Swami Sahajanand College of Computer Science

B.C.A. SEM - V

Subject: Web Application Development Using PHP

UNIT 2

Basic of PHP

- Conditional Statement
- Looping Statement
- Array- Types of Array(Numeric, Associative, Multidimensional)
- PHP Server variables
- Built-in-functions: String, Numeric, Date and Time.

♦ Conditional Statements

- **Decision Making Statement in PHP**
- In any language to solve a real time problem one definitely needs to put some conditional statement. Those statements are known as Decision Making Statement.
- PHP supports 4 Decision Making Statement.
- If Statement
- If…else statement
- If…elseif…else statement
- Switch Statement

1) If Statement:

- If Statement is used to execute some code only if a specified condition is true
- It is a simple decision making statement.

Syntax

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

Example

```
<?php
  $a=5;
  $b=7;
  If($a<$b)
  {
     echo "B value is higher.";
  }
?>
```

2) If…Else Statement:

- Use the if....else statement to execute some code if a condition is true and another code if a condition is false.
- It is called two way decision making statements.

Syntax

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

Example

```
<?php
  $a=5;
```

```
    $b=7;
    If($a<$b)
    {
       echo “B value is higher.”;
    }
    else
    {
        echo “A value is higher.”;
    }
?>
```

3) If…elseif…else Statement:

- if....elseif…else statement is used when the number of condition are more.
- It is useful when developer need to check multiple conditions at same level.
- If first condition is evaluated as false then second condition is checked.
- If second condition is evaluated as false then third condition is checked and so on.
- If first condition is true then statement set – A executes. And remaining condition will not be checked.
- If second condition is true then statement set – B executes and remaining condition will not be checked and so on.
- If all condition are false then statement set – X given with else part executes.

Syntax

```
if (condition)
{
        // statement set - A;
}
elseif(condition)
{
        // statement set – B;
}
else
{
        // statement set - X;
}
```

Example

```
<?php
    $a=5;
    $b=7;
    If($a<$b)
    {
       echo “B value is higher.”;
```

```
}
Elseif($a>b)
{
    echo “A value is higher.”;
}
else
{
    echo “A and B value are same”;
} ?>
```

4) Switch Statement:

- Switch statement is used when multiple option are there and to select specific option based upon equality of value.
- Switch statement are also known as Branching Statement.
- Switch statement can be used with character and integer value only.
- Switch statement can not be used with relational operator like < > <= ?=>= != etc.

Syntax

```
switch (n)
{
  case label1:
  code to be executed if n=label1;
  break;

  case label2:
  code to be executed if n=label2;
  break;

  default:
  code to be executed if n is different from both label1 and label2;
}
```

Example

```
<?php
 $x=1;
 switch ($x)
 {
        case 1:
          echo "Number 1";
          break;
        case 2:
          echo "Number 2";
          break;
        case 3:
```

```
        echo "Number 3";
        break;
      default:
        echo "No number between 1 and 3";
} ?>
```

♦ Looping Structure in PHP

- When you want to perform some task repeatedly that time loop is usful.
- In PHP we have following Looping Structures.
  1. While Loop
  2. Do-while Loop
  3. For loop
  4. Foreach loop

1) While Loop

- While loop is entry control loop that it checks the condition at time of entry in a loop.
- Syntax and Example are shown below illustrate declaration While Loop.

Syntax

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

Example

```
<?php
    $i=1;
    while($i<=5)
    {
        echo "The number is " . $i . "<br />";
        $i++;
    }
?>
```

Output

```
The number is 1
The number is 2
The number is 3
The number is 4
The number is 5
```

2) Do While Loop

- Do While loop is exit control loop that it checks the condition at time of exit in a loop.
- The body of loop is executing at least one time if condition is false from beginning.
- Syntax and Example are shown below illustrate declaration Do While Loop.

Syntax

```
do
```

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

Example

```
<?php
    $i=1;
    do
    {
        $i++;
        echo "The number is " . $i . "<br />";
    } while ($i<=5);
?>
```

Output

```
The number is 2
The number is 3
The number is 4
The number is 5
The number is 6
```

3) For Loop

- for loop is used when you know in advance how many times the script should run.
- Syntax and Example are shown below illustrate declaration For Loop.

