



GIFT School of Engineering and Applied Sciences

Fall 2022

Database Systems-Lab

Lab-2 Manual

Introduction to PHP Basics

Introduction to Lab:

This lab introduces students the server-side scripting language. In the previous lab we have studied the client-side scripting language. This Lab introduces the basic and fundamentals of this programming language. The purpose of studying php in database course is to make interactive and dynamic web pages. So, let's get started!

The main topics cover in this lab includes:

1. Introduction to PHP
2. Request Response Cycle
3. PHP Syntax
4. PHP echo and Print statement
5. PHP variables
6. PHP Super Globals
7. PHP Arrays
8. PHP if/else/elseif Statement
9. PHP Loops
10. PHP User defined Functions
11. PHP Built-in Functions
12. PHP Include and Require
13. PHP Form Validation

Objective of this Lab

At the end of this lab students shall be able to learn

- Understand how server-side programming works on the web.
- PHP Basic syntax for variable types and calculations.
- Creating conditional structures
- Storing data in arrays
- Using PHP built-in functions and creating custom functions
- Understanding POST and GET in form submission.
- How to receive and process form submission data.

1. Introduction to PHP

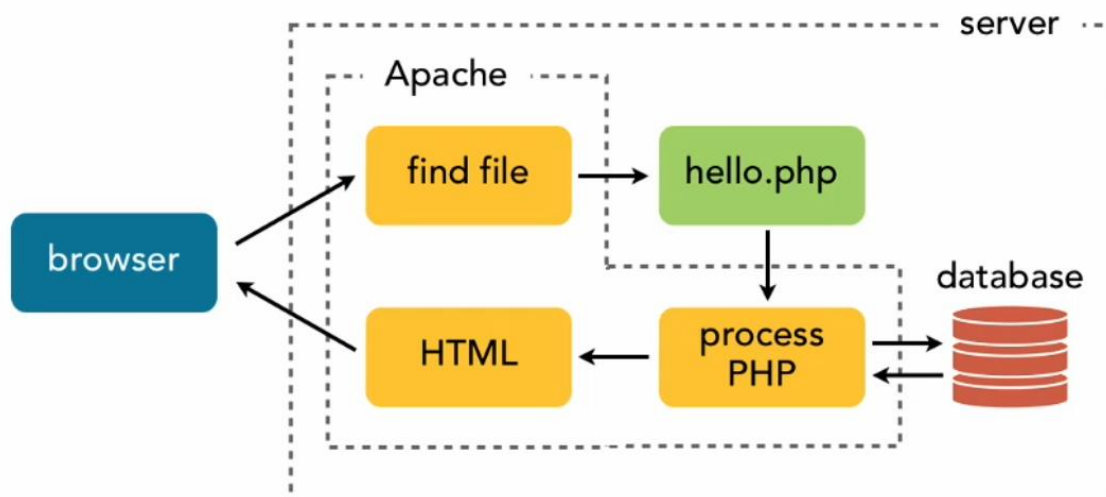
PHP, an acronym for Hypertext Preprocessor, is a widely-used open source general-purpose scripting language. It is a cross-platform, HTML-embedded server-side scripting language and is especially suited for web development.

Where

- Server-side means that PHP scripts execute on the Web server, not within the browser on your local machine.
- Cross-platform means that PHP scripts can run on many different operating systems and Web servers. PHP is available for the two most popular Web server configurations IIS and Apache.
- HTML embedded scripting language means that PHP statements and commands are actually embedded in your HTML documents. When the Web server sees the PHP statements in the Web page, the server executes the statements and sends the resulting output along with the rest of the HTML. PHP commands are parsed by the server much like Active Server Pages or Cold Fusion tags.

The basic syntax of PHP is similar to C, Java, and Perl, and is easy to learn. PHP is used for creating interactive and dynamic web pages quickly, but you can do much more with PHP.

2. Request Response Cycle



3. PHP Syntax

The default syntax starts with "<? php" and ends with "?>". A PHP scripting block can be placed anywhere in the document.

