Syllabus

- UNIT 6: Server Side Scripting using PHP (8 Hrs.)
  - PHP Syntax
  - Variables
  - Data Types
  - Strings
  - Constants
  - Operators
  - Control Structure
  - Functions
  - Array
  - Creating Class and Objects
  - PHP Forms
  - Accessing Form Elements
  - Form Validation
  - Events
  - Cookies and Sessions
  - Working with PHP and MySQL
  - Connecting to Database
  - Creating, Selecting, Deleting, Updating records in a table
  - Inserting Multiple Data
  - Introduction to CodeIgniter, Laravel, Wordpress, etc.

UNIT 6: Server Side Scripting using PHP

PHP

- PHP is an acronym for Hypertext Preprocessor.
- PHP is a widely-used open source scripting language that are executed on the server side.
- PHP borrowed its primary syntax from C++ & C.
- Many of the programming techniques you've previously learned will work in PHP (assignments, comparisons, loops, etc.) with little to no syntax difference.
- There are, however, major changes in how data is manipulated in relationship to C/C++.
- C/C++ are type-specific languages, requiring the user to define a specific, singular type for each variable that they use.
- PHP commonly assigns its variables "by value", meaning a variable takes on the value of the source variable (expression) and is assigned to the destination variable.
- A variable can therefore change its type "on the fly".
- Therefore, variables are not declared as they are in most type-specific languages like C/C++.

What can you do with PHP?

- Generate pages and files dynamically.
- Create, open, read, write and close files on the server.
- Collect data from a web form such as user information, email, phone number, etc.
- Send emails to the users of your website.
- Send and receive cookies to track the visitor of your website.
- Store, delete, and modify information in your database.
- Restrict unauthorized access to your website.
- Encrypt data for safe transmission over internet.
- . . . . . and many more.

Advantages of PHP over Other Languages

- Easy to learn – PHP is easy to learn and use. For beginner programmers who just started out in web development, PHP is often considered as the preferable choice of language to learn.
- Open source – PHP is an open-source project. It is developed and maintained by a worldwide community of developers who make its source code freely available to download and use.
- Portability – PHP runs on various platforms such as Microsoft Windows, Linus Mac OS, etc. and it is compatible with almost all servers used today such as Apache, IIS, etc.
- Fast Performance – Scripts written in PHP usually execute or runs faster than those written in other scripting languages like ASP, Ruby, Python, Java, etc.
- Vast Community – Since PHP is supported by the worldwide community, finding help or documentation related to PHP online is extremely easy.

PHP Syntax

- PHP has quite easy syntax, if you are familiar with any C-type language.
- It has all the same structures that you are familiar with other programming languages.
- A PHP script can be placed anywhere in the document.
- It starts with <?php and ends with ?>

```
<?php
//PHP codes;
?>
```

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

Simple PHP Example

```
<html>
<body>
<h1>My first PHP page</h1>
<?php
echo "Hello World!";
echo "<br>";
print "<br>";
print "This is my first program in PHP.";
?>
</body>
</html>
```

Comments in PHP

- To write a single-line comment either start the line with two slashes (//) or a hash symbol (#).
- However to write multi-line comments, start the comment with a slash followed by an asterisk (/*) and end the comment with an asterisk followed by a slash (*/).

```
<?php
// This is a single line comment
# This is also a single line comment
/*
This is a multiple line comment block
that spans across more than
one line
*/
echo "Hello, world!";
?>
```

Case Sensitivity in PHP

- Variable names in PHP are case-sensitive.
- As a result the variables $color, $Color, and $COLOR are treated as three different variables.
- However the keywords, functions and classes name are case-insensitive.
- As a result calling the gettype() or GETTYPE() produce the same result.

PHP Variables

- In PHP, a variable does not need to be declared before adding a value to it.
- PHP automatically converts the variable to the correct data type, depending on its value.
- After declaring a variable it can be reused throughout the code.
- The assignment operator (=) is used to assign value to a variable.
- In PHP, variable can be declared as

  $var_name = value;

PHP Variables

```
<html>
<body>
<h1>Declaring Variables</h1>
<?php
$txt = "Hello world!";
$x = 5;
$y = 10.5;
echo $txt;
echo "<br>";
echo $x;
echo "<br>";
echo $y;
?>
</body>
</html>
```

Naming Conventions for PHP Variables

- All variables in PHP start with a $ 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 in PHP can only contain alpha-numeric characters and underscores (A-z, 0-9, and _).
- A variable name cannot contain spaces.

Defining a Constant in PHP

- A constant is a name or an identifier for a fixed value.
- Constant are like variables, except that once they are defined, they cannot be undefined or changed.
- Constants are very useful for storing data that doesn't change while the script is running.
- Common examples of such data include configuration settings such as database username and password, website's base URL, company name, etc.
- Constants are defined using PHP's define() function, which accepts two arguments: the name of the constant, and its value.
- Naming conventions for PHP Constants are similar to that of PHP Variables.

