

PHP

PHP Arrays

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

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

```
<?php
```

```
$cars = array ("Volvo", "BMW", "Toyota");  
echo $cars[0]; // Volvo  
?>
```

Three types of array →

1) Indexed Array. → arrays with a numeric index.

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

0th index (assigned automatically)

or assigned manually → ~~\$cars~~ · \$cars[0] = "volvo";

Loop through an indexed Array

{ < ? php

\$cars = array ("BMW", "volvo");

\$arraylength = count(\$cars);

for
loop

for (\$x = 0; \$x < \$arraylength; \$x++)

{ echo \$cars[\$x];

echo "
";

}

for each loop works on arrays and objects only.
and is used to loop through
each key/value pair in an array.

foreach (\$array as \$value)

{
stmt;

: {

{ ej →

foreach (\$cars as \$Value).

{ echo "\$value
"; // BMW
volvo.

{

Or:

foreach (\$array as \$key => \$value)

{

{ ej →

{ echo "{\$key} => {\$value}";

{ } // O => BMW.

(2) Associative Arrays

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

```
$age = array ("Peter" => "35", "Ben" => "37",
              "Joe" => "45");
```

access \Rightarrow echo \$age['Peter']; // 35

loop through

```
{
    $age
    {
        foreach ($age as $x => $x-value)
            echo "Key = ". $x . ", Value = ". $x-value;
    }
}
```

1. Key = Peter, value = 35 & count

each()

each function is meant to be used in loops over collections such as arrays. each time through the array, it returns the current element's key and value & then moves to next element.

To handle a multiple item return value from an array we use list fn. { ~~return~~ assign the two return values from each to separate variables }

```
if [which (list ($key, $value) = each ($cars))
    { echo "Key: $key; Value: $value\n"; } ]
```

⇒ PHP → Hypertext Preprocessor.
widely used, open-source scripting language.
PHP scripts are executed on servers.

why PHP →

- runs on various platforms (Windows, Linux, etc.)
- compatible with all servers (Unix, etc.)
(Apache, IIS etc.)

Syntax →

<?php

|| → stmt ends with a colon ;

?>

Case Sensitivity

- In PHP, all keywords, classes, functions, and user-defined vars are NOT case sensitive.
- all variables names are case-sensitive.

Variables

- starts with \$ sign, followed by name

- o PHP is a loosely typed language i.e. we did not have to tell PHP which data type the variable is. It automatically converts the variable to the correct Data type, depending on its value.

- all variable names are Case Sensitive.

→ VARIABLES

a variable starts with \$ sign, followed by the name.

```
$txt = "Hello World"; // string type
$var = 5; // Integer type
```

- PHP is a loosely typed language, meaning that PHP converts automatically, the variable to the correct data type, depending on its value.

→ Variable SCOPE

part of script where the variable can be referenced / used.

- Local → A variable declared within a function has a local scope & can be only be accessed within that function.

</php

```
function myTest()
{
    $x = 5;
    echo $x; // 5
```

}

myTest()

```
?> echo $x; // error cannot be
           == accessed outside
```

(2) GLOBAL

variable declared outside a function has a global scope and can only be accessed outside a function.

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

```
function myTest()
```

```
{  
    echo $x; // error
```

```
}
```

```
echo $x; // 5
```

```
?>
```

• GLOBAL Keyword

used to access a global variable from within a function.

```
<?php
```

```
$x = 5;
```

```
function myTest()
```

```
{  
    use this global $x;
```

```
    echo $x; // 5
```

```
    keyword }
```

PHP stores all global Variables in an array called

- also $\$GLOBALS$ [index]
- accessed from within the function. (holds the name of the variable)

$\$GLOBALS['y'] = \$GLOBALS['x'] + \$GLOBALS['y'] ;$

STATIC

↳ when we want a local Variable not to be deleted.

<?php

function myTest()

{ static \$x = 0;

echo \$x;

\$x++;

}

myTest(); // 0

myTest(); // 1

myTest(); // 2

?>

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

⇒ Echo / Print

both used to output data to the screen.

- has no return value
- can take multiple arguments
- faster.

