Unit-1: PHP fundamentals

Concepts of Php and introduction

PHP started out as a small open source project that evolved as more and more people found out how useful it was. Rasmus Lerdorf unleashed the first version of PHP way back in 1994.

- PHP is a recursive acronym for "PHP: Hypertext Preprocessor".
- PHP is a server side scripting language that is embedded in HTML. It is used to manage dynamic content, databases, session tracking, even build entire e-commerce sites.
- It is integrated with a number of popular databases, including MySQL, PostgreSQL, Oracle, Sybase, Informix, and Microsoft SQL Server.
- PHP is pleasingly zippy in its execution, especially when compiled as an Apache module on the Unix side. The MySQL server, once started, executes even very complex queries with huge result sets in record-setting time.
- PHP supports a large number of major protocols such as POP3, IMAP, and LDAP. PHP4 added support for Java and distributed object architectures (COM and CORBA), making n-tier development a possibility for the first time.
- PHP is forgiving: PHP language tries to be as forgiving as possible.
- PHP Syntax is C-Like.

Common uses of PHP

- PHP performs system functions, i.e. from files on a system it can create, open, read, write, and close them.
- PHP can handle forms, i.e. gather data from files, save data to a file, through email you can send data, return data to the user.
- You add, delete, modify elements within your database through PHP.
- Access cookies variables and set cookies.
- Using PHP, you can restrict users to access some pages of your website.
- It can encrypt data.

Characteristics of PHP

Five important characteristics make PHP's practical nature possible −

- Simplicity
- Efficiency
- Security
- Flexibility
- Familiarity

"Hello World" Script in PHP

To get a feel for PHP, first start with simple PHP scripts. Since "Hello, World!" is an essential example, first we will create a friendly little "Hello, World!" script.

As mentioned earlier, PHP is embedded in HTML. That means that in amongst your normal HTML (or XHTML if you're cutting-edge) you'll have PHP statements like this −

Live Demo

```
<html>

  <head>
```

```
    <title>Hello World</title>
  </head>

  <body>
    <?php echo "Hello, World!";?>
  </body>

</html>
```

Output:

Hello, World!

Variables

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

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

Rules for PHP variables:

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

Note: Remember that PHP variable names are case-sensitive!

Variable Scope

PHP has three types of variable scopes:

1. Local variable
2. Global variable
3. Static variable

Global

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

Example

Variable with global scope:

```
<?php
    $name = "Sanaya Sharma";        //Global Variable
    function global_var()
    {
        global $name;
        echo "Variable inside the function: ". $name;
        echo "</br>";
```

```
    }
    global_var();
    echo "Variable outside the function: ". $name;
?>
```

Another way to use the global variable inside the function is predefined $GLOBALS array.

```
<?php
    $num1 = 5;      //global variable
    $num2 = 13;     //global variable
    function global_var()
    {
        $sum = $GLOBALS['num1'] + $GLOBALS['num2'];
        echo "Sum of global variables is: " .$sum;
    }
    global_var();
?>
```

Local

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

Example

Variable with local scope:

```
<?php
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>";
?>
```

Static

It is a feature of PHP to delete the variable, once it completes its execution and memory is freed. Sometimes we need to store a variable even after completion of function execution. Therefore, another important feature of variable scoping is static variable. We use the static keyword before the variable to define a variable, and this variable is called as **static variable**.

Static variables exist only in a local function, but it does not free its memory after the program execution leaves the scope. Understand it with the help of an example:

Example:

```
 <?php
function static_var()
{
    static $num1 = 3;       //static variable
    $num2 = 6;          //Non-static variable
    //increment in non-static variable
    $num1++;
```

```
        //increment in static variable
        $num2++;
        echo "Static: " .$num1 ."</br>";
        echo "Non-static: " .$num2 ."</br>";
    }

//first function call
    static_var();

    //second function call
    static_var();
?>
```

Output

```
Static: 4
Non-static: 7
Static: 5
Non-static: 7
```

PHP $ and $$ Variables

The **$var** (single dollar) is a normal variable with the name var that stores any value like string, integer, float, etc.

The **$$var** (double dollar) is a reference variable that stores the value of the $variable inside it.

To understand the difference better, let's see some examples.

Example

```
<?php
$x = "abc";
$$x = 200;
echo $x."<br/>";
echo $$x."<br/>";
echo $abc;
?>
```

Output

```
abc
200
200
```

Constants

PHP constants are name or identifier that can't be changed during the execution of the script except .

1. Using define() function
2. Using const keyword

Syntax

```
define(name, value)
```

Example

```
<?php
    define("MSG1","Hello JavaTpoint PHP");
    echo MSG1;
    const MSG2="Hello const by JavaTpoint PHP";
    echo MSG2;
?>
```

echo

The echo statement can be used with or without parentheses: echo or echo().

Display Text

The following example shows how to output text with the echo command

Example:

```
<?php
echo "<h2>PHP is Fun!</h2>";
echo "Hello world!<br>";
$txt1 = "Learn PHP";
$txt2 = "W3Schools.com";
$x = 5;
$y = 4;

echo "<h2>" . $txt1 . "</h2>";
echo "Study PHP at " . $txt2 . "<br>";
echo $x + $y;
echo "Hello by PHP echo
this is multi line
text printed by
PHP echo statement
";
echo "Hello escape \"sequence\" characters";

?>
```

Print

The print statement can be used with or without parentheses: print or print().

Display Text

The following example shows how to output text with the print command (notice that the text can contain HTML markup):

Example

```
<?php
print "<h2>PHP is Fun!</h2>";
print "Hello world!<br>";
print ("Hello by PHP print()");
?>
```

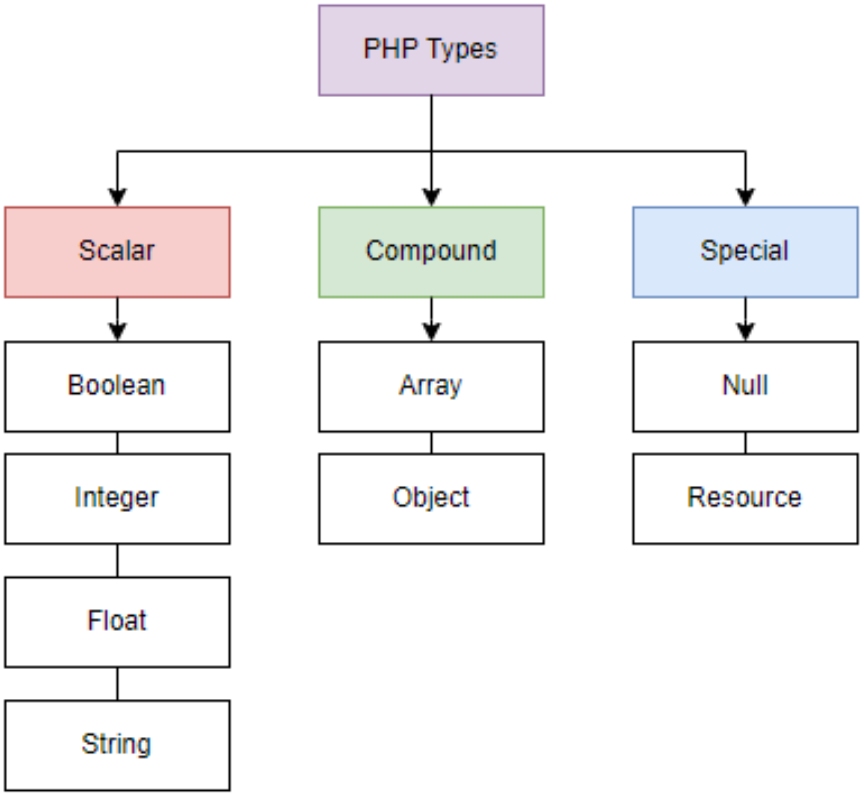
Echo VS Print

Echo	Print
1. echo does not return any value.	1. print always returns an integer value,

2. We can pass multiple strings separated by comma (,) in echo. 3. echo is faster than print statement.	which is 1. 2. Using print, we cannot pass multiple arguments. 3. print is slower than echo statement.

Data types

A type specifies the amount of memory that allocates to a value associated with it.



Scalar type:

It hold single value only:

1. Boolean: Booleans are the simplest data type works like switch. It holds only two values: **TRUE (1)** or **FALSE (0)**
2. Integer:Integer means numeric data with a negative or positive sign. It holds only whole numbers, i.e., numbers without fractional part or decimal points. The range of an integer must be lie between 2,147,483,648 and 2,147,483,647 i.e., -2^31 to 2^31.
3. Float: A floating-point number is a number with a decimal point.
4. String: A string is a non-numeric data type. It holds letters or any alphabets, numbers, and even special characters.String values must be enclosed either within single quotes or in double quotes. But both are treated differently.
   Example: $name = "Raman";

```
//both single and double quote statements will treat different
    echo "Hello $name";
    echo "</br>";
    echo 'Hello $name';
```

   Output:

Hello Javatpoint
Hello $company

Compound Type:

It hold multiple values

1. Array: An array is a compound data type. It can store multiple values of same data type in a single variable.

Example:

$scores = [1, 2, 3];

2. Objects are the instances of user-defined classes that can store both values and functions.

Special type

1.Resource: Resources are not the exact data type in PHP. Basically, these are used to store some function calls or references to external PHP resources. For example - a database call.

2. Null: Null is a special data type that has only one value: NULL

Operators

PHP divides the operators in the following groups:

- Arithmetic operators: The PHP arithmetic operators are used to perform common arithmetic operations such as addition, subtraction, etc. with numeric values.

Operator	Name	Example	Explanation
+	Addition	$a + $b	Sum of operands
-	Subtraction	$a - $b	Difference of operands
*	Multiplication	$a * $b	Product of operands
/	Division	$a / $b	Quotient of operands
%	Modulus	$a % $b	Remainder of operands
**	Exponentiation	$a ** $b	$a raised to the power $b

- Assignment operators: The assignment operators are used to assign value to different variables. The basic assignment operator is "=".

Operator	Name	Example	Explanation
=	Assign	$a = $b	The value of right operand is assigned to the left operand.
+=	Add then Assign	$a += $b	Addition same as $a = $a + $b
-=	Subtract then Assign	$a -= $b	Subtraction same as $a = $a - $b
*=	Multiply then Assign	$a *= $b	Multiplication same as $a = $a * $b

/=	Divide then Assign (quotient)	$a /= $b	Find quotient same as $a = $a / $b
%=	Divide then Assign (remainder)	$a %= $b	Find remainder same as $a = $a % $b

- Conditional assignment operators

The bitwise operators are used to perform bit-level operations on operands. These operators allow the evaluation and manipulation of specific bits within the integer.

Operator	Name	Example	Explanation
&	And	$a & $b	Bits that are 1 in both $a and $b are set to 1, otherwise 0.
|	Or (Inclusive or)	$a | $b	Bits that are 1 in either $a or $b are set to 1
^	Xor (Exclusive or)	$a ^ $b	Bits that are 1 in either $a or $b are set to 0.
~	Not	~$a	Bits that are 1 set to 0 and bits that are 0 are set to 1
<<	Shift left	$a << $b	Left shift the bits of operand $a $b steps
>>	Shift right	$a >> $b	Right shift the bits of $a operand by $b number of places

- Comparison Operators

Comparison operators allow comparing two values, such as number or string. Below the list of comparison operators are given:

Operator	Name	Example	Explanation
==	Equal	$a == $b	Return TRUE if $a is equal to $b
===	Identical	$a === $b	Return TRUE if $a is equal to $b, and they are of same data type
!==	Not identical	$a !== $b	Return TRUE if $a is not equal to $b, and they are not of same data type
!=	Not equal	$a != $b	Return TRUE if $a is not equal to $b
<>	Not equal	$a <> $b	Return TRUE if $a is not equal to $b
<	Less than	$a < $b	Return TRUE if $a is less than $b
>	Greater than	$a > $b	Return TRUE if $a is greater than $b

<=	Less than or equal to	$a <= $b	Return TRUE if $a is less than or equal $b
>=	Greater than or equal to	$a >= $b	Return TRUE if $a is greater than or equal $b
<=>	Spaceship	$a <=>$b	Return -1 if $a is less than $b Return 0 if $a is equal $b Return 1 if $a is greater than $b

- Incrementing/Decrementing Operators

The increment and decrement operators are used to increase and decrease the value of a variable.

Operator	Name	Example	Explanation
++	Increment	++$a	Increment the value of $a by one, then return $a
		$a++	Return $a, then increment the value of $a by one
--	decrement	--$a	Decrement the value of $a by one, then return $a
		$a--	Return $a, then decrement the value of $a by one

- Logical Operators

The logical operators are used to perform bit-level operations on operands. These operators allow the evaluation and manipulation of specific bits within the integer.

Operator	Name	Example	Explanation
And	And	$a and $b	Return TRUE if both $a and $b are true
Or	Or	$a or $b	Return TRUE if either $a or $b is true
Xor	Xor	$a xor $b	Return TRUE if either $ or $b is true but not both
!	Not	! $a	Return TRUE if $a is not true
&&	And	$a && $b	Return TRUE if either $a and $b are true
||	Or	$a || $b	Return TRUE if either $a or $b is true

- String Operators

The string operators are used to perform the operation on strings. There are two string operators in PHP, which are given below:

Operator	Name	Example	Explanation
.	Concatenation	$a . $b	Concatenate both $a and $b
.=	Concatenation and Assignment	$a .= $b	First concatenate $a and $b, then assign the concatenated string to $a,

			e.g. $a = $a . $b

- Array Operators

The array operators are used in case of array. Basically, these operators are used to compare the values of arrays.

Operator	Name	Example	Explanation
+	Union	$a + $y	Union of $a and $b
==	Equality	$a == $b	Return TRUE if $a and $b have same key/value pair
!=	Inequality	$a != $b	Return TRUE if $a is not equal to $b
===	Identity	$a === $b	Return TRUE if $a and $b have same key/value pair of same type in same order
!==	Non-Identity	$a !== $b	Return TRUE if $a is not identical to $b
<>	Inequality	$a <> $b	Return TRUE if $a is not equal to $b

Conditional Statements

to write code that perform different actions based on the results of a logical or comparative test conditions at run time.

- The **if** statement
- The **if...else** statement
- The **if...elseif....else** statement
- The **switch...case** statement

The if Statement

The *if* statement is used to execute a block of code only if the specified condition evaluates to true.

Syntax:

```
if(condition){
   // Code to be executed
}
```

Example:

output "Have a nice weekend!" if the current day is Friday:

```
<?php
$d = date("D");
if($d == "Fri"){
   echo "Have a nice weekend!";
}
?>
```

The if...else Statement

You can enhance the decision making process by providing an alternative choice through adding an *else* statement to the *if* statement. The *if...else* statement allows you to execute one

block of code if the specified condition is evaluates to true and another block of code if it is evaluates to false. It can be written, like this:
Syntax:

```
if(condition){
   // Code to be executed if condition is true
} else{
   // Code to be executed if condition is false
}
```

Exaple:Output "Have a nice weekend!" if the current day is Friday, otherwise it will output "Have a nice day!"

```
<?php
$d = date("D");
if($d == "Fri"){
   echo "Have a nice weekend!";
} else{
   echo "Have a nice day!";
}
?>
```

The if...elseif...else Statement

The *if...elseif...else* a special statement that is used to combine multiple *if...else* statements.
Syntax:

```
if(condition1){
   // Code to be executed if condition1 is true
} elseif(condition2){
   // Code to be executed if the condition1 is false and condition2 is true
} else{
   // Code to be executed if both condition1 and condition2 are false
}
```

Example:
output "Have a nice weekend!" if the current day is Friday, and "Have a nice Sunday!" if the current day is Sunday, otherwise it will output "Have a nice day!"

```
<?php
$d = date("D");
if($d == "Fri"){
   echo "Have a nice weekend!";
} elseif($d == "Sun"){
   echo "Have a nice Sunday!";
} else{
   echo "Have a nice day!";
}
?>
```

The switch-case statement is an alternative to the if-elseif-else statement, which does almost the same thing. The switch-case statement tests a variable against a series of values until it finds a match, and then executes the block of code corresponding to that match.

```
switch(n){
   case label1:
```

```
        // Code to be executed if n=label1
        break;
    case label2:
        // Code to be executed if n=label2
        break;
    ...
    default:
        // Code to be executed if n is different from all labels
}
```

Consider the following example, which display a different message for each day.

Example

```
<?php
$today = date("D");
switch($today){
    case "Mon":
        echo "Today is Monday. Clean your house.";
        break;
    case "Tue":
        echo "Today is Tuesday. Buy some food.";
        break;
    case "Wed":
        echo "Today is Wednesday. Visit a doctor.";
        break;
    case "Thu":
        echo "Today is Thursday. Repair your car.";
        break;
    case "Fri":
        echo "Today is Friday. Party tonight.";
        break;
    case "Sat":
        echo "Today is Saturday. Its movie time.";
        break;
    case "Sun":
        echo "Today is Sunday. Do some rest.";
        break;
    default:
        echo "No information available for that day.";
        break;
}
?>
```

Arrays

It is used to hold multiple values of similar type in a single variable.

Advantage of PHP Array

Less Code: We don't need to define multiple variables.

Easy to traverse: By the help of single loop, we can traverse all the elements of an array.

Sorting: We can sort the elements of array.

There are 3 types of array in PHP.

1. Indexed Array
2. Associative Array
3. Multidimensional Array

Indexed Array

PHP index is represented by number which starts from 0. We can store number, string and object in the PHP array. All PHP array elements are assigned to an index number by default.

There are two ways to define indexed array:

```
$season=array("summer","winter","spring","autumn");
echo "Season are: $season[0], $season[1], $season[2] and $season[3]";
```

OR

```
$season[0]="summer";
$season[1]="winter";
$season[2]="spring";
$season[3]="autumn";
echo "Season are: $season[0], $season[1], $season[2] and $season[3]";
```

Associative Array

We can associate name with each array elements in PHP using => symbol.

There are two ways to define associative array:

```
$salary=array("Sonoo"=>"350000","John"=>"450000","Kartik"=>"200000");
echo "Sonoo salary: ".$salary["Sonoo"]."<br/>";
echo "John salary: ".$salary["John"]."<br/>";
echo "Kartik salary: ".$salary["Kartik"]."<br/>";
```

OR

```
$salary["Sonoo"]="350000";
$salary["John"]="450000";
$salary["Kartik"]="200000";
echo "Sonoo salary: ".$salary["Sonoo"]."<br/>";
echo "John salary: ".$salary["John"]."<br/>";
echo "Kartik salary: ".$salary["Kartik"]."<br/>";
```

Multidimensional Array

PHP multidimensional array is also known as array of arrays. It allows you to store tabular data in an array. PHP multidimensional array can be represented in the form of matrix which is represented by row * column.

Definition

```
$emp = array
  (
  array(1,"sonoo",400000),
```

```
  array(2,"john",500000),
  array(3,"rahul",300000)
  );
 for ($row = 0; $row < 3; $row++) {
  for ($col = 0; $col < 3; $col++) {
    echo $emp[$row][$col]."  ";
          }
         echo "<br/>";
        }
```

Php Loops

the following loop types:

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

while Loop

The while loop executes a block of code as long as the specified condition is true.

Syntax

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

Examples

The example below displays the numbers from 1 to 5:

Example

```
<?php
$x = 1;

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

do...while Loop

The do...while loop will always execute the block of code once, it will then check the condition, and repeat the loop while the specified condition is true.

Syntax

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

Examples

The example below first sets a variable $x to 1 ($x = 1). Then, the do while loop will write some output, and then increment the variable $x with 1. Then the condition is checked (is $x less than, or equal to 5?), and the loop will continue to run as long as $x is less than, or equal to 5:

Example

```
<?php
$x = 1;

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

for Loop

The for loop is used when you know in advance how many times the script should run.

Syntax

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

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

Examples

The example below displays the numbers from 0 to 10:

Example

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

foreach Loop

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

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.

Examples

The following example will output the values of the given array ($colors):

Example

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

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

Break

You have already seen the break statement used in an earlier chapter of this tutorial. It was used to "jump out" of a switch statement.

The break statement can also be used to jump out of a loop.

This example jumps out of the loop when **x** is equal to **4**:

Example

```
<?php
for ($x = 0; $x < 10; $x++) {
```

```
  if ($x == 4) {
    break;
  }
  echo "The number is: $x <br>";
}
?>
```

Continue

The continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

This example skips the value of **4**:

Example

```
<?php
for ($x = 0; $x < 10; $x++) {
  if ($x == 4) {
    continue;
  }
  echo "The number is: $x <br>";
}
?>
```

UNIT 2:

PHP FUNCTIONS

USER-DEFINED FUNCTIONS

We can declare and call user-defined functions easily

Syntax

```
function functionname(arguments){
//code to be executed
Return;
}
```

Example:

```
<?php
function sayHello(){
echo "Hello PHP Function";
}
sayHello();//calling function
?>
```

FUNCTION ARGUMENTS

We can pass the information in PHP function through arguments which is separated by comma. PHP supports **Call by Value** (default), **Call by Reference**, **Default argument values** and **Variable-length argument list**.