Defining a Constant in PHP

```
<html>
<body>
<h1>Defining a Constant</h1>
<?php
define("SITE_URL", "https://www.texasintl.edu.np/");
echo "Thank you for visiting - " . SITE_URL;
?>
</body>
</html>
```

PHP echo & print Statements

- The echo statement can output one or more strings.
- In general terms, the echo statement can display anything that can be displayed to the browser, such as string, numbers, variables values, the results of expressions, etc.
- Like echo, you can also use the print statement to display output to the browser.
- Both echo & print statement works exactly the same way except that the print statement can only output one string, and always returns 1.
- That's why the echo statement is considered marginally faster than the print statement since it doesn't return any value.
- Since echo and print both are language construct and not actually a function (like if statement), you can use it without parentheses. For example. echo or echo(), and print or print().

PHP Data Types

- The values assigned to a PHP variable may be of different data types including simple string and numeric types to more complex data types like arrays and objects.
- PHP supports total eight primitive data types:
  - Integer
  - Float
  - String
  - Booleans
  - Array
  - Object
  - Resource &
  - NULL

PHP Strings

- A string is a sequence of letters, numbers, special characters and arithmetic values or combination of all.
- The simplest way to create a string is to enclose the string characters in single or double quotation marks (' or ").
- Strings enclosed in single-quotes are treated almost literally, whereas the strings delimited by the double quotes replaces variables with the string representations of their values as well as specially interpreting certain escape sequences.
- The escape-sequence replacements are:
  - \n is replaced by the newline character.
  - \r is replaced by the carriage-return character.
  - \t is replaced by the tab character.
  - \$ is replaced by the dollar sign itself ($).
  - \" is replaced by a single double-quote (").
  - \\ is replaced by a single backslash (\).

PHP Strings – Single & Double Quoted

```
<?php
$my_str = 'World';
// Displays: Hello World!
echo "Hello, $my_str!<br>";
// Displays: Hello, $my_str!
echo 'Hello, $my_str!<br>';
// Displays: Hello\tWorld!
echo '<pre>Hello\tWorld!</pre>';
// Displays: Hello   World!
echo "<pre>Hello\tWorld!</pre>";
// Displays: I'll be back
echo 'I\'ll be back';
?>
```

Calculating the Length of a String strlen()

```
<?php
$my_str = 'Welcome to PHP classes';
echo "The length of the string is " . strlen($my_str);
?>
```

Counting Number of Words in a String str_word_count()

```
<?php
$my_str = 'The quick brown fox jumps over the lazy dog.';
echo str_word_count($my_str);
?>
```

Replacing Text within Strings str_replace()

```
<?php
$my_str = 'If the facts do not fit the theory, change the facts.';
echo str_replace("facts", "truth", $my_str, $count);
echo "<br>";
echo "The text was replaced $count times.";
?>
```

Reversing a String – strrev()

```
<?php
$my_str = 'You can do anything, but not everything.';
echo strrev($my_str);
?>
```

Operators in PHP – Arithmetic

Operator	Description	Example	Result
+	Addition	$x + $y	Sum of $x and $y
-	Subtraction	$x - $y	Difference of $x and $y
*	Multiplication	$x * $y	Product of $x and $y
/	Division	$x / $y	Quotient of $x and $y
%	Modulus	$x % $y	Remainder of $x and $y

```
<?php
$x = 10;
$y = 4;
echo($x + $y); // Outputs: 14
echo($x - $y); // Outputs: 6
echo($x * $y); // Outputs: 40
echo($x / $y); // Outputs: 2.5
echo($x % $y); // Outputs: 2
?>
```

Operators in PHP – Assignment

Operator	Description	Example	Is the same as
=	Assign	$x = $y	$x = $y
+=	Add and assign	$x += $y	$x = $x + $y
-=	Subtract and assign	$x -= $y	$x = $x - $y
*=	Multiply and assign	$x *= $y	$x = $x * $y
/=	Divide and assign quotient	$x /= $y	$x = $x / $y
%=	Divide and assign modulus	$x %= $y	$x = $x % $y

```
<?php
$x = 10;
echo $x;
// Outputs: 10
$x = 20;
$x += 30;
echo $x;
// Outputs: 50
```

```
$x = 50;
$x -= 20;
echo $x;
// Outputs: 30
$x = 5;
$x *= 25;
echo $x;
// Outputs: 125
```

```
$x = 50;
$x /= 10;
echo $x;
// Outputs: 5
$x = 100;
$x %= 15;
echo $x;
// Outputs: 10
?>
```

Operators in PHP – Comparison

Operator	Name	Example	Result
==	Equal	$x == $y	True if $x is equal to $y
===	Identical	$x === $y	True if $x is equal to $y; and they are of the same type
!=	Not equal	$x != $y	True if $x is not equal to $y
<>	Not equal	$x <> $y	True if $x is not equal to $y
!==	Not Identical	$x !== $y	True if $x is not equal to $y; or they are not of the same type
<	Less than	$x < $y	True if $x is less than $y
>	Greater than	$x > $y	True if $x is greater than $y
>=	Greater than or equal to	$x >= $y	True if $x is greater than or equal to $y
<=	Less than or equal to	$x <= $y	True if $x is less than or equal to $y

Operators in PHP – Comparison

```
<?php
$x = 25;
$y = 35;
$z = "25";
var_dump($x == $z);  // Outputs: boolean true
var_dump($x === $z); // Outputs: boolean false
var_dump($x != $y);  // Outputs: boolean true
var_dump($x !== $z); // Outputs: boolean true
var_dump($x < $y);   // Outputs: boolean true
var_dump($x > $y);   // Outputs: boolean false
var_dump($x <= $y);  // Outputs: boolean true
var_dump($x >= $y);  // Outputs: boolean false
?>
```