- has a return value of 1
- can take only 1 argument

```
- echo "Hello --> ?;  
- echo "Hello", "World";  
- echo $txt;
```

⇒ PHP Datatypes

1. string → sequence of 'chars' or "" "
2. Integer → non decimal no.
3. float → decimal or exponential.
4. Boolean → True or False
5. Array → stores multiple values in single variable.
6. Object → data-type which stores data & info. on how to process data.

g →

```
class Car {
```

```
    function Car () {
```

```
        $this->Model = "VW";
```

```
}
```

```
{
```

```
$hubre = new Car();  
echo $hubre->model;
```

- 7. Null Value → Null
 - 8. Resource

⇒ String functions

1. `strlen (" ")` → length of a string.
 2. `str-word-count (" ")` → word count.
 3. `strrev (" ") ;` // Reverse
 4. `strpos (" ", " ", ")` → (if found chr pos
to be scanned else false)
 5. `str_replace (" ", " ", " ", " ") ;`
to be replace with string
 6. `htmlspecialchars()` → converts characters to HTML entities
 7. `strcmp (" ", " ") ;`
string1 string2
(case-sensitive)
↓
0 → equal
10 → string1 < string2
>0 → string1 > string2
 8. `trim (" ", " ") ;`
string what to replace

\Rightarrow Constants

- Is an identifier for a simple value.
 - global across the entire script.

define (name , value, case sensitivity)
default → false.

⇒ Operators → Arithmetic, Assignment, Relational,
 Relational (logical, Bitwise, Ternary, MOD
 condition) → Operator Precedence & associativity.

- arithmetic → +, -, *, /, %, ** (\$x + \$y =
\$x * \$y)
- Assignment → =, +=, -=, *=, /=, %=
- Comparison → (1) == {Equal}
 (2) === {Identical} Value/type
 (3) != {Not equal} or \neq
OR
 (4) !== {Not identical} {Value & type}
 (5) >, <, >=, <=
- Increment / Decrement (++\$x, \$x++, --\$x, \$x--)
- Logical (1) and (both true) &&
 (2) OR / ||
 (3) XOR \oplus
OR (4) NOT \neg
- ~~if/else~~ Conditional
 ?: If condition is True? Then value
 X: Otherwise value Y.

⇒ Unary
 Binary
 Condition (Ternary)
 Assignment

Precedence of PHP
 determines the grouping of terms in an expression. This affects how an expression is evaluated.

⇒ functions not case sensitive

block of stmts that can be used repeatedly in a program.

Default Argument Value

function setheight (\$minheight = 50)

{

}

call setheight (350)

setheight () // uses minheight = 50

⇒ conditional

<?php

\$a = 10;

\$b = 20;

{ \$result = (\$a > \$b) ? \$a : \$b ;

IF cond" is true. assign a otherwise

(b)

??

≡

PHP

- ✓ (1) PHP introduction (inventions, tool, slw degrms.)
 - ✓ (2) Scope of PHP
 - ✓ (3) Syntax [constants , variables]
 - ✓ (4) data types , Expressions , scope of variables
 - ✓ (5) Operators
 - ✓ (6) Operator Precedence & associativity
 - ✓ (7) IF / ELSE , Switch , FOR , WHILE , GOTO , Break , Continue , exit
 - ✓ (8) functions.
 - ✓ (9) call by value / call by reference } scope
 - ✓ (10) String manipulation
 - ✓ (11) Regular Expressions . | PHP injections
 - ✓ (12) Arrays . | → passing vulnerabilities into your webapp . It is a malicious code injection technique.
 - ✓ (13) forms
-

⇒ PHP Sort functions for Arrays

(1) sort()

→ sort array in ascending order

(2) rsort()

→ descending order

(3) asort()

associative - ascending - (value)

(4) ksort()

associative - ascending - (key)

| arsort() - descending

| rsort() → descending

Array functions

①

array_chunk (\$array, size, preserve key)

of each
chunk

(optional)

↓
true or false

\$cars = array ("volvo", "BMW", "Toyota",
"Honda", "Mercedes");

point-> (array_chunk (\$cars, 2));

↓
default
Reindexes
the
chunk
numerical

Output: Array ([0] => array([0] => volvo [1] => BMW)
[1] =>)

②

array_fill (index, number, value);

↓
first
index of
returned
array

↓
no. of
elements
to insert

↓
Value to fill array

\$cars = array_fill (3, 2, "Blue");

Output → Array ([3] => Blue [4] => Blue);

③

array_filter (array, callback function);

↓
(returns an
array with filtered
values)

• ↓ passes each value of
the function to put array
to the callback fun.

