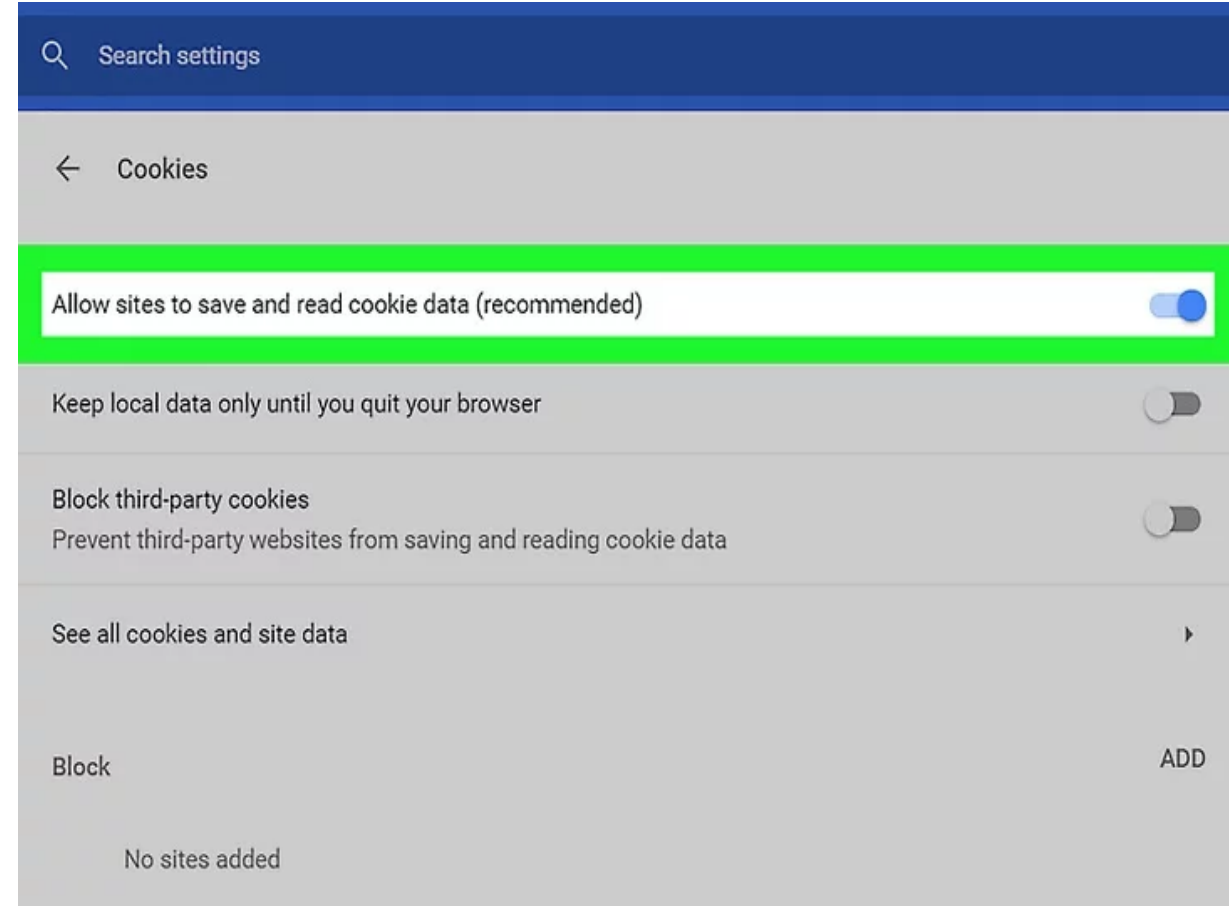
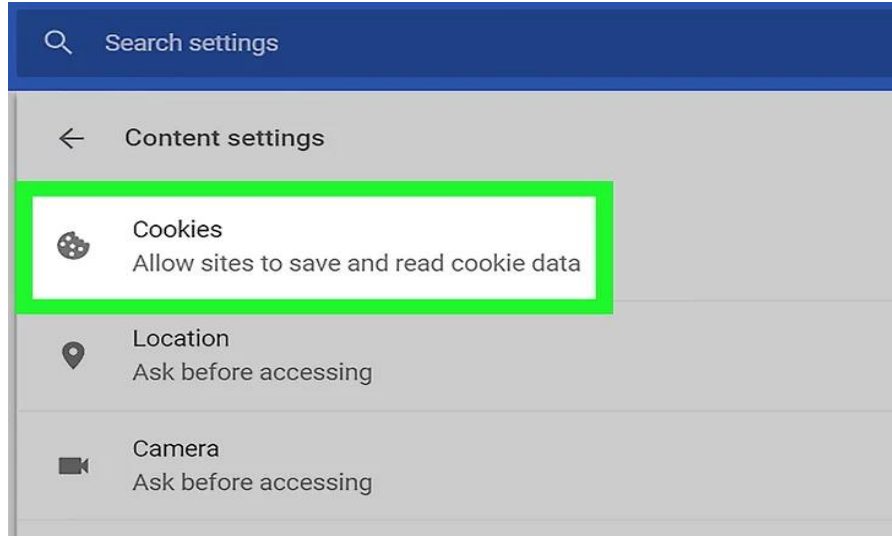
- So,
 - Use require when the file is required by the application.
 - Use include when the file is not required and application should continue when file is not found.