CALL BY VALUE:

```
<?php
function sayHello($name){
echo "Hello $name<br/>";
}
sayHello("Sonoo");
?>
```

CALL BY REFERENCE:

By default, value passed to the function is call by value. To pass value as a reference, you need to use ampersand (&) symbol before the argument name.

```
<?php
function adder(&$str2)
{
   $str2 .= 'Call By Reference';
}
$str = 'Hello ';
adder($str);
echo $str;
?>
```

DEFAULT ARGUMENT VALUE

We can specify a default argument value in function. While calling PHP function if you don't specify any argument, it will take the default argument

```
<?php
function sayHello($name="Sonoo"){
echo "Hello $name<br/>";
}
```

```
sayHello("Rajesh");
sayHello();//passing no value
sayHello("John");
?>
```

RETURNING VALUE

```
<?php
function cube($n){
return $n*$n*$n;
}
echo "Cube of 3 is: ".cube(3);
?>
```

we have passed two parameters **$x** and **$y** inside two functions **add()** and **sub()**.

```
<?php
//add() function with two parameter
function add($x,$y)
{
$sum=$x+$y;
echo "Sum = $sum <br><br>";
}
//sub() function with two parameter
function sub($x,$y)
{
$sub=$x-$y;
echo "Diff = $sub <br><br>";
}
//call function, get  two argument through input box and click on add or sub button
if(isset($_POST['add']))
{
//call add() function
 add($_POST['first'],$_POST['second']);
}
if(isset($_POST['sub']))
{
//call add() function
sub($_POST['first'],$_POST['second']);
}
?>
<form method="post">
Enter first number: <input type="number" name="first"/><br><br>
Enter second number: <input type="number" name="second"/><br><br>
<input type="submit" name="add" value="ADDITION"/>
<input type="submit" name="sub" value="SUBTRACTION"/>
</form>
```

IN-BUILD FUNCTIONS

MATH FUNCTIONS

Function	Description	Example &	Output

		output	
abs()	This function is used to return the absolute (positive) value of a number	abs(-3.5) abs(5) abs(-5)	3.5 5 5
acos()	This function is used to return the arc cosine of a number. value in the range of -1 to +1,otherwise it return NaN Note: same functions asin(),atan(),atan2()	acos(-0.35) acos(5)	1.92 NAN
ceil()	This function is used to round a number up to the nearest integer	ceil(3.35) ceil(-4.35) ceil(14.81)	4 -4 15
cos()	This function is used to return the cosine of a number Note:Same function sin(),tan()	cos(0)	1
floor()	This function is generally used to round a number down to the nearest integer	floor(3.35) floor(-2.35) floor(14.81)	3 -3 14
log()	This function is basically used to return the natural logarithm of a number	log(2)	0.6931
log10()	This function is basically used to return the base-10 logarithm of a number	log10(455)	2.6580
max()	This function is basically used to return the highest value in an array or it can be said that the highest value of various specified values	max(2,4,6,8,10)	10
min()	This function is basically used to return the lowest value in an array or it can be said that the lowest value of various specified values	min(2,4,6,8,10)	2
pi()	This function is simply used to return the value of the PI	Pi()	3.14
pow()	This function is simply used to return x raised to the power of y	pow(2,4)	16
rand()	It is the function that is used in order to generate a random integer	Rand() Rand(10,20)	1132363175 13
round()	This function is generally used to round a floating-point number	round(3.35) round(-2.35)	3 -2
sqrt()	This function is simply used to return the square root of a number	sqrt(9)	3

STRING FUNCTION

Function	Description	Example	Output
chop()	Deletes the whitespace or other characters present in the string from the end of a string	chop(“Hello World”,"World!")	Hello
chr()	Used to get the specific character value of a ascii value.	chr(65)	A
echo()	This function is used to print one or	echo("Hello world!");	Hello

	more string as the output		world!
implode()	Converts the elements of an array into a string.	$arr = array('Hello','World!','Beautiful','Day!'); echo implode(" ",$arr);	Hello World! Beautiful Day!
join()	Alias of implode()	$arr = array('Hello','World!','Beautiful','Day!'); echo join(" ",$arr);	Hello World! Beautiful Day!
lcfirst()	This function Changes the first character of the given string in lowercase	lcfirst("Hello world!")	hello world
ltrim()	This function eliminates the whitespaces or other characters from the left side of the given string	$str = "Hello World!"; echo $str . " "; echo ltrim($str,"Hello");	Hello World! World!
ord()	This function gives the ASCII value of the first character of any given string	ord("A")	65
print()	This function print the Output as one or more strings	print "Hello world!";	Hello world!
printf()	This function is used to print the Output as a formatted string Note: format specifier are according to c language	$number = 9; $str = "Beijing"; printf("There are %d million bicycles in %s.",$number,$str);	There are 9 million bicycles in Beijing.
rtrim()	This function eliminates whitespaces or other characters from the right side of the given string	echo $str; echo rtrim($str,"World!");	Hello World! Hello
similar_text()	This function is used to check the similarity between the two strings	similar_text("Hello World","Hello Peter")	7
str_ireplace()	This function is used to Replace some characters of a string	str_ireplace("WORLD","Peter","Hello good world!")	Hello good Peter!
str_pad()	This function is used for the Padding of a string to a new length	str_pad($str,20,".")	Hello World.....
str_repeat()	Repeats a string as many number of times you want	str_repeat("Wow",3)	WowWow Wow
str_replace()	Replaces parts of a string	str_replace("world","Peter", "Hello world!")	Hello Peter!
str_shuffle()	This function Randomly shuffles all characters of the given string	str_shuffle("Hello World")	lWl eodrloH (randomly shuffle)
str_split()	str_split() function splits(break) a string into an array.	print_r(str_split("Hello"))	Array ([0] => H [1] => e [2] => l [3] => l [4] => o)

str_word_count()	Calculates the number of words in a string	str_word_count("Hello world!")	2
strcasecmp()	Compares two strings	strcasecmp("Hello world!","HELLO WORLD!")	0 (not match)
strchr()	This function Finds the very first presence of a string inside another string.	strchr("Hello world!","world");	world
strcmp()	Compares two strings(case-sensitive) 0 - if the two strings are equal <0 - if string1 is less than string2 >0 - if string1 is greater than string2	strcmp("Hello world!","Hello world!")	0
strcspn()	Counts the number of characters found in a string	strcspn("Hello world!","w")	6
stripos()	This function gives the position of the first presence of a string inside another string	stripos("I love php, I love php too!","PHP")	7
strlen()	Calculates the number of characters in a string	strlen("Hello")	5
strpos()	Gives the position of the first presence of a string inside any other string(case-sensitive)	strpos("I love php, I love php too!","php")	7
strrev()	Retrun reverse of a given string	strrev("Hello World!")	!dlroW olleH
strripos()	Locates the position of the last presence of a string inside another string(case-insensitive)	strripos("I love php, I love php too!","PHP")	19
strrpos()	Locates the position of the last presence of a string inside another string(case-sensitive)	strrpos("I love php, I love php too!","php")	19
strspn()	Gives the count of the characters found in a given string that contains only characters from a specified charlist	strspn("Hello world!","kHlleo")	5
strtolower()	Converts in Lowercases a string	strtolower("Hello WORLD.")	hello world.
strtoupper()	Converts in Uppercases a string	strtoupper("Hello WORLD!");	HELLO WORLD.
substr()	Retrieves a section of a string	substr("Hello world",6,6) substr("Hello world",-4) substr("Hello world",4)	world orld o world
trim()	Removes leading and trailing whitespaces from a string	$str = "Hello World!"; echo trim($str,"Hed!");	llo Worl
ucfirst()	Converts in uppercase the first character of a string	ucfirst("hello world!")	Hello world!

ucwords()	Converts in uppercases the first character of every word of a string	ucwords("hello world");	Hello World
Print_r()	prints the information about a variable in a more human-readable way.	$a = array("red", "green", "blue "); print_r($a);	Array ([0] => red [1] => green [2] => blue)

DATE AND TIME FUNCTIONS

Function	Description	Example	Output
date_create()	Returns a new DateTime object	$date=date_create("2013-03-15"); echo date_format($date,"Y/m/d");	2013/03/15
date_default_timezone_get()	Returns the default timezone used by all date/time functions	date_default_timezone_get();	UTC
date_default_timezone_set()	Sets the default timezone used by all date/time functions	date_default_timezone_set("Asia/India"); echo date_default_timezone_get();	UTC
date_diff()	Returns the difference between two dates	$date1=date_create("2013-03-15"); $date2=date_create("2013-12-12"); $diff=date_diff($date1,$date2);	+272 days
date_format()	Returns a date formatted according to a specified format	$date=date_create("2013-03-15"); echo date_format($date,"Y/m/d H:i:s");	2013/03/15 00:00:00
date_time_set()	Sets the time	$date=date_create("2013-05-01"); date_time_set($date,13,24); echo date_format($date,"Y-m-d H:i:s");	2013-05-01 13:24:00 2013-05-01 12:20:55
date()	Formats a local date and time Note check table 1 for format	echo date("l") . " ";	Thursday
getdate()	Returns date/time information of a timestamp or the current local date/time	print_r(getdate());	Array ([seconds] => 44 [minutes] => 39 [hours] => 6 [mday] => 23 [wday] => 4 [mon] => 6

			[year] => 2022 [yday] => 173 [weekday] => Thursday [month] => June [0] => 1655966384)
time()	Returns the current time as a Unix timestamp	$t=time(); echo(date("Y-m-d",$t));	2022-06-23

Table 1

- d - The day of the month (from 01 to 31)
- D - A textual representation of a day (three letters)
- j - The day of the month without leading zeros (1 to 31)
- l (lowercase 'L') - A full textual representation of a day
- N - The ISO-8601 numeric representation of a day (1 for Monday, 7 for Sunday)
- S - The English ordinal suffix for the day of the month (2 characters st, nd, rd or th. Works well with j)
- w - A numeric representation of the day (0 for Sunday, 6 for Saturday)
- z - The day of the year (from 0 through 365)
- W - The ISO-8601 week number of year (weeks starting on Monday)
- F - A full textual representation of a month (January through December)
- m - A numeric representation of a month (from 01 to 12)
- M - A short textual representation of a month (three letters)
- n - A numeric representation of a month, without leading zeros (1 to 12)
- t - The number of days in the given month
- L - Whether it's a leap year (1 if it is a leap year, 0 otherwise)
- o - The ISO-8601 year number
- Y - A four digit representation of a year
- y - A two digit representation of a year
- a - Lowercase am or pm
- A - Uppercase AM or PM
- B - Swatch Internet time (000 to 999)
- g - 12-hour format of an hour (1 to 12)
- G - 24-hour format of an hour (0 to 23)
- h - 12-hour format of an hour (01 to 12)
- H - 24-hour format of an hour (00 to 23)
- i - Minutes with leading zeros (00 to 59)
- s - Seconds, with leading zeros (00 to 59)
- u - Microseconds (added in PHP 5.2.2)
- e - The timezone identifier (Examples: UTC, GMT, Atlantic/Azores)
- I (capital i) - Whether the date is in daylights savings time (1 if Daylight Savings Time, 0 otherwise)
- O - Difference to Greenwich time (GMT) in hours (Example: +0100)
- P - Difference to Greenwich time (GMT) in hours:minutes (added in PHP 5.1.3)
- T - Timezone abbreviations (Examples: EST, MDT)
- Z - Timezone offset in seconds. The offset for timezones west of UTC is negative (-43200 to 50400)
- c - The ISO-8601 date (e.g. 2013-05-05T16:34:42+00:00)
- r - The RFC 2822 formatted date (e.g. Fri, 12 Apr 2013 12:01:05 +0200)

- U - The seconds since the Unix Epoch (January 1 1970 00:00:00 GMT)

ARRAY FUNCTIONS(INCLUDE SORING ARRAY FUNCTIONS ALSO)

Function	Description	Example	Output
array()	Creates an array	$cars=array("Volvo","BMW","Toyota") $age=array("Peter"=>"35","Ben"=>"37","Joe"=>"43"); $cars=array (array("Volvo",100,96), array("BMW",60,59), array("Toyota",110,100))	
array_change_key_case()	Changes all keys in an array to lowercase or uppercase	$age=array("Peter"=>"35","Ben"=>"37","Joe"=>"43"); print_r(array_change_key_case($age,CASE_LOWER));	Array ([peter] => 35 [ben] => 37 [joe] => 43)
array_chunk()	Splits an array into chunks of arrays	$cars=array("Volvo","BMW","Toyota","Honda","Mercedes","Opel"); print_r(array_chunk($cars,2));	Array ([0] => Array ([0] => Volvo [1] => BMW) [1] => Array ([0] => Toyota [1] => Honda) [2] => Array ([0] => Mercedes [1] => Opel))
array_combine()	Creates an array by using the elements from one "keys" array and one "values" array	$fname=array("Peter","Ben","Joe"); $age=array("35","37","43"); $c=array_combine($fname,$age); print_r($c);	Array ([Peter] => 35 [Ben] => 37 [Joe] => 43)
array_count_values()	Counts all the values of an array	$a=array("A","Cat","Dog","A","Dog"); print_r(array_count_values($a)); ;	Array ([A] => 2 [Cat] => 1 [Dog] => 2)
array_diff()	Compare arrays, and returns the differences (compare values	$a1=array("a"=>"red","b"=>"green","c"=>"blue","d"=>"yellow"); $a2=array("e"=>"red","f"=>"green","g"=>"blue");	Array ([d] => yellow)

	only)	$result=array_diff($a1,$a2); print_r($result);	
array_diff_assoc()	Compare arrays, and returns the differences (compare keys and values)	$a1=array("a"=>"red","b"=>"green","c"=>"blue","d"=>"yellow"); $a2=array("a"=>"red","b"=>"green","c"=>"blue"); $result=array_diff_assoc($a1,$a2); print_r($result);	Array ([d] => yellow)
array_diff_key()	Compare arrays, and returns the differences (compare keys only)	$a1=array("a"=>"red","b"=>"green","c"=>"blue"); $a2=array("a"=>"red","c"=>"blue","d"=>"pink"); $result=array_diff_key($a1,$a2); print_r($result); ?>	Array ([b] => green)
array_fill()	Fills an array with values	$a1=array_fill(3,4,"blue"); print_r($a1);	Array ([3] => blue [4] => blue [5] => blue [6] => blue)
array_flip()	Flips/Exchanges all keys with their associated values in an array	$a1=array("a"=>"red","b"=>"green","c"=>"blue","d"=>"yellow"); $result=array_flip($a1); print_r($result);	$a1=array("a"=>"red","b"=>"green","c"=>"blue","d"=>"yellow"); $result=array_flip($a1); print_r($result);
array_intersect()	Compare arrays, and returns the matches (compare values only)	$a1=array("a"=>"red","b"=>"green","c"=>"blue","d"=>"yellow"); $a2=array("e"=>"red","f"=>"green","g"=>"blue"); $result=array_intersect($a1,$a2); print_r($result);	Array ([a] => red [b] => green [c] => blue)

array_key_exists()	Checks if the specified key exists in the array	$a=array("Volvo"=>"XC90","BMW"=>"X5");
		if (array_key_exists("Volvo",$a))
		  {
		  echo "Key exists!";
		  }
		else
			Key exists!

		{
		echo "Key does not exist!";
		}	

array_merge()	Merges one or more arrays into one array	$a1=array("red","green"); $a2=array("blue","yellow"); print_r(array_merge($a1,$a2));	Array ([0] => red [1] => green [2] => blue [3] => yellow)

array_multisort()	Sorts multiple or multi-dimensional arrays	$a=array("Dog","Cat","Horse","Bear","Zebra");
		array_multisort($a);
		print_r($a);
			Array ([0] => Bear [1] => Cat [2] => Dog [3] => Horse [4] => Zebra)

array_pad()	Inserts a specified number of items, with a specified value, to an array	$a=array("red","green"); print_r(array_pad($a,5,"blue"));	Array ([0] => red [1] => green [2] => blue [3] => blue [4] => blue)

array_pop()	Deletes the last element of an array	$a=array("red","green","blue");
		array_pop($a);
		print_r($a);
			Array ([0] => red [1] => green)

array_push()	Inserts one or more elements to the end of an array	$a=array("red","green");
		array_push($a,"blue","yellow");
		print_r($a);
			Array ([0] => red [1] => green [2] => blue [3] => yellow)

array_replace()	Replaces the values of the first array with the values from following arrays	$a1=array("red","green"); $a2=array("blue","yellow"); print_r(array_replace($a1,$a2));	Array ([0] => blue [1] => yellow)
array_reverse()	Returns an array in the reverse order	$a=array("a"=>"Volvo","b"=>"BMW","c"=>"Toyota"); print_r(array_reverse($a));	Array ([c] => Toyota [b] => BMW [a] => Volvo)
array_search()	Searches an array for a given value and returns the key	$a=array("a"=>"red","b"=>"green","c"=>"blue"); echo array_search("red",$a);	a
array_slice()	Returns selected parts of an array	$a=array("red","green","blue","yellow","brown"); print_r(array_slice($a,2));	Array ([0] => blue [1] => yellow [2] => brown)

array_sum()	Returns the sum of the values in an array	$a=array(5,15,25); echo array_sum($a);	45
array_unique()	Removes duplicate values from an array	$a=array("a"=>"red","b"=>"green","c"=>"red"); print_r(array_unique($a));	Array ([a] => red [b] => green)

arsort()	Sorts an associative array in descending order, according to the value	$age=array("Peter"=>"35","Ben"=>"37","Joe"=>"43");
		arsort($age);
		
		foreach($age as $x=>$x_value)
		{
		echo "Key=" . $x . ", Value=" . $x_value;
		echo "
		";
		}
			Key=Joe, Value=43
			Key=Ben, Value=37
			Key=Peter, Value=35

asort()	Sorts an associative array in ascending order, according to the value	$age=array("Peter"=>"35","Ben"=>"37","Joe"=>"43");
		asort($age);
		
		foreach($age as $x=>$x_value)
		{
		echo "Key=" . $x . ", Value=" . $x_value;
		echo "
		";
		}
			Key=Peter, Value=35
			Key=Ben, Value=37
			Key=Joe, Value=43

compact()	Create array containing variables and their values	$firstname = "Peter"; $lastname = "Griffin"; $age = "41"; $result = compact("firstname", "lastname", "age"); print_r($result);	Array ([firstname] => Peter [lastname] => Griffin [age] => 41)
count()	Returns the number of elements in an array	$cars=array("Volvo","BMW","Toyota"); echo count($cars);	3

in_array()	Checks if a specified value exists in an array	$people = array("Peter", "Joe", "Glenn", "Cleveland");
		
		if (in_array("Glenn", $people))
		{
		echo "Match found";
		}
			Match found

		else
		{
		echo "Match not found";
		}	

krsort()	Sorts an associative array in descending order, according to the key	$age=array("Peter"=>"35","Ben"=>"37","Joe"=>"43");
		krsort($age);
		
		foreach($age as $x=>$x_value)
		{
		echo "Key=" . $x . ", Value=" . $x_value;
		echo "
		";
		}
			Key=Peter, Value=35
			Key=Joe, Value=43
			Key=Ben, Value=37

ksort()	Sorts an associative array in ascending order, according to the key	$age=array("Peter"=>"35","Ben"=>"37","Joe"=>"43");
		krsort($age);
		
		foreach($age as $x=>$x_value)
		{
		echo "Key=" . $x . ", Value=" . $x_value;
		echo "
		";
		}
			Key=Ben, Value=37
			Key=Joe, Value=43
			Key=Peter, Value=35

rsort()	Sorts an indexed array in descending order	$cars=array("Volvo","BMW","Toyota"); rsort($cars);	Volvo Toyota BMW
sort()	Sorts an indexed array in ascending order	$cars=array("Volvo","BMW","Toyota"); sort($cars);	BMW Toyota Volvo

VARIABLE HANDLING FUNCTIONS

Function	**Description**
boolval()	Returns the boolean value of a variable
empty()	Checks whether a variable is empty
is_array()	Checks whether a variable is an array
is_bool()	Checks whether a variable is a Boolean
is_float()/is_double()	Checks whether a variable is of type float
is_int()/is_integer()	Checks whether a variable is of type integer
is_null()	Checks whether a variable is NULL
is_numeric()	Checks whether a variable is a number or a numeric string
is_string()	Checks whether a variable is of type string

isset()	Checks whether a variable is set (declared and not NULL)
print_r()	Prints the information about a variable in a human-readable way
unset()	Unsets a variable
var_dump()	Dumps information about one or more variables

MISCELLANEOUS FUNCTIONS:

Function	**Description**	**Example**
Define	define(*name,value,case_insensitive*) A constant's value cannot be changed after it is set Constant names do not need a leading dollar sign ($) Constants can be accessed regardless of scope Constant values can only be strings and numbers	define("pi",3.14); echo pi; Output:3.14
Exit()	exit(*message*) prints a message and terminates the current script	$x = 1; exit ($x);
Die()	die(*message*) Print a message and terminate the current script Note: exit() is used to stop the execution of the program, and die() is used to throw an exception and stop the execution.	mysql_connect("hostname","mysqlusername","") or die('We are aware of the problem and working on it');
Header	header(*header*, *replace*, *http_response_code*) Sends a raw HTTP header to a client	header("Expires: Mon, 26 Jul 1997 05:00:00 GMT"); header("Cache-Control: no-cache");

GET,POST AND REQUET METHODS

There are two ways the browser client can send information to the web server.

