HYPERTEXT PREPROCESSOR(BCA305)

DEPARTMENT OF COMPUTER SCIENCE

PROGRAMME: BCA

CENTRAL UNIVERSITY OF ORISSA
KORAPUT

PHP

- PHP is a recursive acronym for "PHP: Hypertext Preprocessor".
- PHP is a server side scripting language that is embedded in HTML. It is used to manage dynamic content, databases, session tracking, even build entire e-commerce sites.
- It is integrated with a number of popular databases, including MySQL, PostgreSQL, Oracle, Sybase, Informix, and Microsoft SQL Server.
- PHP supports a large number of major protocols such as POP3, IMAP, and LDAP etc.
- PHP Syntax is C-Like.
- Rasmus Lerdorf unleashed the first version of PHP way back in 1994.
- PHP is a case sensitive language.

Common Uses of PHP

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

Characteristics of PHP

- Simplicity
- Efficiency
- Security
- Flexibility
- Familiarity

Example

```
<html>
<head>
<title>Computer Science</title>
 </head>
<body>
 <? php echo “I am a student of BCA Programme";?>
 </body>
 </html>
```

- All PHP code must be included inside one of the three special markup tags ate are recognized by the PHP Parser.

- ❑ **Canonical PHP tags : <?php PHP code goes here ?>**

- ❑ **Short-open (SGML-style) tags : <?   PHP code goes here ?>**

- ❑ **HTML script tags : <script language="php"> PHP code goes here </script>**
- ❑ **ASP(Active Server Pages)-style tags : <%......%>**

PHP – ENVIRONMENT SETUP

- In order to develop and run PHP Web pages, three vital components need to be installed on your computer system.

- **Web Server -** PHP will work with virtually all Web Server software, including Microsoft's Internet Information Server (IIS) but then most often used is freely available Apache Server.
- **Database -** PHP will work with virtually all database software, including Oracle and Sybase but most commonly used is freely available MySQL database.
- **PHP Parser -** In order to process PHP script instructions, a parser must be installed to generate HTML output that can be sent to the Web Browser.

Running PHP Script from Command Prompt

- test.php file
- $ php test.php
- The opening <?php tells the web server to allow the PHP program to interpret all the following code up to the ?> command. Outside of this construct, everything is sent to the client as direct HTML.
- WAMP, MAMP, and LAMP are abbreviations for “Windows, Apache, MySQL, and PHP,” “Mac, Apache, MySQL, and PHP,” and “Linux, Apache, MySQL, and PHP.
- WAMPs, MAMPs, and LAMPs come in the form of a package that binds the bundled programs together so that you don’t have to install and set them up separately.

Commenting PHP Code

- A comment is the portion of a program that exists only for the human reader and stripped out before displaying the programs result.
- There are two commenting formats in PHP:

❑ **Single-line comments:**

```
# This is a comment
// This is a comment too.
```

❑ **Multi-lines comments:**

```
 /* This is a comment with multiline
    Department: Computer Science
    Programme: BCA
*/
```

PHP Variables

- All variables in PHP are denoted with a leading dollar sign ($)
- The value of a variable is the value of its most recent assignment.
- Variables can, but do not need, to be declared before assignment.
- Variables in PHP do not have intrinsic types - a variable does not know in advance whether it will be used to store a number or a string of characters.
- PHP does a good job of automatically converting types from one to another when necessary.

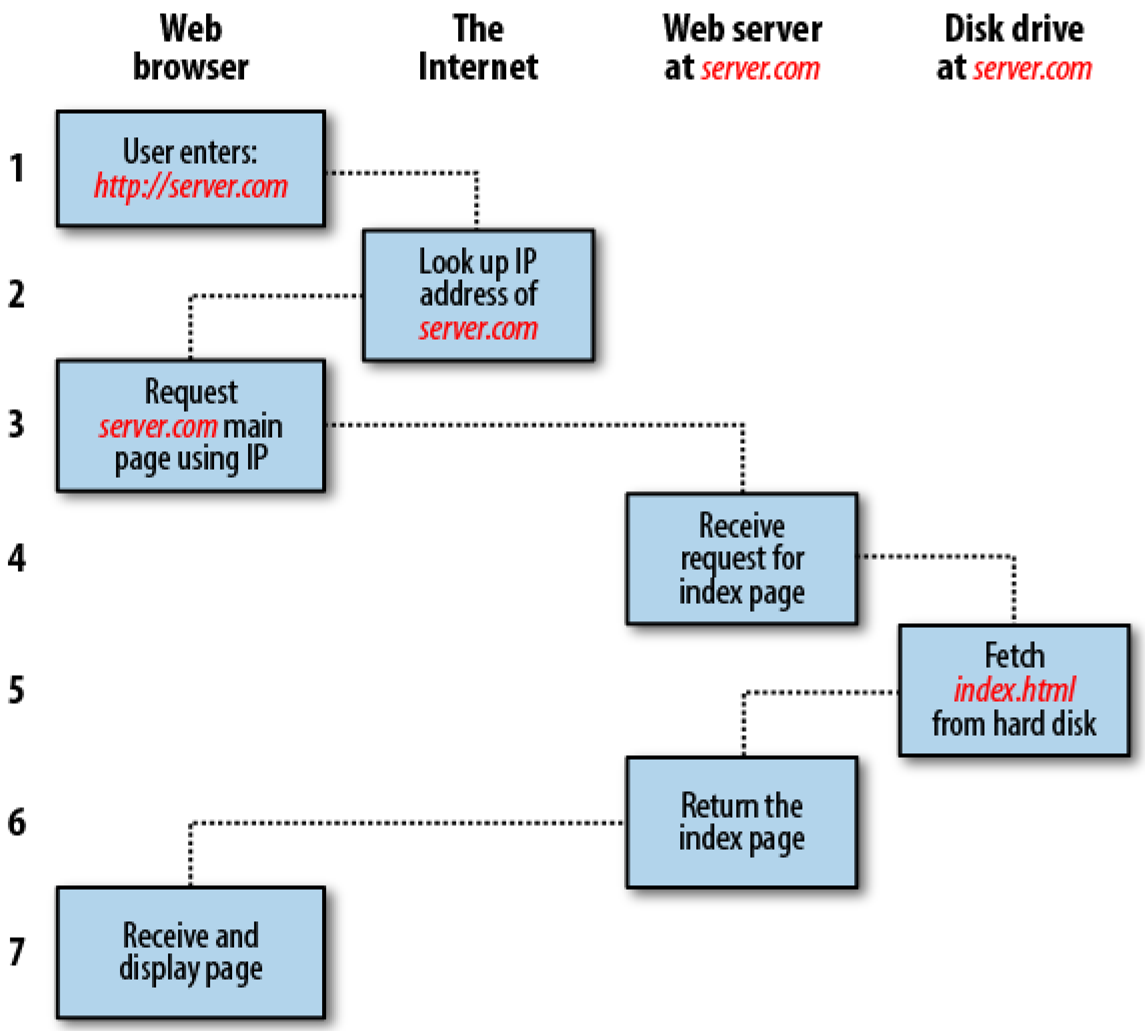
TYPES OF PHP VARIABLES

- **Integers:** are whole numbers, without a decimal point, like 4195.
- **Booleans:** have only two possible values either true or false.
- **NULL:** is a special type that only has one value: NULL.
- **Strings:** are sequences of characters
- **Arrays:** are named and indexed collections of other values.
- **Objects:** are instances of programmer-defined classes, which can package up both other kinds of values and functions that are specific to the class.
- **Resources:** are special variables that hold references to resources external to PHP (such as database connections).
-
```
$a = 1;
$b = "Hello";
$c = array(“BCA", “MCA", “M.Tech");

 echo $c[2];

$d = array(array(‘1', ‘2', ‘3'), array(‘4', ‘5', ‘6'), array(‘7', ‘8', ‘9' ));
```
- The third element in the second row of this array echo $d[1][2];