PHP - Cookies



- Cookies are text files stored on the client computer and they are kept for use tracking purpose.
- PHP transparently supports HTTP cookies.
- There are three steps involved in identifying returning users –
 - Server script sends a set of cookies to the browser. For example name, age, or identification number etc.
 - Browser stores this information on local machine for future use.
 - When next time browser sends any request to web server then it sends those cookies information to the server and server uses that information to identify the user.
- PHP transparently supports HTTP cookies. With PHP, you can both create and retrieve cookie values.

CONT...

- You can find cookies in your browser settings:

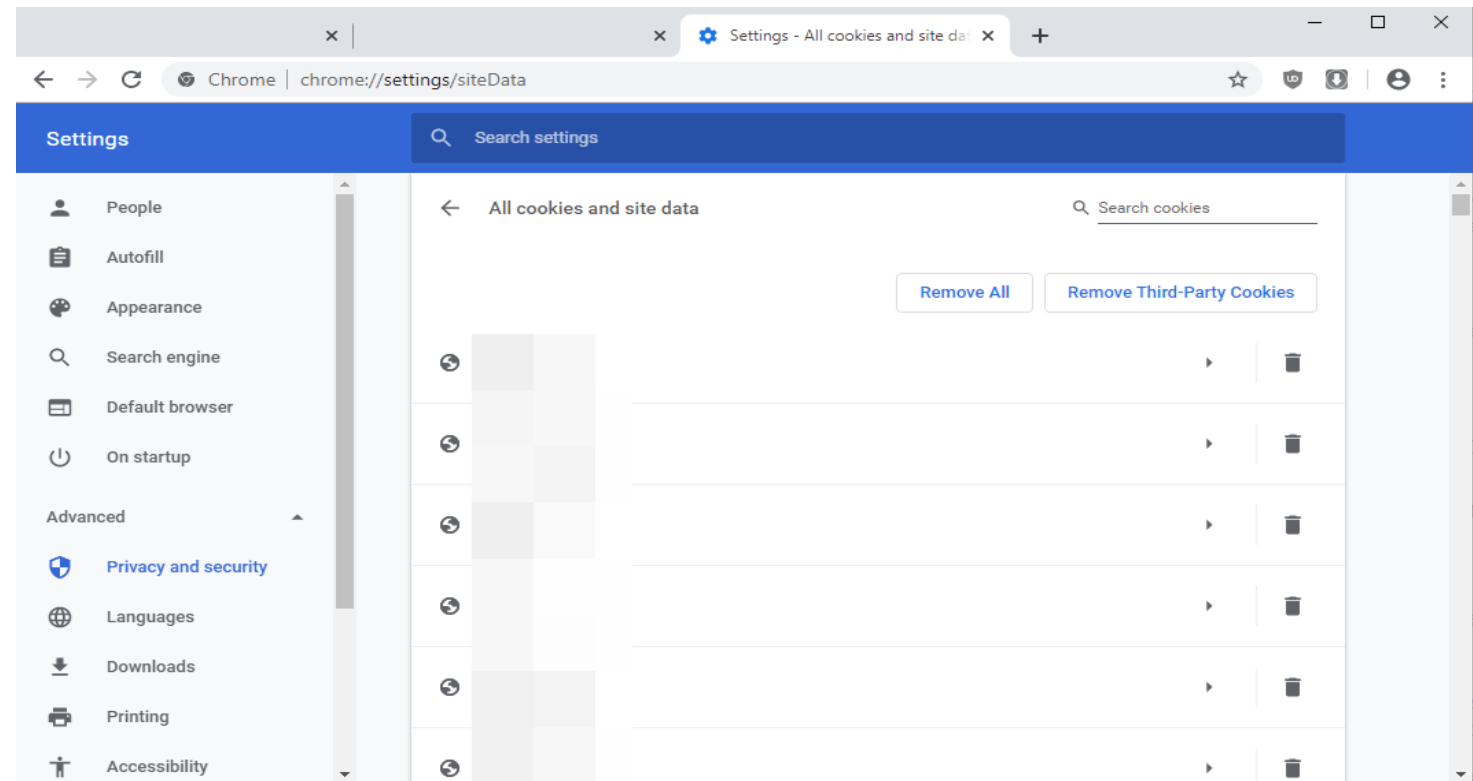


Note:

- Click the grey "Allow sites to save and read cookie data (recommended)" switch 
- It will turn blue 
- Google Chrome will allow cookies from now on.
- If this switch is blue, Google Chrome is already allowing cookies.

CONT...

- You can either load `chrome://settings/siteData` directly to jump straight to the list of cookies or select Menu > Settings > Privacy and Security > Site Settings > Cookies and site data > See all cookies and site data to go there.
- option : “See all cookies and site data”
 - In which you can find stored cookies and their data
 - Also u can also remove the cookies



CONT...

Create Cookies With PHP

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

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

- Only the name parameter is required. All other parameters are optional.
- For each cookie this function has to be called separately.
- Note: The `setcookie()` function must appear BEFORE the `<html>` tag.

CONT...

- **Name** – This sets the name of the cookie. 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 specifies a future time in seconds. 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 means cookie can be sent by regular HTTP.

CONT...

Accessing Cookies with PHP

- PHP provides many ways to access cookies.
- Simplest way is to use either `$_COOKIE` or `$HTTP_COOKIE_VARS` variables.
- You can use **isset()** function to check if a cookie is set or not.

Modify a Cookie Value

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

Delete a Cookie

- To delete a cookie, use the `setcookie()` function with an expiration date in the past

CONT...

- Refer example – (in cookies folder)
 - createcookie.php
 - accesscookie.php
 - checkcookie.php