- The GET Method
- The POST Method

Before the browser sends the information, it encodes it using a scheme called URL encoding. In this scheme, name/value pairs are joined with equal signs and different pairs are separated by the ampersand.

name1=value1&name2=value2&name3=value3

Spaces are removed and replaced with the + character and any other nonalphanumeric characters are replaced with a hexadecimal values. After the information is encoded it is sent to the server.

THE GET METHOD

The GET method sends the encoded user information appended to the page request. The page and the encoded information are separated by the **?** character.
http://www.test.com/index.htm?name1=value1&name2=value2

- The GET method produces a long string that appears in your server logs, in the browser's Location: box.
- The GET method is restricted to send upto 1024 characters only.
- Never use GET method if you have password or other sensitive information to be sent to the server.
- GET can't be used to send binary data, like images or word documents, to the server.
- The data sent by GET method can be accessed using QUERY_STRING environment variable.
- The PHP provides **$_GET** associative array to access all the sent information using GET method.

Try out following example by putting the source code in test.php script.

```
<?php
  if( $_GET["name"] || $_GET["age"] ) {
     echo "Welcome ". $_GET['name']. "<br />";
     echo "You are ". $_GET['age']. " years old.";

     exit();
  }
?>
<html>
  <body>

     <form action = "<?php $_PHP_SELF ?>" method = "GET">
        Name: <input type = "text" name = "name" />
        Age: <input type = "text" name = "age" />
        <input type = "submit" />
     </form>

  </body>
</html>
```

THE POST METHOD

The POST method transfers information via HTTP headers. The information is encoded as described in case of GET method and put into a header called QUERY_STRING.

- The POST method does not have any restriction on data size to be sent.
- The POST method can be used to send ASCII as well as binary data.
- The data sent by POST method goes through HTTP header so security depends on HTTP protocol. By using Secure HTTP you can make sure that your information is secure.
- The PHP provides **$_POST** associative array to access all the sent information using POST method.

Try out following example by putting the source code in test.php script.

```
<?php
  if( $_POST["name"] || $_POST["age"] ) {
     if (preg_match("/[^A-Za-z'-]/",$_POST['name'] )) {
```

```
      die ("invalid name and name should be alpha");
    }
    echo "Welcome ". $_POST['name']. "<br />";
    echo "You are ". $_POST['age']. " years old.";

    exit();
  }
?>
<html>
  <body>

    <form action = "<?php $_PHP_SELF ?>" method = "POST">
      Name: <input type = "text" name = "name" />
      Age: <input type = "text" name = "age" />
      <input type = "submit" />
    </form>

  </body>
</html>
```

THE $_REQUEST VARIABLE

The PHP $_REQUEST variable contains the contents of both $_GET, $_POST, and $_COOKIE. We will discuss $_COOKIE variable when we will explain about cookies.

The PHP $_REQUEST variable can be used to get the result from form data sent with both the GET and POST methods.

Try out following example by putting the source code in test.php script.

```
<?php
  if( $_REQUEST["name"] || $_REQUEST["age"] ) {
    echo "Welcome ". $_REQUEST['name']. "<br />";
    echo "You are ". $_REQUEST['age']. " years old.";
    exit();
  }
?>
<html>
  <body>

    <form action = "<?php $_PHP_SELF ?>" method = "POST">
      Name: <input type = "text" name = "name" />
      Age: <input type = "text" name = "age" />
      <input type = "submit" />
    </form>

  </body>
</html>
```

Here $_PHP_SELF variable contains the name of self script in which it is being called.

FILES COMMANDS

INCLUDE AND REQUIRE STATEMENTS

It is possible to insert the content of one PHP file into another PHP file (before the server executes it), with the include or require statement.

- include will only produce a warning (E_WARNING) and the script will continue
- require will produce a fatal error (E_COMPILE_ERROR) and stop the script

Syntax:

include '*filename*';
or
require '*filename*';

WORKING WITH FILES IN PHP

To create, access, and manipulate files on your web server using the PHP file system functions.

OPENING A FILE WITH PHP FOPEN() FUNCTION

To work with a file you first need to open the file. The PHP fopen() function is used to open a file.

Syntax:

fopen(*filename*, *mode*)

Example:

```
<?php $file = "data.txt"; // Check the existence of file
if(file_exists($file))
{ // Attempt to open the file
$handle = fopen($file, "r");
}
Else
{ echo "ERROR: File does not exist."; }
 ?>
```

Note: check whether a file or directory exists or not before trying to access it, with the PHP file_exists() function.

Mode

Modes	What it does
R	Open the file for reading only.
r+	Open the file for reading and writing.
w	Open the file for writing only and clears the contents of file. If the file does not exist, PHP will attempt to create it.
w+	Open the file for reading and writing and clears the contents of file. If the file does not exist, PHP will attempt to create it.
a	Append. Opens the file for writing only. Preserves file content by writing to the end of the file. If the file does not exist, PHP will attempt to create it.
a+	Read/Append. Opens the file for reading and writing. Preserves file content by writing to the end of the file. If the file does not exist, PHP will attempt to create it.
x	Open the file for writing only. Return FALSE and generates an error if the file

Modes	What it does
	already exists. If the file does not exist, PHP will attempt to create it.
x+	Open the file for reading and writing; otherwise it has the same behavior as 'x'.

FCLOSE() FUNCTION:

Closing a File

```
<?php $file = "data.txt"; // Check the existence of file
if(file_exists($file))
{ // Open the file for reading
$handle = fopen($file, "r") or die("ERROR: Cannot open the file.");
/* Some code to be executed */
// Closing the file handle
        fclose($handle);
}
Else
{
        echo "ERROR: File does not exist.";
}
?>
```

FREAD():

The fread() function can be used to read a specified number of characters from a file.

Syntax:

```
fread(FILE HANDLE, LENGTH IN BYTES)
```

Example

```
<?php $file = "data.txt";
        if(file_exists($file))
        { $handle = fopen($file, "r") or die("ERROR: Cannot open the file.");
                $content = fread($handle, "20");
                 fclose($handle);
                 echo $content; }
     else{ echo "ERROR: File does not exist."; }
?>
```

Note:

1. we can use filesize($filename) at the place of 20 for size of the file in bytes for reading entire file.
2. readfile() function,allows you to read the contents of a file without needing to open it.

FGETS():

The fgets() function is used to read a single line from a file.

```
<?php
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");
echo fgets($myfile);
fclose($myfile);
?>
```

FEOF():

feof() function checks if the "end-of-file" (EOF) has been reached.

```
<?php
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");
// Output one line until end-of-file
while(!feof($myfile)) {
  echo fgets($myfile) . "<br>";
}
fclose($myfile);
?>
```

FWRITE():

The first parameter of fwrite() contains the name of the file to write to and the second parameter is the string to be written.

```
<?php
$myfile = fopen("newfile.txt", "w") or die("Unable to open file!");
$txt = "John Doe\n";
fwrite($myfile, $txt);
$txt = "Jane Doe\n";
fwrite($myfile, $txt);
fclose($myfile);
?>
```

RENAME():RENAME A FILE OR DIRECTORY

rename(Old filename, New file name)

FILE UPLOAD

You can upload any kind of file like images, videos, ZIP files, Microsoft Office documents, PDFs, as well as executables files and a wide range of other file types.

```
<html>
    <body>
    <?php
$target_path = "uploads/";
$target_path = $target_path.basename( $_FILES['fileToUpload']['name']);

if(move_uploaded_file($_FILES['fileToUpload']['tmp_name'], $target_path)) {
    echo "File uploaded successfully!";
} else{
    echo "Sorry, file not uploaded, please try again!";
}
?>
    <form  method="post" enctype="multipart/form-data">
    Select File:
    <input type="file" name="fileToUpload"/>
    <input type="submit" value="Upload Image" name="submit"/>
</form>
  </body>
</html>
```

Note:

Basename()->**function which is used to return the base name of a file**

$_FILES['fileToUpload']['name'])-> file name
$_FILES['fileToUpload']['type'])-> file type
$_FILES['fileToUpload']['size'])-> file size
$_FILES['fileToUpload']['tmp_name'])->file temporary name

FILE DOWNLOAD

```
<?php
$file = "logo_ldg.png"; //Let say If I put the file name Bang.png
echo "<a href='download.php?nama=".$file."'>download</a> ";

?>
```

Download.php

```
<?php
$filename = $_GET["nama"];
$contenttype = "application/force-download";
header("Content-Type: " . $contenttype);
header("Content-Disposition: attachment; filename=\"" . basename($filename) . "\";");
readfile("your file uploaded path".$filename);
exit();
?>
```

- Header: The header function sets the headers for an HTTP Response given by the server.
- Redirect page.
  It is used to redirect a from one web page to another web page in PHP.
  header('Location:give your url here');
- Set Content-Type in header response:
  PHP defaults to sending Content-Type:text/html.If we want to change the Content-Type
- Content-Disposition
  In a regular HTTP response, the Content-Disposition response header is a header indicating if the content is expected to be displayed INLINE in the browser, that is, as a Web page or as part of a Web page, or as an ATTACHMENT, that is downloaded and saved locally.

WORKING WITH DIRECTORIES

We can open a directory and read its contents, create or delete a directory, list all files in the directory, and so on.

MKDIR(): CREATING A NEW DIRECTORY

You can create a new and empty directory by calling the mkdir() function with the path and name of the directory to be created,

```
<?php // The directory path
 $dir = "testdir"; // Check the existence of directory
if(!file_exists($dir)){ // Attempt to create directory
        if(mkdir($dir))
        { echo "Directory created successfully."; }
        Else
        { echo "ERROR: Directory could not be created."; }
}
```

```
Else
{ echo "ERROR: Directory already exists."; }
 ?>
```

COPY(SOURCE,DESTINATION): COPYING FILES FROM ONE LOCATION TO ANOTHER

```
<?php // Source file path
$file = "example.txt"; // Destination file path
$newfile = "backup/example.txt"; // Check the existence of file
 if(file_exists($file))
{ // Attempt to copy file
if(copy($file, $newfile))
{ echo "File copied successfully."; }
Else
{ echo "ERROR: File could not be copied."; }
}
else{ echo "ERROR: File does not exist.";
 } ?>
```

SCANDIR(): LISTING ALL FILES IN A DIRECTORY

scandir() function to list files and directories inside the specified path.

```
<?php
// Define a function to output files in a directory
function outputFiles($path){
    // Check directory exists or not
    if(file_exists($path) && is_dir($path)){
        // Scan the files in this directory
        $result = scandir($path);

        // Filter out the current (.) and parent (..) directories
        $files = array_diff($result, array('.', '..'));

        if(count($files) > 0){
            // Loop through retuned array
            foreach($files as $file){
                if(is_file("$path/$file")){
                    // Display filename
                    echo $file . "<br>";
                } else if(is_dir("$path/$file")){
                    // Recursively call the function if directories found
                    outputFiles("$path/$file");
                }
            }
        } else{
            echo "ERROR: No files found in the directory.";
        }
    } else {
        echo "ERROR: The directory does not exist.";
    }
```

```
}

// Call the function
outputFiles("testdir");
?>
```

Note: glob() function, which matches files based on the pattern.

COOKIES

A cookie is a small text file that lets you store a small amount of data (nearly 4KB) on the user's computer. They are typically used to keeping track of information such as username that the site can retrieve to personalize the page when user visit the website next time.

SET COOKIES: USE TO SET COOKIES

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

Here,

name	The name of the cookie.
value	The value of the cookie. Do not store sensitive information
expires	The expiry date. After this time cookie will become inaccessible. The default value is 0.
path	Specify the path on the server for which the cookie will be available. If set to /, the cookie will be available within the entire domain.
domain	Specify the domain for which the cookie is available to e.g www.example.com.
secure	This field, if present, indicates that the cookie should be sent only if a secure HTTPS connection exists.

Example: create a cookie named username and assign the value value John Carter to it. It also specify that the cookie will expire after 30 days (30 days * 24 hours * 60 min * 60 sec).

```
setcookie("username", "John Carter", time()+30*24*60*60);
```

ACCESS COOKIES:

$_COOKIE superglobal variable is used to retrieve a cookie value.

Example:

```
echo $_COOKIE["username"];
```

REMOVING COOKIES:

Delete a cookie by calling the same setcookie() function with the cookie name and any value or set the expiration date in the past.

Example:

```
setcookie("username", "", time()-3600);
```

Example:

```
<?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>
```

SESSIONS

session stores data on the server rather than user's computer. In a session based environment, every user is identified through a unique number called session identifier or SID. Default session expired after 24 minutes which can be modify by changing in php.ini file.
Which resolve two problems in cookies:

1) Store data using cookies but it has some security issues. Since cookies are stored on user's computer it is possible for an attacker to easily modify a cookie content to insert potentially harmful data in your application that might break your application.
2) Also every time the browser requests a URL to the server, all the cookie data for a website is automatically sent to the server within the request. It means if you have stored 5 cookies on user's system, each having 4KB in size, the browser needs to upload 20KB of data each time the user views a page, which can affect your site's performance.

START SESSION:

store any information in session variables, you must first start up the session.
Example:
session_start();

STORING SESSION DATA

store all your session data as key-value pairs in the $_SESSION[] superglobal array. The stored data can be accessed during lifetime of a session.
Syntax:
$_SESSION["sessionname"] = value;

Example:
$_SESSION["firstname"] = "Peter";

ACCESSING SESSION DATA

To access the session data we set on our previous example from any other page on the same web domain

echo 'Hi, ' . $_SESSION["firstname"];

DESTROYING A SESSION

If you want to remove certain session data
unset($_SESSION[“session name”]);
Example:
unset($_SESSION["firstname"])
Note:
To destroy a session completely, simply call the session_destroy() function.

```
<?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>
```

SEND EMAIL

Using mail() function for creating and sending email messages to one or more recipients
Syntax:
mail(*to*, SUBJECT, MESSAGE, HEADERS, PARAMETERS)

To	The recipient's email address.
Subject	Subject of the email to be sent. This parameter i.e. the subject line cannot contain any newline character (\n).
Message	Defines the message to be sent. Each line should be separated with a line feed-LF (\n). Lines should not exceed 70 characters.
Headers (optional)	This is typically used to add extra headers such as "From", "Cc", "Bcc". The additional headers should be separated with a carriage return plus a line feed-CRLF (\r\n).
Parameters (optional)	Used to pass additional parameters.

Example:

```
<?php $to = 'maryjane@email.com';
 $subject = 'Marriage Proposal';
 $message = 'Hi Jane, will you marry me?';
 $from = 'peterparker@email.com'; // Sending email
if(mail($to, $subject, $message))
{ echo 'Your mail has been sent successfully.'; }
Else
{ echo 'Unable to send email. Please try again.'; }
?>
```

FORMS CREATING

The HTML <form> tag is used for creating a form for user input. A form can contain textfields, checkboxes, radio-buttons and more. Forms are used to pass user-data to a specified URL.

Attribute:

action: Specifies where to send the form-data when a form is submitted. Value is URL.

method: Specifies the HTTP method to use when sending form-data. It will be get or post.

name:Specifies the name of a form in text formate.

target:Specifies where to display the response that is received after submitting the form. It will be _blank, _self,_parent, _top.

Element:

The <form> element can contain one or more of the following form elements:

<input>
<textarea>
<button>
<select>
<option>
<optgroup>
<fieldset>
<legend>
<label>

[1] < INPUT >

The <input> tag specifies an input field where the user can enter data. <input> elements are used within a <form> element to declare input controls that allow users to input data. An input field can vary in many ways, depending on the type attribute.

Attribute:

value: Specifies the value of an <input> element. It may be a *text*.

name:Specifies the name of an <input> element. It may be a text.

type: Specifies the type <input> element to display . it will be button, checkbox, color, date, datetime, email, file, hidden, image, month, number, password, radio, range, reset, search, submit, tel, text, time, url, week.

object	description
checkbox	Checkboxes are used when you want the user to select one or more options of a limited number of choices
color	used for input fields that should contain a color.
date	used for input fields that should contain a date.
Datetime	used for input fields that should contain a date and time.
Email	used for input fields that should contain a email
File	used for input fields that should contain a file.
Hidden	used for input fields that should contain hidden field
Image	used for input fields that should contain image

monthuse for input fields that should contain a month and year.
number use for input fields that should contain a numeric value
password use for input fields that should contain password.
radio use when you want the user to select one of a limited number of choices.
Rangeuse for input fields that should contain value with in a range.
Reset used for input fields that should reset the value.
search used for search fields (a search field behaves like a regular text field).
Submit used for input fields that should submit the value.
text used when you want the user to type letters, numbers, etc. in a form.
Tel used for input fields that should contain a telephone number.
Time used for input fields that should contain a time.
url used for input fields that should contain a url.
week used for input fields that should contain a week and year.

[2] <TEXTAREA>

An input that allows a large amount of text to be entered, and allows the height of input box to be a specified unlike the standard input tag.

Attribute:
name - The unique name assigned to the form field.
rows - The number of rows of text, defines the vertical size of the text area.
cols - The horizontal size of the text box, defined as the number of characters (i.e. columns)

[3] <BUTTON>

A form button is similar to other form inputs

attributes:
name - Unique name for the button to be used by the action script.
type - The button type, either submit or reset, determines whether the form is to be submitted or cleared upon pressing it.
value - Text that appears on the button, such as OK or Submit.
size - Determines the length (or width) of the button.

[4] <SELECT>

A drop-down list, also refered to as a combo-box, allowing a selection to be made from a list of items.
Attribute:
name - Selector name
size - The minimum size (width) of the selection list, usually not required as the size of the items will define the list size.
multiple - Allows a user to select multiple items from the list, normally limited to one.

[5] <OPTION>

An option tag is needed for each item in the list, and must appear within the select tags. The text to be shown for the option must appear between the option tags.

Attribute:

value - The value is the data sent to the action script with the option is selected. This is not the text that appears in the list.

selected - Sets the default option that is automatically selected when the form is shown.

[6] <OPTGROUP>

The <optgroup> is used to group related options in a drop-down list. If you have a long list of options, groups of related options are easier to handle for a user.

Attribute:

label: Specifies a label for an option-group. It is a text.

[7] <FIELDSET>

used to group related elements in a form.

Attribute:

disabled: Specifies that a group of related form elements should be disabled.

form: Specifies one or more forms the fieldset belongs to.

name: Specifies a name for the fieldset.

[8] <LEGEND>

The <legend> tag defines a caption for the <fieldset> element.

Example <fieldset>

```
<legend style="float:right">Personalia:</legend>
<label for="fname">First name:</label>
<input type="text" id="fname" name="fname"><br>
<input type="submit" value="Submit">
</fieldset>
```

[9] <LABEL>

The <label> tag defines a label for an <input> element. The for attribute of the <label> tag should be equal to the id attribute of the related element to bind them together.

Attribute:

for: Specifies which form element a label is bound to

Example:

```
<html>
  <body>
 <form name="input" action="test.html" method="get">
 User Login: <input type="text" name="user" ><br>
Password: <input type="password" name="pass">
<br>
```

```
Address:<textarea>default null</textarea><br>
Gender: <input type="radio" name="gen" value="male">
<input type="radio" name="gen" value="female" checked><br>
Language:<input type="checkbox" name="language" value="spanish"> I speak Spanish<br>
<input type="checkbox" name="language" value="french"> I speak French <br>
College
<select name="college">
<option value="v">vvk</option>
<option value="p">pts</option>
<option value="d">drb</option>
</select><br>
<input type="submit" value="Submit">
</form>
</body>
</html>
```

VALIDATION OF FORMS

An HTML form contains various input fields such as text box, checkbox, radio buttons, submit button, and checklist, etc. These input fields need to be validated, which ensures that the user has entered information in all the required fields and also validates that the information provided by the user is valid and correct.
It may be :Empty String, Validate String, Validate Numbers, Validate Email, Validate URL, Input length and so on:
It will manage using Php filters & regular expressions
Php filter:
To validate data using filter extension you need to use the PHP's filter_var() function.