PHP OPERATORS

- Arithmetic Operators
- Comparison Operators
- Logical (or Relational) Operators
- Assignment Operators
- Conditional (or ternary) Operators

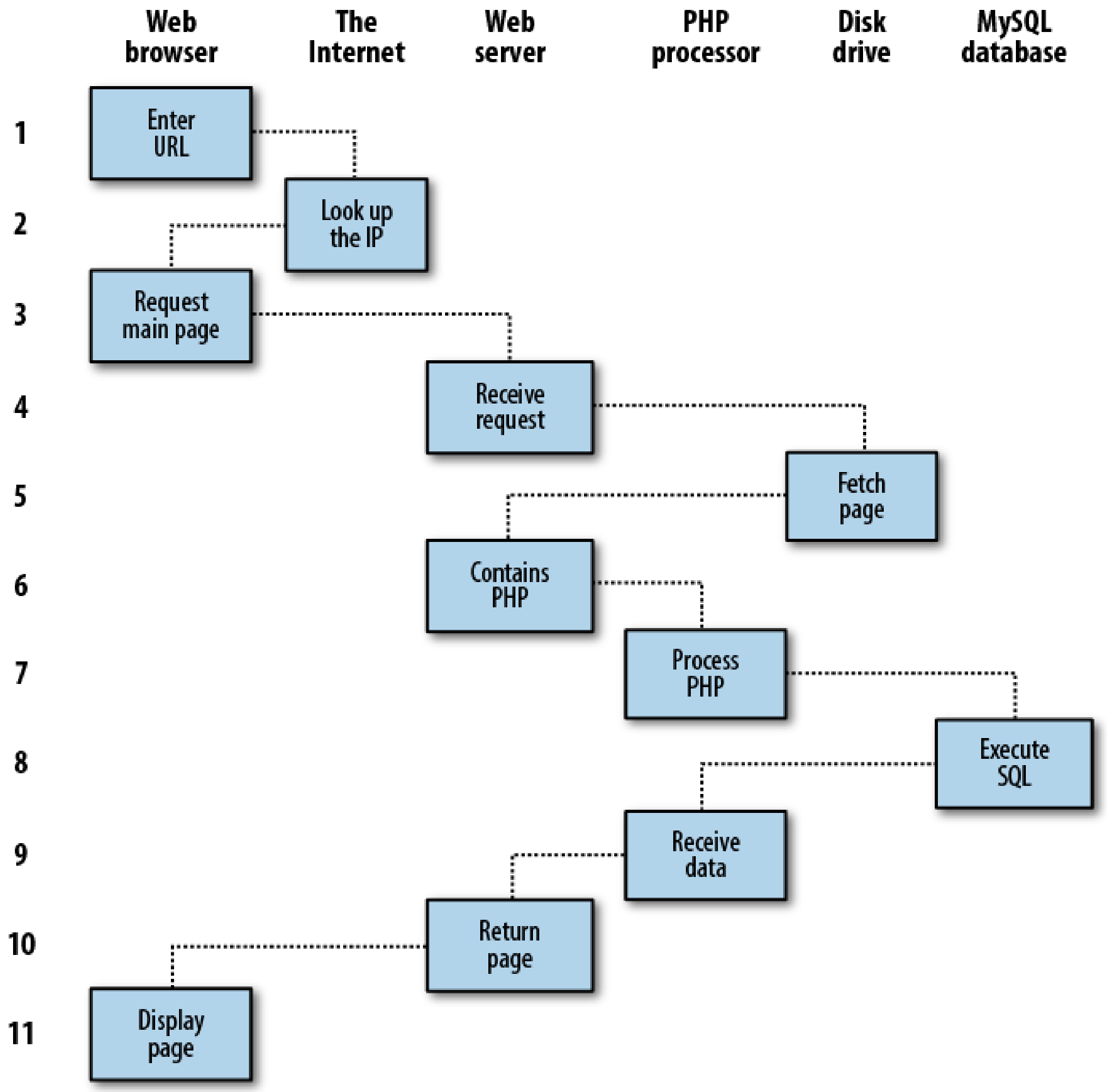
The basic client/server request/response sequence


Web browser
The Internet
Web server at server.com
Disk drive at server.com
1
User enters: http://server.com
2
Look up IP address of server.com
3
Request server.com main page using IP
4
Receive request for index page
5
Fetch index.html from hard disk
6
Return the index page
7
Receive and display page

- HTTP is a communication standard governing the requests and responses that take place between the browser running on the end user’s computer and the web server.
- These are the steps in the request and response sequence:

1. You enter *http://server.com* into your browser’s address bar.
2. Your browser looks up the IP address for *server.com*.
3. Your browser issues a request for the home page at *server.com*.
4. The request crosses the Internet and arrives at the *server.com* web server.
5. The web server, having received the request, looks for the web page on its hard disk.
6. The server retrieves the web page and returns it to the browser.
7. Your browser displays the web page.

A dynamic client/server request/response sequence



- For dynamic web pages, the procedure is a little more involved, because it may bring both PHP and MySQL into the mix :

the steps:

1. You enter *http://server.com* into your browser's address bar.
2. Your browser looks up the IP address for *server.com*.
3. Your browser issues a request to that address for the web server's home page.
4. The request crosses the Internet and arrives at the *server.com* web server.
5. The web server, having received the request, fetches the home page from its hard disk.
6. With the home page now in memory, the web server notices that it is a file incorporating PHP scripting and passes the page to the PHP interpreter.
7. The PHP interpreter executes the PHP code.
8. Some of the PHP contains MySQL statements, which the PHP interpreter now passes to the MySQL database engine.
9. The MySQL database returns the results of the statements back to the PHP interpreter.
10. The PHP interpreter returns the results of the executed PHP code, along with the results from the MySQL database, to the web server.
11. The web server returns the page to the requesting client, which displays it.

Difference Between the echo and print

- There is also an alternative to echo that you can use: print. The two commands are quite similar to each other, but print is an actual function that takes a single parameter, whereas echo is a PHP language construct.
- the echo command will be a tad faster than print in general text output, because—not being a function—it doesn’t set a return value.
- It isn’t a function, echo cannot be used as part of a more complex expression, whereas print can.