 - modifycookie.php
 - deletetecookie.php

CONT ...

Refer example – (in cookies/cookie-example-1 folder)

- When you fill the form and submit it (main.html) , it will create two cookies: (as action="cookie.php" is given)
 - (i) named "Username" with the value which is entered by user in form.
(which you can access via `$_POST` for both the case)
 - (ii) named "Password" with the value which is entered by user in form.
- We then retrieve the value of the cookie "Username" - using the global variable `$_COOKIE`.
- We also use the `isset()` function to find out if the cookie is set.
 - And display message accordingly.

PHP - Sessions

- to make data accessible across the various pages of an entire website
 - use PHP Session
- A session creates a file in a temporary directory on the server where registered session variables and their values are stored.
- This data will be available to all pages on the site during that visit.
- So,
 - 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.

CONT...

- 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).
- So; Session variables hold information about one single user, and are available to all pages in one application.
- A session ends when the user closes the browser or after leaving the site, the server will terminate the session after a predetermined period of time, commonly 30 minutes duration.
- (If you need a permanent storage, you may want to store the data in a database.)

CONT...

Starting a PHP Session

- A PHP session is easily started by making a call to the **session_start()** function.
- This function first checks if a session is already started and if none is started then it starts one.
- It is recommended to put the call to **session_start()** at the beginning of the page.
- Session variables are set with the PHP global variable: `$_SESSION`.
 - for example: `$_SESSION["favcolor"] = "yellow";`
- These session variables can be accessed during lifetime of a session.

CONT...

Destroying a PHP Session

- To remove all global session variables and destroy the session, use `session_unset()` and `session_destroy()`
- A PHP session can be destroyed by **`session_destroy()`** function.
- This function does not need any argument and a single call can destroy all the session variables.
- If you want to destroy a single session variable then you can use **`unset()`** function to unset a session variable.
- To unset a single variable –

```
<?php
```

```
    unset($_SESSION['favcolor']);
```

```
?>
```

CONT...