• (Keys are preserved)

⑦ array_flip (array) → keys → values }
values → keys }

arraymap (for, array) { operation}

⑧ array_map (function, array1, array2, arrays...) =

(makes changes in original array)
to all Values

\$a1 = array ("Dog", "Cat");
\$a2 = array ("Horse", "Kitten");

\$a = array_map (null, \$a1, \$a2);
output ↴

Array [0] ⇒ Array [0] ⇒ Dog [1] ⇒ ~~Cat~~
[1] ⇒ Array [0] ⇒ Cat [1] ⇒ Kitten)

keeps key same (Value changes)

⑨ array_merge (array1, array2 ...)

If two array have same keys. last one overrides

⑩ array_rand (array, number) ↴ (no. of random values
to be generated)

\$a = array ()

\$key_random = array_rand (\$a);

→ echo \$a [\$key_random [0]]

⑧ array_replace (array1, array2, ...)

{
 \$a1 = array ("red", "green");
 \$a2 = array ("blue", "yellow");
 array_replace (\$a1, \$a2);

Output → Array ([0] ⇒ blue, [1] ⇒ yellow)

⑨ array_slice (array, start, length, preserve)

(new indexes)

(first element) (starts at second last element of the array)

\$a = array ("1", "2", "3", "4");
array_slice (\$a, 2);

Output →

array ([0] ⇒ 3, [1] ⇒ 4);

Default
false
reset
keys

optional

⑩ array_splice (array, start, length, array)

(new indexes)

(removes selected elements from an array & replaces with new elements)

(changes are done in this array)

{
 \$a1 = array ("a" => "1", "b" => "2", "c" => "3");
 \$a2 = array ("a" => "one", "b" => "two");
 array_splice (\$a1, 0, 2, \$a2);

Output →

array ([0] ⇒ one, [1] ⇒ two, [2] ⇒ 3)

If length = 0 ⇒ do not remove any element
but add a2 at start position & then continue

array_walk (array, function)

optional 71

⑪ array_walk (array , function , parameters...)

function a (\$value, \$key)

if echo "**\$key \$value**"

{

\$a = array ("a" => "red", "b" => "green", "c" => "blue");
array_walk (\$a, "function a");

output →

fn. runs each array element in a user-defined fn.
The array's keys & values are parameters in fn.

⑫ PHP

count (array)

no. of elements in array.

⑬ \$array_diff (\$a1, \$a2) ⇒ (a1 - a2)

⑭ array_intersect (\$a1, \$a2)

⑮ array_fill_keys (\$array, ~~Value~~) ;

{ \$a = arrays ("a", "b", "c");

\$a1 = array_fill_keys (~~\$a~~ \$a, "Blue");

⇒ Array {

[a] => blue

[b] => blue

[c] => blue

}

- (16) array_key_exists ("key", \$array) True or False
- (17) array_keys (\$array)

(18) mt_rand()
 round()
 ceil()
 floor()
 json_encode()
 json_decode();

Important

{ strtolower(\$str)
strToUpper(\$str)}

⇒ json_decode / encode

\$arr = array ("a" => "1", "b" => "2", "c" => "3")

\$x = json_encode (\$arr);

Output →

~~object~~

{ "a": "1" & "b": "2" , "c": "3" }

\$a = json_decode (\$x);
 associative array

⇒ mt_rand (0, 100);
start (end)

\$heroes[0] = 'Captain Am';
array_pop(\$heroes); //

72

⇒ Cookies

- A cookie is often used to identify a user.
- stored on user's computer in text file format
- cannot hold multiple variables
- we can set expiry for a cookie

Create Cookies

- `setcookie (name, value, expire, path, domain, secure, httponly)` ;
must appear before `</php>` intner>tag.

`$cookie_name = "user";`

`$cookie_value = "John Doe";`

`setcookie ($cookie_name, $cookie_value, time() +
(86400 * 30), "/");` 86400 = 1 day

(it's available
in entire website)

`</php>`

`if (!isset($_COOKIE[$cookie_name]))`

{

`echo "Cookie named ". $cookie_name . " is not set";`

else {

`echo "Cookie ". $cookie_name . " is set ";`

}

`>>`

`=`

⇒ to delete → `setcookie ("user", "", time() - 3600);`

→ enabled → `If (count($_COOKIE) > 0)` to one or to go
`enabled`

new page → instead, they are set/created by
↑ session_start(),

⇒ SESSIONS

- a session is a way to store information (in variables) reused across multiple web pages.
- Info is stored on server
- can hold multiple variables.
- remains active as long as browser is open.
- cannot access data stored in sessions.