Indexed ARRAY

-
  ```
  <?php
  $prgm = array(“BCA", “MCA", “M.Tech."); or $prgm[0]=“BCA”;
  echo "I am a student of " . $prgm [0] " ”;
  ?>
  ```

-
  ```
  <?php
  $prgm = array(“BCA", “MCA", “M.Tech.");
  $length = count($prgm);

  for($x = 0; $x < $length; $x++)
  {
      echo $prgm[$x];
      echo "<br>";
  }
  ?>
  ```

Associative ARRAY

- $mark = array(“Sachin"=>“58", “Sourav"=>“76", “Rahul"=>“64"); or

  $ mark[' Sachin'] = “58";
  $ mark[' Sourav'] = “76";
  $ mark[‘Rahul'] = “64";

```
<?php
  $mark = array(“Sachin"=>“58", “Sourav"=>“76", “Rahul"=>“64");
 echo “Rahul has secured " . $ mark[‘Rahul'] . " in Exam.";
 ?>
```

```
<?php
  $mark = array(“Sachin"=>“58", “Sourav"=>“76", “Rahul"=>“64");

 foreach($mark as $x => $y)
 {
    echo “Mark=" . $x . ,” Value=" . $y;
    echo "<br>";
 }
 ?>
```

User Defined Function in PHP

-
  ```
  <?php
  function display()
   {
     echo “I am a student of BCA Programme";
   }

  display();    // call the function
  ?>
  ```

PHP Function Arguments

-
  ```
  <?php
  function details($fname, $year)
   {
      echo "$fname Born in $year <br>";
  }

   details(“Sachin", "1975");
   details("Sourav", "1978");

  ?>
  ```

PHP Default Argument Value

-
  ```
  <?php
  function getprint($height = 50)
  {
     echo "The height is : $height <br>";
  }

   getprint(350);
   getprint(); // will use the default value of 50
   getprint(135);
   getprint(80);
  ?>
  ```

PHP Functions - Returning values

-
  ```
  <?php
  function sum($x, $y)
  {
      $z = $x + $y;
      return $z;
  }

  echo "5 + 10 = " . sum(5, 10) . "<br>";
  echo "7 + 13 = " . sum(7, 13) . "<br>";
  echo "2 + 4 = " . sum(2, 4);
  ?>
  ```

CALL BY VALUE AND CALL BY REFERENCE

```
<?php

function calval($num)
{
   $num += 2;
   return $num;
}

function calref(&$num)
{
   $num += 10;
   return $num;
}
$n = 10;

calval($n);
echo "The original value is still $n \n";

calref($n);
echo "The original value changes to $n";
?>
```

HTTP

- The Hypertext Transfer Protocol (HTTP) is designed to enable communications between clients and servers.
- HTTP works as a request-response protocol between a client and server.
- A web browser may be the client, and an application on a computer that hosts a web site may be the server.
- A client (browser) submits an HTTP request to the server; then the server returns a response to the client.
- The response contains status information about the request and may also contain the requested content.
- There are two **HTTP** request methods: **GET** and **POST**
- **GET** – Requests data from a specified resource.
- **POST** – Submits data to be processed to a specified resource.

Difference Between GET and POST Method in PHP

GET Method

- Sends information by appending them to the page request.
- Information is visible in the URL.
- Limited amount of information is sent. It is less than 1500 characters.
- Helps to send non-sensitive data.
- Not very secure.
- Possible to bookmark the page.

POST Method

- Transfers information via HTTP header.
- Information is not visible in the URL.
- Unlimited amount of information is sent.
- Helps to send sensitive data (passwords), binary data (word documents, images)and uploading files.
- More secure.
- Not possible to bookmark the page

Difference Between GET and POST Method in PHP

GET Method

- GET can't upload the file.
- GET requests can be cached.
- GET requests remain in the browser history
- GET requests is only used to request data (not modify)
- Restrictions on data type- Only ASCII characters allowed
- Encoding-type-application/x-www-form-urlencoded

POST Method

- POST can upload the files.
- POST requests are never cached
- POST requests do not remain in the browser history
- POST requests have no restrictions on data length.
- No restrictions. Binary data is also allowed.
- Encoding-type-application/x-www-form-urlencoded or multipart/form-data. Use multipart encoding for binary data

Difference Between GET and POST Method in PHP

GET Method	**POST Method**
• Has high performance compared to POST method dues to the simple nature of appending the values in the URL.	• Has lower performance compared to Php_GET method due to time spent encapsulation the Php_POST values in the HTTP body.
• Supports only string data types because the values are displayed in the URL.	• Supports many different data types such as string, numeric, binary etc.

EXAMPLE OF POST METHOD

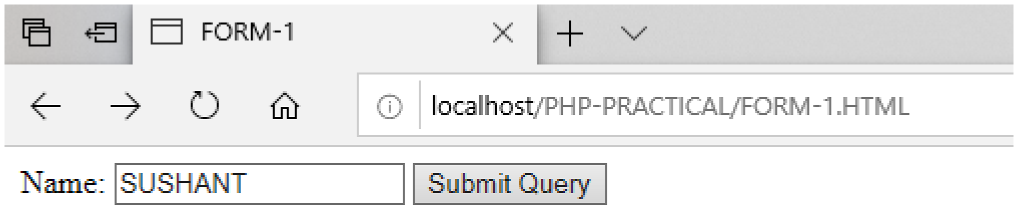
FORM-1.html

```
<html>
 <body>
<form action=“POST-1.php"
      method="post">
 Name: <input type="text" name="name">
<input type="submit">
 </form>
</body>
 </html>
```

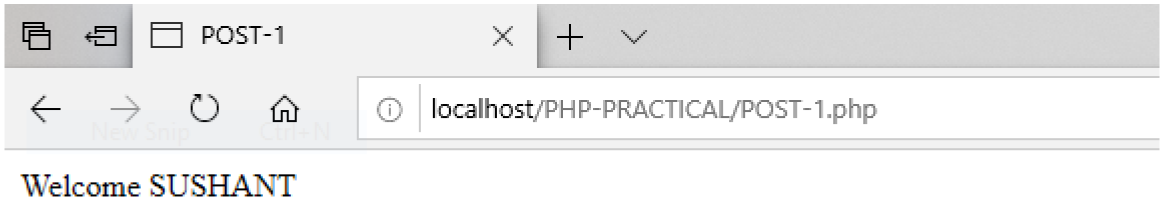
When the user fills out and submits the form, then form data will be sent to PHP file: called POST-1.php.

POST-1.php

```
<html>
 <body>
Welcome <?php echo $_POST["name"];
 </body>
 </html>
```



Name: SUSHANT



Welcome SUSHANT

- 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.