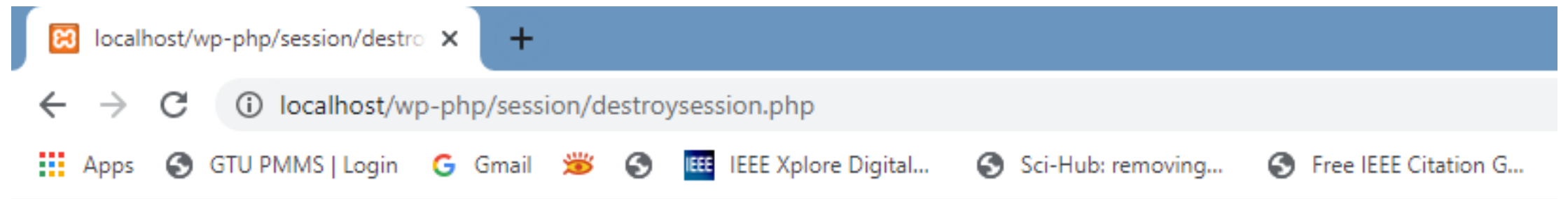
- Refer example :
 - startsession.php
 - In this a new session is started & some session variables are set.
 - getsessionvar.php
 - From this page, we will access the session information we set on the first page ("startsession.php").
 - Notice that session variables are not passed individually to each 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 `$_SESSION` variable.
 - Hence, you can easily access them using `$_SESSION`
 - for example: To print the value of session variable - 'favcolor' which we set in first page, use -
`echo $_SESSION["favcolor"];`

CONT...

- Refer example:
 - modifysession.php
 - To change a session variable, just overwrite it.
 - for example: `$_SESSION["favcolor"] = "red";`
 - After running this file,
 - Try to run **getsessionvar.php file** again, and see output.
 - It is also changed – since session variable is modified. So, now onwards whenever you try to print its value, you will find updated value
 - destroysession.php
 - To remove all global session variables and destroy the session, use `session_unset()` and `session_destroy()`

CONT ...

- Note – to see and understand how to access/ modify/ destroy session variables, you need to first run the file in which you have created session variables. (in our case it is - startsession.php)
 - Otherwise you will find 'Notice: Undefined index:' in getsessionvar.php output (because in that case session variables wouldn't have been set at all – so how and what can be accessed)
 - But if consider the same case – where, session variables are not set and you try to modify the session variable or try to run modifysession.php file –
 - You can see in output that – the variable which we modified (favcolor)– prints its value
 - Because - To change a session variable, we just need to overwrite it.
 - Syntax is same – so here it is considered it as the first time setting of that variable (& not modifying).
 - But if you try to print the value of any other session variable other than this (which isn't previously set or currently modified) (ex: favanimal) – it will give you 'Notice: Undefined index:'
 - Because again – in that case it will consider that – that session variable wouldn't have been set
 - And as far as destroy case is concerned -
 - You can have better idea about working of session_unset() and session_destroy() –
 - if you once try to set session variable and then try to remove it.
 - because after all in both scenario – you set or not set - it will give you 'Notice: Undefined index:' .



Notice: Undefined index: favcolor in C:\xampp\htdocs\WP-php\session\destroysession.php on line 16

Notice: Undefined index: favanimal in C:\xampp\htdocs\WP-php\session\destroysession.php on line 18

CONT...

- **isset()** function - to check if session variable is already set or not.
- Refer example:
 - countersession.php

Cookie Vs. Session

| Cookie | Session |
|---|--|
| •Cookies are client-side files that contain user information | •Sessions are server-side files which contain user information |
| •Cookie ends depending on the lifetime you set for it | •A session ends when a user closes his browser |
| •You don't need to start cookie as it is stored in your local machine | •In PHP, before using <code>\$_SESSION</code> , you have to write <code>session_start()</code> |
| •There is no function named <code>unsetcookie()</code> | •To remove all global session variables and destroy the session, use <code>session_unset()</code> and <code>session_destroy()</code> |

PHP Form Handling

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

PHP Form Validation Example

* required field

Name: *

E-mail: *

Website:

Comment:

Gender: ☐ Female ☐ Male ☐ Other *

Your Input:

CONT...

- **preg_match** - Perform a regular expression match
- The **preg_match()** function searches a string for pattern, returning true if the pattern exists, and false otherwise
- PHP - Validate E-mail
 - The easiest and safest way to check whether an email address is well-formed is to use PHP's **filter_var()** function.
- **filter_var** - Filters a variable with a specified filter
- Note - in detail you can understand from next slides onwards (just for your knowledge purpose)

PHP - Regular Expressions

- Regular expressions are nothing more than a sequence or pattern of characters itself.
- They provide the foundation for pattern-matching functionality.
- Using regular expression you can search a particular string inside a another string, you can replace one string by another string and you can split a string into many chunks.
- PHP offers variety of functions under pattern matching functionality.