FUNCTIONS AND FILTERS

To filter available, use one of the following filter function:
filter_var()- Filter a single variable with a specified filter.
filter_var_array()- Filter several variables with the same or different filters.
filter_input()- Get one input variable and filter it.
filter_input_array – Get several input variables and filter them with the same or different filters.

Syntax: filter_var(*variable*, *filter*, *options*)

The first parameter is the value to be filtered, the second parameter is the ID of the filter to apply, and the third parameter is the array of options related to filter.

ID	Description
FILTER_VALIDATE_BOOLEAN, **FILTER_VALIDATE_BOOL**	Returns true for "1", "true", "on" and "yes". Returns false otherwise. If FILTER_NULL_ON_FAILURE is set, false is returned only for "0", "false", "off", "no", and "", and null is returned for all non-boolean values.
FILTER_VALIDATE_DOMAIN	Validates whether the domain name label lengths are valid.

ID	Description
FILTER_VALIDATE_EMAIL	Validates whether the value is a valid e-mail address.
FILTER_VALIDATE_FLOAT	Validates value as float, optionally from the specified range, and converts to float on success.
FILTER_VALIDATE_INT	Validates value as integer, optionally from the specified range, and converts to int on success.
FILTER_VALIDATE_IP	Validates value as IP address, optionally only IPv4 or IPv6 or not from private or reserved ranges.
FILTER_VALIDATE_MAC	Validates value as MAC address.
FILTER_VALIDATE_REGEXP	Validates value against regexp, a Perl-compatible regular expression.
FILTER_VALIDATE_URL	Validates value as URL

Example: We validate an integer using the **filter_var()** function:

```
<?php
$int=1234;
if(!filter_var($int,FILTER_VALIDATE_INT))
{
echo "Integer is not valid";
} else {
echo "Integer is valid";
}
```

REGULAR EXPRESSION:

A regular expression is a sequence of characters that forms a search pattern. A regular expression can be a single character, or a more complicated pattern.

Regular Expression Functions

Function	Description	
preg_match()	Returns 1 if the pattern was found in the string and 0 if not	$str = "I LIKE apple."; $pattern = "/like/i"; echo preg_match($pattern, $str); // Outputs 1
preg_match_all()	Returns the number of times the pattern was found in the string, which may also be 0	$str = "It is an apple.I like apple"; $pattern = "/apple/"; echo preg_match_all($pattern, $str); // Outputs 2
preg_replace()	Returns a new string where matched patterns have been replaced with another string	$str = " It is an apple.I like apple "; $pattern = "/apple/i"; echo preg_replace($pattern, "orange", $str); // Outputs " It is an orange.I like orange "

Regular Expression Patterns

Expression	Description
[abc]	Find one character from the options between the brackets
[^abc]	Find any character NOT between the brackets
[0-9]	Find one character from the range 0 to 9

Metacharacters

Metacharacter	Description
|	Find a match for any one of the patterns separated by | as in: cat|dog|fish
.	Find just one instance of any character
^	Finds a match as the beginning of a string as in: ^Hello
$	Finds a match at the end of the string as in: World$
\d	Find a digit
\s	Find a whitespace character
\b	Find a match at the beginning of a word like this: \bWORD, or at the end of a word like this: WORD\b
\uxxxx	Find the Unicode character specified by the hexadecimal number xxxx

Quantifiers

Quantifier {quantities}	Description
n+	Matches any string that contains at least one *n*
n*	Matches any string that contains zero or more occurrences of *n*
n?	Matches any string that contains zero or one occurrences of *n*
n{x}	Matches any string that contains a sequence of *X* *n*'s
n{x,y}	Matches any string that contains a sequence of X to Y *n*'s
n{x,}	Matches any string that contains a sequence of at least X *n*'s

EMPTY STRING

Checks that the field is not empty. If the user leaves the required field empty, it will show an error message.

```
if (empty ($_POST["name"])) {
   $errMsg = "Error! You didn't enter the Name.";
         echo $errMsg;
} else {
   $name = $_POST["name"];
}
```

VALIDATE STRING

checks that the field will contain only alphabets and whitespace, for example - name. If the name field does not receive valid input from the user

```
$name = $_POST ["Name"];
if (!preg_match ("/^[a-zA-z]*$/", $name) ) {
   $ErrMsg = "Only alphabets and whitespace are allowed.";
         echo $ErrMsg;
} else {
   echo $name;
}
```

VALIDATE NUMBER

the field will only contain a numeric value.

For example - Mobile no

```
$mobileno = $_POST ["Mobile_no"];
if (!preg_match ("/^[0-9]*$/", $mobileno) ){
   $ErrMsg = "Only numeric value is allowed.";
   echo $ErrMsg;
} else {
   echo $mobileno;
}
```

VALIDATE EMAIL

A valid email must contain @ and . symbols

```
$email = $_POST ["Email"];
$pattern = "^[_a-z0-9-]+(\.[_a-z0-9-]+)*@[a-z0-9-]+(\.[a-z0-9-]+)*(\.[a-z]{2,3})$^";
if (!preg_match ($pattern, $email) ){
   $ErrMsg = "Email is not valid.";
        echo $ErrMsg;
} else {
   echo "Your valid email address is: " .$email;
}
```

INPUT LENGTH VALIDATION

The input length validation restricts the user to provide the value between the specified range, for Example - Mobile Number. A valid mobile number must have 10 digits.

```
$mobileno = strlen ($_POST ["Mobile"]);
```

```
$length = strlen ($mobileno);

if ( $length < 10 && $length > 10) {
    $ErrMsg = "Mobile must have 10 digits.";
          echo $ErrMsg;
} else {
    echo "Your Mobile number is: " .$mobileno;
}
```

VALIDATE URL

code validates the URL of website provided by the user via HTML form

```
$websiteURL = $_POST["website"];
if (!preg_match("/\b(?:(?:https?|ftp):\/\/|www\.)[-a-z0-9+&@#\/%?=~_|!:,.;]*[-a-z0-
9+&@#\/%=~_|]/i",$website)) {
  $websiteErr = "URL is not valid";
  echo $websiteErr;
} else {
    echo "Website URL is: " .$websiteURL;
}
```

BUTTON CLICK VALIDATE

Code validates that the user click on submit button and send the form data to the server one of the following method - get or post.

```
if (isset ($_POST['submit']) {
    echo "Submit button is clicked.";
    if ($_SERVER["REQUEST_METHOD"] == "POST") {
        echo "Data is sent using POST method ";
    }
} else {
    echo "Data is not submitted";
}
```

CLASSES AND OBJECTS IN PHP

Classes and objects are the two main aspects of object-oriented programming.

Class : Fruit

Objects: Apple,Banana,Mango

a class is a template for objects, and an object is an instance of a class.

DEFINE A CLASS

A class is defined by using the class keyword, followed by the name of the class and a pair of curly braces ({}).

SYNTAX

```
<?php
class classname {

Variables/properties declaration

Methods
  // code goes here...
}
?>
```

Example

```
<?php
class Fruit {
  // Properties
  public $name;
  public $color;

  // Methods
  function set_name($name) {
    $this->name = $name;
  }
  function get_name() {
    return $this->name;
  }
}
?>
```

DEFINE OBJECTS

Classes are nothing without objects! We can create multiple objects from a class. Each object has all the properties and methods defined in the class, but they will have different property values.

Objects of a class is created using the new keyword.

Syntax:

```
objectname = new classname;
```

Example:

```
<?php
class Fruit {
  // Properties
  public $name;
  public $color;
```

```
  // Methods
  function set_name($name) {
    $this->name = $name;
  }
  function get_name() {
    return $this->name;
  }
}

$a = new Fruit();
$b = new Fruit();
$a->set_name('Apple');
$b->set_name('Banana');

echo $a->get_name();
echo "<br>";
echo $b->get_name();
?>
```

WHAT IS AN EXCEPTION?

An exception is an object that describes an error or unexpected behaviour of a PHP script.

Exceptions are thrown by many PHP functions and classes.

THE TRY...CATCH... STATEMENT

The try...catch statement to catch exceptions and continue the process..

SYNTAX

```
try

{
  code that can throw exceptions
}

catch(Exception $e)

{
  code that runs when an exception is caught
}
```

User defined functions and classes can also throw exceptions.

THROWING AN EXCEPTION

The throw statement allows a user defined function or method to throw an exception. When an exception is thrown, the code following it will not be executed.

If an exception is not caught, a fatal error will occur with an "Uncaught Exception" message.

SYNTAX:

```
throw new Exception("msg");
```

EXAMPLE 1:

```
<?php
function division($x,$y)
{ if ($x/$y==0)
        throw new exception("Answer is invalid.Division by zero");
        else
        return $x/$y;
}
try
{echo division(5,0);
}
catch(Exception $e)
{ echo $e->getMessage();
}
?>
```

Example 2:

```
<?php
function divide($dividend, $divisor) {
 if($divisor == 0) {
   throw new Exception("Division by zero", 1);
 }
 return $dividend / $divisor;
}

try

{
 echo divide(5, 0);
}

catch(Exception $ex)

{
 $code = $ex->getCode();
 $message = $ex->getMessage();
 $file = $ex->getFile();
 $line = $ex->getLine();
 echo "Exception thrown in $file on line $line: [Code $code]
 $message";

}

?>
```

WHAT IS JSON

JSON stands for **J**ava**S**cript **O**bject **N**otation. JSON is a standard lightweight data-interchange format which is quick and easy to parse and generate.

JSON, like XML, is a text-based format that's easy to write and easy to understand for both humans and computers, but unlike XML, JSON data structures occupy less bandwidth than their XML versions. JSON is based on two basic structures:

Object: This is defined as a collection of key/value pairs (i.e. `key:value`). Each object begins with a left curly bracket `{` and ends with a right curly bracket `}`. Multiple key/value pairs are separated by a comma `,`.

Array: This is defined as an ordered list of values. An array begins with a left bracket `[` and ends with a right bracket `]`. Values are separated by a comma `,`.

In JSON, keys are always strings, while the value can be
a **string**, **number**, **true** or **false**, **null** or even an **object** or an **array**. Strings must be enclosed in double quotes `"` and can contain escape characters such as `\n`, `\t` and `\`.

JSON object:

```
{
   "book": {
      "name": "Harry Potter and the Goblet of Fire",
      "author": "J. K. Rowling",
      "year": 2000,
      "genre": "Fantasy Fiction",
      "bestseller": true
   }
}
```

JSON array:

```
{
   "fruits": [
      "Apple",
      "Banana",
      "Strawberry",
      "Mango"
   ]
```

```
}
```

PARSING JSON WITH PHP

JSON data structures are very similar to PHP arrays. PHP has built-in functions to encode and decode JSON data. These functions are json_encode() and json_decode(), respectively. Both functions only works with UTF-8(Unicode Transformation Format - 8 bits) encoded (this code use for electronic communication in XML/Json/HTML5) string data.

ENCODING JSON DATA IN PHP

In PHP the json_encode() function is used to encode a value to JSON format. The value being encoded can be any PHP data type except a resource, like a database or file handle.

Example to encode a PHP associative array into a JSON object:

```
<?php

// An associative array

$marks = array("Peter"=>65, "Harry"=>80, "John"=>78, "Clark"=>90);


echo json_encode($marks);

?>
```

Output

```
{"Peter":65,"Harry":80,"John":78,"Clark":90}
```

Similarly, you can encode the PHP indexed array into a JSON array, like this:

EXAMPLE

```
<?php

// An indexed array

$colors = array("Red", "Green", "Blue", "Orange", "Yellow");


echo json_encode($colors);

?>
```

Output:

["Red","Green","Blue","Orange","Yellow"]

Note: force json_encode() function to return an PHP indexed array as JSON object by using the JSON_FORCE_OBJECT option, as shown in the example below:

```
<?php

// An indexed array

$colors = array("Red", "Green", "Blue", "Orange");


echo json_encode($colors, JSON_FORCE_OBJECT);

?>
```

The output of the above example will look like this:

{"0":"Red","1":"Green","2":"Blue","3":"Orange"}

As you can see in the above examples a non-associative array can be encoded as array or object. However, an associative array always encoded as object.

Note: error may be occurs in JSON_FORCE_OBJECT due lower version of php.

DECODING JSON DATA IN PHP

Decoding JSON data is as simple as encoding it. You can use the PHP json_decode() function to convert the JSON encoded string into appropriate PHP data type.

Example to decode or convert a JSON object to PHP object.

```
<?php

// Store JSON data in a PHP variable

$json = '{"Peter":65,"Harry":80,"John":78,"Clark":90}';


var_dump(json_decode($json));

?>
```

Output:

object(stdClass)#1 (4) { ["Peter"]=> int(65) ["Harry"]=> int(80) ["John"]=> int(78) ["Clark"]=> int(90) }

By default the json_decode() function returns an object. However, you can optionally specify a second parameter **$assoc** which accepts a boolean value that when set as true JSON objects are decoded into associative arrays. It is false by default.

Example to decode or convert a JSON object to PHP array.

```
<?php
// Store JSON data in a PHP variable
$json = '{"Peter":65,"Harry":80,"John":78,"Clark":90}';


var_dump(json_decode($json, true));
?>
```

Output

```
array(4) { ["Peter"]=> int(65) ["Harry"]=> int(80) ["John"]=> int(78) ["Clark"]=> int(90) }
```

Now let's check out an example that will show you how to decode the JSON data and access individual elements of the JSON object or array in PHP.

```
<?php
// Assign JSON encoded string to a PHP variable
$json = '{"Peter":65,"Harry":80,"John":78,"Clark":90}';


// Decode JSON data to PHP associative array
$arr = json_decode($json, true);
// Access values from the associative array
echo $arr["Peter"];  // Output: 65
echo $arr["Harry"];  // Output: 80
echo $arr["John"];   // Output: 78
echo $arr["Clark"];  // Output: 90


// Decode JSON data to PHP object
$obj = json_decode($json);
// Access values from the returned object
echo $obj->Peter;   // Output: 65
echo $obj->Harry;   // Output: 80
```

```
echo $obj->John;    // Output: 78
echo $obj->Clark;   // Output: 90
?>
```

MYSQL INTRODUCTION

- Mysql is the most popular database system.
- Mysql is freely available and widely used with PHP scripts to create power full anddynamic server side application.
- It is easy to install on a wide range Os include one windows, mac and unix.
- It simple to use and includes some handling administration tools.
- It is fast ,powerful ,client-server system.
- It is SQL compatible.
- A data in Mysql is stored in database object called tables.
- Table is a collection of related data entries and consist of rows and columns.

DATA TYPES OF MYSQL (FIELD TYPE)

- The following is a list of datatypes available in MySQL, which includes
  1. STRING
  2. **Numeric**
  3. DATE/TIME

STRING DATA TYPES

- The following are the **String Data Types** in MySQL:

Data Type Syntax	**Maximum Size**	**Explanation**
CHAR(*size*)	Maximum size of 255 characters.	Where ***size*** is the number of characters to store. Fixed-length strings. Space padded on right to equal ***size*** characters.
VARCHAR(*size*)	Maximum size of 255 characters.	Where ***size*** is the number of characters to store. Variable-length string.
TINYTEXT(*size*)	Maximum size of 255 characters.	Where ***size*** is the number of characters to store.
TEXT(*size*)	Maximum size of 65,535characters.	Where ***size*** is the number of characters to store.
MEDIUMTEXT(*size*)	Maximum size of 16,777,215 characters.	Where ***size*** is the number of characters to store.
LONGTEXT(*size*)	Maximum size of 4GB or4,294,967,295 characters.	Where ***size*** is the number of characters to store.

BINARY(*size*)	Maximum size of 255 characters.	Where ***size*** is the number of binary characters to store. Fixed-length strings. Space padded on right to equal ***size*** characters
VARBINARY(*size*)	Maximum size of 255 characters.	Where ***size*** is the number of characters to store. Variable-length string.
BLOB	Maximum size of 65,535bytes.	Normal sized BLOB.
MEDIUMBLOB	Maximum size of 16,777,215 bytes.	Medium sized BLOB.
LONGBLOB	Maximum size of 4,294,967,295 bytes.	Long sized BLOB.

NUMERIC DATA TYPES

- The following are the **Numeric Data types** in MySQL:

Data Type	**Maximum Size**	**Explanation**
BIT	Signed values range from -128 to 127. Unsigned values range from 0 to 255.	Very small integer value that isequivalent to TINYINT(1).
TINYINT(*m*)	Signed values range from -128 to 127. Unsigned values range from 0 to 255.	Very small integer value.
SMALLINT(*m*)	Signed values range from -32768 to 32767. Unsigned values range from 0 to 65535.	Small integer value.
MEDIUMINT(*m*)	Signed values range from -8388608 to 8388607. Unsigned values range from 0 to 16777215.	Medium integer value.
INT(*m*)	Signed values range from -2147483648 to 2147483647. Unsigned values range from 0 to 4294967295.	Standard integer value.
BIGINT(*m*)	Signed values range from -9223372036854775808 to 9223372036854775807. Unsigned values range from 0 to 18446744073709551615.	Big integer value.
DECIMAL(*m*,*d*)	Unpacked fixed point number. ***m*** defaults to 10, if not specified. ***d*** defaults to 0, if not specified.	Where ***m*** is the total digits and ***d*** is the number of digits after the decimal.
FLOAT(*m*,*d*)	Single precision floating point number.	Where ***m*** is the total digits and ***d*** is the number of digits after the decimal.

DOUBLE(*m*,*d*)	Double precision floating point number.	Where ***m*** is the total digits and ***d*** is the number of digits after the decimal.
FLOAT(p)	Floating point number.	Where ***p*** is the precision.
BOOL	Synonym for TINYINT(1)	Treated as a boolean data type where a value of 0 is considered to be FALSE and any other value is considered to be TRUE.
BOOLEAN	Synonym for TINYINT(1)	Treated as a boolean data type where a value of 0 is considered to be FALSE and any other value is considered to be TRUE.

DATE/TIME DATA TYPES

The following are the **Date/Time Data types** in MySQL:

Data Type Syntax	Maximum Size	Explanation
DATE	Values range from '1000-01-01' to '9999-12-31'.	Displayed as 'YYYY-MM-DD'.
DATETIME	Values range from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'.	Displayed as 'YYYY-MM-DD HH:MM:SS'.
TIMESTAMP(*m*)	Values range from '1970-01-01 00:00:01' UTC to '2038-01-19 03:14:07' UTC.	Displayed as 'YYYY-MM-DD HH:MM:SS'.
TIME	Values range from '-838:59:59' to '838:59:59'.	Displayed as 'HH:MM:SS'.
YEAR[(2|4)]	Year value as 2 digits or 4 digits.	Default is 4 digits.

DATATYPE ATTRIBUTES

1. AUTO_INCREMENT

- The auto increment attribute is used to assign unique integer identifier to newly inserted rows.
- It increments the field value by value when a new row is inserted.
- The auto increment mostly used with primary key.

2. DEFAULT

- The DEFAULT attribute ensures that some constant value will be assigned when no

othervalue is available.

- This value must be constant because MYSQL does not allow functional or expressional valuesto be inserted.
- If the NULL attribute has been assigned to this value the default value will be NULL and if nodefault is specified.

3. NULL

- The null attribute indicates that no value can exist for the given field .
- The NULL attribute is assigned to a field by Default.

4. NOT NULL

- If a column is defined as not null than one cannot insert a null value for this column.

5. PRIMARY KEY

- Keyword **PRIMARY KEY** is used to define a column as primary key.
- It is used guarantee uniqueness for a given row. one cannot insert null values for the primary key column.
- There are two other values to ensure a record uniqueness.
    - **Single-field primary keys:** are used when there is a non-modifiable unique identifier for each row entered into the database. They are never changed once set.
    - **Multi-field primary keys:** are useful when it is not possible to guarantee uniqueness from any single field with a record. Thus, multiple fields are joined to ensure uniqueness.

6. UNIQUE

- A column assigned the UNIQUE attribute will ensure that all values possess different values,except than NULL values are repeatable.

7. ZEROFILL

- It is available to any of the numeric type and will result in the replacement of all remainingfield space with zero.
- When used in conjunction with the optional (nonstandard) attribute ZEROFILL, the defaultpadding of spaces is replaced with zeros.
- For example, for a column declared as INT(4) ZEROFILL, a value of 5 is retrievedas 0005.The ZEROFILL attribute is ignored when a column is involved in expressions.

8. INDEX

- Indexing a column creates a sorted array of key for that column .each of which points to its corresponding table row.
- By creating index the searching will become faster.