EXAMPLE OF GET METHOD

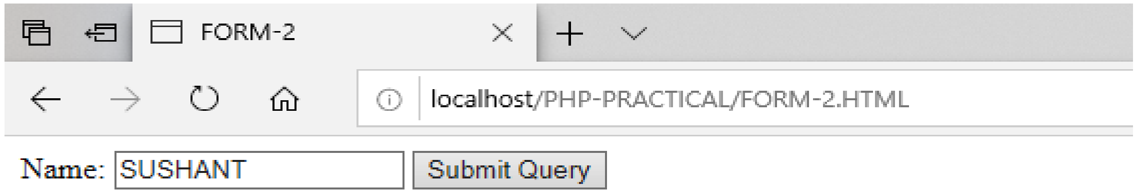
FORM-2.html

```
<html>
 <body>
<form action="GET-1.php" method="get">
 Name: <input type="text" name="name">
<input type="submit">
 </form>
</body>
 </html>
```

When the user fills out and submits the form, then form data will be sent to PHP file: called GET-1.php.

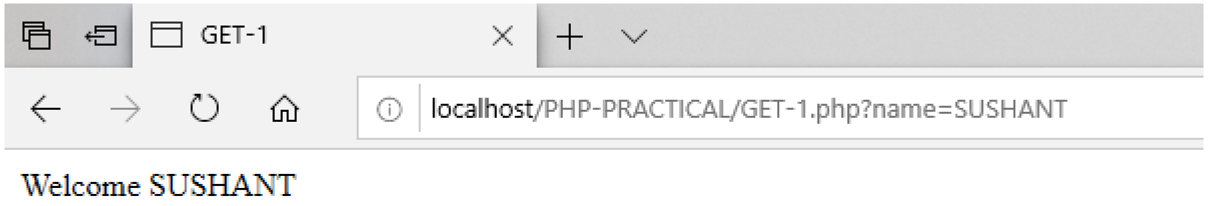
GET-1.php

```
<html>
 <body>
Welcome <?php echo $_GET["name"];
 </body>
 </html>
```


FORM-2

localhost/PHP-PRACTICAL/FORM-2.HTML

Name: SUSHANT Submit Query

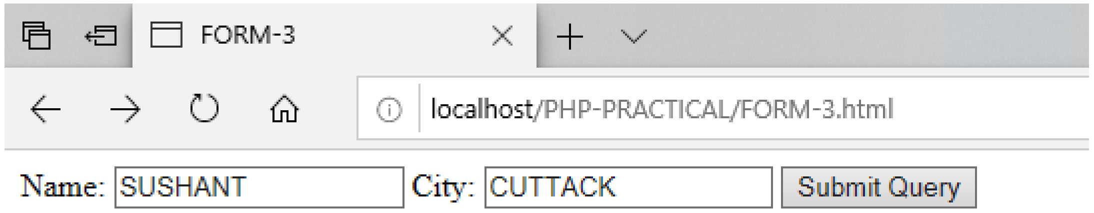

GET-1

localhost/PHP-PRACTICAL/GET-1.php?name=SUSHANT

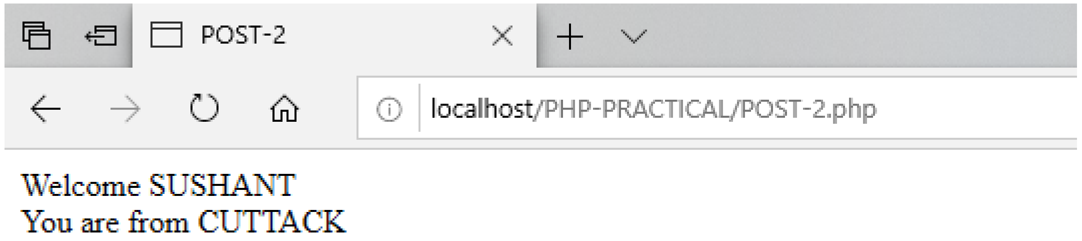
Welcome SUSHANT

- In GET method the data is sent as URL parameters that are usually strings of name and value pairs separated by ampersands (&).
- 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.

EXAMPLE OF POST METHOD

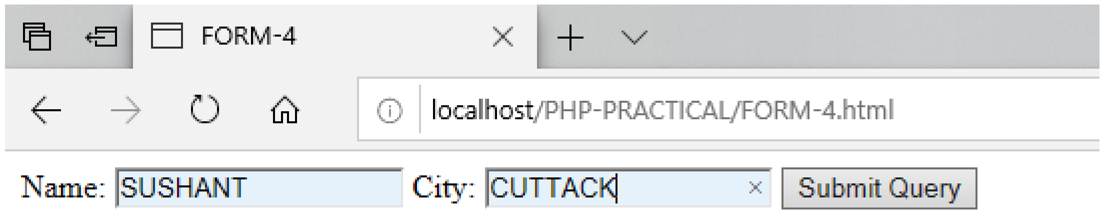


Name: SUSHANT City: CUTTACK

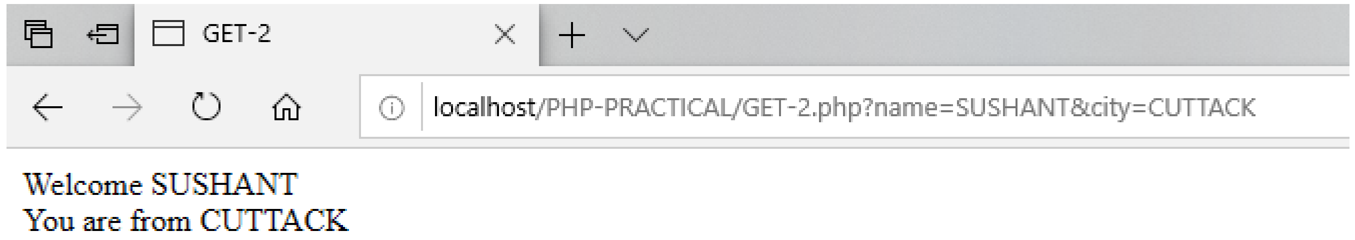


```
Welcome SUSHANT
You are from CUTTACK
```

EXAMPLE OF GET METHOD



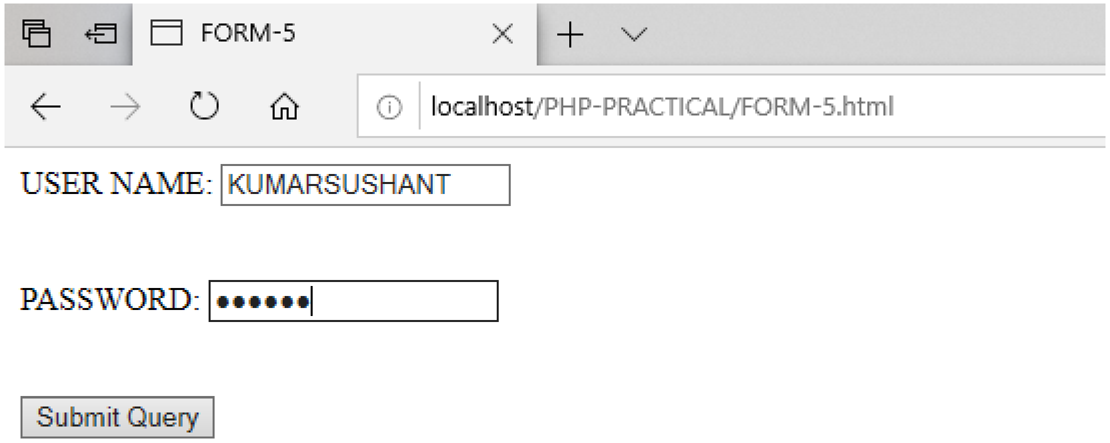
Name: SUSHANT City: CUTTACK



```
localhost/PHP-PRACTICAL/GET-2.php?name=SUSHANT&city=CUTTACK
```