CONT...

- PHP offers functions specific to two sets of regular expression functions, each corresponding to a certain type of regular expression.
- You can use any of them based on your comfort.
 - POSIX Regular Expressions
 - PERL Style Regular Expressions

POSIX Regular Expressions

- To use regular expressions → need to learn syntax of patterns
- The structure of a POSIX regular expression is not dissimilar to that of a typical arithmetic expression: various elements (operators) are combined to form more complex expressions.
- Brackets ([]) - have a special meaning when used in the context of regular expressions. They are used to find a range of characters.

| Expression & Description |
|--|
| [0-9] - It matches any decimal digit from 0 through 9. |
| [a-z] - It matches any character from lower-case a through lowercase z. |
| [A-Z] - It matches any character from uppercase A through uppercase Z. |
| [a-Z] - It matches any character from lowercase a through uppercase Z. |

- The ranges shown in table are general.
- you could also use the range [0-3] to match any decimal digit ranging from 0 through 3.
- the range [b-v] to match any lowercase character ranging from b through v.

CONT...

- Quantifiers - The frequency or position of bracketed character sequences and single characters can be denoted by a special character.
- Each special character having a specific connotation.
- The +, *, ?, {int. range}, and \$ flags all follow a character sequence.

| Expression & Description | |
|--------------------------|--|
| p+ | - It matches any string containing at least one p. |
| p* | - It matches any string containing zero or more p's. |
| p? | - It matches any string containing zero or one p's. |
| p{N} | - It matches any string containing a sequence of N p's |
| p{2,3} | - It matches any string containing a sequence of two or three p's. |
| p{2, } | - It matches any string containing a sequence of at least two p's. |
| p\$ | - It matches any string with p at the end of it. |
| ^p | - It matches any string with p at the beginning of it. |

CONT...

- Examples

| Expression & Description | |
|---------------------------------|--|
| [^a-zA-Z] | - It matches any string not containing any of the characters ranging from a through z and A through Z. |
| p.p | - It matches any string containing p, followed by any character, in turn followed by another p. |
| ^. {2}\$ | - It matches any string containing exactly two characters. |
| (.*) | - It matches any string enclosed within and . |
| p(hp)* | - It matches any string containing a p followed by zero or more instances of the sequence hp. |

CONT...

- Predefined Character Ranges (character classes) - specify an entire range of characters, for example, the alphabet or an integer set

| Expression & Description | |
|---------------------------------|---|
| <code>[[:alpha:]]</code> | - It matches any string containing alphabetic characters aA through zZ. |
| <code>[[:digit:]]</code> | - It matches any string containing numerical digits 0 through 9. |
| <code>[[:alnum:]]</code> | - It matches any string containing alphanumeric characters aA through zZ and 0 through 9. |
| <code>[[:space:]]</code> | - It matches any string containing a space. |

PHP's Regexp POSIX Functions

| Function & Description | |
|------------------------------|---|
| <code>ereg()</code> | - The <code>ereg()</code> function searches a string specified by <code>string</code> for a string specified by <code>pattern</code> , returning <code>true</code> if the pattern is found, and <code>false</code> otherwise. |
| <code>ereg_replace()</code> | - The <code>ereg_replace()</code> function searches for string specified by <code>pattern</code> and replaces <code>pattern</code> with <code>replacement</code> if found. |
| <code>eregi()</code> | - The <code>eregi()</code> function searches throughout a string specified by <code>pattern</code> for a string specified by <code>string</code> . The search is not case sensitive. |
| <code>eregi_replace()</code> | - The <code>eregi_replace()</code> function operates exactly like <code>ereg_replace()</code> , except that the search for <code>pattern</code> in <code>string</code> is not case sensitive. |
| <code>split()</code> | - The <code>split()</code> function will divide a string into various elements, the boundaries of each element based on the occurrence of <code>pattern</code> in <code>string</code> . |
| <code>spliti()</code> | - The <code>spliti()</code> function operates exactly in the same manner as its sibling <code>split()</code> , except that it is not case sensitive. |
| <code>sql_regcase()</code> | - The <code>sql_regcase()</code> function can be thought of as a utility function, converting each character in the input parameter <code>string</code> into a bracketed expression containing two characters. |

PERL Style Regular Expressions

- Perl-style regular expressions are similar to their POSIX counterparts.
- The POSIX syntax can be used almost interchangeably with the Perl-style regular expression functions.
- In fact, you can use any of the quantifiers introduced in the previous POSIX section.

