# PHP

Hypertext Preprocessor

# About PHP

- ✓ PHP is a widely-used, free and open source scripting language
- ✓ PHP scripts are executed on the server
- ✓ It is free to download and use
- ✓ PHP files can contain text, HTML, JavaScript code, and PHP code
- ✓ PHP code are executed on the server, and the result is returned to the browser as plain HTML
- ✓ PHP files have a default file extension of ".php“
- ✓ PHP can generate dynamic page content
- ✓ PHP can create, open, read, write, and close files on the server
- ✓ PHP can collect form data
- ✓ PHP can send and receive cookies
- ✓ PHP can add, delete, modify data in your database
- ✓ PHP can restrict users to access some pages on your website
- ✓ PHP can encrypt data
- ✓ PHP runs on different platforms (Windows, Linux, Unix, Mac OS X, etc.)
- ✓ PHP is compatible with almost all servers used today (Apache, IIS, etc.)
- ✓ PHP has support for a wide range of databases
- ✓ Download PHP from “www.php.net”
- ✓ PHP is easy to learn and runs efficiently on the server side

# PHP Syntax

- ✓ A PHP script always starts with **<?php** and ends with **?>**.
- ✓ A PHP script can be placed anywhere in the document.
- ✓ The default file extension for PHP files is ".php".
- ✓ A PHP file normally contains HTML tags, and some PHP scripting code.

```
<?php
// sample PHP code
?>
```

# PHP Example

```
<!DOCTYPE html>
<html>
<body>
<?php
echo "Hello World!";
?>
</body>
</html>
```

- ✓ Use echo and print to output the text in PHP.
- ✓ Use // for single line comment
- ✓ Use /* */ for comment block

# PHP Variables

- ✓ PHP variables are used to hold values or expressions.
- ✓ Variables in PHP starts with a $ sign, followed by the name of the variable.
- ✓ The variable name must begin with a letter or the underscore character.
- ✓ A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _ ).
- ✓ A variable name should not contain spaces.
- ✓ Variable names are case sensitive.
- ✓ PHP is a Loosely Typed Language

```
<?php
$str=“Web Enabled Technology";
$num=10;
$add=30+40;
$new=add*add;
?>
```

# PHP Variables Scope

The scope of a variable is the portion of the script in which the variable can be referenced.

**Local:** A variable declared **within** a PHP function is local and can only be accessed within that function.

```
<?php
function myTest()
{
$num = 5;
echo $num;
}
myTest();
?>
```

**Output:**
**5**

- ✓ We can have local variables with the same name in different functions.
- ✓ Local variables are deleted as soon as the function is completed.

**Global:**

```
<?php
$a = 1; /* global scope */
function test()
{
echo $a;  /* reference to local scope variable  */
}
test();
?>
```

- ✓ This script will not produce any output because the echo statement refers to a local version of the $a variable, and it has not been assigned a value within this scope.
- ✓ In PHP global variables must be declared global inside a function if they are going to be used in that function.

# Global keyword

```
<!DOCTYPE html>
<html>
<body>
<?php
$x = 5; // global scope
function myTest() {
echo "<p>Variable x inside function
is: $x</p>";
}
myTest();
echo "<p>Variable x outside
function is: $x</p>";
?>
</body>
</html>
```

Output:
Variable x inside function is:
Variable x outside function is: 5

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

Output:
Variable x inside function is: 5
Variable x outside function is:

## Global keyword

```
<!DOCTYPE html>
<html>
<body>
<?php
$x = 5;
$y = 10;
function myTest() {
  global $x, $y;
  $y = $x + $y;
}
myTest();  // run function
echo $y; // output the new value for
variable $y
?>
</body>
</html>
```

Output:
15

## Using $GLOBALS

```
<?php
$x = 5;
$y = 10;

function myTest() {
  $GLOBALS['y'] = $GLOBALS['x'] + $GLOBALS['y'];
}

myTest();
echo $y; // outputs 15
?>
```

Output:
15

# PHP Variables Scope

**Static:** A static variable exists only in a local function scope, but it does not lose its value when program execution leaves this scope.

```
<?php
function myTest() {
  static $x = 0;
  echo $x;
  $x++;
}
myTest();
myTest();
myTest();
?>
```

Output:
0
1
2

# PHP Print Statement

```
<!DOCTYPE html>
<html>
<body>

<?php
print "<h2>PHP is Fun!</h2>";
print "Hello world!<br>";
print "I'm about to learn PHP!";
?>
</body>
</html>
```

Output:

Hello world!
I'm about to learn PHP!

```
<!DOCTYPE html>
<html>
<body>
<?php
$txt1 = "Learn PHP";
$txt2 = "Web Enabled Technologies";
$x = 5;
$y = 4;
print "<h2>" . $txt1 . "</h2>";
print "Study for " . $txt2 . "<br>";
print $x + $y;
?>
</body>
</html>
```

Output:

**Learn PHP**

Study for Web Enabled Technologies
9

# PHP Data Types

Variables can store data of different types, and different data types can do different things.
PHP supports the following data types:

- String
- Integer
- Float (floating point numbers - also called double)
- Boolean
- Array
- Object
- NULL
- Resource

- A string is a sequence of characters.
- A string can be any text inside quotes.
- You can use single or double quotes:

```
<!DOCTYPE html>
<html>
<body>
<?php
$x = "Hello world!";
$y = 'Hello world!';
echo $x;
echo "<br>";
echo $y;
?>
</body>
</html>
```

Output:

Hello world!
Hello world!

An integer data type is a non-decimal number between -2,147,483,648 and 2,147,483,647.

Rules for integers:

- An integer must have at least one digit
- An integer must not have a decimal point
- An integer can be either positive or negative
- Integers can be specified in: decimal (base 10), hexadecimal (base 16), octal (base 8), or binary (base 2) notation

```
<!DOCTYPE html>
<html>
<body>
<?php
$x = 5985;
var_dump($x);
?>
</body>
</html>
```

Output:

int(5985)

A float (floating point number) is a number with a decimal point or a number in exponential form.

```
<!DOCTYPE html>
<html>
<body>
<?php
$x = 10.365;
var_dump($x);
?>
</body>
</html>
```

Output:

float(10.365)

A Boolean represents two possible states: TRUE or FALSE.

$x = true;

$y = false;

Booleans are often used in conditional testing.

An array stores multiple values in one single variable.

```
<!DOCTYPE html>
<html>
<body>
<?php
$cars = array("Volvo","BMW","Toyota");
var_dump($cars);
?>
</body>
</html>
```

Output:

array(3) { [0]=> string(5) "Volvo" [1]=> string(3) "BMW" [2]=> string(6) "Toyota" }

# PHP Arrays

Numeric Array:- Arrays can store numbers, strings and any object but their index will be represented by numbers. By default array index starts from zero.

```
<html>
  <body>
      <?php
      /* First method to create array. */
      $numbers = array( 1, 2, 3, 4, 5);

      foreach( $numbers as $value ) {
        echo "Value is $value <br />";
      }

      /* Second method to create array. */
      $numbers[0] = "one";
      $numbers[1] = "two";
      $numbers[2] = "three";
      $numbers[3] = "four";
      $numbers[4] = "five";

      foreach( $numbers as $value ) {
        echo "Value is $value <br />";
      }
    ?>
```

Output:

Value is 1
Value is 2
Value is 3
Value is 4
Value is 5
Value is one
Value is two
Value is three
Value is four
Value is five

# PHP Arrays

Associative Array:- Arrays can store numbers, strings and any object but their index will be represented by strings.

```
<html>
  <body>

    <?php
      $salaries = array("ABC" => 2000, "PQR" => 1000, "XYZ" => 500);

      echo "Salary of ABC is ". $salaries['ABC'] . "<br />";
      echo "Salary of PQR is ".  $salaries['PQR']. "<br />";
      echo "Salary of XYZ is ".  $salaries['XYZ']. "<br />";

      $salaries['ABC'] = "high";
      $salaries['PQR'] = "medium";
      $salaries['XYZ'] = "low";

      echo "Salary of ABC is ". $salaries['ABC'] . "<br />";
      echo "Salary of PQR is ".  $salaries['PQR']. "<br />";
      echo "Salary of XYZ is ".  $salaries['XYZ']. "<br />";
    ?>

  </body>
</html>
```

Output:

```
Salary of ABC is 2000
Salary of PQR is 1000
Salary of XYZ is 500
Salary of ABC is high
Salary of PQR is medium
Salary of XYZ is low
```

# PHP Arrays

Multidimensional Array:- A multi-dimensional array each element in the main array can also be an array.

```
<html>
  <body>
     <?php
      $marks = array(
        "ABC" => array (
          "SUB1" => 75,
          "SUB2" => 80,
          "Sub3" => 56
        ),
         "PQR" => array (
          "SUB1" => 85,
          "SUB2" => 66,
          "Sub3" => 87
        ),
        );
      echo $marks['ABC']['SUB1'] . "<br />";
      echo $marks['PQR']['SUB2'] . "<br />";
     ?>
   </body>
</html>
```

Output:

75

66

# PHP Strings

```
<!DOCTYPE html>
<html>
<body>

<?php
echo strlen("Hello world!")."<br />";
echo str_word_count("Hello world!")."<br />";
echo strrev("Hello world!")."<br />";
echo strpos("Hello world!", "world")."<br />";
echo str_replace("world", "Dolly", "Hello world!")."<br
/>";
?>
</body>
</html>
```

Output:

12
2
!dlrow olleH
6
Hello Dolly!

# PHP Operators

Operators are used to perform operations on variables and values.