Example:

```
<?php
```

```
Code here.....
```

```
?>
```

A PHP file normally contains HTML tags, just like an HTML file, and some PHP scripting code.

4. PHP echo and Print Statement

In PHP the two basic constructs to get outputs are echo and print. Actually, echo() is not a function, it is a language construct, therefore, you can use it without parentheses.

Syntax

```
echo (arg1, arg2... )
```

Example: Simple string display:

```
<?php
```

```
echo "One line simple string.<br>";
```

```
echo "Two line simple string example<br>";
```

```
echo "Tomorrow I \'ll learn PHP global variables.<br>";
```

```
echo "This is a bad command : del c:\\*. * <br>";
```

```
?>
```

All the above echo commands simply display the corresponding string, here we have used an additional html command
 at the end of each echo statement to generate a line break as \n cannot generate a line break in browser.

Output of above example in web browser:

One line simple string.
Two line simple string example
Tomorrow I 'll learn PHP global variables.
This is a bad command : del c:*.*

4.1. PHP echo and HTML paragraph element:

We can display string, variables with echo function, additionally, we can embedded html commands into echo command. Here we have attached html paragraph element in various form into echo.

Example:

```
<?php  
// simple html statement.  
echo 'One line simple string.<br>';  
// display strings within paragraph with different color.  
echo "<p> <font color=blue>One line simple string in  
blue color</font> </p>";  
echo "<p> <font color=red>One line simple string in red  
color</font> </p>";  
echo "<p> <font color=green> One line simple string in  
green color</font> </p>";  
?>
```

Output of above example in Web Browser:

One line simple string.

One line simple string in blue color

One line simple string in red color

One line simple string in green color

4.2. PHP echo and HTML table element:

We can display string, variables with echo function, additionally, we can embedded html elements into echo command. Here we have attached html table elements into echo.

Example:

```
<?php
$a=1000;
$b=1200;
$c=1400;

echo "<table border=1 cellspacing=0 cellpadding=0>

<tr> <td><font color=blue>Salary of Mr. A is</td>
<td>$a$</font></td></tr>

<tr> <td><font color=blue>Salary of Mr. B is</td>
<td>$b$</font></td></tr>

<tr> <td><font color=blue>Salary of Mr. C is</td>
<td>$c$</font></td></tr>

</table>";

?>
```

Output of above example in Web Browser :

Salary of Mr. A is	1000\$
Salary of Mr. B is	1200\$
Salary of Mr. C is	1400\$

4.3. PHP Print

Print is used to display a string.

Syntax:

```
print(string $val)
```

Parameters:

val: The input data.

Return Values:

Returns 1, always.

Print() is not actually a real function, it is a language construct like echo. There is some difference between the two, echo is marginally faster compare to print as echo does not return any value. Echo can accept multiple parameters without parentheses but print can accept only one parameter.

Example to display simple string with print():

```
<?php  
print 'One line simple string.<br>';  
print 'Two lines simple  
string example<br>';  
print 'Tomorrow I \'ll learn PHP global variables.<br>';  
print 'This is a bad command : del c:\\*. * <br>';
```

?>

All the above print commands simply display the corresponding string, here we have used an additional html command
 at end of each print statement to generate a line break.

Output of above example in Web Browser:

```
One line simple string.
Two lines simple string example
Tomorrow I 'll learn PHP global variables.
This is a bad command : del c:\*.*
```

5. PHP Variables

Variable is a symbol or name that stands for a value. Variables are used for storing values such as numeric values, characters, character strings, or memory addresses so that they can be used in any part of the program.