CONT...

Meta characters

- simply an alphabetical character preceded by a backslash that acts to give the combination a special meaning.

CONT...

- Following is the list of meta characters which can be used in PERL Style Regular Expressions.

| Character | Description |
|---------------|--|
| . | a single character |
| \s | a whitespace character (space, tab, newline) |
| \S | non-whitespace character |
| \d | a digit (0-9) |
| \D | a non-digit |
| \w | a word character (a-z, A-Z, 0-9, _) |
| \W | a non-word character |
| [aeiou] | matches a single character in the given set |
| [^aeiou] | matches a single character outside the given set |
| (foo bar baz) | matches any of the alternatives specified |

CONT ...

Modifiers

- Several modifiers are available that can make your work with regexps much easier, like case sensitivity, searching in multiple lines etc.

| Modifier | Description |
|----------|--|
| i | Makes the match case insensitive |
| m | Specifies that if the string has newline or carriage return characters, the ^ and \$ operators will now match against a newline boundary, instead of a string boundary |
| o | Evaluates the expression only once |
| s | Allows use of . to match a newline character |
| x | Allows you to use white space in the expression for clarity |
| g | Globally finds all matches |
| cg | Allows a search to continue even after a global match fails |

PHP's Regexp PERL Compatible Functions

| Function & Description | |
|-------------------------------|--|
| <code>preg_match()</code> | - The <code>preg_match()</code> function searches string for pattern, returning true if pattern exists, and false otherwise. |
| <code>preg_match_all()</code> | - The <code>preg_match_all()</code> function matches all occurrences of pattern in string. |
| <code>preg_replace()</code> | - The <code>preg_replace()</code> function operates just like <code>ereg_replace()</code> , except that regular expressions can be used in the pattern and replacement input parameters. |
| <code>preg_split()</code> | - The <code>preg_split()</code> function operates exactly like <code>split()</code> , except that regular expressions are accepted as input parameters for pattern. |
| <code>preg_grep()</code> | - The <code>preg_grep()</code> function searches all elements of <code>input_array</code> , returning all elements matching the regexp pattern. |
| <code>preg_quote()</code> | - Quote regular expression characters |

The PHP Filter Extension

- PHP filters are used to validate input.
- The PHP filter extension has many of the functions needed for checking user input, and is designed to make data validation easier and quicker.

Why Use Filters?

- Many web applications receive external input. External input/data can be:
 - User input from a form
 - Cookies
 - Web services data
 - Server variables
 - Database query results
- Invalid submitted data can lead to security problems and break your webpage!
By using PHP filters you can be sure your application gets the correct input!
- **So, you should always validate external data!**

filter_var()

- used to filter a variable with a specified filter
- It takes two pieces of data:
 - The variable you want to check
 - The type of check to use
- The filter_var() function returns filtered data on success, or false on failure.

CONT...

Validate an Integer

```
<?php
$int = 100;

if (!filter_var($int, FILTER_VALIDATE_INT) === false)
{
    echo("Integer is valid");
} else {
    echo("Integer is not valid");
}
?>
```

Output : Integer is valid

- This example uses the filter_var() function to check if the variable \$int is an integer.
- If \$int is an integer, the output will be: "Integer is valid".
- If \$int is not an integer, the output will be: "Integer is not valid"

CONT...

filter_var() and Problem With 0:

- In the example above, if \$int was set to 0, the function will return "Integer is not valid".
- To solve this problem, use:

```
<?php
$int = 0;
if (filter_var($int, FILTER_VALIDATE_INT) === 0 || !filter_var($int, FILTER_VALIDATE_INT) === false) {
    echo("Integer is valid");
} else {
    echo("Integer is not valid");
}
?>
```

Output –

Integer is valid

Note: (example)

```
$int = 0;
echo filter_var($int, FILTER_VALIDATE_INT);
Outputs - 0
```

CONT...

- Validate an Email Address

```
<?php
$email = "john@example.com";

// Validate e-mail
if (!filter_var($email, FILTER_VALIDATE_EMAIL) === false) {
    echo("$email is a valid email address");
} else {
    echo("$email is not a valid email address");
}
?>
```

This example uses the `filter_var()` function to check if `$email` variable is a valid email address

Output - john@example.com is a valid email address

Thank You!!!