9. BINARY

- The binary attribute is only used with char and varchar values when columns are assignedthis attributes they will be stored in case sensitive manner,(According to their ASCII value)

10. UNSIGNED: if specified, disallows negative values.

DATABASE FUNCTIONS

Function	Description
mysql_connect()	Opens a new connection to the MySQL server
mysql_close()	Closes a previously opened database connection
mysql_select_db()	Selects the default database for database queries
mysql_query()	Performs a query on the database
mysql_affected_rows()	Returns the number of affected rows in the previous MySQL operation
mysql_error()	Returns the last error message for the most recent MySQL function call
mysql_fetch_array()	Fetches a result row as an associative, a numeric array, or both
mysql_free_result()	Frees the memory associated with a result
mysql_num_rows()	Returns the number of rows in a result set
mysql_fetch_row()	Fetches one row of data from the result set and returns it as an enumerated array
mysql_fetch_assoc()	Fetches a result row as an associative array
mysql_result()	Retrieves the content of one cell(field)
mysql_fetch_object()	Returns row from a result set as an object

MYSQL_CONNECT()

- The mysql_connect() function opens a new connection to the MySQL server.
- It creates a connection to a MySQL server. This function takes three parameters and returns a MySQL link identifier on success or FALSE on failure.

SYNTAX:

resource mysql_connect(string hostname, string username, string password);

Parameter	Description
Hostname/server	Optional - The host name running database server. If not specified, then default value is localhost:3036.
Username	Optional - The username accessing the database. If not specified, then default is the name of the user that owns the server process.
Password	Optional - The password of the user accessing the database. If not specified, then default is an empty password.

EXAMPLE :

```
<?php
$conn = mysql_connect("localhost", "root", "") ordie("ERROR: Could not connect. ");
// ....some PHP code...
 ?>
```

MYSQL_CLOSE()

- The mysql_close() function is used to close an open MySQL connection.
- The link to the MySQL server is closed when the script is terminated.
- It takes one argument which is the resource of the database. If the connection is not specified as a parameter within the mysql_close(), the last opened link is used.
- If a resource parameter is not specified then last opened database is closed. This function returns true if it closes connection successfully otherwise it returns false.

SYNTAX:

bool mysql_close ([resource $link]);

Parameter	Description
Link	Required. Specifies the MySQL connection link to close .

EXAMPLE

```
<?PHP
    $conn=mysql_connect("localhost","root","") die("ERROR: Could not connect. ");
    // ....some PHP code...
    Mysql_close($conn)
    ?>
```

MYSQL_SELECT_DB()

- It is used to select a database.
- It returns true on success and false on failure ,the name of the database as an argumentand the resource name is optional.

SYNTAX:

bool mysql_select_db(String $databasename,[,resource $link]);

Parameter	Description
Databasename/db_name	Required - MySQL Database name to be select.
Link/connection	Optional - name of the mysql connection. If not specified, then last opened connection by mysql_connect() will be used.

EXAMPLE:

```
<?php
  $conn=mysql_connect("localhost","root"," ") or die("ERROR: Could not connect. ");
      mysql_select_db("Employee",$conn) or die("ERROR: Could not find database. ");

            // ....some PHP code...
     mysql_close($conn);
?>
```

MYSQL_QUERY()

- The mysql_query() function executes a query on a MySQL database connection.
- It sends the query to the currently active database on the server. It takes two arguments. one is the query and the other is the resource which is optional.
- This function returns the query handle for SELECT queries, TRUE/FALSE for other queries,or FALSE on failure.

SYNTAX:

resource mysql_query (String $query [, resource $link])

Parameter	Description
Query	Required. Specifies the SQL query to send (should not end with a semicolon)
Link/connection	Optional. Specifies the MySQL connection. If not specified, the lastconnection opened by mysql_connect() or mysql_connect() is used.

EXAMPLE:INSERT RECORD IN THE TABLE

```
<?php
    $link=mysql_connect("localhost","root","") or die("ERROR: Could not connect. ");
    //open connection
    mysql_select_db("Employee",$conn) or die("ERROR: Could not find database.");

    //selects a database Employee

    mysql_query("insert into emp_per values(101,'Raman')",$conn) or die("ERROR: Could
    not inserted. ");

    //insert in to emp_per table

    mysql_close($conn);//close a connection
```

EXAMPLE: UPDATE RECORD IN THE TABLE

```
<?php
        $conn=mysql_connect("localhost","root","") or die("ERROR: Could not
    connect. ");
        mysql_select_db("Employee",$conn) or die("ERROR: Could not find
    database. ");
        mysql_query("Update emp_per set name='Raj' where eno=101",$conn) or
    die("ERROR:      Could not updated. ");
    //Update in to emp_per table
    mysql_close($conn);
    //close a connection
?>
```

EXAMPLE:DELETE RECORD IN THE TABLE

```
<?php
    $conn=mysql_connect("localhost","root","") or die("ERROR: Could not connect. ");
    mysql_select_db("Employee",$conn) or die("ERROR: Could not find database. ");
    mysql_query("delete from emp_per where rollno=101",$conn) or die("ERROR: Could
    notdeleted. ");
    //delete from student table
    mysql_close($conn);
?>
```

MYSQL_NUM_ROWS

- This function Retrieves the number of rows from a result set. This command is only validfor statements like SELECT or SHOW that return an actual result set.
- Resultset will be the array that is returned by mysql_query() when used with select. Thisfunction returns False on failure.

SYNTAX:

int mysql_num_rows (resource result)

Example:

```
<?php
$conn=mysql_connect("localhost","root","") or die("ERROR: Couldnot connect.);
mysql_select_db("Employee",$conn) or die("ERROR: Could not find
                                    database. ");
```

```
$query="select * from emp_per";
$resultset=mysql_query($query,$conn);
$r=mysql_num_rows($resultset);
echo "<br>Rows:    ".$r;
//$r contains total no. of rows in a resultset
mysql_close($conn);
?>
```

This command is only valid for statements like SELECT that return an actual statment.To retrieve the number of rows affected by a INSERT, UPDATE, REPLACE or DELETE query,use mysql_affected_rows()

MYSQL_AFFECTED_ROWS()

- The mysql_affected_rows() function is used to get the number of affected rows by the last MySQL query.
- If you run a mysql query to insert, update, replace or delete records, and want to know how many records are being affected by that query, you have to use mysql_affected_rows().
- This function returns the number of affected rows on success, or -1 if the last operation failed.

SYNTAX:

int mysql_affected_rows ([resource $link])

Parameter	Description
Link/connection	Optional. Specifies the MySQL connection. If not specified, the lastconnection opened by mysql_connect() is used.

EXAMPLE:

```
<?php
$link=mysql_connect("localhost","root","") or die("ERROR: Could not connect.
");
mysql_select_db("Employee",$conn) or die(mysql_error());
mysql_query("delete from emp_per where  rollno=101",$conn) or die(mysql_error());
$r=mysql_affected_rows();
echo "<br>Deleted Records Are:    ".$r;
 //Prints total no. of rows deleted
```

```
mysql_close($link);
//close a connection
?>
```

MYSQL_FETCH_ARRAY()

- This function fetches rows from the mysql_query() function and returns an array on success .or false on failure or when there are no more rows.
- Array may be an associative array, a numeric array, or both.
- It moves the internal data pointer ahead.

SYNTAX:

array mysql_fetch_array (resource $resultset [, int $result_type=MYSQL_BOTH])

Name	Description
resultset	Refers to the resource return by a valid mysql query (calling by mysql_query() function).
resultset_type	The type of the result is an array. Possible values :
	MYSQL_ASSOC - Associative array MYSQL_NUM - Numeric array MYSQL_BOTH - Both associative and numeric array Default : MYSQL_BOTH

EXAMPLE: DISPLAY ALL RECORDS

```
<?php
    $link=mysql_connect("localhost","root","") or die("ERROR: Could not connect. ");
    mysql_select_db("college",$conn) or die("ERROR: Could not find database. ");
    $resultset=mysql_query("select * from emp_per",$conn) or die(mysql_error());
    while($row=mysql_fetch_array($resultset))
    {
        echo $row[0]."-".$row[1]."<br>";
    }
    mysql_free_result($resultset); mysql_close($conn);//close a connection
?>
```

MYSQL_FREE_RESULT()

- The mysql_free_result() function frees memory used by a resultset handle. mysql_free_result() only needs to be called if you are concerned about how much memory is being used for queries

that return large result sets.

- All associated result memory is automatically freed at the end of the script's execution. This function returns TRUE on success, or FALSE on failure.

SYNTAX:

```
bool mysql_free_result ( resource $resultset )
```

Example:

```
<?php
    $conn=mysql_connect("localhost","root","") or die("ERROR: Could not
    connect. ");

    mysql_select_db("Employee",$conn) or die("ERROR: Could not find database.
    ");
    $resultset=mysql_query("select * from emp_per",$conn) or die(mysql_error());
    while($row=mysql_fetch_array($resultset))
    {
        echo $row[0]."-".$row[1]."<br>";
    }
    mysql_free_result($resultset);
    mysql_close($conn);
?>
```

MYSQL_RESULT()

- The mysql_result() function returns the value of a field in a result set. This functionreturns the field value on success, or FALSE on failure.
- Iit requires three arguments: first is the result set, second will be the row number and third is optional argument which is the field. Its default value is zero.

SYNTAX:

String mysql_result (resource $resultset, int $row [, mixed $field=0])

Name	Description
resultset	Refers to the resource return by a valid mysql query (calling by mysql_query()).
Row	The row number from the result that's being retrieved. Row numbers start at 0.
Field	The name or position of the field being retrieved.

EXAMPLE:

```
<?php
    $conn=mysql_connect("localhost","root","") or die("ERROR: Could not
    connect. ");
    mysql_select_db("Employee",$conn) or die("ERROR: Could not find database.
    ");
    $resultset=mysql_query("select * from emp_per",$link) or die(mysql_error());
    echo mysql_result($resultset,2);//Outputs third student's rollno
    echo mysql_result($resultset,2,1);//Outputs third student's name
    mysql_close($conn);//close a connection
?>
```

MYSQL_ERROR()

- The mysql_error() function returns the error description of the last MySQL operation.
- This function returns an empty string ("") if no error occurs.

SYNTAX:

String mysql_error ([resource $link])

Parameter	Description
$link	Optional. Specifies the MySQL connection. If not specified, the lastconnection opened by mysql_connect()is used.

EXAMPLE:

```
<?php
    $conn=mysql_connect("localhost","root","") or die("ERROR: Could not
    connect. ");

    if(!$conn)
    {
```

```
        die(mysql_error());
    }
    mysql_close($conn);
?>
```

MYSQL_FETCH_ROW()

- The mysql_fetch_row() function returns a row from a resultset as a numeric array.
- This function gets a row from the mysql_query() function and returns an array onsuccess, or FALSE on failure or when there are no more rows.

SYNTAX:

array mysql_fetch_row (resource $resultset)

Name	Description
resultset	Refers to the resource return by a valid mysql query (calling by mysql_query() function).

EXAMPLE:

```
<?php
    $link=mysql_connect("localhost","root","") or die("ERROR: Could not connect. ");
    mysql_select_db("Employee",$conn)  or  die("ERROR:Couldnot
                                find database.  ");
    $resultset=mysql_query("select * from emp_per",$conn) or die(mysql_error());
    while($row=mysql_fetch_row($resultset)
    {
        echo $row[0]; //returns 1st record's first column value
        echo $row[1]; //returns 1st record's second column value
    }
    mysql_close($conn);//close a connection
?>
```

MYSQL_FETCH_OBJECT()

- The mysql_fetch_object() function returns a row from a resultset as an object.
- This function gets a row from the mysql_query() function and returns an object on success, or FALSE on failure or when there are no more rows.

SYNTAX:

object mysql_fetch_object (resource $resultset)

Name	Description
Resultset	Refers to the resource return by a valid mysql query (calling by mysql_query() function).

EXAMPLE:

```
<?php
        $conn=mysql_connect("localhost","root","") or die("ERROR: Could not
        connect. ");
        mysql_select_db("Employee",$conn) or die("ERROR: Could not find database.
        ");
        $query="select * from emp_per";
        $resultset=mysql_query($query,$conn);
        $while($row=mysql_fetch_object($resultset)
        {
                echo $row->rollno."<br/>";
        }
        mysql_close($conn);
?>
```

MYSQL_FETCH_ASSOC()

- The mysql_fetch_assoc() used to retrieve a row of data as an associative array from a MySQL result handle.
- It Returns an associative array that corresponds to the fetched row and moves the internal pointer ahead, or FALSE if there are no more rows.

SYNTAX:

array mysql_fetch_assoc (resource $resultset)

Name	Description
resultset	Refers to the resource return by a valid mysql query (calling by mysql_query() function).

EXAMPLE:

```
<?php
        $conn=mysql_connect("localhost","root","") or die("ERROR: Could not
        connect. ");
        mysql_select_db("Employee",$conn) or die("ERROR: Could not find database.
        ");
        $resultset=mysql_query("select * from emp_per",$link) or die(mysql_error());
        while($row=mysql_fetch_assoc($resultset))
        {
                echo $row['rollno']."<br>";
                echo $row['name']."<br>";
        }
```

```
    mysql_free_result($resultset);
    mysql_close($conn);
?>
```

mysql_num_fields()

- This function returns number of fields in result set or success , or FALSE on failure.

SYNTAX:

int mysql_num_fields (resource $resultset)

IT EXAMPLE:

```
<?php
    $conn=mysql_connect("localhost","root","") or die("ERROR: Could not
    connect. ");
    mysql_select_db("Employee",$conn) or die("ERROR: Could not find database.
    ");
    $resultset=mysql_query("select * from emp_per",$link) or die(mysql_error());
    echo "Total fields are:".mysql_num_fields($resultset);
    mysql_close($conn);
?>
```

ORDER BY

- The order by keyword is used to sort the data in result set.
- The order by keyword sorts the record in ascending order by default in order to sort record is descending order you can use DESC keyword.

SYNTAX:

SELECT *column_name(s)*

FROM *table_name* *Order by column_name(s)[asc |desc]*

This example selects all the records stored in the student table and sorts them byage column.

Database: Employee

Table: Emp_per(eno,ename,age)

Example:

```
<?php
        $conn=mysql_connect("localhost","root","") or die("ERROR: Could not
        connect. ");
        mysql_select_db("Employee",$conn) or die("ERROR: Could not find database.
        ");
        $resultset=mysql_query("select * from emp_per order by age",$conn) or
        die(mysql_error());
        while($row=mysql_fetch_array($resultset))
        {
                echo $row[0]. $row[1]. $row[2].<br>";
                        }
        mysql_free_result($resultset);
        mysql_close($conn);
?>
```

- It is also possible to order by more than one column.
- When ordering by more than one column the 2nd column is only used the values in the 1stcolumn are equal.

SYNTAX:

SELECT *column_name(s)*

FROM *table_name* *Order by column1,column2*

WHAT IS AJAX?

AJAX = Asynchronous JavaScript and XML.

AJAX is a technique for creating fast and dynamic web pages.

AJAX allows web pages to be updated asynchronously by exchanging small amounts of data with the server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.

Example: Google Maps, Gmail, Youtube, and Facebook tabs

Code explanation:

First, check if the input field is empty (str.length == 0). If it is, clear the content of the txtHint placeholder and exit the function.

However, if the input field is not empty, do the following:

- Create an XMLHttpRequest object
- Create the function to be executed when the server response is ready
- Send the request off to a PHP file (gethint.php) on the server
    - State=1request has been setup
    - State=2 request sent
    - state=3 request process
    - State=4 request complete
    - Status=404 for file not found
    - Status=200 for file found
- Notice that q parameter is added to the url (gethint.php?q="+str)
- And the str variable holds the content of the input field

Ajaxcall.php

```
<html>
<head>
<script>
function showUser(str)
{
  if (str == "")
      {
      document.getElementById("txtHint").innerHTML = "";
      return;
      } else {
      var xmlhttp = new XMLHttpRequest();
      xmlhttp.onreadystatechange = function()
            {
            if (this.readyState == 4 && this.status == 200) {
            document.getElementById("txtHint").innerHTML = this.responseText;
            }
      };
    xmlhttp.open("GET","getuser.php?q="+str,true);
//true for asynchronous data
    xmlhttp.send();
  }
}
</script>
```

```
</head>
<body>
<form>
<select name="users" onchange="showUser(this.value)">
  <option value="">Select a person:</option>
  <option value="1">Raj</option>
  <option value="2">Radha</option>
  </select>
</form>
<br>
<div id="txtHint"><b>Person info will be listed here...</b></div>

</body>
</html>
```

Get data from this file-> getuser.php

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

<?php
$q = $_GET['q'];
$con = mysql_connect("localhost","root","");
mysql_select_db("studend",$con);
$result = mysql_query("SELECT * FROM stud_per WHERE sno='$q'",$con);

echo "<table>
<tr>
<th>Firstname</th>
<th>City</th>
</tr>";
while($row = mysql_fetch_array($result)) {
  echo "<tr>";
  echo "<td>" . $row['sno'] . "</td>";
  echo "<td>" . $row['city'] . "</td>";
  echo "</tr>";
}
echo "</table>";
mysql_close($con);
?>
</body>
```

```
</html>
```

- A variable `xmlhttp` is declared. Then, a new XMLHttpRequest object is created. If a your target audience use browsers older than Internet Explorer 8, ActiveXObject is used to create XMLHttpRequest.
- 'onreadystatechange' is a property of XMLHttpRequest object which is called whenever 'readyState' attribute is changed.
- We check whether the value of the 'readyState' property is 4, which denotes that the operation is complete.
- If the operation is completed, the status of the response to the request is checked. It returns the HTTP result code. Result code 200 states that the response to the request is successful.
- Now we set the value of the string to be displayed within the div whose id is 'suggestion' as 'responseText' property of the XMLHttpRequest object. 'responseText' is the response to the request as text.
- By using 'open' method of XMLHttpRequest object, a new request to the server is initialized. There are three parameters passed by this method. 'POST' determines the type of the httprequest. 'book-suggestion.php' sets the server side file and setting the third parameter 'true' states that the request should be handled asynchronously.
- 'send' method is used to send data contained in the 'data' variable to the server.

PYTHON INTEGRATION

PHP and Python are two of the most popular and influential programming languages. They both have their strengths and weaknesses and share many standard software development capabilities. Although Rasmus Lerdorf developed PHP as a web language, and Guido van Rossum created Python as a general-purpose programming language. Assessing each language and comparing their strengths across several core elements can help you make the correct choice.

Development Environment (PHP < Python)
Language Complexity (PHP > Python)
Extendibility (PHP > Python)
Security (PHP > Python)
Scalability and Adaptability (=)
Documentation and Community Support(=)

EXECUTING PYTHON SCRIPT USING PHP:

To run a Python script from PHP, we can use the shell_exec function.

Step :1 create python file "x.py" in "www\wamp"

```
Print("hello")
```

Step:2 create php file for calling(execute) python script.