Operators in PHP
Increment & Decrement

Operator	Name	Effect
++$x	Pre-increment	Increments $x by one, then returns $x
$x++	Post-increment	Returns $x, then increments $x by one
--$x	Pre-decrement	Decrements $x by one, then returns $x
$x--	Post-decrement	Returns $x, then decrements $x by one

```
<?php
$x = 10;
echo ++$x; // Outputs: 11
echo $x;   // Outputs: 11
$x = 10;
echo $x++; // Outputs: 10
echo $x;   // Outputs: 11
```

```
$x = 10;
echo --$x; // Outputs: 9
echo $x;   // Outputs: 9
$x = 10;
echo $x--; // Outputs: 10
echo $x;   // Outputs: 9
?>
```

Operators in PHP – Logical

Operator	Name	Example	Result
and	And	$x and $y	True if both $x and $y are true
or	Or	$x or $y	True if either $x or $y is true
xor	Xor	$x xor $y	True if either $x or $y is true, but not both
&&	And	$x && $y	True if both $x and $y are true
||	Or	$x || $y	True if either $x or $y is true
!	Not	!$x	True if $x is not true

Operators in PHP – String

Operator	Description	Example	Result
.	Concatenation	$str1.$str2	Concatenation of $str1 and $str2
.=	Concatenation assignment	$str1.=$str2	Appends the $str2 to the $str1

```
<?php
$x = "Hello";
$y = "World!";
echo $x . $y;
// Output: Hello World!
$x .= $y;
echo $x;
//Output: Hello World!
?>
```

PHP Conditional Statements

- The if statement

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

- The if . . . else statement

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

PHP Conditional Statements

- The if . . . elseif . . . else statement

```
if (condition1){
        //code to be executed if condition1 is true
} elseif(condition2){
        //code to be executed if condition2 is true
} else{
        //code to be executed if both conditions are false
}
```

- The switch . . . case statement

```
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
}
```

PHP Conditional Statements – Example

- The if statement

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

PHP Conditional Statements – Example

- The if . . . else statement

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

PHP Conditional Statements – Example

- The if . . . elseif . . . else statement

```
<?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!";
}
?>
```

PHP Conditional Statements – Example

- The switch . . . case statement

```
<?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;
}
?>
```

PHP Ternary Operator (?)

- The if . . . else statement

```
<?php
$age=18;
if($age < 18){
    echo 'Child'; // Display Child if age is less than 18
} else{
    echo 'Adult'; // Display Adult if age is greater than or

equal to 18
}
?>
```

- The ternary operator statement

```
echo ($age < 18) ? 'Child' : 'Adult';
```

PHP Arrays

- Arrays are complex variables that allow us to store more than one value or a group of values under a single variable name.
- Storing the colors one by one in a variable could look like this:

```
<?php
$color1 = "red";
$color2 = "green";
$color3 = "blue";
?>
```

- But what if you want to store more than three colors?
- It is quite hard, boring, and a bad idea to store each colors in a separate variable.
- And here array comes into play.
- Types of Arrays
  - Indexed Array – array with a numeric key
  - Associative Array – array where each key has its own specific value
  - Multidimensional Array – array containing one or more arrays within itself

PHP Arrays – Indexed Arrays

- An indexed (or numeric) array stores each array element with a numeric index.
- Storing the colors one by one in a variable could look like this:

```
<?php
$color1 = "red";
$color2 = "green";
$color3 = "blue";
?>
```

- The following example shows how an indexed array is created:

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

PHP Arrays – Associative Arrays

- In an associative array, the keys assigned to values can be arbitrary and user defined strings.
- In the following example the array uses keys instead of index numbers:

```
<?php
$ages = array("Peter"=>22, "Clark"=>32, "John"=>28);
?>
```

- The following example displays the associative array:

```
<?php
$ages = array("Peter"=>22, "Clark"=>32, "John"=>28);
foreach($ages as $key_arr=>$val_arr)
echo $key_arr ."=" . $val_arr . "<br>";
?>
```

PHP Arrays – Multidimensional Arrays

- An array in which each element can also be an array and each element in the sub-array can be an array or further contain array within itself and so on.

```
<?php
$contacts = array(
    array(
        "name" => "Peter Parker",
        "email" => "peterparker@mail.com",
    ),
    array(
        "name" => "Harry Potter",
        "email" => "harrypotter@mail.com",
    )
);
echo "His/Her name is ".$contacts[0]["name"] . " and email-id is: " .
$contacts[0]["email"];
echo "<br>";
echo "His/Her name is ".$contacts[1]["name"] . " and email-id is: " .
$contacts[1]["email"];
?>
```

Sorting PHP Arrays

- PHP comes with a number of built-in functions designed specifically for sorting array elements in different ways like alphabetically or numerically in ascending or descending order.
- **sort()** and **rsort()** – for sorting indexed arrays
- **asort()** and **arsort()** – for sorting associative arrays by value
- **ksort()** and **krsort()** – for sorting associative arrays by key