## Arithmetic

| Operator | Name | Example | Result |
|---|---|---|---|
| + | Addition | $x + $y | Sum of $x and $y |
| - | Subtraction | $x - $y | Difference of $x and $y |
| * | Multiplication | $x * $y | Product of $x and $y |
| / | Division | $x / $y | Quotient of $x and $y |
| % | Modulus | $x % $y | Remainder of $x divided by $y |
| ** | Exponentiation | $x ** $y | Result of raising $x to the $y'th power |

## Assignment

| Assignment | Same as... | Description |
|---|---|---|
| x = y | x = y | The left operand gets set to the value of the expression on the right |
| x += y | x = x + y | Addition |
| x -= y | x = x - y | Subtraction |
| x *= y | x = x * y | Multiplication |
| x /= y | x = x / y | Division |
| x %= y | x = x % y | Modulus |

## Increment/Decrement

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

## String

| Operator | Name | Example | Result |
|---|---|---|---|
| . | Concatenation | $txt1 . $txt2 | Concatenation of $txt1 and $txt2 |
| .= | Concatenation assignment | $txt1 .= $txt2 | Appends $txt2 to $txt1 |

# PHP Operators

## Comparison

| Operator | Name | Example | Result |
|---|---|---|---|
| == | Equal | $x == $y | Returns true if $x is equal to $y |
| === | Identical | $x === $y | Returns true if $x is equal to $y, and they are of the same type |
| != | Not equal | $x != $y | Returns true if $x is not equal to $y |
| <> | Not equal | $x <> $y | Returns true if $x is not equal to $y |
| !== | Not identical | $x !== $y | Returns true if $x is not equal to $y, or they are not of the same type |
| > | Greater than | $x > $y | Returns true if $x is greater than $y |
| < | Less than | $x < $y | Returns true if $x is less than $y |
| >= | Greater than or equal to | $x >= $y | Returns true if $x is greater than or equal to $y |
| <= | Less than or equal to | $x <= $y | Returns true if $x is less than or equal to $y |
| <=> | Spaceship | $x <=> $y | Returns an integer less than, equal to, or greater than zero, depending on if $x is less than, equal to, or greater than $y. Introduced in PHP 7. |

## 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 |

# PHP Operators

## Array

| Operator | Name | Example | Result |
|---|---|---|---|
| + | Union | $x + $y | Union of $x and $y |
| == | Equality | $x == $y | Returns true if $x and $y have the same key/value pairs |
| === | Identity | $x === $y | Returns true if $x and $y have the same key/value pairs in the same order and of the same types |
| != | Inequality | $x != $y | Returns true if $x is not equal to $y |
| <> | Inequality | $x <> $y | Returns true if $x is not equal to $y |
| !== | Non-identity | $x !== $y | Returns true if $x is not identical to $y |

## Conditional assignment

| Operator | Name | Example | Result |
|---|---|---|---|
| ?: | Ternary | $x = *expr1* ? *expr2* : *expr3* | Returns the value of $x.<br>The value of $x is *expr2* if *expr1* = TRUE.<br>The value of $x is *expr3* if *expr1* = FALSE |
| ?? | Null coalescing | $x = *expr1* ?? *expr2* | Returns the value of $x.<br>The value of $x is *expr1* if *expr1* exists, and is not NULL.<br>If *expr1* does not exist, or is NULL, the value of $x is *expr2*.<br>Introduced in PHP 7 |

# PHP Expressions

```
<!DOCTYPE html>
<html>
<body>

<?php
$x=100;
$y=$x++;
echo "x = $x y = $y"."<br/>";
$marks=60;
$result= $marks<50 ? "fail" : "pass";
echo $result;
?>
</body>
</html>
```

Output:
x = 101 y = 100
pass

# PHP Regular Expressions

1. A regular expression is a sequence of characters that forms a search pattern.
2. When you search for data in a text, you can use this search pattern to describe what you are searching for.
3. A regular expression can be a single character, or a more complicated pattern.
4. Regular expressions can be used to perform all types of text search and text replace operations.
5. PHP

| Function | Description |
|---|---|
| preg_match() | Returns 1 if the pattern was found in the string and 0 if not |
| preg_match_all() | Returns the number of times the pattern was found in the string, which may also be 0 |
| preg_replace() | Returns a new string where matched patterns have been replaced with another string |

ons.

# PHP Regular Expressions- preg_match()

```
<!DOCTYPE html>
<html>
<body>

<?php
$str = "Web enabled technology";
$pattern = "/tech/i";
echo preg_match($pattern, $str);
?>

</body>
</html>
```

Output:
1

```
<!DOCTYPE html>
<html>
<body>

<?php
$str = "Web enabled technology";
$pattern = "/at/i";
echo preg_match($pattern, $str);
?>

</body>
</html>
```

Output:
0

# PHP Regular Expressions- preg_match_all()

```
<!DOCTYPE html>
<html>
<body>

<?php
$str = "If you pain others pain gives you pain.";
$pattern = "/pain/i";
echo preg_match_all($pattern, $str);
?>

</body>
</html>
```

Output:
3

```
<!DOCTYPE html>
<html>
<body>

<?php
$str = "If you pain others pain gives you pain.";
$pattern = "/apain/i";
echo preg_match_all($pattern, $str);
?>

</body>
</html>
```

Output:
0

# **PHP Regular Expressions-** preg_replace()

```
<!DOCTYPE html>
<html>
<body>

<?php
$str = "If you pain others pain gives you pain!";
$pattern = "/pain/i";
echo preg_replace($pattern, "gain", $str);
?>

</body>
</html>
```

Output:
If you gain others gain gives you gain!

# PHP Conditional statements

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

```
<!DOCTYPE html>
<html>
<body>
<?php
$t = date("H");
if ($t < "12") {
  echo "Good morning!";
}
?>
 </body>
</html>
```

Output:
Good morning!

```
<!DOCTYPE html>
<html>
<body>
<?php
$t = date("H");
if ($t < "12") {
  echo "Good morning!";
} else {
  echo "Good night!";
}
?>
 </body>
</html>
```

Output:
Good morning!

# PHP Conditional statements

```
<!DOCTYPE html>
<html>
<body>
<?php
$t = date("H");
if ($t < "12") {
  echo "Good morning!";
} elseif ($t < "18") {
  echo "Good evening!";
} else {
  echo "Good night!";
}
?>
</body>
</html>
```

Output:
Good morning!

```
<!DOCTYPE html>
<html>
<body>
<?php
$branch = "cse";
switch ($branch) {
  case "cse":
    echo "Your branch is CSE!";
    break;
  case "ece":
    echo "Your branch is ECE!";
    break;
  case "civil":
    echo "Your branch is Civil!";
    break;
  default:
    echo "BSC!";
}
?>
</body>
</html>
```

Output:
Your branch is CSE!

# PHP- Loop Statements

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

In PHP, we have 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

## ...ile

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

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

## Do-While

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

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

**or**

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

Output:

The number is: 0
The number is: 1
The number is: 2
The number is: 3
The number is: 4
The number is: 5
The number is: 6
The number is: 7
The number is: 8
The number is: 9
The number is: 10

**foreach**

```
<!DOCTYPE html>
<html>
<body>

<?php
$colors = array("red", "green", "blue",
"yellow");
foreach ($colors as $value) {
  echo "$value <br>";
}
?>
</body>
</html>
```

Output:

red
green
blue
yellow

# ons

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

Output:
Hello world!

```
<!DOCTYPE html>
<html>
<body>
<?php
function saleInfo(int $discount = 50) {
  echo "The height is : $discount <br>";
}
saleInfo(25);
saleinfo();
?>
</body> </html>
```

Output:
25
50

```
<!DOCTYPE html>
<html>
<body>

<?php
function familyName($fname) {
  echo "$fname Sirisha.<br>";
}
familyName("Potluri");
?>
</body></html>
```

Output:
Potluri Sirisha.

```
<!DOCTYPE html>
<html>
<body>
<?php
function sum(int $x, int $y) {
  $z = $x + $y;
  return $z;
}
echo "10 + 20 = " . sum(10,20) . "<br>";
?>
</body>
</html>
```

Output:
10 + 20 = 30

# To Learn PHP With Database

**MySQL Database**

Creating Your First Database

•Most web hosts do not allow you to create a database directly through a PHP script.
•Instead they require that you use the PHP/MySQL administration tools on the web host control panel to create these databases.
•Create a database and assign a new user to this database.
•For all of our beginning examples we will be using the following information:

.:. Server - localhost
.:. Database - test
.:. Table - example
.:. Username - admin
.:. Password - 1admin

**MySQL Connect**

•Before you can do anything with MySQL in PHP you must first establish a connection to your web host's MySQL database.

•This is done with the MySQL connect function.

**MySQL localhost**

•When the PHP script and MySQL are on the same machine, you can use localhost as the address you wish to connect to.

• localhost is a shortcut to just have the machine connect to itself.

• If your MySQL service is running at a separate location you will need to insert the IP address or URL in place of localhost.

**PHP & MySQL Code:**

```
<?php
mysql_connect("localhost", "admin", "1admin") or die(mysql_error());
echo "Connected to MySQL<br />";
?>
```

**Display:**
Connected to MySQL

The mysql_connect function takes three arguments. Server, username, and password. In our example above these arguments were:

.:. Server - localhost
.:. Username - admin
.:. Password - 1admin

The "die(mysql..." code displays an error message in your browser if --you've probably guessed it -- there is an error in processing the connection! Double-check your username, password, or server if you receive this error.

**Choosing the Working Database**

After establishing a MySQL connection with the code above, you then need to choose which database you will be using with this connection. This is done with the mysql_select_db function.

**PHP & MySQL Code:**

```
<?php
mysql_connect("localhost", "admin", "1admin") or
die(mysql_error());
echo "Connected to MySQL<br />";
mysql_select_db("test") echo "Connected to Database";
?>
```