5.1. Declaring PHP variables:

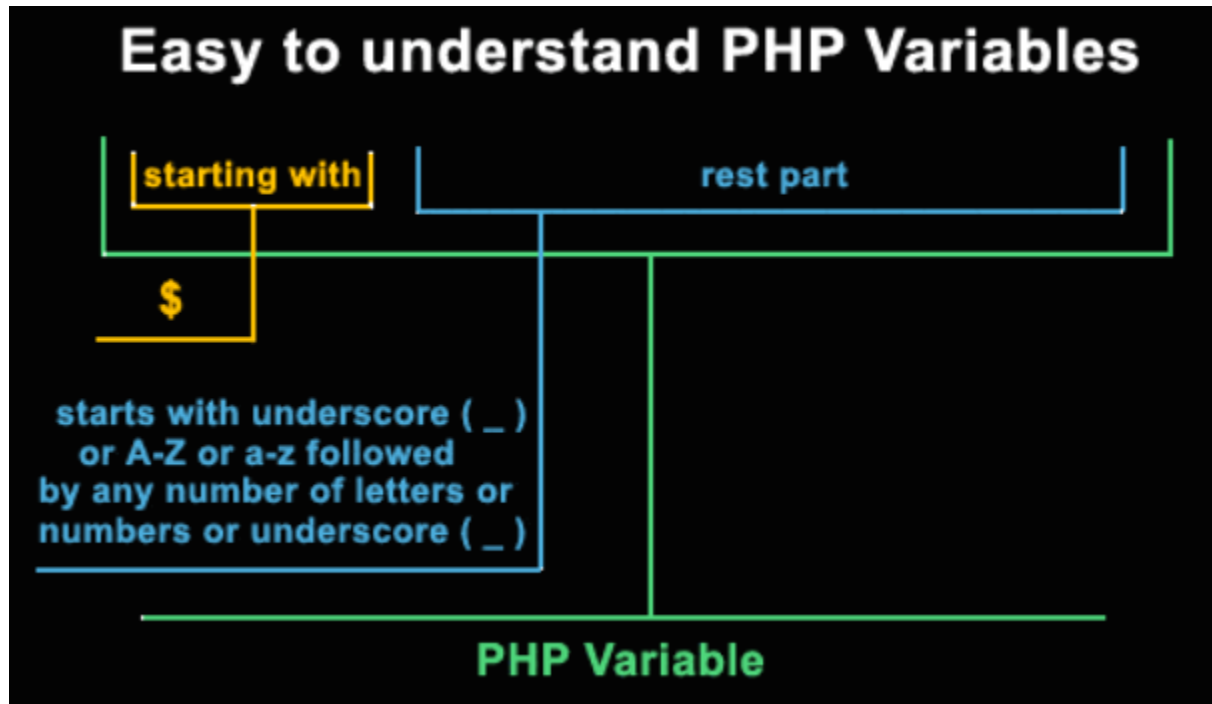
All variables in PHP start with a \$ (dollar) sign followed by the name of the variable

A valid variable name starts with a letter (A-Z, a-z) or underscore (_), followed by any number of letters, numbers, or underscores.

If a variable name is more than one word, it can be separated with an underscore (for example \$employee_code instead of \$employeecode).

'\$' is a special variable that cannot be assigned.

5.2. Pictorial presentation of PHP variable naming:



Example: Valid and invalid PHP variables:

```
<?php
```

```
$abc = 'Welcome'; //valid
```

```
$Abc = 'daraz.pk'; //valid
```

```
$9xyz = 'Hello world'; //invalid; starts with a number
```

```
$_xyz = 'Hello world'; //valid; starts with an underscore
```

```
$_9xyz = 'Hello world'; //valid
```

```
$aäa = 'Hello world'; //valid; 'ä' is (Extended) ASCII 228.
```

```
?>
```

6. PHP Super Globals

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

The PHP superglobal variables are:

- \$_REQUEST
- \$_POST
- \$_GET

6.1. PHP: \$_REQUEST

\$_REQUEST is a super global variable which is widely used to collect data after submitting html forms.