Welcome SUSHANT
You are from CUTTACK

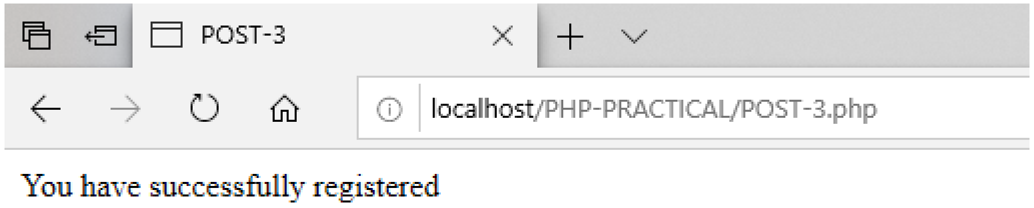
EXAMPLE OF POST METHOD


FORM-5

localhost/PHP-PRACTICAL/FORM-5.html

USER NAME: KUMARSUSHANT

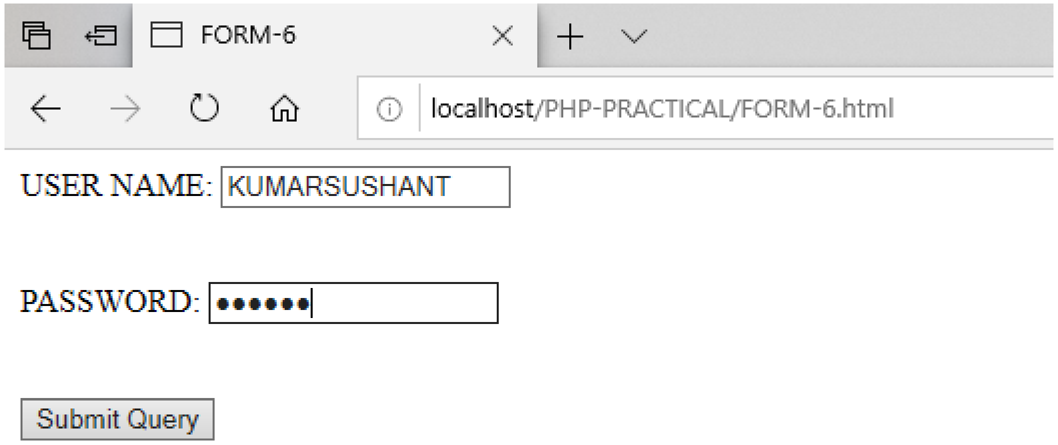
PASSWORD: ••••••


POST-3

localhost/PHP-PRACTICAL/POST-3.php

You have successfully registered

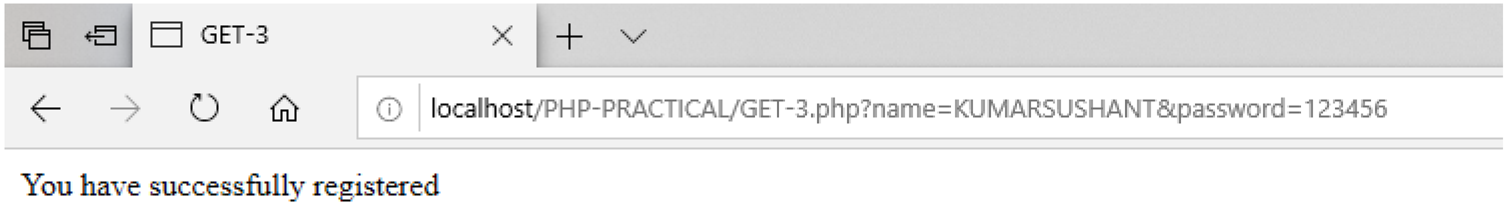
EXAMPLE OF GET METHOD



localhost/PHP-PRACTICAL/FORM-6.html

USER NAME: KUMARSUSHANT

PASSWORD: ●●●●●●

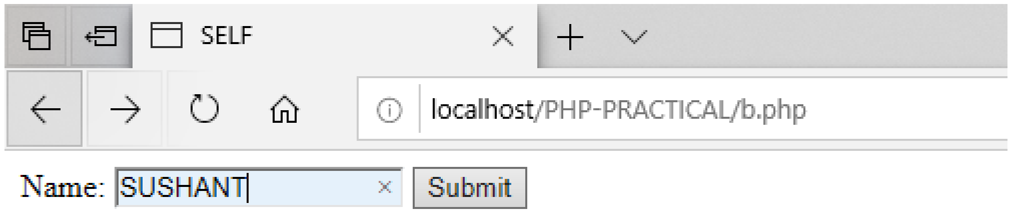


localhost/PHP-PRACTICAL/GET-3.php?name=KUMARSUSHANT&password=123456

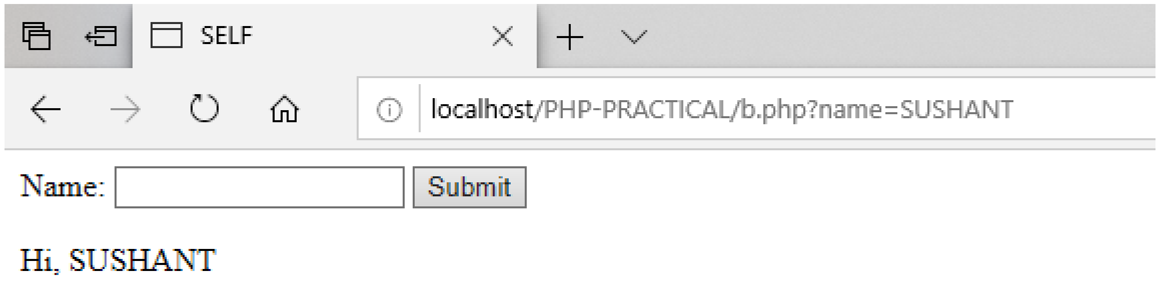
You have successfully registered

$_SERVER["PHP_SELF"] variable

- <form method="post" action="<?php echo $_SERVER["PHP_SELF"];?>">
- The $_SERVER["PHP_SELF"] is a super global variable that returns the filename of the currently executing script.
- So, the $_SERVER["PHP_SELF"] sends the submitted form data to the page itself, instead of jumping to a different page.
- This way, the user will get error messages on the same page as the form.
- The $_SERVER["PHP_SELF"] variable can be used by hackers!
- If PHP_SELF is used in your page then a user can enter a slash (/) and then some Cross Site Scripting (XSS) commands to execute.
- **Cross-site scripting (XSS) is a type of computer security vulnerability typically found in Web applications. XSS enables attackers to inject client-side script into Web pages viewed by other users.**



SELF
localhost/PHP-PRACTICAL/b.php
Name: SUSHANT
Submit



SELF
localhost/PHP-PRACTICAL/b.php?name=SUSHANT
Name:
Submit

```
Hi, SUSHANT
```

String Functions

- **1. trim ( ):** Removes whitespace or other characters from both sides of a string.