Syntax

```
for (init; condition; increment)
{
    code to be executed;
}
```

Parameters:

- **Init:** Mostly used to set a counter (but can be any code to be executed once at the beginning of the loop)
- **Condition:** Evaluated for each loop iteration. If it evaluates to TRUE, the loop continues. If it evaluates to FALSE, the loop ends.
- **Increment:** Mostly used to increment a counter (but can be any code to be executed at the end of the iteration)

Example

```
<?php
    for ($i=1; $i<=5; $i++)
    {
        echo "The number is " . $i . "<br />";
    }
?>
```

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

4) Foreach Loop

- The foreach loop is used with the array.
- For every loop iteration, the value of the current array element is assigned to $value (and the array pointer is moved by one) - so on the next loop iteration, you'll be looking at the next array value.
- Syntax and Example are shown below illustrate declaration Foreach Loop.

Syntax

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

Example

```
<?php
   $x=array("one","two","three");
   foreach ($x as $value)
  {
     echo $value . "<br />";
  }
?>
```

Output
one
two
three

♦ What is an Array? Explain Types of Array.

- Before we learn about array let us revise what is variable?
- A variable is a storage area holding a number or text. The problem is, a variable will hold only one value.
- **An array is a special variable, which can store multiple values in one single variable.**
- If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:
- $cars1="MARUTI";
  $cars2="SANTRO";
  $cars3="HONDA";
- It is not possible to use loop with above three variable to process the value of each variable
  More about array
- An array can hold all your variable values under a single name.
- And you can access the any of the values by referring to the array name.
- Each element in the array has its own index so that it can be easily accessed.

- In PHP, there are three kind of arrays:

–**Numeric array**

- An array with a numeric index

–**Associative array**

- An array where each ID key is associated with a value

–**Multidimensional array**

- An array containing one or more arrays

Numeric Arrays

- A numeric array stores each array element with a numeric index.
- There are two methods to create a numeric array.
- In the following example the index are automatically assigned (the index starts at 0):
- $cars=array("MARUTI","AUDI","BMW","Toyota");
- In the following example we assign the index manually:

```
$cars[0]=" MARUTI ";
$cars[1]=" AUDI ";
$cars[2]=" BMW ";
$cars[3]=" Toyota ";
```

Associative Arrays

- An associative array, each ID key is associated with a value.
- When storing data about specific named values, a numerical array is not always the best way to do it.
- With associative arrays we can use the values as keys and assign values to them.

•**Example 1**

- In this example we use an array to store various attributes of single person.
- $ages = array("name"=>"mohan", "age"=>30, "gender"=>true);

♦ PHP Server Veriables

- **$_SERVER** is an array which holds information of headers, paths, script locations.
- Web server creates the entries in the array.
- This is not assured that every web server will provide similar information, rather some servers may include or exclude some information which are not listed here.
- $_SERVER has following basic properties:
- Set by web server.
- Directly related to the runtime environment of the current php script.
- It does the same job as $HTTP_SERVER_VARS used to do in previous versions of PHP

'PHP_SELF'

- The filename of the currently executing script, relative to the document root. For instance, $_SERVER['PHP_SELF'] in a script at the address http://example.com/test.php/foo.bar would be /test.php/foo.bar.

'SERVER_NAME'

- The name of the server host under which the current script is executing.
- If the script is running on a virtual host, this will be the value defined for that virtual host.

SERVER_SOFTWARE'

- Server identification string, given in the headers when responding to requests.

'***SERVER_PROTOCOL***'
- Name and revision of the information protocol via which the page was requested; i.e. '*HTTP/1.0*';

'***REQUEST_METHOD***'
- Which request method was used to access the page; i.e. '*GET*', '*HEAD*', '*POST*', '*PUT*'.

'***REQUEST_TIME***'
- The timestamp of the start of the request. Available since PHP 5.1.0.

'***QUERY_STRING***'
- The query string, if any, via which the page was accessed.

'***DOCUMENT_ROOT***'
- The document root directory under which the current script is executing, as defined in the server's configuration file.

'*HTTP_USER_AGENT*'
- Contents of the *User-Agent:* header from the current request, if there is one.
- This is a string denoting the user agent being which is accessing the page. A typical example is: Mozilla/4.5 [en] (X11; U; Linux 2.2.9 i586).

REMOTE_ADDR'
- The IP address from which the user is viewing the current page.

'*REMOTE_PORT*'
- The port being used on the user's machine to communicate with the web server.

'*SCRIPT_FILENAME*'
- The absolute pathname of the currently executing script.

SCRIPT_NAME'
- Contains the current script's path. This is useful for pages which need to point to themselves. The __FILE__ constant contains the full path and filename of the current (i.e. included) file.

'*REQUEST_URI*'
- The URI which was given in order to access this page; for instance, '*/index.html*'.

♦ Built in Functions

- **Array()**
- It is used to create an array.
- **array()** is a language construct, and not a regular function.
- Returns an array of the parameters. The parameters can be given an index with the => operator.