Sorting PHP Arrays

- **Sorting Indexed Arrays**

```
<?php
// Define array
$colors = array("Red", "Green", "Blue", "Yellow");
// Sorting and printing array
sort($colors);          // Ascending Order
print_r($colors);
echo "<br>";
rsort($colors);         // Descending Order
print_r($colors);
?>
```

- **Output**

```
Array ( [0] => Blue [1] => Green [2] => Red [3] => Yellow )
Array ( [0] =>Yellow [1] => Red [2] => Green [3] => Blue )
```

Sorting PHP Arrays

- **Sorting Associative Arrays by Value**

```
<?php
// Define array
$age = array("Peter"=>20, "Harry"=>14, "John"=>45, "Clark"=>35);
// Sorting array by value and print
asort($age);  // Ascending Order
print_r($age);
echo "<br>";
arsort($age);  // Descending Order
print_r($age);
?>
```

- **Output**

```
Array ( [Harry] => 14 [Peter] => 20 [Clark] => 35 [John] => 45 )
Array ( [John] => 45 [Clark] => 35 [Peter] => 20 [Harry] => 14 )
```

Sorting PHP Arrays

- **Sorting Associative Arrays by Key**

```
<?php
// Define array
$age = array("Peter"=>20, "Harry"=>14, "John"=>45, "Clark"=>35);
// Sorting array by key and print
ksort($age);  // Ascending Order
print_r($age);
echo "<br>";
krsort($age);  // Descending Order
print_r($age);
?>
```

- **Output**

```
Array ( [Clark] => 35 [Harry] => 14 [John] => 45 [Peter] => 20 )
Array ( [Peter] => 20 [John] => 45 [Harry] => 14 [Clark] => 35 )
```

PHP Loops

- Loops are used to execute the same block of code again and again, until a certain condition is met.
- The basic idea behind a loop is to automate the repetitive tasks within a program to save the time and effort.
- PHP supports four different types of loops:
  - while – loops through a block of code until the condition is evaluate to true.
  - do . . . while – the block of code executes once and then condition is evaluated. If the condition is true the statement is repeated as long as the specified condition is true.
  - for – loops through a block of code until the counter reaches a specified number.
  - foreach – loops through a block of code for each element in an array.

PHP Loops – while

- Syntax:

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

- Example:

```
<?php
$i = 1;
while($i <= 3){
    $i++;
    echo "The number is " . $i . "<br>";
}
?>
```

PHP Loops – do...while

- Syntax:

```
do{
    // Code to be executed
}
while(condition);
```

- Example:

```
<?php
$i = 1;
do{
    $i++;
    echo "The number is " . $i . "<br>";
}
while($i <= 3);
?>
```

PHP Loops – for

- Syntax:

```
for(initialization; condition; increment){
    // Code to be executed
}
```

- Example:

```
<?php
for($i=1; $i<=3; $i++){
    echo "The number is " . $i . "<br>";
}
?>
```

PHP Loops – foreach

- Syntax:

```
foreach($array as $value){
    // Code to be executed
}
```

- Example:

```
<?php
$colors = array("Red", "Green", "Blue");
foreach($colors as $value){
    echo $value . "<br>";
}
?>
```

PHP Functions

- A function is a self-contained block of code that performs a specific task.
- PHP has a huge collection of internal or built-in functions that you can call directly within your PHP scripts to perform a specific task, like gettype(), print_r(), var_dump, etc.
- In addition to the built-in functions, PHP also allows you to define your own functions.
- It is a way to create reusable code packages that perform specific tasks and can be kept and maintained separately from main program.

PHP Functions – Advantages

- Functions reduces the repetition of code within a program
  - It allows you to extract commonly used block of code into a single component.
  - Now you can perform the same task by calling this function wherever you want within your script without having to copy and paste the same block of code again and again.
- Functions makes the code much easier to maintain
  - Since a function created once can be used many times, so any changes made inside a function automatically implemented at all the places without touching the several files.
- Functions makes it easier to eliminate the errors
  - When the program is subdivided into functions, if any error occur you know exactly what function is causing the error and where to find it.
  - Therefore, fixing errors becomes much easier.
- Functions can be reused in other application
  - A function is separated from the rest of the script
  - So it's easy to reuse the same function in other applications just by including the php file containing those functions.

PHP Functions – Example

```
<?php
// Defining function
function whatIsToday(){
    echo "Today is " . date("l");
}
whatIsToday();
?>
```

PHP Functions (with parameters) – Example

```
<?php
// Defining function
function getSum($num1, $num2){
$sum = $num1 + $num2;
echo "Sum of the two numbers $num1 and $num2 is : $sum";
}
// Calling function
getSum(10, 20);
?>
```

PHP date() method

d - The day of the month (from 01 to 31)

D - A textual representation of a day (three letters)

l (lowercase 'L') - A full textual representation of a day

w - A numeric representation of the day (0 for Sunday, 6 for Saturday)

z - The day of the year (from 0 through 365)

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)

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

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)

PHP GET and POST

- GET and POST are the methods that sends information to Server.
- Both GET and POST are HTTP methods
- Both methods pass the information differently and have different advantages and disadvantages.
- GET method carries request parameter appended in URL string **WHEREAS**, POST method carries request parameter in message body which makes it more secure way of transferring data from client to server.
- Both GET and POST are treated as $_GET and $_POST. These are superglobals, which means that they are always accessible, regardless of scope – and can be accessed from any function, class, or file without having to do anything special.