Syntax: trim(**string,characterlist**)

'**characterlist**' is Optional. Specifies which characters to remove from the string. If omitted, all of the following characters are removed.

```
<?php
$str = “Central University";
echo $str . "<br>";
echo trim($str, “Cesity");     // ntral Univer
?>
```

i) ltrim( ) - Removes whitespace or other predefined characters from the left side of a string.

ii) rtrim( ) - Removes whitespace or other predefined characters from the right side of a string.

- ltrim(**string, characterlist**)

```
<?php
 $str = “Central University";
 echo $str . "<br>";
 echo ltrim($str, “Cen");     // tral University
 ?>
```

- rtrim(**string, characterlist**)

```
<?php
 $str = “Central University";
 echo $str . "<br>";
 echo rtrim($str, “sity");     // Central Univer
 ?>
```

- 2. **strlen():** Returns the length of a string (in bytes) on success, and 0 if the string is empty.

Syntax: strlen(***string***)

```
<?php
  echo strlen(“Central University"); // 16
  ?>
```

3. **strcmp() :**The strcmp() function compares two strings. The strcmp() function is binary-safe and case-sensitive.

Syntax: strcmp(**string1,string2**)

This function returns: **0** - if the two strings are equal, **<0** - if string1 is less than string2 and **>0** - if string1 is greater than string2.

```
<?php
echo strncmp("CENTRAL","CENTRAL");  // 0
echo "<br>";
echo strncmp("CENTRAL","central");  //  -1
echo "<br>";
echo strncmp("central","CENTRAL”);  //  1
?>
```

- 4. **strncmp():** The strcmp() function compares two strings. This function is similar to the strcmp() function, with the difference that you can specify the number of characters from each string to be used in the comparison with strncmp(). The strcmp() function is binary-safe and case-sensitive.

Syntax: strncmp(***string1,string2,length***)

This function returns: **0** - if the two strings are equal, **<0** - if string1 is less than string2 and **>0** - if string1 is greater than string2.

```
<?php
echo strncmp("CENTRAL","CENTRAL",2);  // 0
echo "<br>";
echo strncmp("CENTRAL","central",2);  //  -1
echo "<br>";
echo strncmp("central","CENTRAL",2);  //  1
?>
```

- 5. i) **strpos() :**The strpos() function finds the position of the first occurrence of a string inside another string. The strpos() function is binary-safe and case-sensitive.
- Syntax: strpos(**string, find, start**)
- **string** -Specifies the string to search, **find**-Specifies the string to find & **start** - Optional. Specifies where to begin the search. If start is a negative number, it counts from the end of the string.

Returns the position of the first occurrence of a string inside another string, or FALSE if the string is not found.

Note: String positions start at 0, and not 1.

- ii) **stripos() :**The stripos() function finds the position of the first occurrence of a string inside another string. The stripos() function is binary-safe and **case-insensitive**.
- Syntax: stripos(**string, find, start**)

string -Specifies the string to search, **find**-Specifies the string to find & **start** - Optional. Specifies where to begin the search. If start is a negative number, it counts from the end of the string.

Returns the position of the first occurrence of a string inside another string, or FALSE if the string is not found.

iii) **strripos() :**The strripos() function finds the position of the last occurrence of a string inside another string. The strripos() function is binary-safe and **case-insensitive**.

- Syntax: strripos(**string, find, start**)

string -Specifies the string to search, **find-**Specifies the string to find & **start -** Optional. Specifies where to begin the search. If start is a negative number, it counts from the end of the string.

Returns the position of the last occurrence of a string inside another string, or FALSE if the string is not found.

iv) **strrpos() :**The strrpos() function finds the position of the last occurrence of a string inside another string. The strrpos() function is binary-safe and **case-sensitive**.

- Syntax: strrpos(**string, find, start**)

string -Specifies the string to search, **find-**Specifies the string to find & **start -** Optional. Specifies where to begin the search. If start is a negative number, it counts from the end of the string.

Returns the position of the last occurrence of a string inside another string, or FALSE if the string is not found.

- **v) strstr() :**The strstr() function searches for the first occurrence of a string inside another string. The strrpos() function is binary-safe and **case-sensitive**. For a case-insensitive search, use stristr() function.
- Returns the rest of the string (from the matching point), or FALSE, if the string to search for is not found.
- Syntax: strstr(**string, search, before_search**)
- **String** - Specifies the string to search, **search** - Specifies the string to search for. If this parameter is a number, it will search for the character matching the ASCII value of the number & **before_search** - Optional which is a boolean value whose default is "false". If set to "true", it returns the part of the string before the first occurrence of the search parameter.

- **vi) stristr() :** The stristr() function searches for the first occurrence of a string inside another string. The stristr() function is binary-safe and **case-insensitive**.
- Returns the rest of the string (from the matching point), or FALSE, if the string to search for is not found.
- Syntax: stristr(***string, search, before_search***)
- **string** - Specifies the string to search, **search** - Specifies the string to search for. If this parameter is a number, it will search for the character matching the ASCII value of the number & **before_search** – Optional which is a boolean value whose default is "false". If set to "true", it returns the part of the string before the first occurrence of the search parameter.

- <?php
- echo strpos("CENTRAL UNIVERSITY OF ODISHA OF CENTRAL GOVERNMENT","OF");//**19**
- echo "<br>";
- echo stripos("CENTRAL UNIVERSITY OF ODISHA OF CENTRAL GOVERNMENT","OF");//**19**
- echo "<br>";
- echo strripos("CENTRAL UNIVERSITY OF ODISHA OF CENTRAL GOVERNMENT","OF");//**29**
- echo "<br>";
- echo strrpos("CENTRAL UNIVERSITY OF ODISHA OF CENTRAL GOVERNMENT","OF");//**29**
- echo "<br>";
- echo "<br>";
- echo strstr("CENTRAL UNIVERSITY OF ODISHA OF CENTRAL GOVERNMENT","OF", **TRUE**);
- echo "<br>"; // **CENTRAL UNIVERSITY**

- echo stristr("CENTRAL UNIVERSITY OF ODISHA OF CENTRAL GOVERNMENT","OF", **FALSE**);
- echo "<br>"; // **OF ODISHA OF CENTRAL GOVERNMENT**
- ?>

- 6. **strtr() :**The strtr() function translates certain characters in a string. If the from and to parameters are different in length, both will be formatted to the length of the shortest.
- Syntax: strtr(***string, from, to***) or strtr(***string, array***)
- ***string*** - Specifies the string to translate, ***from*** - Specifies what characters to change, ***to*** - Specifies what characters to change into & ***array*** - An array containing what to change from as key, and what to change to as value.
- **Returns** the translated string. If the array parameter contains a key which is an empty string (""), it will return FALSE.

```
<?php
echo strtr("CENTRAL UNIVERSITY","NR", "MS");
echo "<br>"; //  CEMTSAL UMIVESSITY
echo "<br>";
$arr = array("CENTRAL" => "GOOD", "UNIVERSITY" => "STUDENT");
echo strtr("CENTRAL UNIVERSITY", $arr); // GOOD STUDENT
?>
```

- 7. **substr( ) :** The substr() function returns a part of a string.