→

Start - `<?php`
session_start(); ; } // top of the page
?>

`<body>`

`<?php`
\$SESSION["favcolor"] = "green";
\$SESSION["favanimal"] = "cat";

?>

Destroy / unset

session_unset(); // remove all session variables.

session_destroy() // destroy session.

Q → P Diff. b/w \$var & \$\$var

Variable Interpolation.

`$a = "abc"`

`$$a = "5"`

`echo $abc` → 5

String

optional

① chop (\$string, "word");

removes characters from right end of a string.
trim()

if chop (\$string)

it removes → "10", "lt", "ln".
 & white space

② echo (str).

\$color = "red";

{ difference of single & double quotes.

echo "Roses are \$color"; // Roses are red

echo 'Roses are \$color'; // Roses are \$color

str-split (\$string, length)

③ explode (separator, string, limit)

→ 0, +ve, -ve

(from last)

↓ optional (no. of elements to be returned)

[breaks a string into an array]

where ↓
break string

(the string to split)

{ ex → \$str = "Hello world. It is a day";

explode(" ", \$str));

Output ↴

Array ([0] => Hello [1] => world [2] => It [3] =>
 [4] => a [5] => day) ↴

④ htmlspecialchars (\$str).

optional

⑤ implode (separator, array)

[returns a string from the elements of array]

{ \$arr = array('Hello', 'world');
 echo implode(" ", \$arr);

⑥ ltrim (\$str , "Hello");
↳ removes from left

⑦ str_repeat ("str" , repeat) ;

numerical value

⑧ str_replace (find , replace , string , count)
(optional)

⑨ { echo str_replace ("world" , "Peter" , "Hello world") ;
output → Hello Peter! }

echo \$a = array ("Blue" , "Red" , "Yellow") ;
print_r (str_replace ("Red" , "Pink" , \$a , \$c)) ;
output →
array [0] => Blue [1] => Pink. [2] => Yellow) .

⑩ Str_word_count (str) // returns no. of words

⑪ Strrchr (str , search , before search) ;
↳ search ↳ search (optional)
→ returns last of the str

⑫ strcmp (str1 , str2) → 0 - equal
< 0 - str1 < str2
> 0 - str1 > str2

⑬ ~~Strpos~~ Strpos (str , find , start)
(returns start pos of str) optional

⑭ ~~Strlen~~ strlen (str)
↳ returns length of a str.

(14) stripos (string)

(15) trim (string, charlist)

optional

(removes
charlist
from both
sides)

→ Global Variables — Superglobals

{ Pre-defined variables in PHP
that are always accessible regardless
of scope.

(1) `$GLOBALS` → used to access global variables
→ PHP stores all global variables in an array
called `$GLOBALS` [index].
↳ name of variable.

(2) `$_SERVER` → holds info about headers, paths
& script locations

- `$_SERVER['PHP_SELF']`
returns filename of the currently executing script
- `$_SERVER['SERVER_NAME']`
returns name of the host server
- `['HTTP_HOST']`
returns host header from current request.
- `['HTTP_REFERER']` `['HTTP_USER_AGENT']` `['SCRIPT_NAME']`
(use of current page)
- `['QUERY_STRING']` → query string if the page is accessed via a query string.

{ (3) \$-REQUEST

used to collect data after submitting an HTML form

(7) \$-POST

method = "post".

(5) \$-GET

method = "get"

also collect data send in URL.

⇒ \$-COOKIE, \$-SESSION, \$-FILES, \$-ENV.



What is query string? Explain with an example

Ans →

The information can be sent across the web pages. This information is called query string.

- This query string can be passed from one page to another by appending it to the address of the page.
- You can pass more than one query string by inserting the '&' sign b/w query strings.
contains
querystring ID & its value.