PHP GET Method

- In GET method the data is sent as URL parameters that are usually strings of name and value pairs separated by ampersands (&).
- In general, a URL with GET data will look like this:

  http://www.example.com/action.php?**name**=*john*&**age**=*24*
- The bold parts in the URL are the GET parameters and the italic parts are the value of those parameters.
- More than one parameter=value can be embedded in the URL by concatenating with ampersands (&).
- One can only send simple text data via GET method.
- The GET method is restricted to send up to 1024 characters only.

PHP GET Method
Advantages & Disadvantages

- Since the data sent by the GET method are displayed in the URL, it is possible to bookmark the page with specific query string values.
- The GET method is not suitable for passing sensitive information such as the username and password, because these are fully visible in the URL query string as well as potentially stored in the client browser's memory as a visited page.
- Because the GET method assigns data to a server environment variable, the length of the URL is limited. So there is a limitation for the total data to be sent.
- PHP provides the super global variable $_GET to access all the information sent either through the URL or submitted through an HTML form using the method="GET"

PHP POST Method

- In POST method the data is sent to the server as a package in a separate communication with the processing script.
- Data sent through POST method will not be visible in the URL.
- There is no restriction on data size to be sent in the POST method.
- 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.

PHP POST Method
Advantages & Disadvantages

- It is more secure than GET because user-entered information is never visible in the URL query string or in the server logs.
- There is a much larger limit on the amount of data that can be passed and one can send text data as well as binary data (uploading a file) using POST.
- Since the data sent by the POST method is not visible in the URL, it is not possible to bookmark the page with specific query.
- PHP provides the super global variable $_POST to access all the information sent via post method or submitted through an HTML form using the method="POST"

Handling Errors

- Sometimes your application will not run as it is supposed to do, resulting an error.
- There are number of reasons that may cause errors, for example:
  - The web server might run out of disk space.
  - A user might have entered an invalid value in a form field.
  - The file or database record that you were trying to access may not exist.
  - The application might not have permission to write to a file on the disk.
  - A service that the application needs to access might be temporarily unavailable.
- These types of errors are known as runtime errors, because they occur at the time the script runs.
- They are distinct from syntax errors that needs to be fixed before the script will run.
- A professional application must have the capabilities to handle such runtime error gracefully.
- Usually this means informing the user about the problem more clearly and precisely.

Handling Errors – die() Function

```
<?php
//Try to open a non-existent file
$file = fopen("sample.txt", "r");
?>
```

- If the file does not exist you will get an error like this:

  Warning: fopen(sample.txt): failed to open stream: No such file or directory in **C:\xampp1\htdocs\project\DEMO.php** on line **3**

- If we follow some simple steps we can prevent the users from getting such error message.

```
<?php
if(file_exists("sample.txt")){
    $file = fopen("sample.txt", "r");
} else{
    die("Error: The file you are trying to access doesn't exist.");
}
?>
```

- Now if you run the above script, you will get the error message like this:

  Error: The file you are trying to access doesn't exist.

Sessions & State

- A session is a way to store information (in variables) to be used across multiple pages.
- Unlike a cookie, the information is not stored on the users computer.
- When you work with an application, you open it, do some changes, and then you close it. This is much like a session.
- The computer knows who you are and it knows when you start application and when you end it.
- But on the internet there is a problem: the web server does not know who you are or what you do, because the HTTP address doesn't maintain state.
- Session variables solve this problem by storing user information to be used across multiple pages.
- By default, session variables last until the user closes the browser.
- So, session variables hold information about one single user, and are available to all pages in one application.

PHP Session – Start

- A session is started with the session_start() function.
- The session_start() function must be the very first thing in your document before any HTML tags.

```
<?php
// Start the session
session_start();
?>
<?php
// Set session variables
$_SESSION["favcolor"] = "green";
$_SESSION["favanimal"] = "cat";
echo "Session variables are set.";
?>
```

PHP – Get Session Variable Values

- We create another page and will access the session information that we set on the first page.

```
<?php
session_start();
?>
<?php
// Echo session variables that were set on previous page
echo "Favorite color is " . $_SESSION["favcolor"] . ".<br>";
echo "Favorite animal is " . $_SESSION["favanimal"] . ".";
?>
```

- Another way to see all the session variable values that has been set:

```
<?php
session_start();
?>
<?php
print_r($_SESSION);
?>
```

PHP Session – Modify

- To change a session variable, just overwrite it.

```
<?php
session_start();
?>
<?php
// to change a session variable, just overwrite it
$_SESSION["favcolor"] = "yellow";
print_r($_SESSION);
?>
```

PHP Session – Destroy

- To remove all global session variables and destroy the session, we use session_unset() and session_destroy().

```
<?php
session_start();
?>
<?php
// remove all session variables
session_unset();
// destroy the session
session_destroy();
echo "All session variables are now removed, and the session is destroyed."
?>
```

Database Connectivity

- In order to store or access the data inside a MySQL database, you first need to connect to the MySQL database server.
- PHP offers two different ways to connect to MySQL server: MySQLi (Improved MySQL) and PDO (PHP Data Objects) extensions.
- While the PDO extension is more portable and supports more than twelve different databases, MySQLi extension as the name suggests supports MySQL database only.
- MySQLi extension however provides an easier way to connect to, and execute queries on, a MySQL database server.