**Display:**

Connected to MySQL
Connected to Database

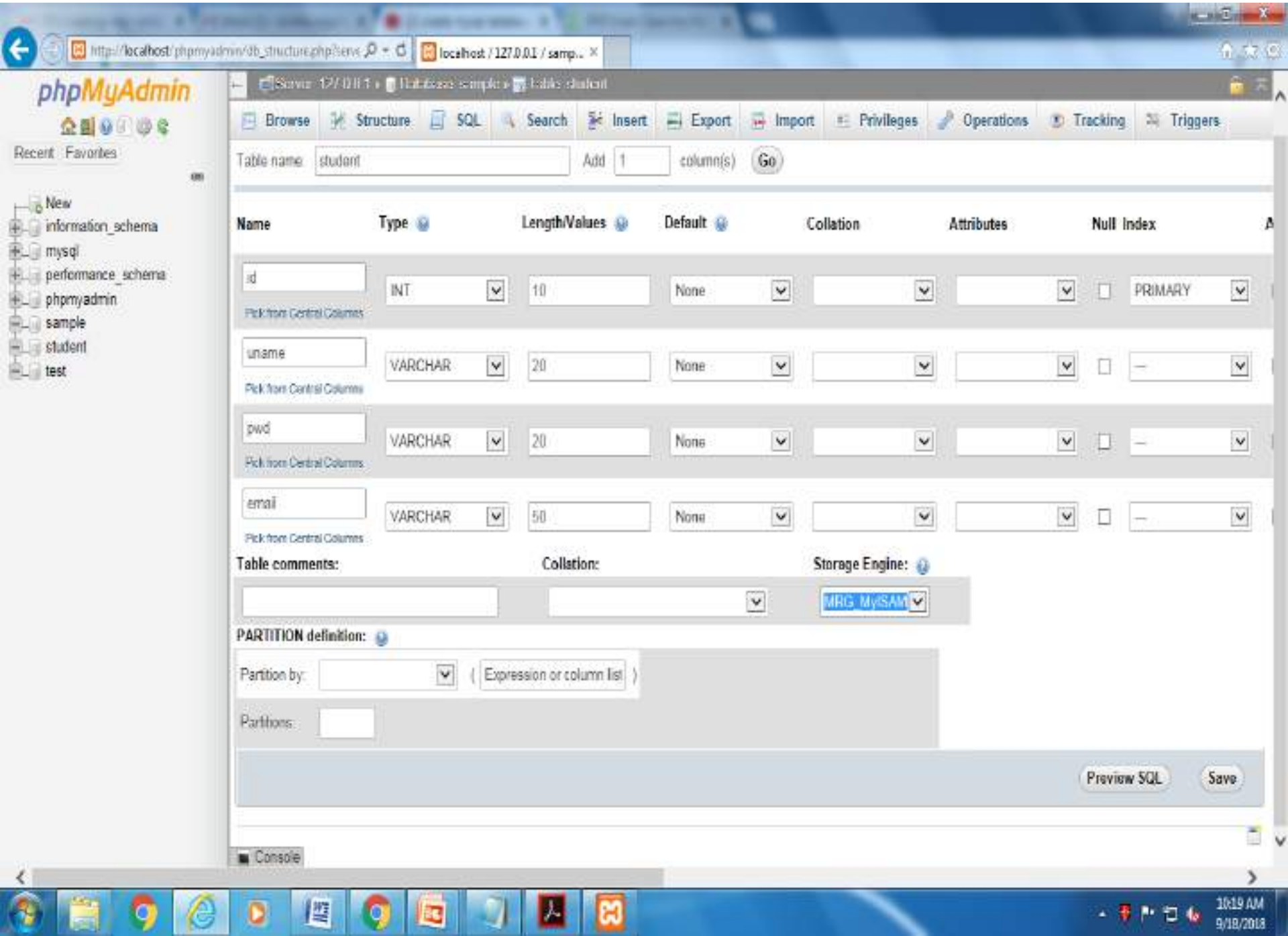
**Create Table MySQL**

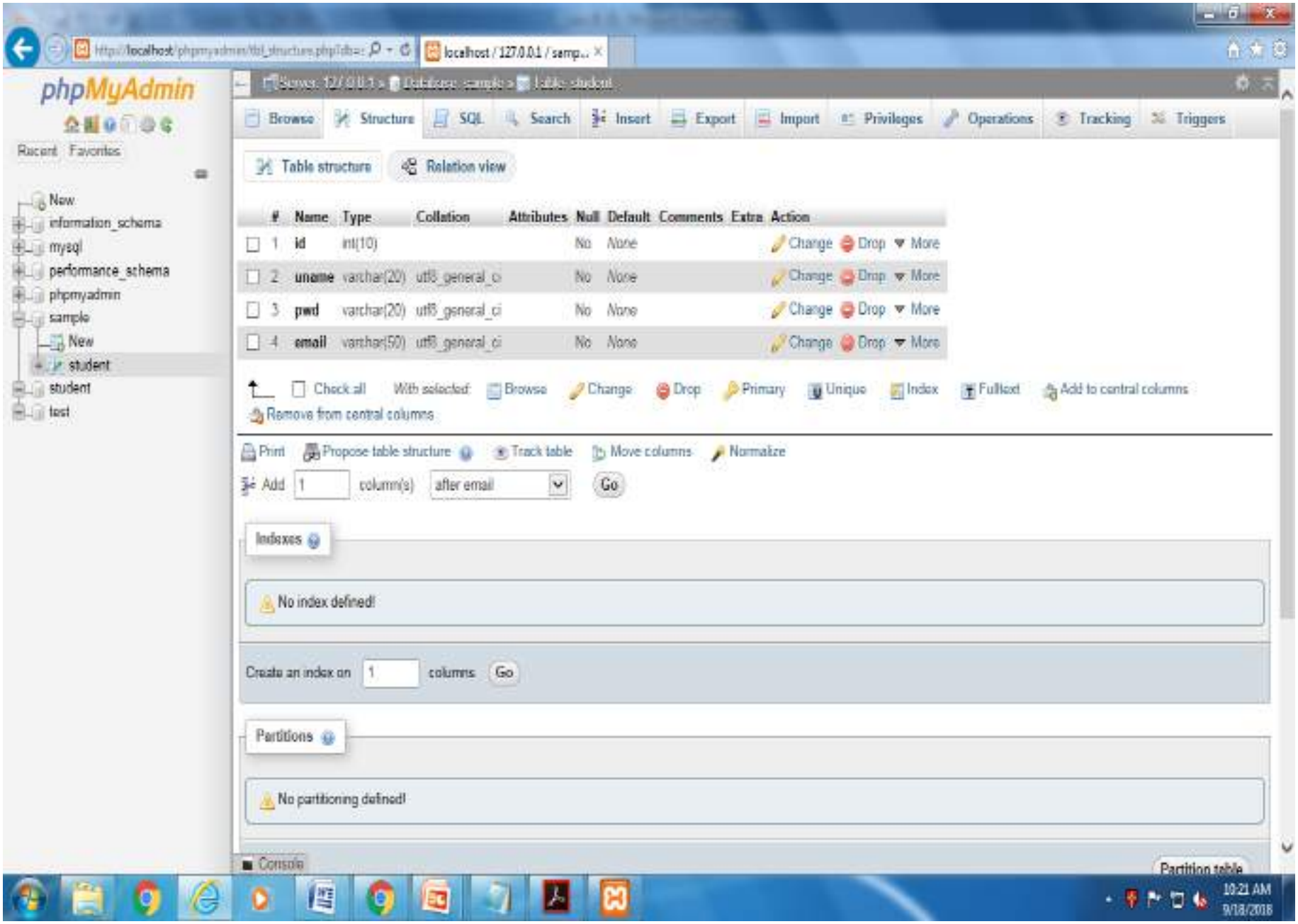
PHP & MySQL Code:

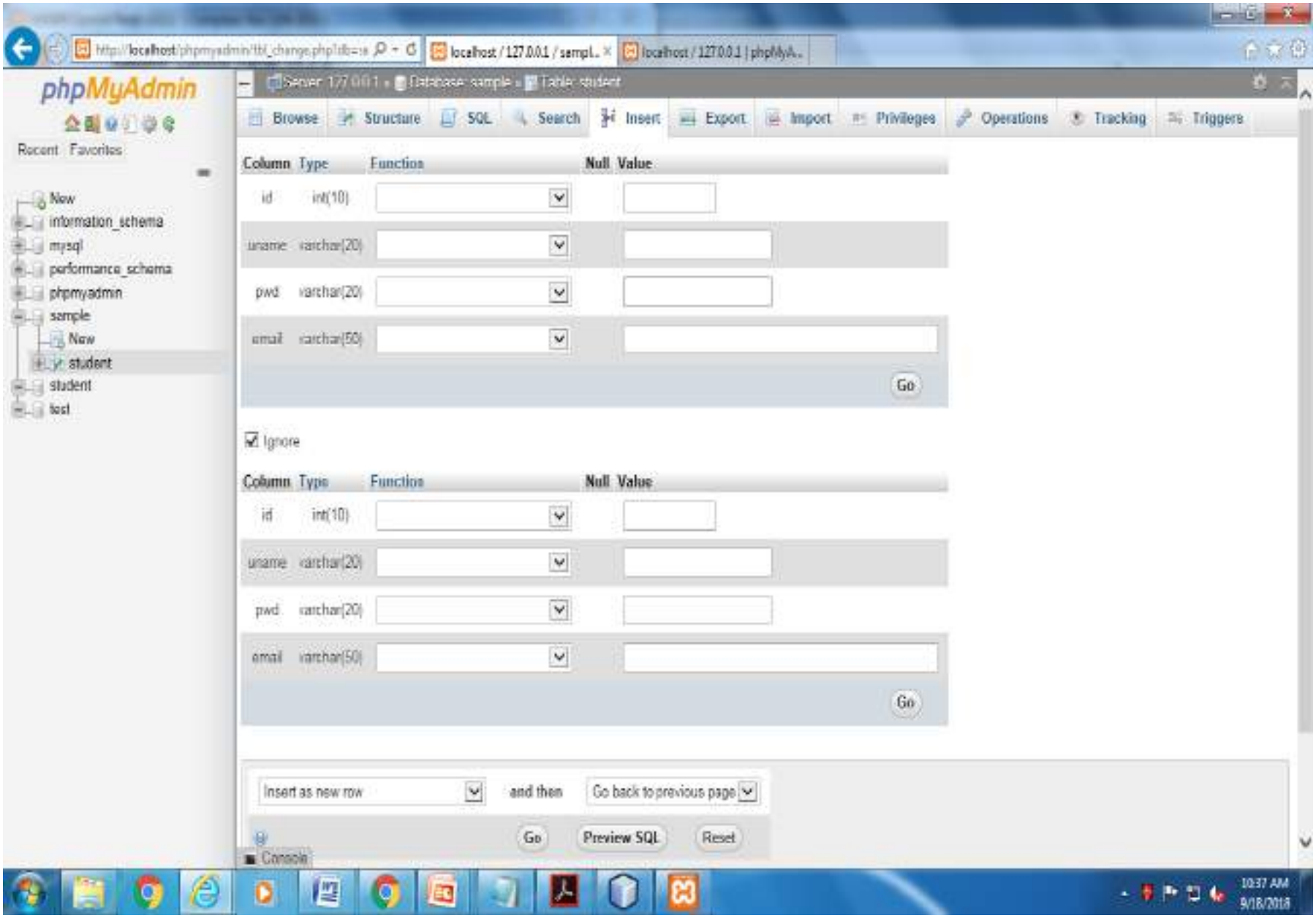
```
<?php
// Make a MySQL Connection
mysql_connect("localhost", "admin", "1admin")
// Create a MySQL table in the selected database
mysql_query("CREATE TABLE example(id INT NOT NULL
AUTO_INCREMENT, PRIMARY KEY(id),name VARCHAR(30), age INT)"));
echo "Table Created!";
?>
```

**Display:**

Table Created!