Here is an example:

```
<!DOCTYPE HTML>

<html>

<head>

<title>HTML form example </title>

</head>

<body>

<form name="signin" action="demo.php" method="POST" >

Name: <input type="text" name="fname" id="name" size="30" placeholder="Enter
your name" /><br /><br />

Email: <input type="email" name="email" id="email" size="30"
placeholder="Enter your email" /><br /><br />

Roll No: <input type="text" name="rollno" id="rollno" size="28"
placeholder="Enter 4 digit Roll No" /><br /><br />

<input type="submit" value="Sign in" / >

</form>

</body>

</html>
```

we can collect the data entered by the user in different fields using \$_REQUEST. Suppose we want to see what data have been entered by the user in the name field, then code to do that will be:

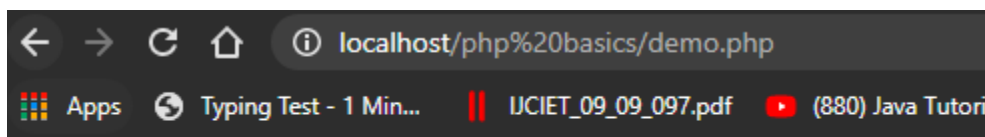
```
<?php
$name=$_REQUEST['fname'];
echo "Your name is: ".$name;
?>
```

Output of above Example in Web browser:

Name:

Email:

Roll No:



Your name is: Tonny

Note: when you hit the sign in button, you are redirected to the php file you have mentioned in the action attribute of the form.

6.2. PHP: \$_GET and \$_POST

Both GET and POST create an array (e.g. array(key1 => value1, key2 => value2, key3 => value3, ...)). This array holds key/value pairs, where keys are the names of the form controls and values are the input data from the user.

Both GET and POST are treated as \$_GET and \$_POST. These are superglobals, which means that they are always accessible, regardless of scope - and you can access them from any function, class or file without having to do anything special.

\$_GET is an array of variables passed to the current script via the URL parameters.

\$_POST is an array of variables passed to the current script via the HTTP POST method.

Here is the Example of Get method:

HTML CODE:

```
<!DOCTYPE HTML>

<html>

<head>

<title>HTML form example </title>

</head>

<body>

<form name="signin" action="demo.php" method="GET" >

Name: <input type="text" name="fname" id="name" size="30" placeholder="Enter
your name" /><br /><br />

Email: <input type="email" name="email" id="email" size="30"
placeholder="Enter your email" /><br /><br />

Roll No: <input type="text" name="rollno" id="rollno" size="28"
placeholder="Enter 4 digit Roll No" /><br /><br />

<input type="submit" value="Sign in" / >

</form>

</body>

</html>
```

PHP CODE:

```
<?php
```

```

$name = $_REQUEST['fname'];

echo "Your name is: ".$name."<br>";

$email = $_REQUEST['email'];

echo "Your email is: ".$email."<br>";

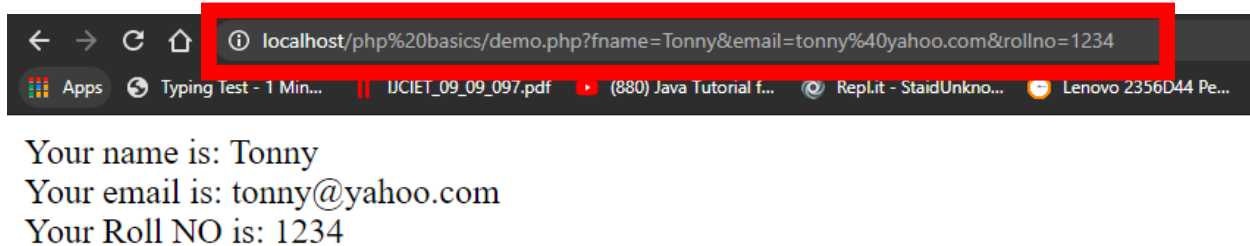
$rollno = $_REQUEST['rollno'];

echo "Your Roll NO is: ".$rollno."<br>";

?>

```

Output of above Example in Web Browser:



Here is the Example of POST method:

HTML CODE:

```

<!DOCTYPE HTML>

<html>

<head>

<title>HTML form example </title>

</head>

<body>

<form name="signin" action="demo.php" method="POST" >

Name: <input type="text" name="fname" id="name" size="30" placeholder="Enter
your name" /><br /><br />