•For example

```
<?php
$array = array(1, 1, 1, 1,  1, 8 => 1,  4 => 1, 19, 3 => 13);
print_r($array);
?>
```

Sort()
- This function sorts an array.
- Elements will be arranged from lowest to highest when this function has completed.
- This function assigns new keys for the elements in *array*.
- It will remove any existing keys you may have assigned, rather than just reordering the keys.

•It has following syntax

•bool **sort** (array &array [, int sort_flags])

- **SORT_REGULAR** - compare items normally (don't change types)
- **SORT_NUMERIC** - compare items numerically
- **SORT_STRING** - compare items as strings

asort()

- arsort -- Sort an array in given order and maintain index association.
- This function sorts an array such that array indices maintain their correlation with the array elements they are associated with.
- This is used mainly when sorting associative arrays where the actual element order is important.
- bool **asort** (array &array [, int sort_flags])
- The second argument is flag to sort the value of array and it is same as sort() function.

Count()

- count -- Count elements in an array, or properties in an object
- **count()** may return 0 for a variable that isn't set, but it may also return 0 for a variable that has been initialized with an empty array. 0
- It has following sytax.
  int **count** (mixed var)

Sizeof()

- This function is actually alias of count() function.
- It also return the size of array passed as an argument in the sizeof() function.
- It has following syntax
- int **sizeof** (mixed var)

extract()

- Import variables into the current symbol table from an array
- It takes an associative array *var_array* and treats keys as variable names and values as variable values.
- For each key/value pair it will create a variable in the current symbol table, subject to *extract_type* and *prefix* parameters.
- It has following syntax.
- int extract (array var_array [, int extract_type [, string prefix]]).

List()

- Assign variables as if they were an array.
- Like **array()**, this is not really a function, but a language construct. **list()** is used to assign a list of variables in one operation.
- **list()** only works on numerical arrays and assumes the numerical indices start at 0.
- It has following syntax.
- void **list** (mixed varname, mixed ...)

•For example

```
$info = array('coffee', 'brown', 'caffeine');

list($drink, $color, $power) = $info;

echo "$drink is $color and $power makes it special.\n";
```

array_merge()

- Merge one or more arrays and then It returns the resulting new array.
- **array_merge()** merges the elements of one or more arrays together so that the values of second are appended to the end of the first array.

- If the input arrays have the same string keys, then the second value for that key will overwrite the first one.
- If, however, the arrays contain numeric keys, the later value will **not** overwrite the original value, but will be appended.
- If only one array is given and the array is numerically indexed, the keys get reindexed in a continuous way.
- It has following syntax.
- array array_merge (array array1 [, array array2 [, array ...]])

Array_push

- Push one or more elements onto the end of array.
- a**rray_push()** treats *array* as a stack, and pushes the passed variables onto the end of *array*.
- The length of *array* increases by the number of variables pushed.
- it has following syntax
- int **array_push** (array &array, mixed var [, mixed ...])

Example

```
<?php
$stack = array("orange", "banana");
array_push($stack, "apple", "raspberry");
print_r($stack);
?>
```

array_pop

- Pop the element from the end of array.
- **array_pop()** pops and returns the last value of the *array*, shortening the *array* by one element.
- If *array* is empty (or is not an array), **NULL** will be returned.
- It has following syntax
- mixed **array_pop** (array &array)

Example

```
<?php
$stack = array("orange", "banana", "apple", "raspberry");
$fruit = array_pop($stack);
print_r($stack);
?>
```

array_reverse()

- Return an array with elements in reverse order
- **array_reverse()** takes input *array* and returns a new array with the order of the elements reversed, preserving the keys if *preserve_keys* is **TRUE**.

•It has following syntax

•array **array_reverse** (array array [, bool preserve_keys])

```
<?php
$input  = array("php", 4.0, array("green", "red"));
$result = array_reverse($input);
```

```
$result_keyed = array_reverse($input, true);
?>
```

array_sum

- array_sum -- Calculate the sum of values in an array
- **array_sum()** returns the sum of values in an array as an integer or float.
- It has following syntax
- number **array_sum** (array array)
  example

```
<?php
$a = array(2, 4, 6, 8);
echo "sum(a) = " . array_sum($a) . "\n"; ?>
```

Implode()

- implode -- Join array elements with a string

syntax

- string implode (string glue, array pieces)
- Returns a string containing a string representation of all the array elements in the same order, with the glue string between each element.

implode() example

```
<?php
$array = array('lastname', 'email', 'phone');
$comma_separated = implode(",", $array);
echo $comma_separated; // lastname,email,phone
?>
```