Syntax: **substr(*string, start,l ength*)**

string - Specifies the string to return a part of,

start - Specifies where to start in the string(A positive number - Start at a specified position in the string, A negative number - Start at a specified position from the end of the string & 0 - Start at the first character in string)

- ***length*** - Optional. Specifies the length of the returned string. Default is to the end of the string.(A positive number - The length to be returned from the start parameter, Negative number - The length to be returned from the end of the string & If the length parameter is 0, NULL, or FALSE - it return an empty string)
- **Returns** the extracted part of a string, or FALSE on failure, or an empty string.

```
<?php
echo substr("Central University",10)."<br>";  // iversity
echo substr("Central University",1)."<br>";   // entral University
echo substr("Central University",3)."<br>";   // tral University
echo substr("Central University",7)."<br>";   //  University

echo substr("Central University",-1)."<br>";   //  y
echo substr("Central University",-10)."<br>";  // University
echo substr("Central University",-8)."<br>";    //  iversity
echo substr("Central University",-4)."<br>";    //  sity


echo substr("Central University",0,10)."<br>";    //  Central Un
echo substr("Central University",1,8)."<br>";     //  entral U
echo substr("Central University",0,5)."<br>";     //  Centr
echo substr("Central University",6,6)."<br>";     //  l Univ

echo substr("Central University",0,-1)."<br>";    //  Central Universit
echo substr("Central University",-10,-2)."<br>";  // Universi
echo substr("Central University",0,-6)."<br>";    // Central Univ
?>
```

- 8. **strrchr() :** The strrchr() function finds the position of the last occurrence of a string within another string, and returns all characters from this position to the end of the string. This function is binary-safe.
- Syntax: strrchr(***string, char***)
- *string* - Specifies the string to search & *char* - Specifies the string to find. If this is a number, it will search for the character matching the ASCII value of that number.
- **Returns** all characters from the last occurrence of a string within another string, to the end of the main string, or FALSE if the character is not found.

-
  ```
  <?php
  echo strrchr("CENTRAL UNIVERSITY", "IV"); //ITY
   ?>
  ```

- 9. **strchr() :** The strchr() function searches for the first occurrence of a string inside another string.
- This function is an alias of the strstr() function.
- **Note:** This function is binary-safe. This function is case-sensitive. For a case-insensitive search, use stristr() function.
- Syntax: strchr(**string, search, before_search**);

String - Specifies the string to search, **search** - Specifies the string to search for. If this parameter is a number, it will search for the character matching the ASCII value of the number & **before_search** – Optional which is a boolean value whose default is "false". If set to "true", it returns the part of the string before the first occurrence of the search parameter.

Returns the rest of the string (from the matching point), or FALSE, if the string to search for is not found.

-
  ```
  <?php
  echo strchr("CENTRAL UNIVERSITY", "IV")."<br>";  // IVERSITY
  echo strchr("CENTRAL UNIVERSITY", "IV", TRUE)."<br>";  //  CENTRAL UN
  ?>
  ```

- 10. **substr_compare() :** The substr_compare() function compares two strings from a specified start position. This function is binary-safe and optionally case-sensitive.
- Syntax: substr_compare(**string1,string2,startpos,length,case**)
- **string1** - Specifies the first string to compare, **string2** - Specifies the second string to compare, **startpos** - Specifies where to start comparing in string1. If negative, it starts counting from the end of the string, **length** – Optional which specifies how much of string1 to compare & **case** – Optional which a boolean value that specifies whether or not to perform a case-sensitive compare(FALSE - Default. Case-sensitive & TRUE - Case-insensitive).
- This function **returns:** 0 - if the two strings are equal, <0 - if string1 (from startpos) is less than string2 & >0 - if string1 (from startpos) is greater than string2. If length is equal or greater than length of string1, this function returns FALSE.

- echo substr_compare("CENTRAL UNIVERSITY","UNIV",6)."<br>"; // **-1**
- echo substr_compare("CENTRAL UNIVERSITY","IV",1,2)."<br>"; // **-1**
- echo substr_compare("CENTRAL UNIVERSITY","ER",-2,2)."<br>"; // **1**
- echo substr_compare("CENTRAL UNIVERSITY","RS",1,2)."<br>"; // **-1**
- echo substr_compare("CENTRAL UNIVERSITY","TR",1,2,TRUE)."<br>"; // **-15**
- echo substr_compare("CENTRAL UNIVERSITY","NT",1,3)."<br>"; // **-1**
- echo substr_compare("CENTRAL UNIVERSITY","NI",1,2)."<br>"; // **-1**
- echo substr_compare("CENTRAL UNIVERSITY","CENTRAL UNIVERSITY",0)."<br>"; // the two strings are equal   // **0**
- echo substr_compare("CENTRAL UNIVERSITY","CENTRAL",0)."<br>"; // string1 is greater than string2 //   **11**
- echo substr_compare("CENTRAL UNIVERSITY","CENTRAL UNIVERSITY OF ODISHA",0)."<br>"; // str1 is less than str2   // **-10**

- 11. **substr_count() :** The substr_count() function counts the number of times a substring occurs in a string. The substring is case-sensitive. This function does not count overlapped substrings .
- **Note:** This function generates a warning if the start parameter plus the length parameter is greater than the string length.
- Syntax: substr_count(**string, substring, start, length**)
- **string -** Specifies the string to check, **substring -** Specifies the string to search for, **start –** Optional which specifies where in string to start searching. If negative, it starts counting from the end of the string& **length –** Optional which specifies the length of the search.
- **Returns** the number of times the substring occurs in the string.

- $str = "This is nice";
- echo strlen($str)."<br>"; // Using strlen() to return the string length // **12**
- echo substr_count($str,"is")."<br>"; // The number of times "is" occurs in the string // **2**
- echo substr_count($str,"is",2)."<br>"; // The string is now reduced to "is is nice“ // **2**
- echo substr_count($str,"is",3)."<br>"; // The string is now reduced to "s is nice“ // **1**
- echo substr_count($str,"is",3,3)."<br>"; // The string is now reduced to "s i“ // **0**

- $str = "abcabcab";
- echo substr_count($str,"abcab"); // This function does not count overlapped substrings // **1**

- echo $str = "This is nice";
- substr_count($str,"is",3,9); // **This is nice**

- 12. substr_replace() :The substr_replace() function replaces a part of a string with another string. If the start parameter is a negative number and length is less than or equal to start, length becomes 0. This function is binary-safe.
- Syntax: substr_replace(***string, replacement, start, length***)
- **string -** Specifies the string to check, **replacement -** Specifies the string to insert, **start -** Specifies where to start replacing in the string(A positive number - Start replacing at the specified position in the string, Negative number - Start replacing at the specified position from the end of the string & 0 - Start replacing at the first character in the string),**length -** Optional. Specifies how many characters should be replaced. Default is the same length as the string(A positive number - The length of string to be replaced, A negative number - How many characters should be left at end of string after replacing & 0 - Insert instead of replace).
- **Returns** the replaced string. If the string is an array then the array is returned

- echo substr_replace("Hello world","earth",6)."<br>"; // **Hello earth**
- echo substr_replace("Hello world","earth",-5)."<br>"; // **Hello earth**
- echo substr_replace("world","Hello ",0,0)."<br>"; // **Hello world**
- $replace = array("1: AAA","2: AAA","3: AAA");
- echo implode("<br>",substr_replace($replace,'BBB',3,3)); //

  1: BBB
  2: BBB
  3: BBB

- **13. print( ):** The print() function outputs one or more strings. The print() function is slightly slower than echo().
- Syntax: **print(*strings*)**
- **Return :** Always returns 1
- Example:

-
  ```
  <?php
  $str = ”Computer Science";
  print $str;
  print ”$str”;
  print ’$str’;
  ?>
  ```
- Single quotes will print the variable name, not the value whereas double quotes will print the value.