```

```
Email: <input type="email" name="email" id="email" size="30"
placeholder="Enter your email" /><br /><br />
```

```
Roll No: <input type="text" name="rollno" id="rollno" size="28"
placeholder="Enter 4 digit Roll No" /><br /><br />
```

```
<input type="submit" value="Sign in" / >
```

```
</form>
```

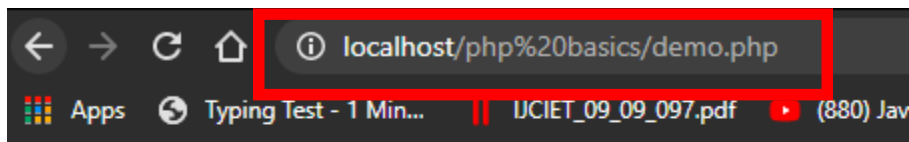
```
</body>
```

```
</html>
```

PHP CODE:

```
<?php
$name = $_REQUEST['fname'];
echo "Your name is: ".$name.'<br>';
$email = $_REQUEST['email'];
echo "Your email is: ".$email.'<br>';
$rollno = $_REQUEST['rollno'];
echo "Your Roll NO is: ".$rollno.'<br>';
?>
```

Output of above example in Web Browser:



Your name is: Tonny
Your email is: tonny@yahoo.com
Your Roll NO is: 1234

Note: GET should NEVER be used for sending passwords or other sensitive information! So, Developers prefer POST for sending the form data.

7. PHP Arrays

An array in PHP is a collection of key/value pairs. This means that it maps values to keys. Array keys (or indexes) may be either integers or string whereas values can be any type.

7.1. Indexed and Associative Arrays


In PHP there is two kinds of arrays : indexed array and associative array. The only difference is that numeric values are used as 'keys' in indexed array start from zero (0) and in associative array, strings are used as 'keys'. PHP does not differentiate between indexed and associative arrays, therefore a PHP array may contain strings as well as integers as 'keys'.

Example: Indexed arrays without key

```
<?php
$courses = array("Database Systems", "SDA", "SQA");
var_dump($courses);
?>
```

Here var_dump() function is used to display structured information of an array.

Output of above example in Web browser:



```
array(3) { [0]=> string(16) "Database Systems" [1]=> string(3) "SDA" [2]=> string(3) "SQA" }
```

Example: Associative arrays with key

There are two ways to create an associative array:

```
<?php

$course_teacher = array('Ahmed' => 'SDA'
, 'Zunaina' => 'SQA', 'Tayabba' => 'Database Systems' );
```

```
var_dump($course_teacher);
```

```
?>
```

or:

```
<?php
```

```
$course_teacher = array();
```

```
$course_teacher['Ahmed'] = "SDA";
```

```
$course_teacher['Zunaina'] = 'SQA';
```

```
$course_teacher['Tayabba'] = 'Database Systems';
```

```
var_dump($course_teacher);
```

```
?>
```

In the above example, Ahmed, Zunaina and Tayabba are the keys and SDA, SQA and Database Systems are the values of the array \$course_teacher.

 string(3) "SDA" ["Zunaina"]=> string(3) "SQA" ["Tayabba"]=> string(16) "Database Systems" }" data-bbox="114 495 884 548"/>

```
array(3) { ["Ahmed"]=> string(3) "SDA" ["Zunaina"]=> string(3) "SQA" ["Tayabba"]=> string(16) "Database Systems" }
```

8. PHP if / else / elseif Statement

The if statement executes a single statement or a group of statements if a certain condition is met. It cannot do anything if the condition is false. For this purpose, else is used.

Syntax:

```
if (condition)
    execute statement(s) if condition is true;
else
    execute statement(s) if condition is false;
```

9. PHP Loops

Looping statements in PHP are used to execute the same block of code a specified number of times.

In PHP we have the following looping statements:

- **while** - loops through a block of code if and as long as a specified condition is true
- **do...while** - loops through a block of code once, and then repeats the loop as long as a special 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

9.1. The While Statement

The while statement will execute a block of code **if and as long as** a condition is true.