The query string passed across the web pages is stored in \$-REQUEST variable, \$-GET or \$-POST.

```

<html>
  <head>
    <title> File —
  </head>
  <body>
    <form action = "welcome.php" method = "get" >
      <input type = "text" name = "username" >
      <input type = "text" name = "email" >
      <input type = "submit" name = "sub" value = "Submit" >
    </form>

```

welcome.php ↴

```

echo "Welcome $_GET['username']"  

$_GET['email']

```

Q → Give the syntax to pass & capture the variables between PHP web pages during navigation

① url

 —

② GET / POST method form.

access → \$_REQUEST['id'] -

③ sessions startSession()

\$_SESSION['id'] = "Tanya";

8 →

`$ Strif1 = "into onto unto";`

`$ Sub = "to";`

`IF (strpos ($strif1, $Sub) ;`

`str_replace ($Sub, "Tanya", $Strif1)`

→ strstr()

`Strstr ($strif, $Search, $before)`



/

to
in
which

↓

to search
for

optional

false
default

true

returns part
of \$strif before
the first occurrence
of \$search.

returns

search

rest of the \$strif from
matching point

`strstr ($use, "@")`

`$use = "nachiketan@example.com";`

Output → @example.com

`ltrim ("@example.com", "@");`

Output → example.com

→ Regular Expressions

- ↳ Are powerful pattern matching algorithm that can be performed in a single expression.
- ↳ helps in validating email, IP address etc.

functions

(1) preg-match()

used to perform match on a string. It returns true if a match is found & false otherwise

```
{
    ↪ PHP
    $my_url = "www.guru99.com";
    IF (preg-match (" / guru / ", $my_url)).
        {
            ↪
            " / pattern / ", $string .
        }
    ↪
}
```

(2) preg-split()

used to perform a pattern match on a string and then split the results into a numeric array.

```
{
    ↪ PHP
    $my_text = "I love Regular Expressions";
    $my_array = preg-split (" / / ", $mytext);
    ↪
    Output → Array ([0] ⇒ I [1] ⇒ love. [2] ⇒ regular
                    [3] ⇒ Expressions) .
```

③ preg_replace()

\$ used to perform a pattern match on a string & then replace the match with the specified text.

```
{ $text = "We at Guru99 to make, quality";  
$text = preg_replace( "/Guru1/",  
'<span style="background: yellow">Guru</span>',  
$text );
```

⇒
 \d → any digit (0-9) [\d → not digit]
 \s → any white space [\w ⇒ not word]
 \w → any word char (a-z) (A-Z) (0-9) (-)
 . → any char
 [abc] = a, b or c [(abc) {3} → abc abc
 [^a-z] ⇒ not a-z abc]

{ 3 } → match only three of these

{ } → on zero or one char

* → zero or more matches

+ → One or more

^ abc → beginning

abc \$ → cut the end

(abc) → group

⇒ Email first-name.last-name@domain.com'

\$regexp = '/ ^[\w.] + @ [\w.] + [a-zA-Z]{2,4} \$ /c';

switch

for case insensitive
match

Database

① configuration file

<?php

```
$servername = "localhost";  
$password = " ";  
$username = "root";  
$db = "myDB";  
$conn = mysqli_connect($servername, $username,  
                      $password, $db);  
?>
```

② (Included only once)

createdb

<?php
\$su -
pas -
usn -

```
$conn = mysqli_connect($su, $pas, $usn);  
IF ($conn)
```

```
{ echo "connection";  
$sql = "create database myDB";  
$res = mysqli_query($conn, $sql);  
IF ($res)
```

```
{ echo " Database";  
else {
```

```
? else { }
```

③ Create Table (only once)

<?php

```
$conn = include('configuration.php');
if ($conn)
{
    echo "success";
    $sql = "create table users (id int not null primary
key, name varchar(250) not null,
email --,
pass --- ) ;
$res = mysqli_query($conn, $sql);
if ($res)
{
    echo "success";
}
```

?>

④ Login.php

<?php

```
include('config.php');
echo "<form action='login.php' method='Post'>


```

```
if ($rescheck > 0) {  
    $row = mysqli_fetch_assoc($res);  
    $name = $row['name'];  
    echo "Hello, $name";  
}  
else {  
    echo "Invalid email or password.";  
}  
}
```

⑤ Insert

```
$sql = "Insert values into employee()  
        values ( , );";  
$sql = ' _____';  
$sql = ' _____';  
if (mysqli_multi_query($conn, $sql)) {  
    echo "Multiple rows inserted";  
}  
else {  
    _____;  
};
```

⇒ Magic Constants in PHP

- PHP provides a set of special defined constants that change depending on where they are used
— LINE — , — FILE — , — DIR — .

- ⇒ `include()` → gives warning if file is not present
`require()` → does same, but open failure generates fatal error & halts execution.
`include_once()` → checks if file already has been included & so it will not include the file again.