```
<?PHP
$command_exec = escapeshellcmd('python x.py');
$str_output = shell_exec($command_exec);
echo $str_output;
 ?>
```

To call escapeshellcmd to escape the command string.Then we call shell_exec to run the $command.

And we get the output from $output

ESCAPESHELLCMD() METHOD:

escapeshellcmd() escapes any characters in a string that might be used to trick a shell command into executing arbitrary commands. This function should be used to make sure that any data coming from user input is escaped before this data is passed to the exec() or system() functions, or to the backtick operator.

The command line is a dangerous place for unescaped characters. Never pass unmodified user input to one of PHP's shell-execution functions. Always escape the appropriate characters in the command and the arguments. Following characters are preceded by a backslash: &#;`|*?~<>^()[]{}$\, \x0A and \xFF. ' and " are escaped only if they are not paired. On Windows, all these characters plus % and ! are preceded by a caret (^).

Syntax:

escapeshellcmd (string $command)

Example:

```
<?php
$command = "\\%!** Hello  World";
$escaped_command = escapeshellcmd($command);
 echo ($escaped_command);
?>
```

Output

!Hello World

SHELL_EXEC() METHOD :

used to execute the commands via shell and return the complete output as a string. The shell_exec is an alias for the backtick operator

syntax:

shell_exec($cmd)

example: display list of file and directory

```
$output = shell_exec('dir');
echo "$output";
```

EXEC() METHOD:

The exec()is used to execute an external program and returns the last line of the output. It also returns NULL if no command run properly.

Syntax:

exec($command, $output)

example: display last line of output

```
exec("dir",$output1);
echo $output1;
```

SYSTEM() :

it is for executing a system command and immediately displaying the output - presumably text.

Example:

```
system("dir",$output1);
echo $output1;
```

THE OS MODULE

The **OS** **module** in **Python** provides functions for interacting with the **operating system**. **OS** comes under **Python's** standard utility **modules**. we will be covering the following:

- os.chdir()
- os.mkdir()
- os.remove()
- os.rmdir()
- os.path
- read()
- os.getcwd()
- os.makedirs()
- os.rename()
- os.walk()
- write()
- close()

firstly "import os" for executing all commands under os module.

OS.CHDIR() AND OS.GETCWD()

The **os.chdir** function allows us to change the directory that we're currently running our Python session in. If you want to actually know what path you are currently in, then you would call **os.getcwd().**

```
>>> os.getcwd()
'C:\\Python27'
>>> os.chdir("c:\Users\mike\Documents")
>>> os.getcwd()
'c:\\Users\\mike\\Documents'
```

The code above shows us that we started out in the Python directory by default when we run this code in IDLE. Then we change folders using **os.chdir()**. Finally we call os.getcwd() a second time to make sure that we changed to the folder successfully.

OS.MKDIR() AND OS.MAKEDIRS()

used for creating directories. The first one is **os.mkdir()**, which allows us to create a single folder. Let's try it out:

```
>>> os.mkdir("test")
>>> path = 'C:\wamp\www\test\pytest'
>>> os.mkdir(path)
```

The first line of code will create a folder named **test** in the current directory.

The **os.makedirs()** function will create all the intermediate folders in a path if they don't already exist. Basically this means that you can created a path that has nested folders in it. I find myself doing this a lot when I create a log file that is in a dated folder structure, like Year/Month/Day. Let's look at an example:

```
>>> path = ' C:\wamp\www\test\pytest\2014\02\19'
>>> os.makedirs(path)
```

the **pytest** folder in your system, then it just added a **2014** folder with another folder inside of it which also contained a folder. Try it out for yourself using a valid path on your system.

OS.REMOVE() AND OS.RMDIR()

The **os.remove()** and **os.rmdir()** functions are used for deleting files and directories respectively. Let's look at an example of **os.remove()**:

```
>>> os.remove("test.txt")
```

This code snippet will attempt to remove a file named **test.txt** from your current working directory. If it cannot find the file, you will likely receive some sort of error.

an example of **os.rmdir()**:

```
>>> os.rmdir("pytest")
```

The code above will attempt to remove a directory named **pytest** from your current working directory. If it's successful, you will see that the directory no longer exists. An error will be raised if the directory does not exist, you do not have permission to remove it or if the directory is not empty. You might also want to take a look at **os.removedirs()** which can remove nested empty directories recursively.

OS.RENAME(SRC, DST)

The **os.rename()** function will rename a file or folder. Let's take a look at an example where we rename a file:

```
>>> os.rename("test.txt", "pytest.txt")
```

In this example, we tell **os.rename** to rename a file named **test.txt** to **pytest.txt**. This occurs in our current working directory. You will see an error occur if you try to rename a file that doesn't exist or that you don't have the proper permission to rename the file.

OS.WALK()

The **os.walk()** method gives us a way to iterate over a root level path. What this means is that we can pass a path to this function and get access to all its sub-directories and files. Let's use one of the Python folders that we have handy to test this function with. We'll use: C:\Python27\Tools

```
>>> path = 'C:\Python27\Tools'
>>> for root, dirs, files in os.walk(path):
        print(root)
```

```
C:\Python27\Tools
C:\Python27\Tools\i18n
C:\Python27\Tools\pynche
C:\Python27\Tools\pynche\X
C:\Python27\Tools\Scripts
C:\Python27\Tools\versioncheck
C:\Python27\Tools\webchecker
```

If you want, you can also loop over **dirs** and **files** too. Here's one way to do it:

```
>>> for root, dirs, files in os.walk(path):
        print(root)
        for _dir in dirs:
            print(_dir)
        for _file in files:
            print(_file)
```

This piece of code will print a lot of stuff out, so I won't be showing its output here, but feel free to give it a try. Now we're ready to learn about working with paths!

OS.PATH

The **os.path** sub-module of the **os** module has lots of great functionality built into it. We'll be looking at the following functions:

- basename
- dirname
- exists
- isdir and isfile
- join
- split

There are lots of other functions in this sub-module.

OS.PATH.BASENAME

The **basename** function will return just the filename of a path. Here is an example:

```
>>> os.path.basename(' C:\wamp\www\test\pytest \p1.py')
'p1.py'
```

I have found this useful whenever I need to use a filename for naming some related file, such as a log file. This happens a lot when I'm processing a data file.

OS.PATH.DIRNAME

The **dirname** function will return just the directory portion of the path. It's easier to understand if we take a look at some code:

```
>>> os.path.dirname(' C:\wamp\www\test\pytest\p1.py')
' C:\wamp\www\test\pytest '
```

In this example, we just get the directory path back. This is also useful when you want to store other files next to the file you're processing, like the aforementioned log file.

OS.PATH.EXISTS

The **exists** function will tell you if a path exists or not. All you have to do is pass it a path. Let's take a look:

```
>>> os.path.exists(' C:\wamp\www\test\pytest\p1.py')
True
>>> os.path.exists(' C:\wamp\www\test\pytest\fake.py')
False
```

In the first example, we pass the **exists** function a real path and it returns **True**, which means that the path exists. In the second example, we passed it a bad path and it told us that the path did not exist by returning **False**.

OS.PATH.ISDIR / OS.PATH.ISFILE

The **isdir** and **isfile** methods are closely related to the **exists** method in that they also test for existence. However, **isdir** only checks if the path is a directory and **isfile** only checks if the path is a file. If you want to check if a path exists regardless of whether it is a file or a directory, then you'll want to use the **exists** method.

Examples:

```
>>> os.path.isfile(' C:\wamp\www\test\pytest\p1.py')
True
>>> os.path.isdir(' C:\wamp\www\test\pytest\p1.py')
False
>>> os.path.isdir(' C:\wamp\www\test\pytest ')
True
>>> os.path.isfile(' C:\wamp\www\test\pytest1')
False
```

OS.LISTDIR

This function just lists out the files and directories present in the current working directory.

```
>>> import os
>>> os.listdir()
```

OS.PATH.JOIN

The **join** method give you the ability to join one or more path components together using the appropriate separator. For example, on Windows, the separator is the backslash, but on Linux, the separator is the forward slash. Here's how it works:

```
>>> os.path.join(' C:\wamp\www\test\pytest ', 'p1.py')
' C:\wamp\www\test\pytest \p1.py'
```

In this example, we joined a directory path and a file path together to get a fully qualified path.

OS.PATH.SPLIT

The **split** method will split a path into a tuple that contains the directory and the file.

```
>>> os.path.split(' C:\wamp\www\test\pytest \p1.py')
(' C:\wamp\www\test\pytest ', 'p1.py')
```

This example shows what happens when we path in a path with a file. Let's see what happens if the path doesn't have a filename on the end:

```
 >>> os.path.split(' C:\wamp\www\test\pytest ')
(' C:\wamp\www\test\pytest ', 'p1')
```

As you can see, it took the path and split it in such a way that the last sub-folder became the second element of the tuple with the rest of the path in the first element.

For our final example, I thought you might like to see a commmon use case of the **split**:

```
>>> dirname, fname = os.path.split(' C:\wamp\www\test\pytest\p1.py')
>>> dirname
' C:\wamp\www\test\pytest '
>>> fname
'p1.py'
```

This shows how to do multiple assignment. When you split the path, it returns a two-element tuple. Since we have two variables on the left, the first element of the tuple is assigned to the first variable and the second element to the second variable.

OS.OPEN()

Python method open() opens the file file and set various flags according to flags and possibly its mode according to mode.The default mode is 0777 (octal), and the current umask value is first masked out.

SYNTAX:

os.open(file, flags[, mode]);

PARAMETERS

file − File name to be opened.

flags − The following constants are options for the flags. They can be combined using the bitwise OR operator |. Some of them are not available on all platforms.

- os.O_RDONLY − open for reading only
- os.O_WRONLY − open for writing only
- os.O_RDWR − open for reading and writing
- os.O_NONBLOCK − do not block on open
- os.O_APPEND − append on each write
- os.O_CREAT − create file if it does not exist
- os.O_TRUNC − truncate size to 0
- os.O_EXCL − error if create and file exists
- os.O_SHLOCK − atomically obtain a shared lock
- os.O_EXLOCK − atomically obtain an exclusive lock
- os.O_DIRECT − eliminate or reduce cache effects
- os.O_FSYNC − synchronous writes
- os.O_NOFOLLOW − do not follow symlinks

mode − This work in similar way as it works for chmod()

OS. READ():

os.read() method in Python is used to read at most n bytes from the file associated with the given file descriptor.

SYNTAX:

os.read(fd, n)

PARAMETER:

fd: A file descriptor representing the file to be read.

n: An integer value denoting the number of bytes to be read from the file associated with the given file descriptor fd.

OS.CLOSE()

os.close() method in Python is used to close the given file descriptor, so that it no longer refers to any file or other resource and may be reused.

SYNTAX:

os.close(fd)

PARAMETER:

fd: A file descriptor, which is to be closed.

Example of (open,read and close)

```
import os

# Open the file and get # the file descriptor associated # with it using os.open() method
fd = os.open("data.csv", os.O_RDONLY)

# Number of bytes to be read
n = 50

# Read at most n bytes  # from file descriptor fd # using os.read() method
readBytes = os.read(fd, n)

# Print the bytes read
print(readBytes)

# close the file descriptor
os.close(fd)
```

OS.WRITE()

os.write() method in Python is used to write a bytestring to the given file descriptor.

SYNTAX:

os.write(fd, str)

PARAMETER:

fd: The file descriptor representing the target file.
str: A bytes-like object to be written in the file.

Example:

```
import os
 # Open the file and get# the file descriptor associated# with it using os.open() method
fd = os.open("x.txt", os.O_RDWR|os.O_CREAT )
 # String to be written
s = "I like apple"
 # Convert the string to bytes
line = str.encode(s)

# Write the bytestring to the file # associated with the file # descriptor fd and get the number of
# Bytes actually written
numBytes = os.write(fd, line)
print("Number of bytes written:", numBytes)
# close the file descriptor
os.close(fd)
```

SUBPROCESS MODULE IN PYTHON:

Subprocess in Python is a module used to run new codes and applications by creating new processes. It lets you start new applications right from the Python program you are currently writing. So, if you want to run external programs from a git repository or codes from C or C++ programs, you can use subprocess in Python.

The subprocess Module

The **subprocess** module gives the developer the ability to start processes or programs from Python. In other words, you can start applications and pass arguments to them using the subprocess module. The subprocess module was added way back in Python 2.4 to replace the **os** modules set of os.popen, os.spawn and os.system calls as well as replace popen2 and the old **commands** module. We will be looking at the following aspects of the subprocess module:

- the call function
- the Popen class
- how to communicate with a spawned process

THE CALL FUNCTION

The subprocess module provides a function named **call**. This function allows you to call another program, wait for the command to complete and then return the return code. It accepts one or more arguments as well as the following keyword arguments (with their defaults): stdin=None, stdout=None, stderr=None, shell=False.

Let's look at a simple example:

```
>>> import subprocess
>>> subprocess.call("notepad.exe")
0
```

If you run this on a Windows machine, you should see Notepad open up. You will notice that IDLE waits for you to close Notepad and then it returns a code zero (0). This means that it completed successfully. If you receive anything except for a zero, then it usually means you have had some kind of error.

THE POPEN CLASS

The **Popen** class executes a child program in a new process. Unlike the **call** method, it does not wait for the called process to end unless you tell it to using by using the **wait** method. It is like os.system command. Python method **popen()** opens a pipe to or from command.The return value is an open file object connected to the pipe, which can be read or written depending on whether mode is 'r' (default) or 'w'.

SYNTAX:

```
os.popen(command[, mode])
```

EXAMPLE

```
>>> program = "notepad.exe"
>>> subprocess.Popen(program)
```

EXAMPLE:

```
import subprocess
cm='dir'

p1= subprocess.Popen(cm,shell=True)
p1.wait()
if p1.returncode==0:
   print("sucessful run")
else:
   print(p1.stderr)
```

EXPLANATION:

- Import subprocess module.
- Run ‘dir’ command and wait for process completion.
- If process completed successful it’s return 0. Otherwise 1(none).
- If no error found display “successfully run” otherwise display error using stderr.

SOME OTHER PARAMETERS OF POPEN:

```
#display output on terminal
        p1= subprocess.Popen(cm,shell=True)
        print(p1)
#display output store in p1 variable using subprocess.pipe, print on terminal using print(p1)
        p1= subprocess.Popen(cm,stdout=subprocess.PIPE,shell=True)
#TO PRINT RETURNCODE
        print(p1.returncode)
#PRINT ARGUMENTS
        print(p1.args)
#not display output and pass to DEVNULL file using subprocess.DEVNULL
        p1= subprocess.Popen(cm,stdout=subprocess.DEVNULL,shell=True)
```

Note : that using the **wait** method can cause the child process to deadlock when using the stdout/stderr=PIPE commands when the process generates enough output to block the pipe. You can use the **communicate** method to alleviate this situation.

COMMUNICATE

There are several ways to communicate with the process you have invoked. to use the subprocess module’s **communicate** method.

SYNTAX:

Variableofincommingprocess.communicate()

EXAMPLE:

```
Import subprocess
args = 'dir'
```

```
p1=
subprocess.Popen(args,stdout=subprocess.PIPE,stderr=subprocess.PIPE,shell=True,universal_newlines=True)
o,e=p1.communicate()
print('out',format(o))
print('error',format(e))
if p1.returncode==0:
   print("sucessful run")
else:
   print(p1.stderr)
```

EXPLANATION:

it takes output of first process(p1) for further process using p1.communicate().
We can store value of stdout and stderr in variable 'o' and 'e', because it will pass using stdout=subprocess.PIPE and stderr=subprocess.PIPE respectively.
Output will display using print()

In this code example, we create an **args** variable to hold our list of arguments. Then we redirect standard out (stdout) to our subprocess so we can communicate with it. The **communicate** method itself allows us to communicate with the process we just spawned. We can actually pass input to the process using this method. But in this example, we just use communicate to read from standard out. You will notice when you run this code that communicate will wait for the process to finish and then returns a two-element tuple that contains what was in stdout and stderr. That last line that says "None" is the result of stderr, which means that there were no errors.

CHECK-CALL():

to run new applications or programs through Python code by creating new processes. It also helps to obtain the input/output/error pipes as well as the exit codes of various commands. check_call() **returns as soon as /bin/sh process exits without waiting for descendant processes** (assuming shell=True as in your case). check_output() waits until all output is read.

subprocess.check_call(args, *, stdin=None, stdout=None, stderr=None, shell=False)

Parameters:
args=The command to be executed.Several commands can be passed as a string by separated by ";".
stdin=Value of standard input stream to be passed as (os.pipe()).
stdout=Value of output obtained from standard output stream.
stderr=Value of error obtained(if any) from standard error stream.
shell=Boolean parameter.If True the commands get executed through a new shell environment.

EXAMPLE:

```
cm=['echo','hello']
p1=subprocess.check_call(cm,shell=True,universal_newlines=True)
```

```
print(p1)
```

CHECK-OUTPUT():

check_output() is **used to get the output of the calling program in python**. It has 5 arguments; args, stdin, stderr, shell, universal_newlines. The args argument holds the commands that are to be passed as a string.

check_output(args, *, stdin=None, stderr=None, shell=False, universal_newlines=False)
Parameters:
args=The command to be executed. Several commands can be passed as a string by separated by ";".
stdin=Value of standard input stream to be passed as pipe(os.pipe()).
stdout=Value of output obtained from standard output stream.
stderr=Value of error obtained(if any) from standard error stream.
shell=boolean parameter.If True the commands get executed through a new shell environment.
universal_newlines=Boolean parameter.If true files containing stdout and stderr are opened in universal newline mode.

EXAMPLE:

```
Import subprocess
cm='dir'
p1=subprocess.check_output(cm,shell=True,universal_newlines=True)
print(p1)
```

DECODE():

This method is used to convert from one encoding scheme, in which argument string is encoded to the desired encoding scheme. This works opposite to the encode. It accepts the encoding of the encoding string to decode it and returns the original string.
Syntax :

decode(encoding, error)

Parameters:
encoding : Specifies the encoding on the basis of which decoding has to be performed.
error : Decides how to handle the errors if they occur, e.g 'strict' raises Unicode error in case of exception and 'ignore' ignores the errors occurred. decode with error replace implements the 'replace' error handling.

EXAMPLE:

```
a = 'This is a bit möre cömplex sentence.'
print('Original string:', a)
 # Encoding in UTF-8
encoded_bytes = a.encode('utf-8', 'replace')
# Trying to decode via ASCII, which is incorrect
decoded_correct = encoded_bytes.decode('utf-8', 'replace')
print('Correctly Decoded string:', decoded_correct)
```

EXECUTE PHP FILE IN PYTHON

step;1 create p1.php

<?php

Echo “hello”;

?>

Step:2 create python file at same path.

```
import subprocess
proc = subprocess.Popen("php c:/wamp/www/test/p1.php", shell=True,
stdout=subprocess.PIPE)
script_response = proc.stdout.read()
print(script_response)
```

Note if not error like ->> set environmental variable for php

UNIT 5: PYTHON WEB FRAMEWORK: FLASK

INSTALLATION OF FLASK AND ENVIRONMENT SETUP

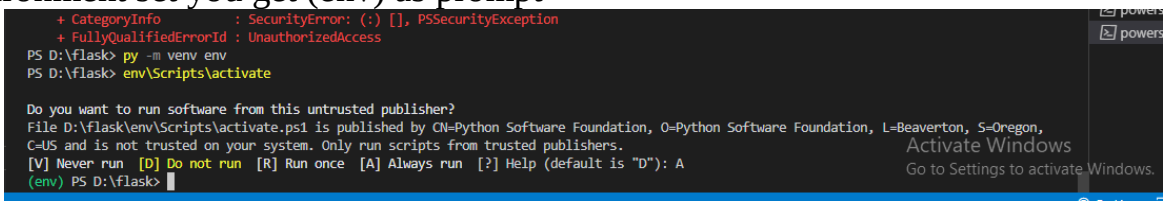
1. Create a new folder
2. Rightclick open with vscode
   3. Go to terminal and create virtual environment
      > pip install virtualenv
      >virtualenv env
      [isolate programs from others,because it not effect on other's environment]

   If you get error like "PermissionError: [WinError 5] Access is denied: 'D:\\flask\\env'", right click on folder and change permission from properties.
   4. follow command "py –m venv env" and then "env\Scripts\activate". If still you get error do the step 6. Otherwise step 5

5. For Activate the environment-> env\Scripts\activate

6. If u get any problem to create virtual environment run-> Set-ExecutionPolicy unrestricted and Set → "A"
   - Still error,run ->"Set-ExecutionPolicy -Scope CurrentUser -ExecutionPolicy Unrestricted" on vscode terminal.

   If environment set you get (env) as prompt



7. Install flask-> pip install flask
   Create a python file