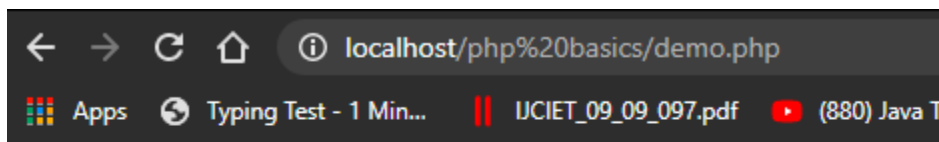
Syntax:

```
while (condition) code to
be executed;
```

Example:

```
<?php
$courses = array("DB" , "SQA" , "SDA");
$i = 0;
$arrLength = count($courses);
while ($i < $arrLength) {
    # code...
    echo $courses[$i]."<br>";
    $i++;
}
?>
```

Output of above Example in Web Browser:



DB
SQA
SDA

9.2. The For Statement

The for statement is the most advanced of the loops in PHP.

Syntax:

```
for (init; cond; incr)
{
```

```
code to be executed;
}
```

In its simplest form, the for statement is used when you know how many times you want to execute a statement or a list of statements.

Parameters:

- **init:** Is mostly used to set a counter, but can be any code to be executed once at the beginning of the loop statement.
- **cond:** Is evaluated at beginning of each loop iteration. If the condition evaluates to TRUE, the loop continues and the code executes. If it evaluates to FALSE, the execution of the loop ends.
- **incr:** Is mostly used to increment a counter, but can be any code to be executed at the end of each loop.

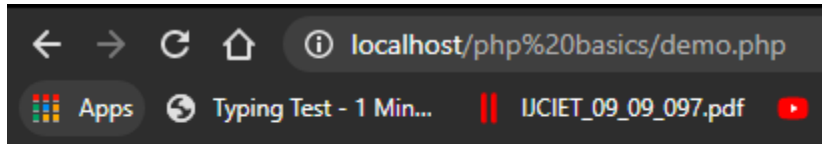
Note: Each of the parameters can be empty or have multiple expressions separated by commas.

- **cond:** All expressions separated by a comma are evaluated but the result is taken from the last part. This parameter being empty means the loop should be run indefinitely. This is useful when using a conditional break statement inside the loop for ending the loop.

Example:

```
<?php
$courses = array("DB" , "SQA" , "SDA" , "Algorithms");
for ($i=0; $i < count($courses) ; $i++) {
    # code...
    echo $courses[$i]."<br>";
}
?>
```

Output of above Example in Web Browser:



DB
SQA
SDA
Algorithms

9.3. The Foreach Statement

The foreach statement is used to loop through arrays.

For every loop, the value of the current array element is assigned to \$value (and the array pointer is moved by one) - so on the next loop, you'll be looking at the next element.

Syntax:

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

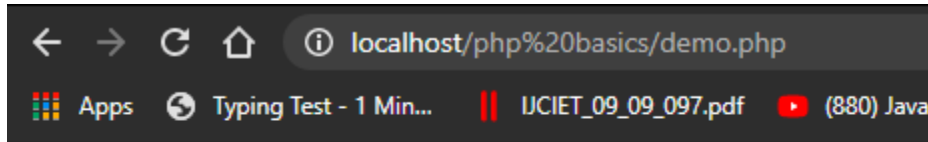
Example:

```
<?php
$course_teacher = array('Ahmed' => 'SDA', 'Zunaina' => 'SQA', 'Tayabba' => 'Database
Systems' );

foreach ($course_teacher as $key => $value) {
    # code...
    echo "Key:". $key. "=". " Value: ". $value. "<br>";
}

?>
```

Output of above Example in Web Browser:



Key:Ahmed= Value: SDA

Key:Zunaina= Value: SQA

Key:Tayabba= Value: Database Systems

10. PHP User Defined Function

In all programming and scripting language, a function is a block of statements that can be used repeatedly in a program. In PHP, the concept of the function is the same as in another language like 'C'. There are more than 1,000 in-built functions into the standard PHP distribution. Besides these, we can define functions as per our requirements. These are called 'User Defined Function'.