phpMyAdmin
Recent Favorites
New
information_schema
mysql
performance_schema
phpmyadmin
sample
student
test
Browse
Structure
SQL
Search
Insert
Export
Import
Privileges
Operations
Tracking
Triggers
Table name: student
Add 1 column(s)
Go
Name
Type
Length/Values
Default
Collation
Attributes
Null
Index
id
INT
10
None
PRIMARY
uname
VARCHAR
20
None
pwd
VARCHAR
20
None
email
VARCHAR
50
None
Pick from Central Columns
Table comments:
Collation:
Storage Engine:
MRG_MyISAM
PARTITION definition:
Partition by:
Expression or column list
Partitions:
Preview SQL
Save
Console


phpMyAdmin
Recent Favorites
New
information_schema
mysql
performance_schema
phpmyadmin
sample
New
student
student
test
Server: 127.0.0.1 » Database: sample » Table: student
Browse
Structure
SQL
Search
Insert
Export
Import
Privileges
Operations
Tracking
Triggers
Table structure
Relation view
# Name Type Collation Attributes Null Default Comments Extra Action
1 id int(10) No None Change Drop More
2 uname varchar(20) utf8_general_ci No None Change Drop More
3 pwd varchar(20) utf8_general_ci No None Change Drop More
4 email varchar(50) utf8_general_ci No None Change Drop More
Check all
With selected:
Browse
Change
Drop
Primary
Unique
Index
Fulltext
Add to central columns
Remove from central columns
Print
Propose table structure
Track table
Move columns
Normalize
Add 1 column(s) after email Go
Indexes
No index defined!
Create an index on 1 columns Go
Partitions
No partitioning defined!
Console
Partition table


phpMyAdmin
Recent Favorites
New
information_schema
mysql
performance_schema
phpmyadmin
sample
New
student
student
test
Server: 127.0.0.1 » Database: sample » Table: student
Browse
Structure
SQL
Search
Insert
Export
Import
Privileges
Operations
Tracking
Triggers
Column Type Function Null Value
id int(10)
uname varchar(20)
pwd varchar(20)
email varchar(50)
Go
Ignore
Column Type Function Null Value
id int(10)
uname varchar(20)
pwd varchar(20)
email varchar(50)
Go
Insert as new row
and then
Go back to previous page
Go
Preview SQL
Reset
Console
10:37 AM
9/18/2018

**Reserved MySQL Keywords:**

.:. **INT** - This stands for integer or whole number.  'id' has been defined to be an integer.

.:. **NOT NULL** - An entry is NOT NULL only if it has some value, while something with no value is NULL.

.:. **AUTO_INCREMENT** - Each time a new entry is added the value will  be incremented by 1.

**'PRIMARY KEY** (id)':  PRIMARY KEY is used as a unique identifier for the rows.

'name **VARCHAR**(30),': Here we make a new column with the name "name"! VARCHAR stands for "variable character". "Character" means that you can put in any kind of typed information in this column (letters, numbers, symbols, etc).

'age **INT**,': Our third and final column is age, which stores an integer.
The possible integer values that can be stored in an "INT" are -2,147,483,648 to 2,147,483,647.

**MySQL Insert**

When data is put into a MySQL table it is referred to as inserting data

**Inserting Data Into Your Table**

PHP & MySQL Code:

```
<?php
// Make a MySQL Connection
mysql_connect("localhost", "admin", "1admin") mysql_select_db("test")
// Insert a row of information into the table "example"
 mysql_query("INSERT INTO example(name, age) VALUES('Timmy Mellowman',
'23' ) ");
mysql_query("INSERT INTO example(name, age) VALUES('Sandy Smith', '21' ) ");
mysql_query("INSERT INTO example(name, age) VALUES('Bobby Wallace', '15' ) ");
echo "Data Inserted!";
?>
```

**Display:**

Data Inserted!

```php
<?php
$servername = "localhost";
$username = "root";
$password = "root";
$dbname = "sample";
// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
   die("Connection failed: " . $conn->connect_error);
}
$sql = "INSERT INTO student (id, uname, pwd,email)
VALUES ('100', 'abc', 'xyz','abc@xyz.com')";
if ($conn->query($sql) === TRUE) {
   echo "New record created successfully";
} else {
   echo "Error: " . $sql . "<br>" . $conn->error;
}
$conn->close();
?>
```

```php
<?php
$servername = "localhost";
$username = "root";
$password = "root";
$dbname = "sample";
// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
//Insert Query of SQL
$query = "INSERT INTO  student(id, uname, pwd, email) VALUES ('$_POST[id]',
'$_POST[uname]', '$_POST[pwd]', '$_POST[email]')";
 if ($conn->query($query)==TRUE)
 {
     echo "Data Inserted successfully...!!";
}
else
{
echo "Insertion Failed   Some Fields are Blank....!!";
}
$conn->close();
?>
```

**MySQL Query**
**Using MySQL SELECT & FROM**

PHP & MySQL Code:

```
<?php
// Make a MySQL Connection
mysql_connect("localhost", "admin", "1admin") or
mysql_select_db("test"));
//        Retrieve all the data from the "example" table
$result = mysql_query("SELECT * FROM example");
//        store the record of the "example" table into $row
  $row = mysql_fetch_array( $result );
//        Print out the contents of the entry
echo "Name: ".$row['name']; echo " Age: ".$row['age'];
?>
```

**Display:**

Name: Timmy Mellowman Age: 23

**Fetch Array While Loop**

PHP and MySQL Code:

```
<?php

// Make a MySQL Connection $query = "SELECT * FROM example";

$result = mysql_query($query) or die(mysql_error());
while($row = mysql_fetch_array($result)){
echo $row['name']. " - ". $row['age']; echo "<br />";
}
?>
```

**Display:**

Timmy Mellowman - 23
Sandy Smith - 21
Bobby Wallace - 15

**Retrieving Information from MySQL**

PHP & MySQL Code:

```
<?php
// Make a MySQL Connection
mysql_connect("localhost", "admin", "1admin") or
die(mysql_error()); mysql_select_db("test") or die(mysql_error());
// Get all the data from the "example" table
 $result = mysql_query("SELECT * FROM example") or
die(mysql_error());
echo "<table border='1'>";
echo "<tr> <th>Name</th> <th>Age</th> </tr>";
// keeps getting the next row until there are no more to get
while($row = mysql_fetch_array( $result )) {
// Print out the contents of each row into a table echo "<tr><td>";
echo $row['name']; echo "</td><td>"; echo $row['age']; echo
"</td></tr>";
}
echo "</table>"; ?>
```

**MySQL Where**

WHERE lets you specify requirements that entries must meet in order to be returned in the MySQL result.

**PHP & MySQL Code:**

```
<?php
// Make a MySQL Connection
mysql_connect("localhost", "admin", "1admin"); mysql_select_db("test");
//          Get a specific result from the "example" table
$result = mysql_query("SELECT * FROM example WHERE name='Sandy Smith'")
or die(mysql_error());
//          get the first (and hopefully only) entry from the result
        $row = mysql_fetch_array( $result );
//          Print out the contents of each row into a table
echo $row['name']." - ".$row['age']; ?>
```

**Display:**

Sandy Smith - 21

**MySQL Wildcard Usage '%'**

In MySQL there is a "wildcard" character '%' that can be used to search for partial matches in your database. The '%' tells MySQL to ignore the text that would normally appear in place of the wildcard.

For example '2%' would match the following: 20, 25, 2000000, 2avkldj3jklsaf, and 2!
On the other hand, '2%' would not match the following: 122, a20, and 32.

**PHP & MySQL Code:**

```
<?php
//          Connect to MySQL
//          Insert a row of information into the table "example“
$result = mysql_query("SELECT * FROM example WHERE age LIKE '2%' ") or
die(mysql_error());
// keeps getting the next row until there are no more to get
while($row = mysql_fetch_array( $result )) {
// Print out the contents of each row
echo $row['name']." - ".$row['age']. "<br />";
}
?>
```

**Display:**

Timmy Mellowman - 23
Sandy Smith - 21

**MySQL Order By**

What ORDER BY does is take the a column name that you specify and sort it in alphabetical order (or numeric order if you are using numbers). Then when you use mysql_fetch_array to print out the result, the values are already sorted and easy to read.

```
<?php
// Make a MySQL Connection
mysql_connect("localhost", "admin", "1admin"); mysql_select_db("test";
// Get all the data from the "example" table
$result = mysql_query("SELECT * FROM example ORDER BY age") or
die(mysql_error());
echo "<table border='1'>";
echo "<tr> <th>Name</th> <th>Age</th> </tr>";
// keeps getting the next row until there are no more to get
 while($row = mysql_fetch_array( $result )) {
// Print out the contents of each row into a table echo "<tr><td>";
echo $row['name']; echo "</td><td>"; echo $row['age']; echo "</td></tr>";
}
echo "</table>"; ?>
```

**MySQL Joins**

The act of joining in MySQL refers to smashing two or more tables into a single table.

**PHP and MySQL Code:**

```
<?php
Make a MySQL Connection
Construct our join query
 $query = "SELECT family.Position, food.Meal ". "FROM family, food ".
"WHERE family.Position = food.Position";
$result = mysql_query($query) or die(mysql_error());
 // Print out the contents of each row into a table
while($row = mysql_fetch_array($result)){
 echo $row['Position']. " - ". $row['Meal']; echo "<br />";
}
?>
```

**Display:**

**Dad - Steak**
**Mom - Salad**
**Dad - Tacos**

**The statement "WHERE family.Position = food.Position" will restrict the results to the rows where the *Position* exists in both the "family" and "food" tables.**

# Prepared Statements

❖The MySQL database supports prepared statements.

❖A prepared statement or a parameterized statement is used to execute the same statement repeatedly with high efficiency.

❖Basic workflow The prepared statement execution consists of two stages: **prepare and execute.**

❖At the prepare stage a statement template is sent to the database server.

❖The server performs a syntax check and initializes server internal resources for later use.

❖The MySQL server supports using anonymous, positional placeholder with **?**.

# First stage: prepare

```
<?php
$mysqli = new mysqli(“localhost", "user", "password", "database");
if ($mysqli->connect_errno) {
 echo "Failed to connect to MySQL: " . $mysqli->connect_error;
}
/* Non-prepared statement */
if (!$mysqli->query("DROP TABLE IF EXISTS test")
|| !$mysqli->query("CREATE TABLE test(id INT)"))
{
 echo "Table creation failed: " . $mysqli->error;
}
/* Prepared statement, stage 1: prepare */
if (!($stmt = $mysqli->prepare("INSERT INTO test(id) VALUES (?)")))
{
 echo "Prepare failed: " . $mysqli->error;
}
?>
```

❖Prepare is followed by execute.

❖During execute the client binds parameter values and sends them to the server.