```
from flask import Flask

app = Flask(__name__)
@app.route("/")
def hello_world():
    return "<p>Hello, World!</p>"

if __name__=="__main__":
    app.run(debug=True)
```

   8. run→Flask run
   9. To exit from virtual environment give command->"deactivate" on terminal

   Explanation:

   1. First we imported the **Flask** class. An instance of this class will be our WSGI application.

2. Next we create an instance of this class. The first argument is the name of the application's module or package. __name__ is a convenient shortcut for this that is appropriate for most cases. This is needed so that Flask knows where to look for resources such as templates and static files.
3. We then use the **route()** decorator to tell Flask what URL should trigger our function.
4. The function returns the message we want to display in the user's browser. The default content type is HTML, so HTML in the string will be rendered by the browser.
   Save it as hello.py or something similar. Make sure to not call your application flask.py because this would conflict with Flask itself.

FLASK

- Flask was created by Armin Ronacher of Pocoo, an international group of Python enthusiasts formed in 2004. According to Ronacher, the idea was originally an April Fool's joke that was popular enough to make into a serious application
- Written in: Python
- When Ronacher and Georg Brandl created a bulletin board system written in Python in 2004, the Pocoo projects Werkzeug and Jinja were developed
- In April 2016, the Pocoo team was disbanded and development of Flask and related libraries passed to the newly formed Pallets project
- Flask has become popular among Python enthusiasts. As of October 2020, it has second most stars on GitHub among Python web-development frameworks, only slightly behind Django and was voted the most popular web framework in the Python Developers Survey 2018.

WSGI

WSGI(Web Server Gateway Interface) is a specification that describes the communication between **web servers and Python web applications or frameworks**. It explains how a web server communicates with python web applications/frameworks and applications/frameworks can be chained for processing a request.

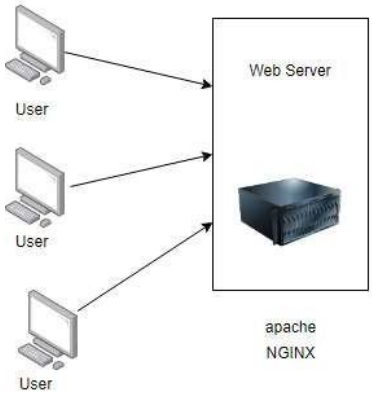
HOW DOES WSGI WORK?

how web Now, let's have a look at how WSGI work. So, to obtain a clear understanding of WSGI, let us assume a case scenario where you have a web application developed in Django or Flask application as shown in the figure

The above figure represents your Django/Flask application.Since a web application is deployed in the web server. The figure below represents the web server that obtains requests from various users.

The web server which obtains requests from the users.

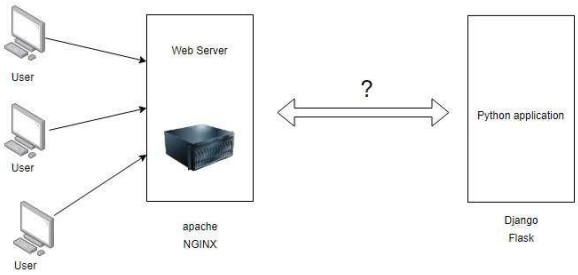
The above web server can be apache,NGINX, etc. server which is responsible for handling various static files and caching purposes. Furthermore,



you can also use the server as a load balancer if you are willing to scale multiple applications.

HOW CAN A WEB SERVER INTERACT WITH THE PYTHON APPLICATION?

The figure representing problem in the interaction between Python application and a web server.



So, now a problem arises as a web server has to interact with a Python application.

Hence, a mediator is required for carrying out the interaction between the web servers and the Python application. So, the standard for carrying out communication between the web server and Python application is WSGI(Web Server Gateway Interface).

Now, web server is able to send requests or communicate with WSGI containers. Likewise, Python application provides a 'callable' object which contains certain functionalities that are invoked by WSGI application which are defined as per the PEP 3333 standard(pyton gateway interface standand). Hence, there are multiple WSGI containers available such as Gunicorn(Green Unicorn' is a Python WSGI HTTP Server for UNIX), uWSGI, etc.

The figure below represents the communication that is carried out between web server, WSGI, and Python application.

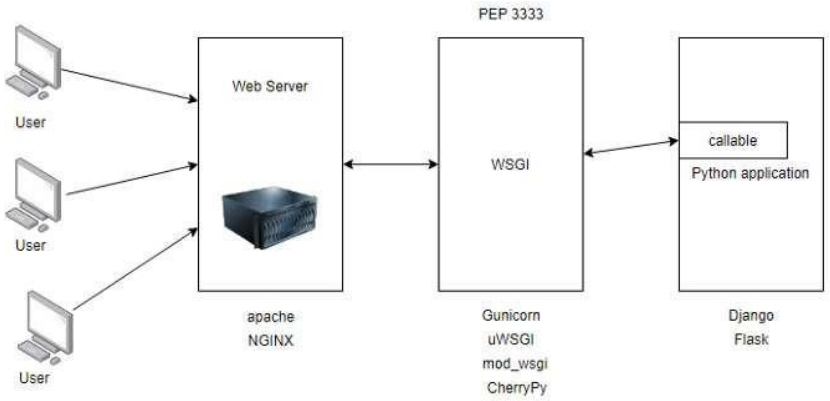
COMMUNICATION BETWEEN USER, WEB SERVER, WSGI, AND PYTHON APPLICATION.

There are multiple WSGI containers that are available today. Hence, a WSGI container is required to be installed in the project so that a web server can

communicate to a WSGI container which further communicates to the Python application and provides the response back accordingly. Finally, when the web server obtains the response, it is sent back to the web browser/users.

WHY USE THE WSGI RATHER THAN DIRECTLY POINTING THE WEB SERVER TO THE DJANGO OR FLASK APPLICATION?

If you directly point your web server to your application, it



reduces the **flexibility** of your application. Since your web server now directly points to your web application, you are unable to swap out web stack components. Now, let's have a look at an example to make you clear about the applicability of WSGI. For instance, today you have decided to deploy your application using Gunicorn but after some years you decide to switch from Gunicorn to mod_wsgi. Now, in this case, you can easily switch to mod_wsgi without making any changes in the application or framework that implements WSGI. Hence, WSGI provides flexibility to your application.

Another reason for using WSGI is due to its **scalability**. Once your application is live, up and running there can be thousands of requests in your application. Hence, WSGI is capable of serving thousands of requests at a time. As we know, the WSGI server is responsible for handling the requests from the web server and takes decision for carrying out the communication of those requests to an application framework's process. Here, we can divide the responsibilities among the servers for scaling web traffic.

WEB TEMPLATE ENGINE (JINJA2)

Jinja2 is a modern day templating language for Python developers. It was made after Django's template. It is used to create HTML, XML or other markup formats that are returned to the user via an HTTP request.

Jinja is a fast, expressive, extensible templating engine. Special placeholders in the template allow writing code similar to Python syntax. Then the template is passed data to render the final document.

It includes:

- Template inheritance and inclusion.
- Define and import macros within templates.
- HTML templates can use autoescaping to prevent XSS from untrusted user input.

- A sandboxed environment can safely render untrusted templates.
- Async support for generating templates that automatically handle sync and async functions without extra syntax.
- I18N support with Babel.
- Templates are compiled to optimized Python code just-in-time and cached, or can be compiled ahead-of-time.
- Exceptions point to the correct line in templates to make debugging easier.
- Extensible filters, tests, functions, and even syntax.

Jinja's philosophy is that while application logic belongs in Python if possible, it shouldn't make the template designer's job difficult by restricting functionality too much.

FOLDER STRUCTURE FOR A FLASK APP

A proper Flask app is going to use multiple files — some of which will be template files. The organization of these files has to follow rules so the app will work. Here is a diagram of the typical structure:

```
my-flask-app
   ├── static/
   |   └── css/
   |       └── main.css
   ├── templates/
   |   ├── index.html
   |   └── student.html
   ├── data.py
   └── students.py
```

1. Everything the app needs is in one folder, here named MY-FLASK-APP.
2. That folder contains two folders, specifically named STATIC and TEMPLATES.

- The STATIC folder contains **assets** used by the templates, including CSS files, JavaScript files, and images. In the example, we have only one asset file, MAIN.CSS. Note that it's inside a CSS folder that's inside the STATIC folder.
- The TEMPLATES folder contains only templates. These have an `.html` extension. As we will see, they contain more than just regular HTML.

3. In addition to the STATIC and TEMPLATES folders, this app also contains `.py` files. Note that these must be OUTSIDE the two folders named STATIC and TEMPLATES.

CREATING THE FLASK CLASS OBJECT

After import flask library file.
Create object using app=Flask(__name__)

CREATING AND HOSTING FIRST BASIC FLASK APP.

```
from flask import Flask

app = Flask(__name__)
@app.route("/")
def hello_world():
   return "<p>Hello, World!</p>"

if __name__=="__main__":
   app.run(debug=True)
```

Importing flask module in the project is mandatory. An object of Flask class is our **WSGI** application.
Flask constructor takes the name of **current module (__name__)** as argument.
The **route()** function of the Flask class is a decorator, which tells the application which URL should call the associated function.
app.route(rule, options)

- The **rule** parameter represents URL binding with the function.
- The **options** is a list of parameters to be forwarded to the underlying Rule object.

In the above example, **'/'** URL is bound with **hello_world()** function. Hence, when the home page of web server is opened in browser, the output of this function will be rendered.
Finally the **run()** method of Flask class runs the application on the local development server.

app.run(host, port, debug, options)

All parameters are optional

Sr.No.	Parameters & Description
1	**Host** Hostname to listen on. Defaults to 127.0.0.1 (localhost). Set to '0.0.0.0' to have server available externally
2	**Port** Defaults to 5000,we can chang by passing parameter port=value
3	**Debug** Defaults to false. If set to true, provides a debug information
4	**Options** To be forwarded to underlying Werkzeug server.

The above given **Python** script is executed from Python shell.
Python Hello.py
A message in Python shell informs you that
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
Open the above URL **(localhost:5000)** in the browser. **'Hello World'** message will be displayed on it.

DEBUG MODE

A **Flask** application is started by calling the **run()** method. However, while the application is under development, it should be restarted manually for each change in the code. To avoid this inconvenience, enable **debug support**. The server will then reload itself if the code changes. It will also provide a useful debugger to track the errors if any, in the application.
The **Debug** mode is enabled by setting the **debug** property of the **application** object to **True** before running or passing the debug parameter to the **run()** method.

```
app.debug = True
app.run()
app.run(debug = True)
```

Modern web frameworks use the routing technique to help a user remember application URLs. It is useful to access the desired page directly without having to navigate from the home page.
The **route()** decorator in Flask is used to bind URL to a function. For example −

```
@app.route('/hello')
def hello_world():
   return 'hello world'
```

Here, URL **'/hello'** rule is bound to the **hello_world()** function. As a result, if a user visits **http://localhost:5000/hello** URL, the output of the **hello_world()** function will be rendered in the browser.
The **add_url_rule()** function of an application object is also available to bind a URL with a function as in the above example, **route()** is used.
A decorator's purpose is also served by the following representation −

```
def hello_world():
   return 'hello world'
app.add_url_rule('/', 'hello', hello_world)
```

FLASK – VARIABLE RULES(APP ROUTING)

It is possible to build a URL dynamically, by adding variable parts to the rule parameter. This variable part is marked as **<variable-name>**. It is passed as a keyword argument to the function with which the rule is associated.
In the following example, the rule parameter of **route()** decorator contains **<name>** variable part attached to URL **'/hello'**. Hence, if the **http://localhost:5000/hello/ami** is entered as a **URL** in the browser, **'ami'** will be supplied to **hello()** function as argument.

```
from flask import Flask
app = Flask(__name__)

@app.route('/hello/<name>')
def hello_name(name):
   return 'Hello %s!' % name

if __name__ == '__main__':
   app.run(debug = True)
```

Save the above script as **hello.py** and run it from Python shell. Next, open the browser and enter URL **http://localhost:5000/hello/ami.**

The following output will be displayed in the browser.
Hello ami!
In addition to the default string variable part, rules can be constructed using the following converters −

Sr.No.	Converters & Description
1	**Int -** accepts integer
2	**Float-** For floating point value
3	**Path-** accepts slashes used as directory separator character

In the following code, all these constructors are used.

```
from flask import Flask
app = Flask(__name__)

@app.route('/blog/<int:postID>')
def show_blog(postID):
   return 'Blog Number %d' % postID

@app.route('/rev/<float:revNo>')
def revision(revNo):
   return 'Revision Number %f' % revNo

if __name__ == '__main__':
   app.run()
```

Run the above code from Python Shell. Visit the URL **http://localhost:5000/blog/11** in the browser.
The given number is used as argument to the **show_blog()** function. The browser displays the following output −
Blog Number 11

FLASK – URL BUILDING

THE ADD_URL_RULE() FUNCTION

There is one more approach to perform routing for the flask web application that can be done by using the add_url() function of the Flask class. The syntax to use this function is given below.

add_url_rule(<url rule>, <endpoint>, <view function>)

This function is mainly used in the case if the view function is not given and we need to connect a view function to an endpoint externally by using this function.
Example:

```
from flask import Flask
app = Flask(__name__)

def about():
   return "This is about page";
```

```
app.add_url_rule("/about","about",about)

if __name__ =="__main__":
   app.run(debug = True)
```

THE URL_FOR() FUNCTION

It is very useful for dynamically building a URL for a specific function. The function accepts the name of a function as first argument, and one or more keyword arguments, each corresponding to the variable part of URL.

The following script demonstrates use of **url_for()** function.

```
from flask import Flask, redirect, url_for
app = Flask(__name__)

@app.route('/admin')
def hello_admin():
   return 'Hello Admin'

@app.route('/guest/<guest>')
def hello_guest(guest):
   return 'Hello %s as Guest' % guest

@app.route('/user/<name>')
def hello_user(name):
   if name =='admin':
      return redirect(url_for('hello_admin'))
   else:
      return redirect(url_for('hello_guest',guest = name))

if __name__ == '__main__':
   app.run(debug = True)
```

The above script has a function **user(name)** which accepts a value to its argument from the URL.

The **User()** function checks if an argument received matches **'admin'** or not. If it matches, the application is redirected to the **hello_admin()** function using **url_for()**, otherwise to the **hello_guest()** function passing the received argument as guest parameter to it.

Save the above code and run from Python shell.

Open the browser and enter URL as − **http://localhost:5000/user/admin**

The application response in browser is −

Hello Admin

Enter the following URL in the browser − **http://localhost:5000/user/ami**

The application response now changes to −

Hello ami as Guest

FLASK – HTTP METHODS

Http protocol is the foundation of data communication in world wide web. Different methods of data retrieval from specified URL are defined in this protocol.
The following table summarizes different http methods −

Sr.No.	Methods & Description
1	**GET**-Sends data in unencrypted form to the server. Most common method.
2	**HEAD**-Same as GET, but without response body
3	**POST**-Used to send HTML form data to server. Data received by POST method is not cached by server.
4	**PUT**-Replaces all current representations of the target resource with the uploaded content.
5	**DELETE**-Removes all current representations of the target resource given by a URL

By default, the Flask route responds to the **GET** requests. However, this preference can be altered by providing methods argument to **route()** decorator.
In order to demonstrate the use of **POST** method in URL routing, first let us create an HTML form and use the **POST** method to send form data to a URL.
Save the following script as login.html

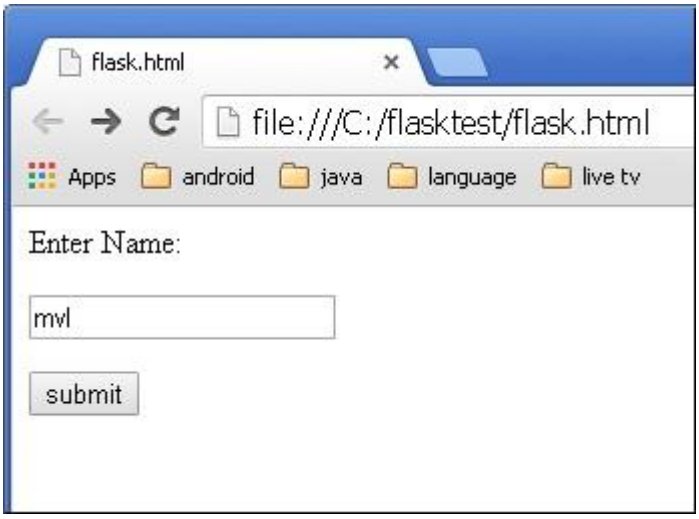
```
<html>
   <body>
      <form action = "http://localhost:5000/login" method = "post">
         <p>Enter Name:</p>
         <p><input type = "text" name = "nm" /></p>
         <p><input type = "submit" value = "submit" /></p>
      </form>
   </body>
</html>
```

Now enter the following script in Python shell.

```
from flask import Flask, redirect, url_for, request
app = Flask(__name__)

@app.route('/success/<name>')
def success(name):
   return 'welcome %s' % name

@app.route('/login',methods = ['POST', 'GET'])
def login():
   if request.method == 'POST':
      user = request.form['nm']
      return redirect(url_for('success',name = user))
   else:
      user = request.args.get('nm')
```

```
    return redirect(url_for('success',name = user))

if __name__ == '__main__':
   app.run(debug = True)
```

After the development server starts running, open **login.html** in the browser, enter name in the text field and click **Submit**.

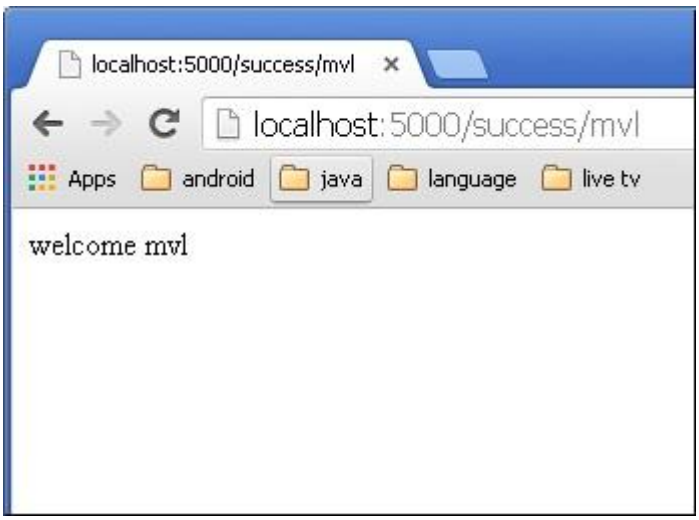


Form data is POSTed to the URL in action clause of form tag.

http://localhost/login is mapped to the **login()** function. Since the server has received data by **POST** method, value of 'nm' parameter obtained from the form data is obtained by −

user = request.form['nm']

It is passed to **'/success'** URL as variable part. The browser displays a **welcome** message in the window.



Change the method parameter to **'GET'** in **login.html** and open it again in the browser. The data received on server is by the **GET** method. The value of 'nm' parameter is now obtained by −

User = request.args.get('nm')

Here, **args** is dictionary object containing a list of pairs of form parameter and its corresponding value. The value corresponding to 'nm' parameter is passed on to '/success' URL as before.

FLASK – TEMPLATES(EMBEDDING PYTHON STATEMENT IN HTML)

It is possible to return the output of a function bound to a certain URL in the form of HTML. For instance, in the following script, **hello()** function will render **'Hello World'** with **<h1>** tag attached to it.

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def index():
   return '<html><body><h1>Hello World</h1></body></html>'

if __name__ == '__main__':
   app.run(debug = True)
```

However, generating HTML content from Python code is cumbersome, especially when variable data and Python language elements like conditionals or loops need to be put. This would require frequent escaping from HTML.

This is where one can take advantage of **Jinja2** template engine, on which Flask is based. Instead of returning hardcode HTML from the function, a HTML file can be rendered by the **render_template()** function.

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def index():
   return render_template('hello.html')

if __name__ == '__main__':
   app.run(debug = True)
```

Flask will try to find the HTML file in the templates folder, in the same folder in which this script is present.

- Application folder
  - Hello.py
  - templates
    - hello.html

The term **'web templating system'** refers to designing an HTML script in which the variable data can be inserted dynamically. A web template system comprises of a template engine, some kind of data source and a template processor.

Flask uses **jinja2** template engine. A web template contains HTML syntax interspersed placeholders for variables and expressions (in these case Python expressions) which are replaced values when the template is rendered.

The following code is saved as **hello.html** in the templates folder.

```
<!doctype html>
<html>
   <body>

      <h1>Hello {{ name }}!</h1>
```

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

Next, run the following script from Python shell.

```
from flask import Flask, render_template
app = Flask(__name__)

@app.route('/hello/<user>')
def hello_name(user):
   return render_template('hello.html', name = user)

if __name__ == '__main__':
   app.run(debug = True)
```

As the development server starts running, open the browser and enter URL as − **http://localhost:5000/hello/mvl**

The **variable** part of URL is inserted at **{{ name }}** place holder.

THE JINJA2 TEMPLATE ENGINE

It uses the following delimiters for escaping from HTML.

- {% ... %} for Statements
- {{ ... }} for Expressions to print to the template output
- {# ... #} for Comments not included in the template output
- # ... ## for Line Statements

In the following example, use of conditional statement in the template is demonstrated. The URL rule to the **hello()** function accepts the integer parameter. It is passed to the **hello.html** template. Inside it, the value of number received (marks) is compared (greater or less than 50) and accordingly HTML is conditionally rendered.

The Python Script is as follows −

```
from flask import Flask, render_template
app = Flask(__name__)

@app.route('/hello/<int:score>')
def hello_name(score):
   return render_template('hello.html', marks = score)

if __name__ == '__main__':
   app.run(debug = True)
```

HTML template script of **hello.html** is as follows −

```
<!doctype html>
<html>
   <body>
      {% if marks>50 %}
         <h1> Your result is pass!</h1>
      {% else %}
         <h1>Your result is fail</h1>
      {% endif %}
```

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

Note that the conditional statements **if-else** and **endif** are enclosed in delimiter **{%..%}**.

Run the Python script and visit URL **http://localhost/hello/60** and then **http://localhost/hello/30** to see the output of HTML changing conditionally.

The Python loop constructs can also be employed inside the template. In the following script, the **result()** function sends a dictionary object to template **results.html** when URL **http://localhost:5000/result** is opened in the browser.

The Template part of **result.html** employs a **for loop** to render key and value pairs of dictionary object **result{}** as cells of an HTML table.

Run the following code from Python shell.

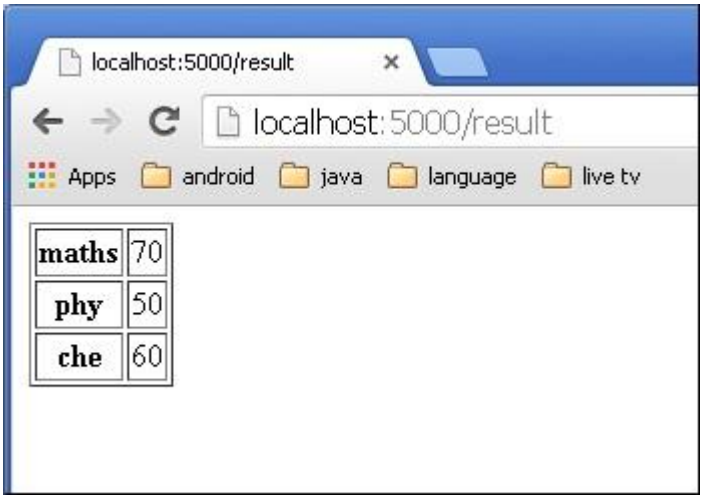
```
from flask import Flask, render_template
app = Flask(__name__)

@app.route('/result')
def result():
   dict = {'phy':50,'che':60,'maths':70}
   return render_template('result.html', result = dict)

if __name__ == '__main__':
   app.run(debug = True)
```

Save the following HTML script as **result.html** in the templates folder.