Syntax:

```
function function-name()
{
    statement 1 :
    statement 2 :
    statement 3 :
    .....
}
```

Elements of a function:

function: A function declaration starts with the special word 'function'.

Name of the function:

The function name is defined by the user.

A valid function name starts with a letter or underscore, followed by any number of letters, numbers, or underscores.

Remember that function names are case-insensitive.

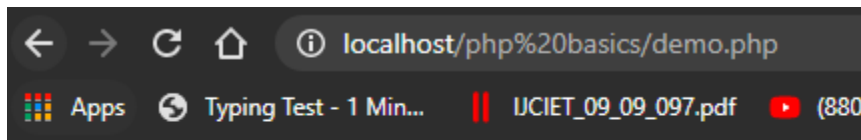
Opening and Closing curly braces ({ })

The function body enclosed within a pair of braces which may contain variable names and actual function code. The opening curly brace ({) indicates the beginning of the function code and the closing curly (}) brace indicates the termination of the function.

Example: PHP function

```
<?php
function myfunction()
{
    echo "Welcome to Database Systems Lab";
}
myfunction();
?>
```

When we call the above function, it will print Welcome to Database Systems Lab



Welcome to Database Systems Lab

11. PHP Built-in Functions

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

11.1. PHP Variable Handling Functions

The PHP variable handling functions are part of the PHP core. No installation is required to use these functions.

Function	Description
boolval()	Returns the boolean value of a variable
debug_zval_dump()	Dumps a string representation of an internal zend value to output
doubleval()	Alias of floatval()
empty()	Checks whether a variable is empty
floatval()	Returns the float value of a variable
get_defined_vars()	Returns all defined variables, as an array
get_resource_type()	Returns the type of a resource
gettype()	Returns the type of a variable
intval()	Returns the integer value of a variable
is_array()	Checks whether a variable is an array
is_bool()	Checks whether a variable is a boolean
is_callable()	Checks whether the contents of a variable can be called as a function
is_countable()	Checks whether the contents of a variable is a countable value
is_double()	Alias of is_float()
is_float()	Checks whether a variable is of type float
is_int()	Checks whether a variable is of type integer
gettype()	Returns the type of a variable
intval()	Returns the integer value of a variable
is_array()	Checks whether a variable is an array

is_integer()	Alias of is_int()
is_iterable()	Checks whether the contents of a variable is an iterable value
is_long()	Alias of is_int()
is_null()	Checks whether a variable is NULL
is_numeric()	Checks whether a variable is a number or a numeric string
is_object()	Checks whether a variable is an object
is_real()	Alias of is_float()
is_resource()	Checks whether a variable is a resource
is_scalar()	Checks whether a variable is a scalar
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
serialize()	Converts a storable representation of a value
settype()	Converts a variable to a specific type
strval()	Returns the string value of a variable
unserialize()	Converts serialized data back into actual data
var_dump()	Dumps information about one or more variables
var_export()	Returns structured information (valid PHP code) about a variable

var_dump()	Dumps information about one or more variables
var_export()	Returns structured information (valid PHP code) about a variable
var_dump()	Dumps information about one or more variables

12. PHP Include and Require

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.

The include and require statements are identical, except upon failure:

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

So, if you want the execution to go on and show users the output, even if the include file is missing, use the include statement. Otherwise, in case of Framework, CMS, or a complex PHP application coding, always use the require statement to include a key file to the flow of execution. This will help avoid compromising your application's security and integrity, just in-case one key file is accidentally missing.

Including files saves a lot of work. This means that you can create a standard header, footer, or menu file for all your web pages. Then, when the header needs to be updated, you can only update the header include file.