❖The server creates a statement from the statement template and the bound values to execute it using the previously created internal resources

```
<?php
/* Prepared statement, stage 2: bind and execute */
$id = 1;
if (!$stmt->bind_param("i", $id)) {
 echo "Binding parameters failed: " . $stmt->error;
}
if (!$stmt->execute())
{
 echo "Execute failed: " . $stmt->error;
}
?>
```

## Repeated execution

- A prepared statement can be executed repeatedly.
- Upon every execution the current value of the bound variable is evaluated and sent to the server.
- The statement is not parsed again.
- The statement template is not transferred to the server again.

```php
<?php
$mysqli = new mysqli(“localhost", "user", "password", "database");
if ($mysqli->connect_errno) {
 echo "Failed to connect to MySQL: " . $mysqli->connect_error;
}
/* Non-prepared statement */
if (!$mysqli->query("DROP TABLE IF EXISTS test") || !$mysqli->query("CREATE
TABLE test(id INT)")) {
 echo "Table creation failed: " . $mysqli->error;
}
/* Prepared statement, stage 1: prepare */
if (!($stmt = $mysqli->prepare("INSERT INTO test(id) VALUES (?)"))) {
 echo "Prepare failed: " . $mysqli->error;
}
/* Prepared statement, stage 2: bind and execute */
$id = 1;
if (!$stmt->bind_param("i", $id)) {
 echo "Binding parameters failed: " . $stmt->error;
}
if (!$stmt->execute()) {
 echo "Execute failed: " . $stmt->error;
}
```

```
/* Prepared statement: repeated execution, only data transferred from client to server */
for ($id = 2; $id < 5; $id++) {
 if (!$stmt->execute()) {
 echo "Execute failed: " . $stmt->error;
 }
}
/* explicit close recommended */
$stmt->close();
/* Non-prepared statement */
$result = mysql_query("SELECT * FROM test") or die(mysql_error());
echo "<table border='1'>";
echo "<tr> <th>id</th> </tr>";
// keeps getting the next row until there are no more to get
while($row = mysql_fetch_array( $result )) {
// Print out the contents of each row into a table echo "<tr><td>";
echo $row[‘id']; echo "</td></tr>";
}
echo "</table>"; ?>
```

❖Every prepared statement occupies server resources.

❖ Statements should be closed explicitly immediately after use.

❖If not done explicitly, the statement will be closed when the statement handle is freed by PHP.

❖ Using a prepared statement is not always the most efficient way of executing a statement.

❖A prepared statement executed only once causes more client-server round-trips than a non-prepared statement.

❖This is why the SELECT is not run as a prepared statement.

❖Consider the use of the MySQL multi-INSERT SQL syntax for INSERTs. It requires less round-trips between the server and client than the prepared statement.

## Less round trips using multi-INSERT SQL

```
<?php

query("INSERT INTO test(id) VALUES (1), (2), (3), (4)"))
 {
echo "Multi-INSERT failed: " . $mysqli->error;
}
?>
```

# Stored Procedures

❖The MySQL database supports stored procedures.

❖A stored procedure is a subroutine stored in the database catalog. Applications can call and execute the stored procedure.

❖The CALL SQL statement is used to execute a stored procedure. Parameter Stored procedures can have IN, INOUT and OUT parameters, depending on the MySQL version.

❖The mysqli interface has no special notion for the different kinds of parameters.

❖ IN parameter Input parameters are provided with the CALL statement. Please, make sure values are escaped correctly.

```
<?php
$mysqli = new mysqli("example.com", "user", "password", "database");
if ($mysqli->connect_errno)
{
 echo "Failed to connect to MySQL: " . $mysqli->connect_error;
}
if (!$mysqli->query("DROP TABLE IF EXISTS test") ||
!$mysqli->query("CREATE TABLE test(id INT)"))
{
 echo "Table creation failed: " . $mysqli->error;
}
if (!$mysqli->query("DROP PROCEDURE IF EXISTS p") ||
 !$mysqli->query("CREATE PROCEDURE p(IN id_val INT) BEGIN INSERT INTO
test(id) VALUES(id_val); END;"))
{
 echo "Stored procedure creation failed: " . $mysqli->error;
}
```

```
if (!$mysqli->query("CALL p(1)")) {
 echo "CALL failed: " . $mysqli->error;
}
$result = mysql_query("SELECT * FROM test") or die(mysql_error());
echo "<table border='1'>";
echo "<tr> <th>id</th> </tr>";
// keeps getting the next row until there are no more to get
while($row = mysql_fetch_array( $result ))
 {
// Print out the contents of each row into a table echo "<tr><td>";
echo $row[‘id']; echo "</td></tr>";
}
echo "</table>"; ?>
```

# PHP 5 Cookies

A cookie is often used to identify a user.

# What is a Cookie?

A cookie is often used to identify a user. A cookie is a small file that the server embeds on the user's computer. Each time the same computer requests a page with a browser, it will send the cookie too. With PHP, you can both create and retrieve cookie values.

# Create Cookies With PHP

A cookie is created with the setcookie() function.

## Syntax

`setcookie(`*`name, value, expire, path, domain, secure, httponly`*`);` Only the *name* parameter is required. All other parameters are optional.

# PHP Create/Retrieve a Cookie

The following example creates a cookie named "user" with the value "John Doe". The cookie will expire after 30 days (86400 * 30). The "/" means that the cookie is available in entire website (otherwise, select the directory you prefer). We then retrieve the value of the cookie "user" (using the global variable $_COOKIE). We also use the isset() function to find out if the cookie is set:

## 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>
```

**Note:** The setcookie() function must appear BEFORE the <html> tag.

**Note:** The value of the cookie is automatically URLencoded when sending the cookie, and automatically decoded when received (to prevent URLencoding, use setrawcookie() instead).

# Modify a Cookie Value

To modify a cookie, just set (again) the cookie using the setcookie() function:

## Example

```
<?php
$cookie_name = "user";
$cookie_value = "Alex Porter";
setcookie($cookie_name, $cookie_value, time() + (86400 * 30), "/");
?>
<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>
```

# Delete a Cookie

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

## Example

```
<?php
// set the expiration date to one hour ago
setcookie("user", "", time() - 3600);
?>
```

```
<html>
<body>

<?php
echo "Cookie 'user' is deleted.";
?>

</body>
</html>
```

# Check if Cookies are Enabled

The following example creates a small script that checks whether cookies are enabled. First, try to create a test cookie with the setcookie() function, then count the $_COOKIE array variable:

## Example

```
<?php
setcookie("test_cookie", "test", time() + 3600, '/');
?>
<html>
<body>

<?php
if(count($_COOKIE) > 0) {
    echo "Cookies are enabled.";
} else {
    echo "Cookies are disabled.";
}
?>

</body>
</html>
```

# PHP 5 Sessions

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.

# What is a PHP Session?

When you work with an application, you open it, do some changes, and then you close it. This is much like a Session. The computer knows who you are. It knows when you start the application and when you end. But on the internet there is one problem: the web server does not know who you are or what you do, because the HTTP address doesn't maintain state.

Session variables solve this problem by storing user information to be used across multiple pages (e.g. username, favorite color, etc). 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.

**Tip:** If you need a permanent storage, you may want to store the data in a database.

# Start a PHP Session

A session is started with the session_start() function. Session variables are set with the PHP global variable: $_SESSION. Now, let's create a new page called "demo_session1.php". In this page, we start a new PHP session and set some session variables:

## Example

```
<?php
// Start the session
session_start();
?>
<!DOCTYPE html>
<html>
<body>

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

</body>
</html>
```

**Note:** The session_start() function must be the very first thing in your document. Before any HTML tags.

# Get PHP Session Variable Values

Next, we create another page called "demo_session2.php". From this page, we will access the session information we set on the first page ("demo_session1.php"). Notice that session variables are not passed individually to each new page, instead they are retrieved from the

session we open at the beginning of each page (session_start()). Also notice that all session variable values are stored in the global $_SESSION variable:

## Example

```
<?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>
```

Another way to show all the session variable values for a user session is to run the following code:

## Example

```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
print_r($_SESSION);
?>

</body>
</html>
```

**How does it work? How does it know it's me?**

Most sessions set a user-key on the user's computer that looks something like this: 765487cf34ert8dede5a562e4f3a7e12. Then, when a session is opened on another page, it scans the computer for a user-key. If there is a match, it accesses that session, if not, it starts a new session.

# Modify a PHP Session Variable

To change a session variable, just overwrite it:

## Example

```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>

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

</body>
</html>
```

# Destroy a PHP Session

To remove all global session variables and destroy the session, use session_unset() and session_destroy():

## Example

```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
// remove all session variables
session_unset();

// destroy the session
session_destroy();
?>

</body>
</html>
```

# Form Elements: Letting the User Work with Data

- The data from an HTML form is sent to a specific location and processed.

- The PHP script receives the data from the form and uses it to perform an action such as updating the contents of a database, sending an e - mail, testing the data format, and so on.

- **An interactive web site requires user input, which is generally gathered through forms.**

This chapter welcomes you into a world of PHP/MySQL interaction by covering the following:

Creating forms using buttons, text boxes, and other form elements.

Creating PHP scripts to process HTML forms.

Passing hidden information to the form - processing script via hidden form controls and a URL query string

**print_r()**

displays information about a variable in a way that's readable by humans.