```
<!doctype html>
<html>
   <body>
      <table border = 1>
         {% for key, value in result.items() %}
            <tr>
               <th> {{ key }} </th>
               <td> {{ value }} </td>
            </tr>
         {% endfor %}
      </table>
   </body>
</html>
```

Here, again the Python statements corresponding to the **For** loop are enclosed in {%..%} whereas, the expressions **key and value** are put inside **{{ }}**.

After the development starts running, open **http://localhost:5000/result** in the browser to get the following output.



STATIC FILES

A web application often requires a static file such as a **javascript** file or a **CSS** file supporting the display of a web page. Usually, the web server is configured to serve them for you, but during the development, these files are served from *static* folder in your package or next to your module and it will be available at **/static** on the application.
A special endpoint 'static' is used to generate URL for static files.
In the following example, a **javascript** function defined in **hello.js** is called on **OnClick** event of HTML button in **index.html**, which is rendered on **'/'** URL of the Flask application.

```
from flask import Flask, render_template
app = Flask(__name__)

@app.route("/")
def index():
   return render_template("index.html")

if __name__ == '__main__':
   app.run(debug = True)
```

The HTML script of **index.html** is given below.

```
<html>
   <head>
      <script type = "text/javascript"
         src = "{{ url_for('static', filename = 'hello.js') }}" ></script>
   </head>

   <body>
      <input type = "button" onclick = "sayHello()" value = "Say Hello" />
   </body>
</html>
```

hello.js contains **sayHello()** function.

```
function sayHello() {
   alert("Hello World")
```

```
}
```

FLASK REQUEST OBJECT: (FORM, ARGS, FILES, REDIRECT)

The data from a client's web page is sent to the server as a global request object. In order to process the request data, it should be imported from the Flask module.
Important attributes of request object are listed below −

- **Form** − It is a dictionary object containing key and value pairs of form parameters and their values.
- **args** − parsed contents of query string which is part of URL after question mark (?).
- **Cookies** − dictionary object holding Cookie names and values.
- **files** − data pertaining to uploaded file.
- **method** − current request method.

FORM REQUEST OBJECT, RENDER_TEMPLATE() METHOD, FORM DATA HANDLING

We have already seen that the http method can be specified in URL rule. The **Form** data received by the triggered function can collect it in the form of a dictionary object and forward it to a template to render it on a corresponding web page.
In the following example, **'/'** URL renders a web page (student.html) which has a form. The data filled in it is posted to the **'/result'** URL which triggers the **result()** function.
The **results()** function collects form data present in **request.form** in a dictionary object and sends it for rendering to **result.html**.
The template dynamically renders an HTML table of **form** data.
Given below is the Python code of application −

```
from flask import Flask, render_template, request
app = Flask(__name__)

@app.route('/')
def student():
   return render_template('student.html')

@app.route('/result',methods = ['POST', 'GET'])
def result():
   if request.method == 'POST':
      result = request.form
      return render_template("result.html",result = result)

if __name__ == '__main__':
   app.run(debug = True)
```

Given below is the HTML script of **student.html**.

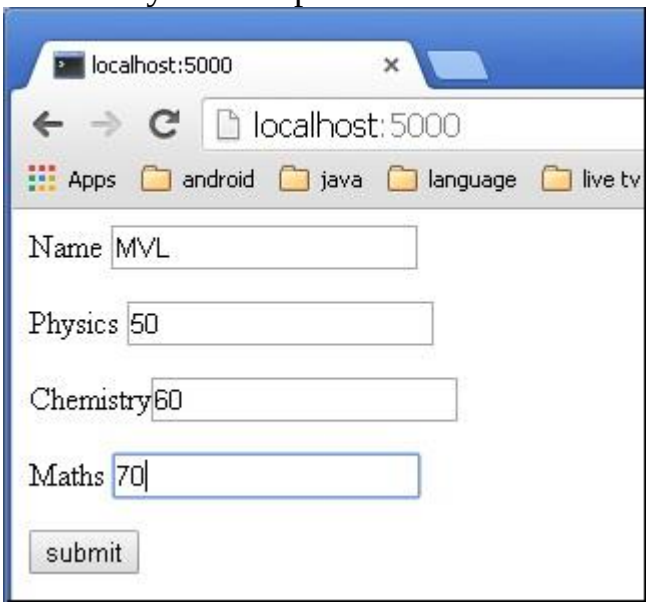
```
<html>
   <body>
      <form action = "http://localhost:5000/result" method = "POST">
```

```
      <p>Name <input type = "text" name = "Name" /></p>
      <p>Physics <input type = "text" name = "Physics" /></p>
      <p>Chemistry <input type = "text" name = "chemistry" /></p>
      <p>Maths <input type ="text" name = "Mathematics" /></p>
      <p><input type = "submit" value = "submit" /></p>
   </form>
  </body>
</html>
```

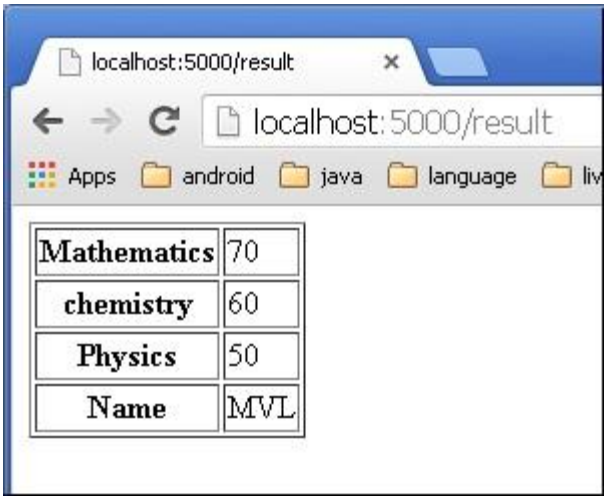
Code of template **(result.html)** is given below −

```
<!doctype html>
<html>
  <body>
    <table border = 1>
      {% for key, value in result.items() %}
        <tr>
          <th> {{ key }} </th>
          <td> {{ value }} </td>
        </tr>
      {% endfor %}
    </table>
  </body>
</html>
```

Run the Python script and enter the URL **http://localhost:5000/** in the browser.



When the **Submit** button is clicked, form data is rendered on **result.html** in the form of HTML table.

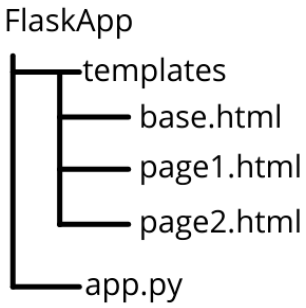


USING FLASK TEMPLATES FOR DYNAMIC DATA

We have already discussed using templates in flask for storing the non-changing data, but we can also use them dynamically to show data using Jinja. Jinja is used to write python-like syntax in HTML files, which helps in using variables like functionality. In other words, we can make dynamic templates also.
We will use them in a small flask application to get better understanding of how we use them.

FILE STRUCTURE

The file structure will look like the image given below.



FLASK APPLICATION

CREATING APP.PY

Our app.py files consists of only two routes with their templates and a value to pass to that particular template. Down below is code, for the reference.

```
from flask import Flask, render_template, request,redirect,url_for
 app = Flask(__name__)
```

```
@app.route("/")
def index():
    data = "This is the body of homepage"
    return render_template("page1.html",content = data)

@app.route("/page2")
def page2():
    data = ['A','B','C']
    return render_template("page2.html",content = data)

if __name__ == "__main__":
    app.run(debug=True)
```

CREATING TEMPLATES

Now creating a folder **templates** and inside it creating three HTML files, BASE.HTML, PAGE1.HTML and PAGE2.HTML. Firstly adding some code to BASE.HTML, as the name suggests it's base file, that means it will be included in the another HTML file using `{% extends "base.html" %}`.
One thing to note that is we use that synatx in the file which we have to include something.Following code is for BASE.HTML.

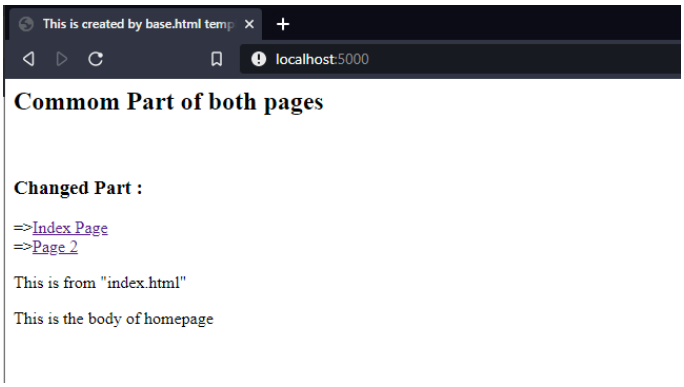
```
<html>
<head><title>  This is created by base.html template  </title></head>
<body>
    <h2>Commom Part of both pages</h2>
    <br>
    <h3>Changed Part :</h3>
    <div>
        =><a href = '{{ url_for("index") }}'>Index Page</a><br>
        =><a href = '{{ url_for("page2") }}'>Page 2</a>
    {% block page_content %}
    {% endblock %}
    </div>
</body>
</html>
```

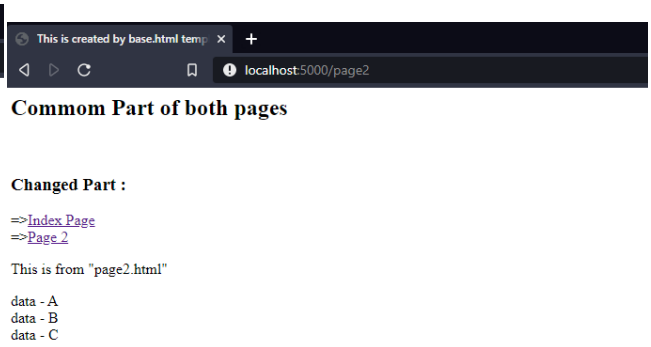
Now adding the code for other files, PAGE1.HTML and PAGE2.HTML.

```
<!-- page1.html -->
{% extends "base.html" %}

{% block page_content %}
<p> This is from "index.html"</p>
        {{ content }}
{% endblock %}
<!-- page2.html -->
{% extends "base.html" %}

{% block page_content %}
```

```
<p> This is from "page2.html"</p>
{% for i in content %}
   data - {{ i }} <br>
{% endfor %}

{% endblock %}
```

Let's discuss what we have done in another two files, firstly PAGE1.HTML, at starting we used this {% extends "base.html" %} to add base.html in page1.html. Then, we created a block named **page_content**. Inside it put the value **content** which we pass when we render the template, then we close the block we created. In this case **content** is a string.
Now , you will notice we have done the same with **page2.html**. But all that's changed is instead of passing an string in RENDER_TEMPLATE(). This time we passed a list, so to access them individually we use JINJA'S **for** loop.





FLASK SESSION, CREATING SESSION, SESSION VARIABLE, SESSION.POP()

Like Cookie, Session data is stored on client. Session is the time interval when a client logs into a server and logs out of it. The data, which is needed to be held across this session, is stored in the client browser.
A session with each client is assigned a **Session ID**. The Session data is stored on top of cookies and the server signs them cryptographically. For this encryption, a Flask application needs a defined **SECRET_KEY**.
Session object is also a dictionary object containing key-value pairs of session variables and associated values.

For example, to set a **'username'** session variable use the statement −
Session['username'] = 'admin'

TO RELEASE A SESSION VARIABLE USE **POP()** METHOD.

session.pop('username', None)
The following code is a simple demonstration of session works in Flask. URL **'/'** simply prompts user to log in, as session variable **'username'** is not set.

```
@app.route('/')
def index():
```

```
  if 'username' in session:
    username = session['username']
      return 'Logged in as ' + username + '<br>' + \
      "<b><a href = '/logout'>click here to log out</a></b>"
  return "You are not logged in <br><a href = '/login'></b>" + \
    "click here to log in</b></a>"
```

As user browses to '/login' the login() view function, because it is called through GET method, opens up a login form.
A Form is posted back to **'/login'** and now session variable is set. Application is redirected to **'/'**. This time session variable **'username'** is found.

```
@app.route('/login', methods = ['GET', 'POST'])
def login():
  if request.method == 'POST':
    session['username'] = request.form['username']
    return redirect(url_for('index'))
  return '''

  <form action = "" method = "post">
    <p><input type = text name = username/></p>
    <p<<input type = submit value = Login/></p>
  </form>

  '''
```

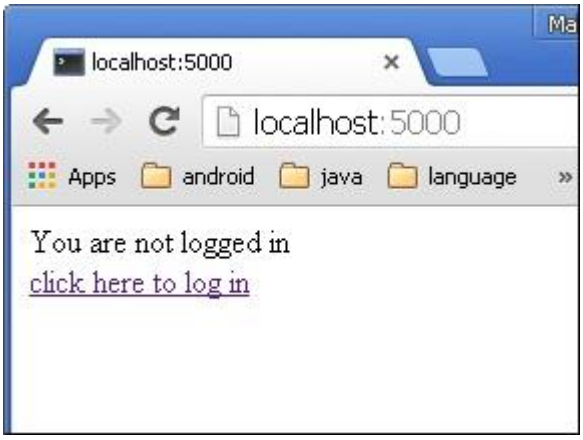
The application also contains a **logout()** view function, which pops out **'username'** session variable. Hence, **'/'** URL again shows the opening page.

```
@app.route('/logout')
def logout():
  # remove the username from the session if it is there
  session.pop('username', None)
  return redirect(url_for('index'))
```

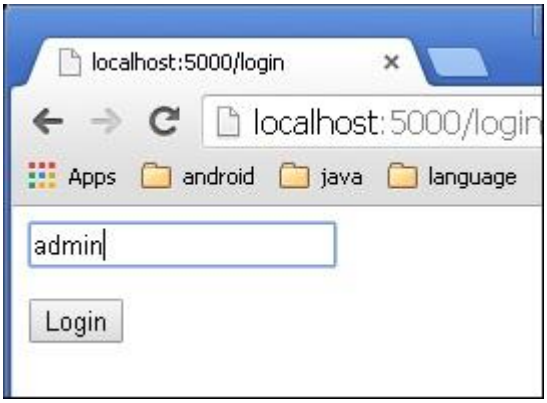
Run the application and visit the homepage. (Ensure to set **secret_key** of the application)

```
from flask import Flask, session, redirect, url_for, escape, request
app = Flask(__name__)
app.secret_key = 'any random string'
```

The output will be displayed as shown below. Click the link **"click here to log in"**.



The link will be directed to another screen. Type ‘admin’.



The screen will show you the message, **‘Logged in as admin’**.

FILE UPLOADING: REQUEST.FILES[] OBJECT, SAVE() METHOD, SAVING FILE TO SPECIFIC FOLDER.

Handling file upload in Flask is very easy. It needs an HTML form with its enctype attribute set to ‘multipart/form-data’, posting the file to a URL. The URL handler fetches file from **request.files[]** object and saves it to the desired location.

Each uploaded file is first saved in a temporary location on the server, before it is actually saved to its ultimate location. Name of destination file can be hard-coded or can be obtained from filename property of **request.files[file]** object. However, it is recommended to obtain a secure version of it using the **secure_filename()** function.

It is possible to define the path of default upload folder and maximum size of uploaded file in configuration settings of Flask object.

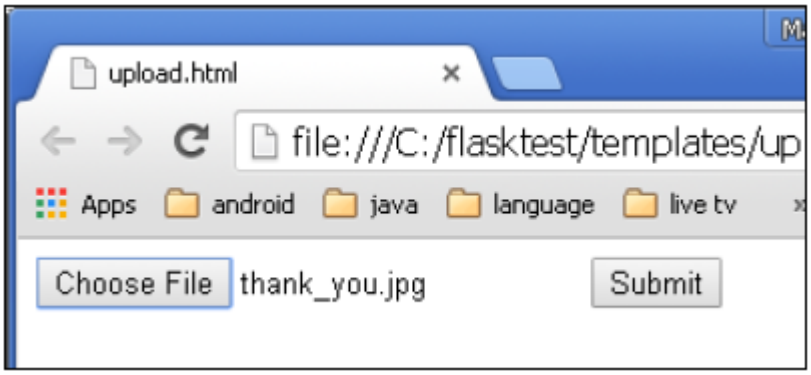
app.config[‘UPLOAD_FOLDER’]	Defines path for upload folder
app.config[‘MAX_CONTENT_PATH’]	Specifies maximum size of file you be

	uploaded – in bytes

The following code has **'/upload'** URL rule that displays **'upload.html'** from the templates folder, and **'/upload-file'** URL rule that calls **uploader()** function handling upload process. **'upload.html'** has a file chooser button and a submit button.

```
<html>
   <body>
      <form action = "http://localhost:5000/uploader" method = "POST"
         enctype = "multipart/form-data">
         <input type = "file" name = "file" />
         <input type = "submit"/>
      </form>
   </body>
</html>
```

You will see the screen as shown below.



Click **Submit** after choosing file. Form's post method invokes **'/upload_file'** URL. The underlying function **uploader()** does the save operation.

Following is the Python code of Flask application.

```
from flask import Flask, render_template, request
from werkzeug import secure_filename
app = Flask(__name__)

@app.route('/upload')
def upload_file():
   return render_template('upload.html')

@app.route('/uploader', methods = ['GET', 'POST'])
def upload_file():
   if request.method == 'POST':
      f = request.files['file']
      f.save(secure_filename(f.filename))
      return 'file uploaded successfully'

if __name__ == '__main__':
   app.run(debug = True)
```

REDIRECTING : REDIRECT() METHOD, LOCATION, STATUS CODE AND RESPONSE.

Flask class has a **redirect()** function. When called, it returns a response object and redirects the user to another target location with specified status code.
Prototype of **redirect()** function is as below −
Flask.redirect(location, statuscode, response)
In the above function −

- **location** parameter is the URL where response should be redirected.
- **statuscode** sent to browser's header, defaults to 302.
- **response** parameter is used to instantiate response.

The following status codes are standardized −

- HTTP_300_MULTIPLE_CHOICES
- HTTP_301_MOVED_PERMANENTLY
- HTTP_302_FOUND
- HTTP_303_SEE_OTHER
- HTTP_304_NOT_MODIFIED
- HTTP_305_USE_PROXY
- HTTP_306_RESERVED
- HTTP_307_TEMPORARY_REDIRECT

The **default status** code is **302**, which is for **'found'**.
In the following example, the **redirect()** function is used to display the login page again when a login attempt fails.

```
from flask import Flask, redirect, url_for, render_template, request
# Initialize the Flask application
app = Flask(__name__)

@app.route('/')
def index():
   return render_template('log_in.html')

@app.route('/login',methods = ['POST', 'GET'])
def login():
   if request.method == 'POST' and request.form['username'] == 'admin' :
      return redirect(url_for('success'))
   else:
      return redirect(url_for('index'))

@app.route('/success')
def success():
   return 'logged in successfully'

if __name__ == '__main__':
   app.run(debug = True)
```

FLASK CLASS HAS **ABORT()** FUNCTION WITH AN ERROR CODE.

Flask.abort(code)
The **Code** parameter takes one of following values −

- **400** − for Bad Request

- **401** − for Unauthenticated
- **403** − for Forbidden
- **404** − for Not Found
- **406** − for Not Acceptable
- **415** − for Unsupported Media Type
- **429** − Too Many Requests

Let us make a slight change in the **login()** function in the above code. Instead of re-displaying the login page, if **'Unauthourized'** page is to be displayed, replace it with call to **abort(401)**.

```
from flask import Flask, redirect, url_for, render_template, request, abort
app = Flask(__name__)

@app.route('/')
def index():
   return render_template('log_in.html')

@app.route('/login',methods = ['POST', 'GET'])
def login():
   if request.method == 'POST':
      if request.form['username'] == 'admin' :
         return redirect(url_for('success'))
      else:
         abort(401)
   else:
      return redirect(url_for('index'))

@app.route('/success')
def success():
   return 'logged in successfully'

if __name__ == '__main__':
   app.run(debug = True)
```