~~★ \$car = array("one" => "BMW", "TWO" => "Toyota");~~

⇒ ~~Var-dump (\$car)~~ → displays no. of elements, datatype & values

~~Output :~~

~~array (2) {~~

~~['One'] => BMW~~

⇒ `$arr = array(' ', false, 72, array("42"));`

① `Var-dump ($arr);`

`array(4) {`

`[0] => string(0) " "`

`[1] => boolean(false)`

`[2] = int(72)`

`[3] = array(1) { [0] => string(2) "42" }`

`}`

② print -e (\$arr)
⇒

Array ([0] =>

[1] => ~~false~~

[2] => 72

[3] => Array ([0] => 72)
)

③ Var-export (\$arr)
⇒

array (

0 => ',',

2 => false,

3 => 72,

3 => array (0 => 72,),

)

=

\$val = 42;

\$valu = '41';

IF (is_numeric (\$val)) // True ~~False~~

IF (is_numeric (\$valu)) // True

Question paper

Q → 6

x6 php

```
$server = "localhost";
$user = "root";
$pass = "";
$db = "admin";
$conn = mysqli_connect ($server, $user, $pass,
$db);
```

IF (! \$conn)

{ die ("connection failed : ". mysqli_connect_error()); }

else {

```
$co $sql = "select id, name, salary from
employee";
```

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

```
$acs = mysqli_num_rows ($result);
```

IF (\$acs > 0)

{

```
echo "<table> <tr> <th> ID </th>
<th> Name </th>
<th> Salary </th>
</tr> ";
```

echo //

```
while ( $row = mysqli_fetch_assoc ($result) )
```

{

```
echo "<tr> <td>". $row ["id"]. "</td>
<td>". $row ["name"]. "</td> <td>
$row [ "salary"] . " </td> </tr> ";
```

```
echo "</table>";  
 }  
 else {  
     echo "0 results";  
     mysqli_close($conn);  
 }
```

⇒

=
= PHP

\$x = 3;

function fn (\$var)

```
{  
    global $x;  
    static $a = 0;  
    $b = 2;  
    $b++;  
    $a++;  
    $x++;  
    echo $b;  
    echo $a;  
    echo $x;  
    echo "<br>";  
}
```

```
}  
$arr = array(1, 2, 3, 4);  
array_walk($arr, "fn");  
}
```

Output:

\$n=3. \$a=4, \$b=2.3
2 3 4 2 3
echo → 3 1 7

\$ 523 325 →

★ GET method()

- Displayed in URL.
- Remain in Browser history and can never be cached.
- Can be bookmarked.
- Should not be used with sensitive data , used only to retrieve data.
- Have length restrictions of upto 1024 characters only.
- Can't be used to send binary data like images or word documents to the server.

★ POST method()

- Not displayed in URL.
- Do not remain in browser history.
- Can't be bookmarked.
- Should be used with sensitive data.
- Do not have any length restrictions.
- Can be used to send text as well as binary data.

★ SuperGlobal

It is a predefined variable that is always accessible through any function, class or file.

\$GLOBALS: Contains a reference to every variable within the global scope of the script . The keys are the names of the global variable.

\$-SERVER: An array containing information such as headers , paths and script locations. The entries in this array are created by web server.

\$-REQUEST: An associative array consisting of contents of **\$-GET**, **\$-POST** and **\$-COOKIE**.

\$-GET: An associative array of variables passed to the current script via HTTP GET method.

\$-POST: An associative array of variables passed to the current script via HTTP POST method.

\$-FILES: An associative array of items uploaded to current script via HTTP POST method.

\$-COOKIE: An associative array of variable passed to current script via HTTP cookie.

\$-SESSION: An associative array containing session variable to the current script.

\$-PHP_SELF: A string containing PHP script file name in which it is called.

\$php_errormsg: Variable containing the text of last error message generated by PHP.

\$-SERVER['PHP_SELF']: Returns the filename of currently executing script.

\$-SERVER['SERVER_ADDR']: Returns the IP address of the host server.

\$-SERVER['SERVER_NAME']: Returns name of host server.

\$-SERVER['REQUEST_METHOD']: Returns the request method used to access the page.

\$-SERVER['QUERY_STRING']: Returns the query string if the page is accessed via a query string.

\$-SERVER['HTTP_HOST']: Returns the host header from the current request.

\$-SERVER['HTTP_REFERER']: Returns the complete URL of the current page.