```
<pre>
<?php
$a = array (‘1' => ’abc', ’2' => ’50‘, ’3' => ’90.7‘,
’4' => array(’x‘,’y’,’z’));
print_r ($a);
?>
</pre>
```

**Output:**

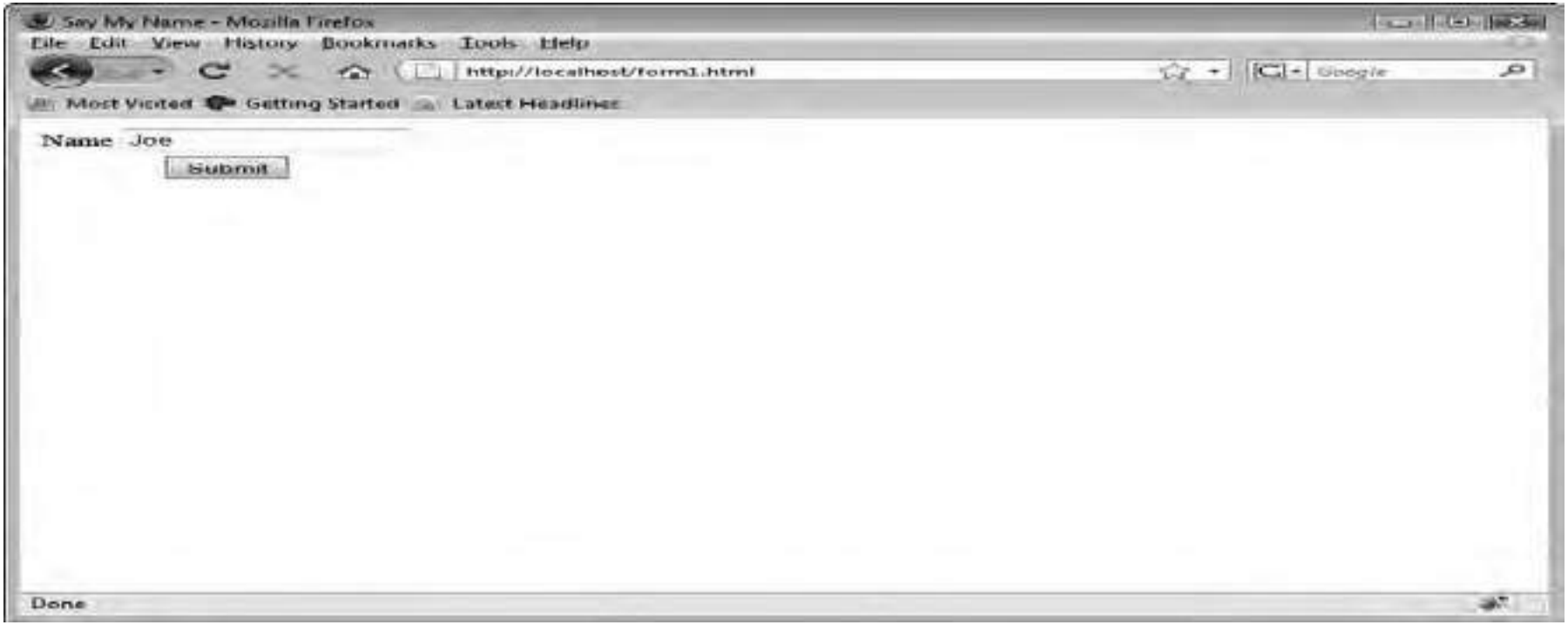
```
<pre>
Array (
        [1] => abc
        [2] => 50
        [3] => 90.7
        [4] => Array
                (
                [0] => x
                [1] => y
                [2] => z
                )
        )
</pre>
```

**//Form1.html**

```
< html > < head >
< title > Say My Name < /title >
< /head >
< body >
 < form action=”formprocess1.php” method=”post” >
< table > < tr > < td > Name < /td >
< td > < input type=”text” name=”name” / > < /td >
< /tr > < tr > < td colspan=”2” style=”text-align: center;” >
< input type=”submit” name=”submit” value=”Submit” / >
< /td > < /tr >
< /table >
< /form >
< /body >
< /html >
```

**//formprocess1.php**

```
< html >
< head > < title > Say My Name < /title >
< /head >
 < body >
< ?php
 echo ‘ < h1 > Hello ‘ . $_POST[‘name’] . ‘! < /h1 > ’;
 ? >
< pre > < strong > DEBUG: < /strong >
 < ?php print_r($_POST); ? >
< /pre >
 < /body >
 < /html >
```


Say My Name - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost/form1.html
Most Visited Getting Started Latest Headlines
Name Joe
Submit
Done


Say My Name - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost/formprocess1.php
Most Visited Getting Started Latest Headlines
Hello Joe!
DEBUG:
ARRAY
(
[name] => Joe
[submit] => Submit
)
Done

# Taking the User Input

- The form in this example allows you to lead the user to choose values from a set of values you provide.
- Defining a value set is done through the use of specific HTML elements, such as list boxes, radio buttons, and check boxes.
- Two kinds of predefined user input are in HTML forms.
- The first kind allows you to choose one item from the available options;
- The second allows the user to choose multiple items.
- Drop - down list boxes and radio buttons allow for one selection only.
- Check boxes and multiline list boxes provide for multiple choices.

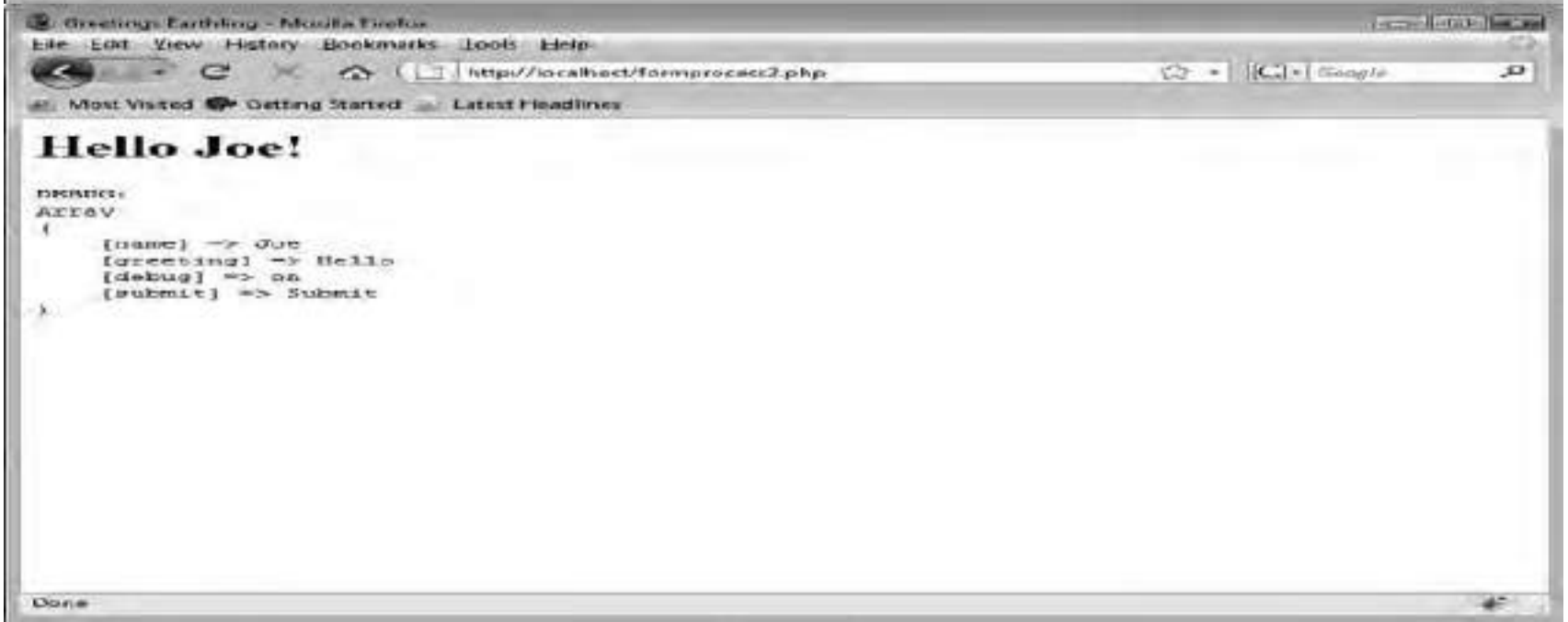
**//form2.html**

< html >
< head > < title > Greetings Earthling < /title > < /head >
< body >
< form action=”formprocess2.php” method=”post” >
< table > < tr > < td > Name < /td > < td >
< input type=”text” name=”name” / > < /td > < /tr >
< tr > < td > Greetings < /td >
< td > < select name=”greeting” >
< option value=”Hello” > Hello < /option >
< option value=”Hola” > Hola < /option >
< option value=”Bonjour” > Bonjour < /option >
< /select > < /td > < /tr >
< tr > < td >< /td > < td >
< input type=”checkbox” name=”debug” checked=”checked”/ >
Display Debug info < /td > < /tr > < tr >
< td colspan=”2” style=”text-align: center” >
< input type=”submit” name=”submit” value=”Submit” / > < /td > < /tr > <
/table > </form > < /body > < /html >

**//formprocess2.php**

```
< html >
< head >
 < title > Greetings Earthling < /title >
 < /head >
< body >
< ?php
echo ‘ < h1 > ’ . $_POST[‘greeting’] . ‘ ‘ . $_POST[‘name’] . ‘! < /h1 > ’;
if (isset($_POST[‘debug’]))
 {
echo ‘ < pre > < strong > DEBUG: < /strong > ’ . “\n”;
 print_r($_POST);
 echo ‘ < /pre > ’; } ? >
 < /body >
< /html >
```


Greetings Earthling - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost/form2.html
Most Visited Getting Started Latest Headlines
Name Joe
Greetings Hello
Display Debug info
Submit
Done


Greetings Earthling - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost/formprocacc2.php
Most Visited Getting Started Latest Headlines
Hello Joe!
POST:
Array
(
[name] => Joe
[greeting] => Hello
[debug] => on
[submit] => Submit
)
Done

**To get value of multiple select option from select tag, name attribute in HTML <select> tag should be initialize with an array [ ]**

```
< html >
 < head > < title > Greetings Earthling < /title > < /head >
 < body >
 < form action=”formprocess2.php” method=”post” >
< table > < tr > < td > Name < /td > < td >
< input type=”text” name=”name” / > < /td > < /tr >
 < tr > < td > Greetings < /td >
< td > < select name=”greeting[]” >
 < option value=”Hello” > Hello < /option >
< option value=”Hola” > Hola < /option >
< option value=”Bonjour” > Bonjour < /option >
 < /select > < /td > < /tr >
< input type=”submit” name=”submit” value=”Submit” / > < /td > <
/tr > < /table > </form > < /body > < /html >
```

**//formprocess3.php**

```
<html>
<body>
< ?php
if (isset($_POST[‘debug’]))
{
echo ‘ < pre > ’;
print_r($_POST);
echo ‘ < /pre > ’;
}
foreach ($_POST['movie_type '] as $select)
 {
 echo "You have selected :" .$select; // Displaying Selected Value
}
? >
< /body >
< /html >
```

**PHP script for RADIO BUTTON FIELD:**

```
<form action="" method="post">
<input type="radio" name="radio" value="1">option 1
<input type="radio" name="radio" value=" 2">option 2
<input type="radio" name="radio" value="3">option 3
<input type="submit" name="submit" value="submit" />
</form>
<?php
if (isset($_POST['submit']))
{
if(isset($_POST['radio']))
{
echo "You have selected :".$_POST['radio'];
}
?>
```

# String Manipulation and Regular Expressions

# Regular Expression

✓A regular expression is a sequence of characters that forms a search pattern.
✓When you search for data in a text, you can use this search pattern to describe what you are searching for.
✓A regular expression can be a single character, or a more complicated pattern.
✓Regular expressions can be used to perform all types of text search and text replace operations.
✓In PHP, regular expressions are strings composed of delimiters, a pattern and optional modifiers.

**Syntax**
**$exp = "/welcome/i";**

**/ is the delimiter**
**welcome is the pattern that is being searched for**
**i is a modifier that makes the search case-insensitive**

✓The delimiter can be any character that is not a letter, number, backslash or space.
✓The most common delimiter is the forward slash (/),
✓When your pattern contains forward slashes it is convenient to choose other delimiters such as # or ~.

# Regular Expression Functions

| Function | Description |
| --- | --- |
| preg_match() | Returns 1 if the pattern was found in the string and 0 if not |
| preg_match_all() | Returns the number of times the pattern was found in the string, which may also be 0 |
| preg_replace() | Returns a new string where matched patterns have been replaced with another string |

# preg_match()

The `preg_match()` function will tell you whether a string contains matches of a pattern.