Connecting to MySQL Database

- In PHP, you can easily connect to MySQL database using the mysqli_connect() function.
- All communication between PHP and the MySQL database server takes place through this connection.
- The connection to the MySQL database server will be closed automatically as soon as the execution of the script ends.
- However, if you want to close it earlier, you can do this by simply calling the PHP mysqli_close() function.

Connecting to MySQL Database

- Basic Syntax (MySQLi, Procedural way)

```
$link = mysqli_connect("hostname", "username", "password", "database");
```

- Basic Syntax (MySQLi, Object Oriented way)

```
$mysqli = new mysqli("hostname", "username", "password", "database");
```

- Basic Syntax (PHP Data Objects way)

```
$pdo = new PDO("mysql:host=hostname;dbname=database", "username", "password");
```

Connecting to MySQL Database

```
<?php
$link = mysqli_connect("localhost", "root", "");
// Check connection
if($link === false){
die("ERROR: Could not connect. " . mysqli_connect_error());
}
// Print host information
echo "Connect Successfully. Host info: " .
mysqli_get_host_info($link);
// Close connection
mysqli_close($link);
?>
```

PHP MySQL Create Database

```
<?php
/* Attempt MySQL server connection. Assuming you are running MySQL
server with default setting (user 'root' with no password) */
$link = mysqli_connect("localhost", "root", "");
// Check connection
if($link === false){
die("ERROR: Could not connect. " . mysqli_connect_error());
}
// Attempt create database query execution
$sql = "CREATE DATABASE demo";
if(mysqli_query($link, $sql)){
echo "Database created successfully";
} else{
echo "ERROR: Could not able to execute $sql. " . mysqli_error($link);
}
// Close connection
mysqli_close($link);
?>
```

PHP MySQL Create Table

```
<?php
/* Attempt MySQL server connection. Assuming you are running MySQL
server with default setting (user 'root' with no password) */
$link = mysqli_connect("localhost", "root", "", "demo");
// Check connection
if($link === false){
die("ERROR: Could not connect. " . mysqli_connect_error());
}
// Attempt create table query execution
$sql = "CREATE TABLE persons(
id INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
first_name VARCHAR(30) NOT NULL,
last_name VARCHAR(30) NOT NULL,
email VARCHAR(70) NOT NULL UNIQUE
)";
if(mysqli_query($link, $sql)){
echo "Table created successfully.";
} else{
echo "ERROR: Could not able to execute $sql. " . mysqli_error($link);
}
// Close connection
mysqli_close($link);
?>
```

PHP MySQL Insert Data

```
<?php
/* Attempt MySQL server connection. Assuming you are running MySQL
server with default setting (user 'root' with no password) */
$link = mysqli_connect("localhost", "root", "", "demo");
// Check connection
if($link === false){
die("ERROR: Could not connect. " . mysqli_connect_error());
}
// Attempt insert query execution
$sql = "INSERT INTO persons (first_name, last_name, email) VALUES ('Peter', 'Parker', 'peterparker@mail.com')";
if(mysqli_query($link, $sql)){
echo "Records inserted successfully.";
} else{
echo "ERROR: Could not able to execute $sql. " . mysqli_error($link);
}
// Close connection
mysqli_close($link);
?>
```

PHP MySQL Insert Multiple Data

```
<?php
/* Attempt MySQL server connection. Assuming you are running MySQL
server with default setting (user 'root' with no password) */
$link = mysqli_connect("localhost", "root", "", "demo");
// Check connection
if($link === false){
die("ERROR: Could not connect. " . mysqli_connect_error());
}
// Attempt insert query execution
$sql = "INSERT INTO persons (first_name, last_name, email) VALUES
('John', 'Rambo', 'johnrambo@mail.com'),
('Clark', 'Kent', 'clarkkent@mail.com'),
('John', 'Carter', 'johncarter@mail.com'),
('Harry', 'Potter', 'harrypotter@mail.com')";
if(mysqli_query($link, $sql)){
echo "Records added successfully.";
} else{
echo "ERROR: Could not able to execute $sql. " . mysqli_error($link);
}
// Close connection
mysqli_close($link);
?>
```

PHP MySQL Select Query

```
<?php
/* Attempt MySQL server connection. Assuming you are running MySQL
server with default setting (user 'root' with no password) */
$link = mysqli_connect("localhost", "root", "", "demo");
// Check connection
if($link === false){
die("ERROR: Could not connect. " . mysqli_connect_error());
}
// Attempt select query execution
$sql = "SELECT * FROM persons";
if($result = mysqli_query($link, $sql)){
if(mysqli_num_rows($result) > 0){
echo "<table>";
echo "<tr>";
echo "<th>id</th>";
echo "<th>first_name</th>";
echo "<th>last_name</th>";
echo "<th>email</th>";
echo "</tr>";
while($row = mysqli_fetch_array($result)){
echo "<tr>";
echo "<td>" . $row['id'] . "</td>";
echo "<td>" . $row['first_name'] . "</td>";
echo "<td>" . $row['last_name'] . "</td>";
echo "<td>" . $row['email'] . "</td>";
echo "</tr>";
}
echo "</table>";
// Free result set
mysqli_free_result($result);
} else{
echo "No records matching your query were found.";
}
} else{
echo "ERROR: Could not able to execute $sql. " . mysqli_error($link);
}
// Close connection
mysqli_close($link);
?>
```