Syntax:

```
include('name of the php file with path');
or
require('name of the php file with path');
```

Example:

File header.php:

```
<?php
```

```
echo "Hello! Welcome to Database Systems Lab."."<br>";
```



```
?>
```

File demo.php, which includes header.php :

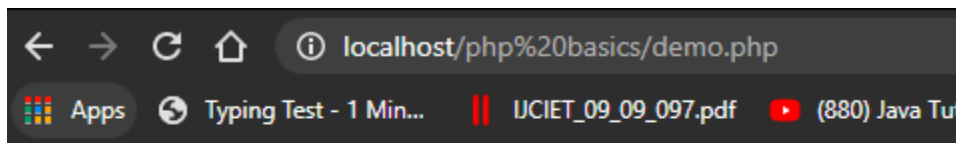
```
<?php
```

```
include ('header.php');
```

```
echo "We are studying PHP Includes.";
```

```
?>
```

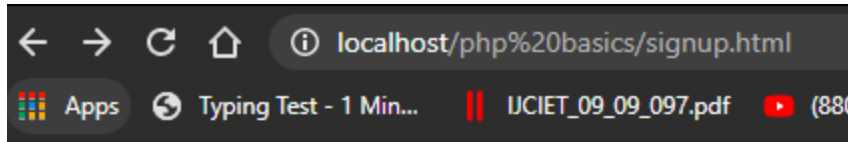
Output of above Example in Web Browser:



Hello! Welcome to Database Systems Lab.
We are studying PHP Includes.

13. PHP Form Validation

The example below displays a simple sign up form with six input fields and a sign-up button:



Register Yourself

First Name

Last Name

Email:

Password:

Confirm Password:

Date of birth



When the user fills out the form above and clicks the submit button, the form data is sent for processing to a PHP file named "validation.php". The form data is sent with the HTTP POST method.

To validation the submitted data you can simply use the PHP built-in functions that we discussed earlier.

```

<?php
if (isset($_POST['signup'])) {
    if (empty($_POST['fname'])) {
        echo "Please enter the first name.<br>";
    }
    else{
        $firstname = trim($_POST['fname']);
    }
    if (empty($_POST['lname'])) {
        echo "Please enter the last name.<br>";
    }
    else{
        $lastname = trim($_POST['lname']);
    }
    if (empty($_POST['email'])) {
        echo "Please enter the email.<br>";
    }
    else{
        $email = trim($_POST['email']);
    }
    if (!empty($_POST['password']) && !empty($_POST['cpassword'])) {
        if ($_POST['password'] != $_POST['cpassword']) {
            echo "Both Passwords should be same.<br>";
        }
        else{
            $password = $_POST['password'];
        }
    }
    else{
        echo "Please enter the password.<br>";
    }
    if (empty($_POST['dob'])) {
        echo "Please enter the date of birth.<br>";
    }
    else{
        $dob = trim($_POST['dob']);
    }
}

```

To display the submitted data you could simply echo all the variables. The "validation.php" looks like this:

```
echo "<table border=1>
    <tr>
        <th>First Name</th>
        <th>Last Name</th>
        <th>Email</th>
        <th>Password</th>
        <th>Date of Birth</th>
    </tr>
    <tr>
        <td>".$firstname."</td>
        <td>".$lastname."</td>
        <td>".$email."</td>
        <td>".$password."</td>
        <td>".$dob."</td>
    </tr>
</table>";

}
else{
    echo "Please Sign UP The Form";
}
```

Output of above example in the Web Browser:

Register Yourself


First Name

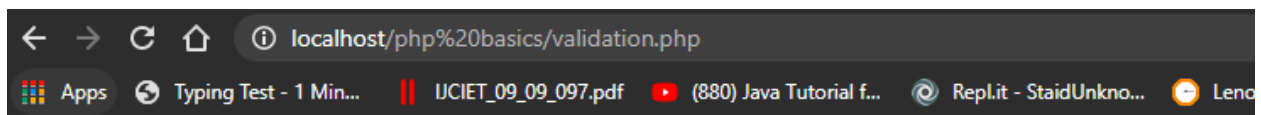
Last Name

Email:

Password:

Confirm Password:

Date of birth
 



First Name	Last Name	Email	Password	Date of Birth
Tonny	Gaddis	tonny@yahoo.com	abcdefgh	2020-06-20

The End