```
<!DOCTYPE html>
<html>
<body>

<?php
$str = "Welcome to India";
$pattern = "/India/i";
echo preg_match($pattern, $str);
?>

</body>
</html>
```

**Output:**
**1**

# preg_match_all()

The preg_match_all() function will tell you how many matches were found for a pattern in a string.

```
<!DOCTYPE html>
<html>
<body>

<?php
$str = "Welcome to India. Explore India";
$pattern = "/india/i";
echo preg_match_all($pattern, $str);
?>

</body>
</html>
```

**Output:**
**2**

# preg_replace()

The preg_replace() function will replace all of the matches of the pattern in a string with another string.

```
<!DOCTYPE html>
<html>
<body>

<?php
$str = "Welcome to India. Explore India";
$pattern = "/India/i";
echo preg_replace($pattern,
"Hyderabad", $str);
?>

</body>
</html>
```

**Output:**
**Welcome to Hyderabad. Explore Hyderabad**

**Modifiers:** These can change how a search is performed.

| Modifier | Description |
|---|---|
| i | Performs a case-insensitive search |
| m | Performs a multiline search (patterns that search for the beginning or end of a string will match the beginning or end of each line) |
| u | Enables correct matching of UTF-8 encoded patterns |

**Brackets:** These are used to find a range of characters:

| 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:** Metacharacters are characters with a special meaning:

| 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:** Quantifiers define quantities:

| Quantifier | 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 |

**Grouping:** You can use parentheses ( ) to apply quantifiers to entire patterns. They also can be used to select parts of the pattern to be used as a match.

```
<!DOCTYPE html>
<html>
<body>

<?php
$str = "Banana is sweeter than Apple.";
$pattern = "/ba(na){2}/i";
echo preg_match($pattern, $str);
?>

</body>
</html>
```

**Output:**
**1**

# Form Validation Example

**PHP - Validate Name**

```
$name = test_input($_POST["name"]);
if (!preg_match("/^[a-zA-Z-' ]*$/",$name)) {
  $nameErr = "Only letters and white space allowed";
}
```

**Validate E-mail**

```
$email = test_input($_POST["email"]);
if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
  $emailErr = "Invalid email format";
}
```

**Validate URL**

```
$website = test_input($_POST["website"]);
if (!preg_match("/\b(?:(?:https?|ftp):\/\/|www\.)[-a-z0-
9+&@#\/%?=~_|!:,.;]*[-a-z0-
9+&@#\/%=~_|]/i",$website)) {
  $websiteErr = "Invalid URL";
}
```

```
<!DOCTYPE html>
<html>
<body>

<?php
// Variable to check
$email = "sirisha.vegunta@gmail.com";

// Validate email
if (filter_var($email, FILTER_VALIDATE_EMAIL))
{
  echo("$email is a valid email address");
} else {
  echo("$email is not a valid email address");
}
?>

</body>
</html>
```

# File Handling in PHP

# Files in PHP

PHP file is a plain-text file which contains code written in the PHP programming language.

PHP is a server-side scripting language, the code written in the PHP file is executed on the server.

File handling is an important part of any web application.

PHP has several functions for creating, reading, uploading, and editing files.

# Opening and Closing files

- Files are opened in PHP using the fopen command.
- The command takes two parameters, the file to be opened, and the mode in which to open the file.
- The function returns a file pointer if successful, otherwise zero (false).
- Files are opened with fopen for reading or writing.

```
$fp = fopen("myfile.txt", "r");
```

- The fclose function is used to close a file when you are finished with it.

```
fclose($fp);
```

# fopen

- If fopen is unable to open the file, it returns 0. This can be used to exit the function with an appropriate message.

```
if ( !($fp = fopen("myfile.txt", "r") ) )
exit("Unable to open the input file.");
```

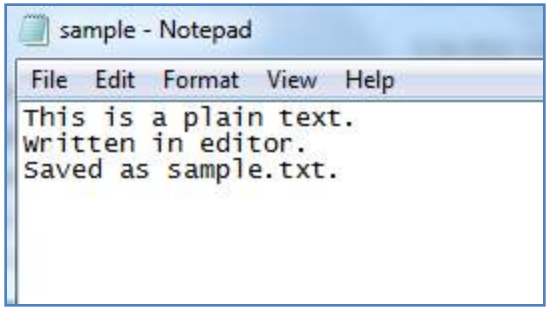
**File Modes**

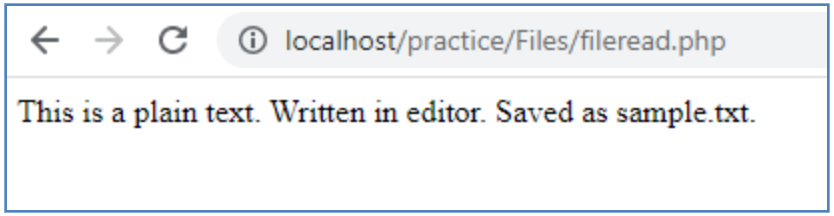
| Mode | Description |
|---|---|
| r | Read Only mode, with the file pointer at the start of the file. |
| r+ | Read/Write mode, with the file pointer at the start of the file. |
| w | Write Only mode. Truncates the file (effectively overwriting it). If the file doesn't exist, fopen will attempt to create the file. |
| w+ | Read/Write mode. Truncates the file (effectively overwriting it). If the file doesn't exist, fopen will attempt to create the file. |
| a | Append mode, with the file pointer at the end of the file. If the file doesn't exist, fopen will attempt to create the file. |
| a+ | Read/Append, with the file pointer at the end of the file. If the file doesn't exist, fopen will attempt to create the file. |

# Reading files- fread()

- The PHP fread() function is used to read the content of the file.
- It accepts two arguments: resource and file size.

```
<!DOCTYPE html>
<html>
<body>

<?php
$fp = fopen("sample.txt", "r")or die("Unable to open file!");
echo fread($fp,filesize("sample.txt"));
fclose($fp);
?>

</body>
</html>
```

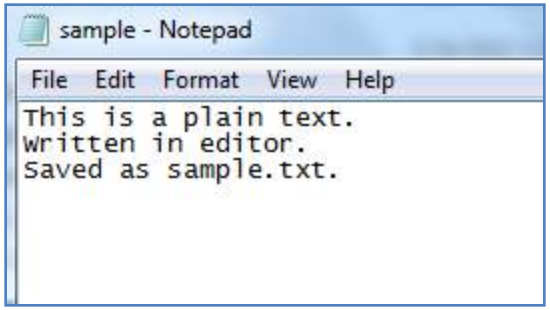


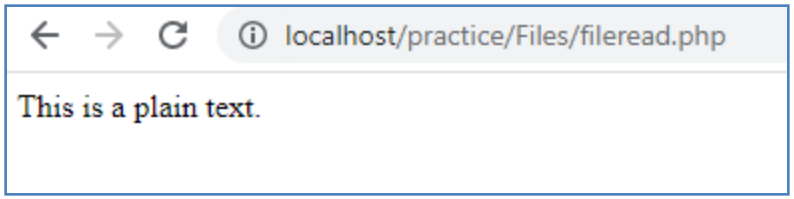



# Reading files- fgets()

- The fgets() function is used to read a single line from a file.
- It accepts one argument: resource.

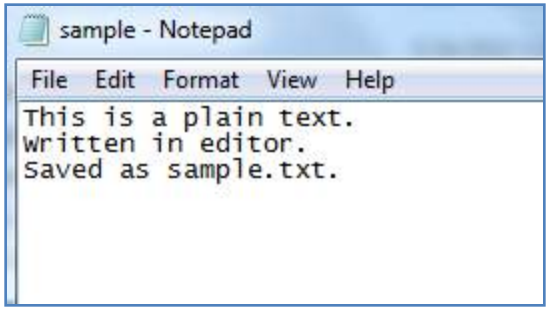
```
<!DOCTYPE html>
<html>
<body>

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

</body>
</html>
```



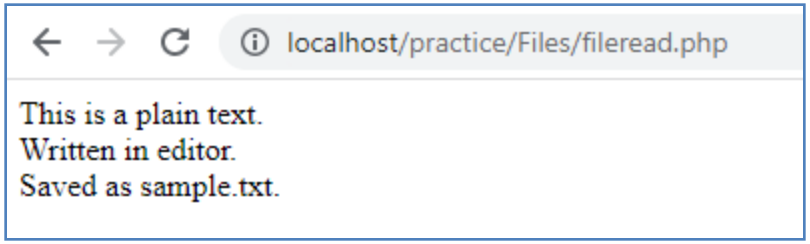



# Reading files- fgets() and feof()

- The feof() function checks if the "end-of-file" (EOF) has been reached.
- The feof() function is useful for looping through data of unknown length.

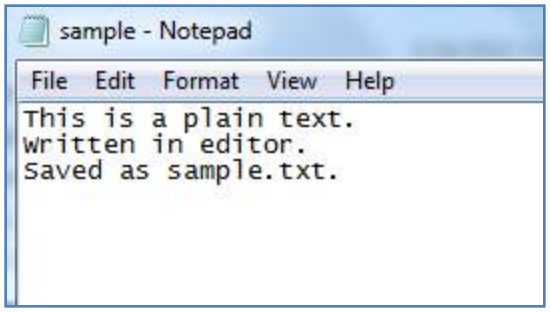



```
<!DOCTYPE html>
<html>
<body>

<?php
$fp = fopen("sample.txt", "r")or die("Unable to open file!");
while(!feof($fp)) {
  echo fgets($fp) . "<br>";
}
fclose($fp);
?>

</body>
</html>
```

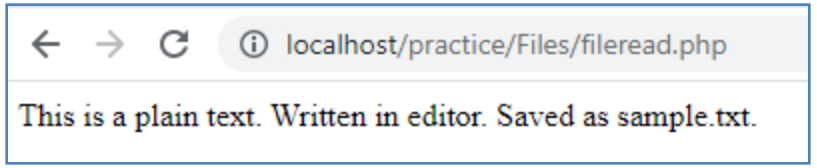


# Reading files- fgetc() and feof()

- The fgetc() function is used to read a single character from a file.