PHP MySQL Update Data in Table

```
<?php
/* Attempt MySQL server connection. Assuming you are running MySQL
server with default setting (user 'root' with no password) */
$link = mysqli_connect("localhost", "root", "", "demo");
// Check connection
if($link === false){
die("ERROR: Could not connect. " . mysqli_connect_error());
}
// Attempt update query execution
$sql = "UPDATE persons SET email='peterparker_new@mail.com' WHERE id=1";
if(mysqli_query($link, $sql)){
echo "Records were updated successfully.";
} else {
echo "ERROR: Could not able to execute $sql. " . mysqli_error($link);
}
// Close connection
mysqli_close($link);
?>
```

PHP MySQL Delete Data in Table

```
<?php
/* Attempt MySQL server connection. Assuming you are running MySQL
server with default setting (user 'root' with no password) */
$link = mysqli_connect("localhost", "root", "", "demo");
// Check connection
if($link === false){
die("ERROR: Could not connect. " . mysqli_connect_error());
}
// Attempt delete query execution
$sql = "DELETE FROM persons WHERE first_name='John'";
if(mysqli_query($link, $sql)){
echo "Records were deleted successfully.";
} else{
echo "ERROR: Could not able to execute $sql. " . mysqli_error($link);
}
// Close connection
mysqli_close($link);
?>
```

PHP MySQL Delete Table

```
<?php
$link = mysqli_connect("localhost", "root", "", "demo");
// Check connection
if($link === false){
die("ERROR: Could not connect. " . mysqli_connect_error());
}
// Attempt delete table query execution
$sql = "DROP TABLE persons;";
if(mysqli_query($link, $sql)){
echo "Table deleted successfully.";
} else{
echo "ERROR: Could not able to execute $sql. " . mysqli_error($link);
}
// Close connection
mysqli_close($link);
?>
```

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 visits the website next time.
- Each time the browser requests a page to the server, all the data in the cookie is automatically sent to the server within the request.

Setting a Cookie in PHP

- The setcookie() function is used to set a cookie in PHP.
- Make sure you call the setcookie() function before any output generated by your script otherwise cookie will not set.
- The basic syntax of this function is as follow:

  setcookie(name, value, expire, path, domain, secure);
- If the expiration time of the cookie is set to 0, or omitted, the cookie will expire at the end of the session. i.e. when the browser closes.

```
<?php
// Setting a cookie
setcookie("username", "John Carter", time()+30*24*60*60);
echo “Cookie has been set.”;
?>
```

Setting a Cookie in PHP

- All the arguments except the name are optional.
- You may also replace an argument with an empty string ("") in order to skip that argument, however to skip the expire argument use a zero (0) instead, since it is an integer.

Parameter	Description
name	The name of the cookie.
value	The value of the cookie.
expires	The expiry date of the cookie. 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. E.g. www.example.com
secure	Indicates that the cookie should be sent only if a secure HTTPS connection exists.

Accessing Cookies Values in PHP

- The PHP $_COOKIE superglobal variable is used to retrieve a cookie value.
- The individual cookie value can be accessed using standard array notation.

```
<?php
// Accessing an individual cookie value
echo $_COOKIE["username"];
?>
```

- An example for accessing cookie:

```
<?php
// Verifying whether a cookie is set or not
if(isset($_COOKIE["username"])){
    echo "Hi " . $_COOKIE["username"];
} else{
    echo "Welcome Guest!";
}
?>
```

Removing Cookies in PHP

- You can delete a cookie by calling the same setcookie() function with the cookie name and any value (such as an empty string).
- However this time you need to set the expiration date in the past.

```
<?php
// Deleting a cookie
setcookie("username", "", time()-3600);
?>
```

- You should pass exactly the same path, domain, and other arguments that you have used when you first created the cookie in order to ensure that the correct cookie is deleted.

File Handling in PHP – Opening

- The PHP fopen() function is used to open a file.
- The basic syntax of this function is:

  fopen(filename, mode)
- The first parameter passed to fopen() specifies the name of the file you want to open and the second parameter specifies which mode the file should be opened.
- For example:

```
<?php
$handle = fopen("data.txt", "r");
echo "File opened successfully.";
?>
```

File Handling in PHP – Opening

- If you try to open a file that doesn't exist, PHP will generate a warning message.
- To avoid these error messages you should always implement a simple check whether a file or directory exists or not before trying to access it, with the PHP file_exists() function.

```
<?php
$file = "data.txt";
// Check the existence of file
if(file_exists($file)){
    echo "The file $file exists." . "<br>";
    // Attempt to open the file
    $handle = fopen($file, "r") or die("ERROR: Cannot open the file.");
    if($handle){
        echo "File opened successfully.";
        // Closing the file handle
        fclose($handle);
    }
} else{
    echo "ERROR: The file $file does not exist.";
}
?>
```

File Handling in PHP – Opening

- The file may be opened in one of the following modes:

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 doesn't exist, PHP will attempt to create it.
w+	Open the file for reading and writing and clears the contents of file. If the file doesn't 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 doesn't 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 doesn't exist, PHP will attempt to create it.
x	Opens the file for writing only. Return FALSE and generates an error if the file already exists. If the file doesn't exist, PHP will attempt to create it.
x+	Open the file for reading and writing; otherwise it has the same behavior as 'x'.