- **14. echo() :**The echo() function outputs one or more strings. if you want to pass more than one parameter to echo(), using parentheses will generate a parse error. The echo() function is slightly faster than print().
- Syntax: **echo(*strings*)**
- **Return value:** No value is returned.
- Ex:

-
  ```
  <?php
  $str = ”Computer Science";
  echo $str;
  ?>
  ```

- **15. strrev() :**The strrev() function reverses a string.
- Syntax: **strrev(string)**
- Return: Returns the reversed string.
- **16. strtok() :**The strtok() function splits a string into smaller strings (tokens).
- Syntax: **strtok(*string, split*)**
- **string:** Specifies the string to split & **split:** Specifies one or more split characters.
- **Return value:** Returns a string token
-
```
<?php
$string = “I am a Computer Science Student";
$t = strtok($string, " ");

while ($t !== false)
{
echo "$t <br>";
$t = strtok(" ");
}
?> // I // am // a  // Computer  // Science // Student
```

- **17. str_repeat() :** The str_repeat() function repeats a string a specified number of times.
- Syntax: **str_repeat(string,repeat)**
- **string** - Specifies the string to repeat & **repeat -** Specifies the number of times the string will be repeated. Must be greater or equal to 0.
- Return value: **Returns the repeated string.**
- <?php
  echo str_repeat(“Hi",5); // **Hi Hi Hi Hi Hi**
  ?>

- **18. str_pad() :**The str_pad() function pads a string to a new length.
- Syntax: **str_pad(*string*, *length*, *pad_string*, *pad_type*)**
- ***string* -** Specifies the string to pad, ***length* -** Specifies the new string length. If this value is less than the original length of the string, nothing will be done, ***pad_string*** - Specifies the string to use for padding. Default is whitespace *&* ***pad_type* – (*STR_PAD_BOTH* -** Pad to both sides of the string. If not an even number, the right side gets the extra padding, ***STR_PAD_LEFT* -** Pad to the left side of the string & ***STR_PAD_RIGHT* -** Pad to the right side of the string. This is default*)*
- <?php
  $str = “DCS";
  echo str_pad($str,3,“U",STR_PAD_BOTH); // **UUUDCSUUU**
  ?>

- **19. str_split() :**The str_split() function splits a string into an array.
- Syntax: **str_split(string, length)**
- **string -** Specifies the string to split & **length** – Optional which specifies the length of each array element. Default is 1.
- Return value: If length is less than 1, the str_split() function will return FALSE. If length is larger than the length of string, the entire string will be returned as the only element of the array.

-
  ```
  <?php
   print_r(str_split("Hello")); // Array ( [0] => H [1] => e [2] => l [3] => l [4] => o )
   ?>
  ```

-
  ```
  <?php
     print_r(str_split("Hello",3));  // Array ( [0] => Hel [1] => lo )
   ?>
  ```

- **20. str_shuffle() :** The str_shuffle() function randomly shuffles all the characters of a string.
- Syntax: **str_shuffle(string)**
-
```
<html>
<body>
<?php
echo str_shuffle("Hello World");  // rHl eldolWo
?>
<p>Try to refresh the page. This function will randomly shuffle all characters each time.</p>
</body>
</html>
```
-

- **21. str_word_count() :** The str_word_count() function counts the number of words in a string.
- Syntax: **str_word_count(string, return, char)**
- **string -** Specifies the string to check, **return –** Optional which Specifies the return value of the str_word_count() function *(*Possible values: 0 - Default. Returns the number of words found, 1 - Returns an array with the words from the string & 2 - Returns an array where the key is the position of the word in the string, and value is the actual word*) &* **char –** Optional which specifies special characters to be considered as words.
- <?php

  echo str_word_count("Hello world!"); // **2**

  print_r(str_word_count("Hello world!",2)); // Array ([0] => Hello [6] => world)

  print_r(str_word_count("Hello world & good morning!",1));

// **Array ([0] => Hello [1] => world [2] => good [3] => morning)**

  print_r(str_word_count("Hello world & good morning!",1,"&"));

// **Array ([0] => Hello [1] => world [2] => & [3] => good [4] => morning)**

  ?>

- **22. strtolower() :** The strtolower() function converts a string to lowercase. This function is binary-safe.
- Syntax: **strtolower(string)**
- **23. strtoupper() :** The strtoupper() function converts a string to uppercase. This function is binary-safe.
- Syntax: **strtoupper(string)**
- <?php
  echo strtolower(“Computer science"); // **computer science**

  echo strtoupper(" Computer science "); // **COMPUTER SCIENCE**
  ?>

- **24. lcfirst() :** The lcfirst() function converts the first character of a string to lowercase.
- Syntax: **lcfirst(string)**
- **25. ucfirst() :** The ucfirst() function converts the first character of a string to lowercase.
- Syntax: **ucfirst(string)**
- **26. ucwords() :** The ucwords() function converts the first character of each word in a string to uppercase. This function is binary-safe.
- Syntax: **ucwords(string, delimiters)**
- **string -** Specifies the string to convert & **delimiters –** Optional which specifies the word separator character.
- <?php
  echo lcfirst(“Computer Science"); // **computer Science**
- echo ucfirst(" computer science"); // **Computer science**
- echo ucwords(" computer science "); // **Computer Science**
- echo ucwords(" Computer | Science ", "|"); // **Computer | Science**
  ?>

- **27. wordwrap() :** The wordwrap() function wraps a string into new lines when it reaches a specific length. This function may leave white spaces at the beginning of a line.
- Syntax: **wordwrap(string, width, break, cut)**
- **string -** Specifies the string to break up into lines,**width** - Specifies the maximum line width. Default is 75,**break** - Specifies the characters to use as break. Default is "\n"& **cut –** Optional which specifies whether words longer than the specified width should be wrapped: (**FALSE - Default. No-wrap & TRUE - Wrap**).
- Return value: **Returns the string broken into lines on success, or FALSE on failure.**
-
```
<?php
$str = “I am a Compute Science student";
echo wordwrap($str,15,"<br>\n",TRUE);
?>
```