\$-SERVER['REMOTE_ADDR']: Returns the IP address from where the user is viewing the current page.

\$-SERVER['REMOTE_HOST']: Returns the host name from where the user is viewing the current page.

\$-SERVER['REMOTE_PORT']: Returns the port being used on the user's machine to communicate with web server.

\$-SERVER['SCRIPT_FILENAME']: Returns the absolute pathname of currently executing script.

\$-SERVER['SCRIPT_NAME']: Returns the path of the current script.

\$-SERVER['SCRIPT_URI']: Returns the URI of current page.

* File Handling functions

`fopen` : opens file or URL.

`fclose` : closes an open file pointer.

`fread` : Binary-safe file read.

`fwrite` : Binary-safe file write.

`feof` : Tests for end of file on a file pointer

`file_exists` : checks whether a file or directory exists.

`fgetc` : gets character from file pointer.

`fgets` : gets line from file pointer.

`fflush` : flushes the output to a file.

`fseek` : seeks on a file pointer.

`ftell` : Returns current position of the file read/write pointer.

`fscanf` : Parses output from a file according to a format

`readfile` : outputs a file.

`rename` : Renames a file or directory

`rewind` : Rewind the position of a file pointer.

`mkdir` : Makes a directory

`rmdir` : Removes a directory

`filesize` : Returns length of file.

`copy` : copies a file

`file` : Reads a file into an array.

`file_get_contents` : Reads a file into a string.

`file_put_contents` : Writes a string to a file.

`fileowner` : Returns user ID (owner) of a file.

`fileperms` : Returns the permissions of a file.

`filetype` : Returns file type.

`flock` : Locks or releases a file.

`ftruncate` : Truncates an open file to a specified length.

unlink: Deletes a file.

is_readable: checks whether a file is readable.

is_writable: checks whether a file is writable.

pathinfo: Returns information about a file path.

stat: Returns information about a file.

chmod: changes file permissions for file.

* Cookies

- A cookie is a small file that the server embeds on the user's computer.
- A cookie is used to identify a user.
- Each time the same computer requests a page with a browser it will send the cookie too.
- cookies are usually set in an HTTP header.
- Cookie header contains a name-value pair, both & domain.

* Creating a cookie

A cookie is created with the setcookie() function.

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

setcookie() function must appear before the <html> tag.

Simplest way to access cookies is with \$COOKIE associative array.

isset() function is used to check if a cookie is set or not.

```

< form action="setcookie.php" method="POST">
    Name : < input type="text" name="name" />
    Age : < input type="text" name="age" />
    < input type="submit" name="submit" />
</ form>

```

<? .php

```

if ( isset( $_POST['submit'] ) )
{
    $nm = $_POST['name'];
    $age = $_POST['age'];
    setcookie("name", $nm);
    setcookie("age", $age, time() + 3600, "/");
}
?>

```

<? .php

```

if ( (isset( $_COOKIE['name'] ) || (isset( $_COOKIE['age'] ) ) )
{
    echo "Name : ". $_COOKIE['name']. "<br> Age : ". $_COOKIE['age']
}
?>

```

* Deleting a cookie

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

<? .php

```

setcookie("age", $age, time() - 3600, "/");

```

?>

Database

L. php

```
$host = "localhost";
$username = "username";
$password = "password";
```

// create connection

```
$conn = mysqli_connect($host, $username, $password);
```

// check connections

```
if (!$conn)
```

```
die("connection failed: ".mysqli_connect_error());
```

// create database

```
$sql = "CREATE DATABASE myDB";
```

```
if (mysqli_query($conn, $sql))
```

```
echo "Database created successfully";
```

```
else
```

```
echo "Error creating database: ".mysqli_error($conn);
```

// close a connection

```
mysqli_close($conn);
```

```
?>
```

To get ID of last inserted record

```
$last_id = mysqli_insert_id($conn);
```

mysqli_query(): used to execute single SQL statement.

mysqli_multi_query(): used to execute multiple SQL statements

* Select Stmt

```
$sql = "SELECT id, firstname, lastname FROM My DB";  
$result = mysqli_query($conn, $sql);
```

```
if (mysqli_num_rows($result) > 0)  
    while ($row = mysqli_fetch_assoc($result))  
        echo "id: " . $row['id'] . " Name: " . $row['firstname'];  
else echo "0 results";
```

num_rows() checks if there are more than 0 rows returned.

fetch_assoc(): puts all the returned rows into an associative array.