File Handling in PHP – Closing

- Once you've finished working with a file, it needs to be closed.
- The fclose() function is used to close the file, as seen in the previous example.
- Although PHP automatically closes all open files when script terminates, but it's a good practice to close a file after performing all the operations.

File Handling in PHP – Reading

- The fread() function can be used to read a specified number of characters from a file.
- The basic syntax of this function can be given as:

  fread(file handle, length in bytes)
- This function takes two parameter – a file handle and the number of bytes to read.
- The following example reads 20 bytes from the "data.txt" file including spaces.

  fread($handle, "20");
- The fread() function can be used in conjugation with the filesize() function to read the entire file at once.
- The fiesize() function returns the size of the file in bytes.

  fread($handle, filesize($file));

File Handling in PHP – Reading

- The easiest way to read the entire contents of a file in PHP is with the readfile() function.
- This function allows you to read the contents of a file without needing to open it.

  readfile($file)
- Another way to read the whole content of a file without needing to open it is with the file_get_contents() function.

  file_get_contents($file)

File Handling in PHP – Reading

```php
<?php
$file = "data.txt";
// Check the existence of file
if(file_exists($file)){
    // Open the file for reading
    $handle = fopen($file, "r");
    // Reading the entire file
    $content = fread($handle, filesize($file));
    // Display the file content
    echo $content;
    // Closing the file handle
    fclose($handle);
} else{
    echo "ERROR: File does not exist.";
}
?>
```

```php
<?php
$file = "data.txt";
// Check the existence of file
if(file_exists($file)){
    // Reads and outputs the entire file
    readfile($file);
} else{
    echo "ERROR: File does not exist.";
}
?>
```

File Handling in PHP – Writing

- The fwrite() function can be used to write data to a file or append to an existing file using PHP.
- The basic syntax of this function can be given as:

  fwrite(file handle, string)
- This function takes two parameter – a file handle and the string of data that is to be written.
- The following example writes the message to the "data.txt" file.

  fwrite($handle, "This line will be written in the file");
- An alternative way is using the file_put_contents() function which is the counterpart of file_get_contents() function and provides an easy method of writing the data to a file without needing to open it.

File Handling in PHP – Writing

```
<?php
$file = "data.txt";
// String of data to be written
$data = "The quick brown fox jumps over the lazy dog.";
// Open the file for writing
$handle = fopen($file, "w");
// Write data to the file
fwrite($handle, $data);
// Closing the file handle
fclose($handle);
echo "Data written to the file successfully.";
?>
```

File Handling in PHP – Renaming

- The rename() function can be used to rename a file or directory using PHP.

```
<?php
$file = "data.txt";
// Check the existence of file
if(file_exists($file)){
    // Attempt to rename the file
    if(rename($file, "newfile.txt")){
        echo "File renamed successfully.";
    } else{
        echo "ERROR: File cannot be renamed.";
    }
} else{
    echo "ERROR: File does not exist.";
}
?>
```

File Handling in PHP – Deleting

- The unlink() function can be used to delete a file or directory using PHP.

```
<?php
$file = "newfile.txt";
// Check the existence of file
if(file_exists($file)){
    // Attempt to delete the file
    if(unlink($file)){
        echo "File removed successfully.";
    } else{
        echo "ERROR: File cannot be removed.";
    }
} else{
    echo "ERROR: File does not exist.";
}
?>
```

Form Handling in PHP

```
<html>
<head>
<title>Contact Form</title>
</head>
<body>
<h2>Contact Us</h2>
<p>Please fill in this form and send us.</p>
<form action="process-form.php" method="post">
<p>
<label for="inputName">Name:<sup>*</sup></label>
<input type="text" name="name" id="inputName">
</p>
<p>
<label for="inputEmail">Email:<sup>*</sup></label>
<input type="text" name="email" id="inputEmail">
</p>
<p>
<label for="inputSubject">Subject:</label>
<input type="text" name="subject" id="inputSubject">
</p>
<p>
<label for="inputComment">Message:<sup>*</sup></label>
<textarea name="message" id="inputComment" rows="5" cols="30"></textarea>
</p>
<input type="submit" value="Submit">
<input type="reset" value="Reset">
</form>
</body>
</html>
```

Form Handling in PHP

```
<html>
<head>
<title>Contact Form</title>
</head>
<body>
<h1>Thank You</h1>
<p>Here is the information you have submitted:</p>
<ol>
<li><em>Name:</em> <?php echo $_POST["name"]?></li>
<li><em>Email:</em> <?php echo $_POST["email"]?></li>
<li><em>Subject:</em> <?php echo $_POST["subject"]?></li>
<li><em>Message:</em> <?php echo $_POST["message"]?></li>
</ol>
</body>
</html>
```

END OF UNIT SIX