```
<!DOCTYPE html>
<html>
<body>

<?php
$fp = fopen("sample.txt", "r")or die("Unable to open file!");
while(!feof($fp)) {
  echo fgetc($fp);
}
fclose($fp);
?>

</body>
</html>
```



# Creating and writing files- fopen(), fwrite()

- The fopen() function is also used to create a file.
- If you use fopen() on a file that does not exist, it will create it, given that the file is opened for writing (w) or appending (a).

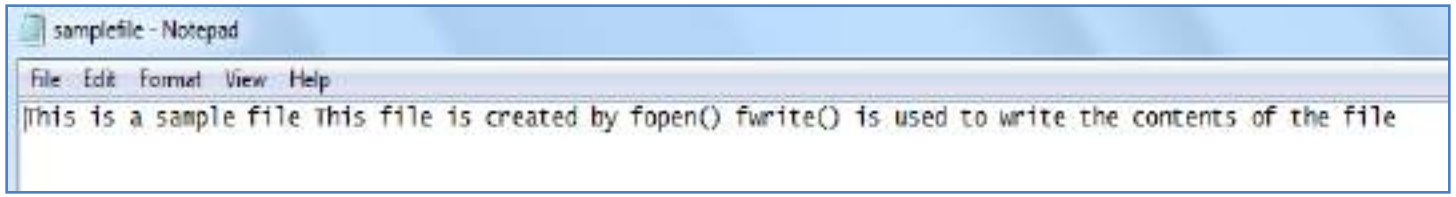
**$fp = fopen(“samplefile.txt", "w")**

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

```
$data = ” This is a sample file”;
fwrite($fp, $data);
```

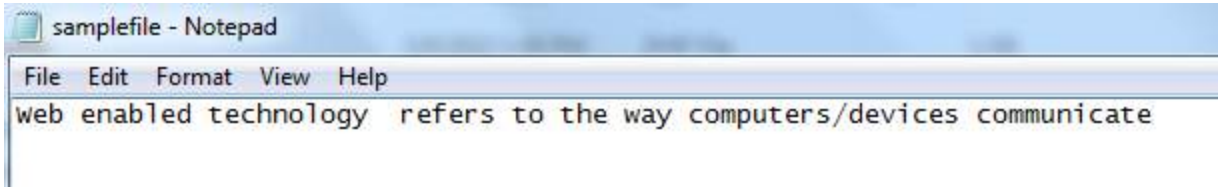
# Creating and writing files- fopen(), fwrite()

```
<!DOCTYPE html>
<html>
<body>

<?php
$fp = fopen("samplefile.txt", "w") or die("Unable to open file!");
$data = "This is a sample file ";
fwrite($fp, $data);
$data = "This file is created by fopen() ";
fwrite($fp, $data);
$data = "fwrite() is used to write the contents of the file ";
fwrite($fp, $data);
fclose($fp);
?>

</body>
</html>
```



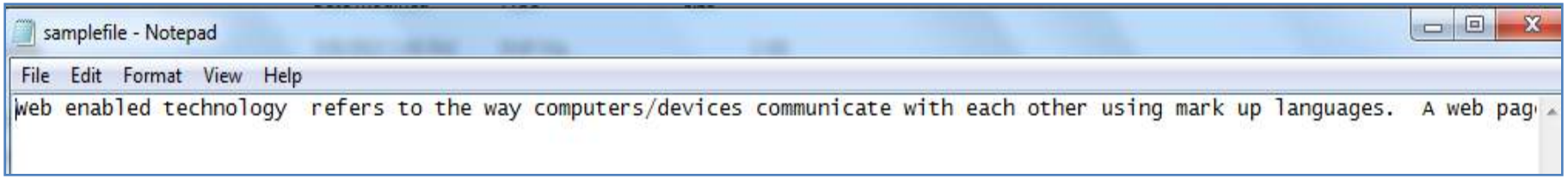
# Overwriting files- fopen(), fwrite()

```
<!DOCTYPE html>
<html>
<body>

<?php
$fp = fopen("samplefile.txt", "w") or die("Unable to open file!");
$data = "Web enabled technology ";
fwrite($fp, $data);
$data = " refers to the way computers/devices communicate ";
fwrite($fp, $data);
fclose($fp);
?>

</body>
</html>
```



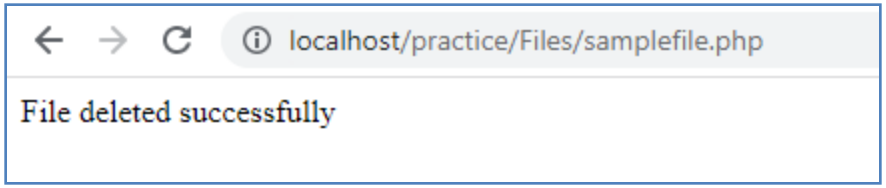
# Appending text in files- fopen(), fwrite()

```
<!DOCTYPE html>
<html>
<body>

<?php
$fp = fopen("samplefile.txt", "a") or die("Unable to open file!");
$data = "with each other using mark up languages. ";
fwrite($fp, $data);
$data = " A web page is a web document which is written in in
HTML ";
fwrite($fp, $data);
fclose($fp);
?>

</body>
</html>
```



# Deleting files- unlink()

```
<!DOCTYPE html>
<html>
<body>

<?php
$fp=unlink("samplefile.txt");
if($fp){
echo "File deleted successfully";
}else{
echo "File not deleted";
}
?>

</body>
</html>
```



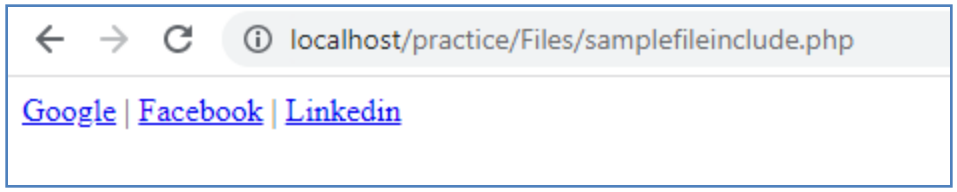
# Include files-include

**Samplefile.php**

```
<!DOCTYPE html>
<html>
<body>
<a href="http://https://www.google.com">Google</a> |
<a href="http://https://www.facebook.com">Facebook</a> |
<a href="http://https://www.linkedin">Linkedin</a>
</body>
</html>
```

**Samplefileinclude.php**

```
<!DOCTYPE html>
<html>
<body>
<?php include("samplefile.php"); ?>
</body>
</html>
```



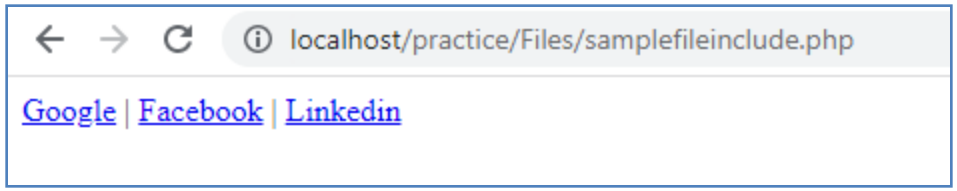
# Include files-require

**Samplefile.php**

```
<!DOCTYPE html>
<html>
<body>
<a href="http://https://www.google.com">Google</a> |
<a href="http://https://www.facebook.com">Facebook</a> |
<a href="http://https://www.linkedin">Linkedin</a>
</body>
</html>
```

**Samplefileinclude.php**

```
<!DOCTYPE html>
<html>
<body>
<?php require("samplefile.php"); ?>
</body>
</html>
```



# PHP File Handling- Binary Files

# Writing to a Binary file

- The open() function opens a file in text format by default.
- To open a file in binary format, add 'b' to the mode parameter.
- Hence the "rb" mode opens the file in binary format for reading, while the "wb" mode opens the file in binary format for writing.
- Unlike text files, binary files are not human-readable.
- When opened using any text editor, the data is unrecognizable.

# Writing to a Binary file

```
file=open("array.bin","wb");
num=[2,4,6,8,10] ;
array=bytearray(num) ;
file.write(array) ;
file.close();
```

```
file=open("array.bin","rb");
number=list(file.read(3)) ;
print (number);
 file.close();
```

# Reading a Binary file

```
file = open(“sample.bin","wb");
sentence = bytearray(“Welcome to IcfaiTech".encode("ascii"));
file.write(sentence);
file.close();


file = open(“sample.bin","rb");
print(file.read(4));
 file.close();
```

# Writing to a Binary file

```
file=open("array.bin","wb");
num=[2,4,6,8,10] ;
array=bytearray(num) ;
file.write(array) ;
file.close();
```

```
file=open("array.bin","rb");
number=list(file.read(3)) ;
print (number);
 file.close();
```

# PHP Directories

# Directory Function

The directory functions allow you to retrieve information about directories and their contents.

| Function | Description |
|---|---|
| chdir() | Changes the current directory |
| chroot() | Changes the root directory |
| closedir() | Closes a directory handle |
| dir() | Returns an instance of the Directory class |
| getcwd() | Returns the current working directory |
| opendir() | Opens a directory handle |
| readdir() | Returns an entry from a directory handle |
| rewinddir() | Resets a directory handle |
| scandir() | Returns an array of files and directories of a specified directory |

# chdir() Function

The chdir() function changes the current directory.
chdir(*directory*)

```
<?php
// Get current directory
echo getcwd() . "<br>";

// Change directory
chdir(“documents");

// Get current directory
echo getcwd();
?>
```

**/home/working**
**/home/working/documents**

# chroot() Function

The chroot() function changes the root directory of the current process to *directory*, and changes the current working directory to "/".

```
<?php
// Change root directory
chroot("/path/to/chroot/");
/

// Get current directory
echo getcwd();
?>
```

# closedir() Function

The closedir() function closes a directory handle.

```
<?php
$dir = "/images/";

// Open a directory, and read its contents
if (is_dir($dir)){
  if ($dh = opendir($dir)){
    while (($file = readdir($dh)) !== false){
      echo "filename:" . $file . "<br>";
    }
    closedir($dh);
  }
}
?>
```

**filename: cat.gif**
**filename: dog.gif**
**filename: horse.gif**

# dir() Function

- The dir() function returns an instance of the Directory class.
- This function is used to read a directory, which includes the following:
- The given directory is opened
- The two properties handle and path of dir() are available
- Both handle and path properties have three methods: read(), rewind(), and close()

```
<?php
$d = dir(getcwd());

echo "Handle: " . $d->handle . "<br>";
echo "Path: " . $d->path . "<br>";

while (($file = $d->read()) !== false){
  echo "filename: " . $file . "<br>";
}
$d->close();
?>
```

**Handle: Resource id #2**
**Path: /etc/php**
**filename: .**
**filename: ..**
**filename: ajax.gif**
**filename: books.xml**