Explode()

- Split a string by string
- Returns an array of strings, each of which is a substring of *string* formed by splitting it on boundaries formed by the string *separator*.
- If *limit* is set, the returned array will contain a maximum of *limit* elements with the last element containing the rest of *string*.
- It has following syntax
- array **explode** (string separator, string string [, int limit])
  example

```
    $pizza  = "piece1 piece2 piece3 piece4 piece5 piece6";
$pieces = explode(" ", $pizza);
echo $pieces[0]; // piece1
echo $pieces[1]; // piece2
```

♦ Maths Related inbuilt Function in PHP

abs()

- Abs() function is used to get absolute value of passed value.
- If the argument number is of type float, the return type is also float, otherwise it is integer
- It has following syntax.
- number **abs** (mixed number)

•Let us see example

```
<?php
$abs = abs(-4.2); // $abs = 4.2; (double/float)
$abs2 = abs(5);   // $abs2 = 5; (integer)
$abs3 = abs(-5);  // $abs3 = 5; (integer)
?>
```

bindec()

- Returns the decimal equivalent of the binary number given as *binary_string* argument.
- bindec() converts a binary number to an integer
- It has following syntax.
- number **bindec** (string binary_string)
  let us see example

```
<?php
echo bindec('110011') . "\n";
echo bindec('000110011') . "\n";
echo bindec('111');
?>
```

decbin()

- It is used to returns a string containing a binary representation of the given *number* argument.
- The largest number that can be converted is 4,29,496,7295 in decimal resulting to a string of 32 1's.
- It has following syntax
- string **decbin** (int number)
  it has following syntax.

```
<?php
echo decbin(12) . "\n";
echo decbin(26);
?>
```

decoct()

- It is used to return string containing an octal representation of the given *number* argument.
- The largest number that can be converted is 4294967295 in decimal resulting to "37777777777".
- It has following syntax.
- string **decoct** (int number)
  let us see an example

```
<?php
echo decoct(15) . "\n";
echo decoct(264);
?>
```

octdec()

- It is used to returns the decimal equivalent of the octal number represented by the *octal_string* argument.
- The largest number that can be converted is 17777777777 or 2147483647 in decimal.
- It has following syntax
- number **octdec** (string octal_string)

now let us see an example

```
<?php
echo octdec('77') . "\n";
echo octdec(decoct(45));
?>
```

hexdec()

- It is used to return decimal equivalent of the hexadecimal number represented by the *hex_string* argument.
- **hexdec()** converts a hexadecimal string to a decimal number. The largest number that can be converted is 7fffffff or 2147483647 in decimal.
- It does not consider any invalid hexadecimal string.

•It has following syntax.
number **hexdec** (string hex_string)
now let us see an example

```
<?php
echo  hexdec ('1AF') . "\n";
?>
```

dechex()

- Returns a string containing a hexadecimal representation of the given *number* argument.
- The largest number that can be converted is 4294967295 in decimal resulting to "ffffffff".
- It has following syntax.
- **string dechex** (int number)

now let us see an example

```
<?php
echo dechex(10) . "\n";
echo dechex(47);
?>
```

ceil()

- Returns the next highest integer value by rounding up *value* if necessary.
- It has following syntax

- float **ceil** (float value)
  let us see an example

```
<?php
echo ceil(4.3);    // 5
echo ceil(9.999);  // 10
?>
```

floor()

- Returns the next lowest integer value by rounding down *value* if necessary.
- It has following syntax

float **floor** (float value)

```
<?php
echo floor(4.3);   // 4
echo floor(9.999); // 9
?>
```

round()

- Returns the rounded value of *argument* to specified *precision*.
- *precision* can also be negative or zero (default).

•It has following syntax

• float **round** (float val [, int precision])

```
<?php
echo round(3.4);         // 3
echo round(3.5);         // 4
echo round(3.6);         // 4
echo round(3.6, 0);      // 4
echo round(1.95583, 2);  // 1.96
echo round(1241757, -3); // 1242000
echo round(5.045, 2);    // 5.05
echo round(5.055, 2);    // 5.06
?>
```

pi()

- Returns an approximation of pi.
- It has following syntax
- float **pi** (void)
  now let us see an example

```
<?php
echo pi(); // 3.1415926535898
```

```
echo M_PI; // 3.1415926535898
?>
```

pow()

- Returns *base* raised to the power of *exp*.
- It has following syntax
- number **pow** (number base, number exp)
  now let us see an example

```
<?php
var_dump(pow(2, 8)); // int(256)
echo pow(-1, 20); // 1
echo pow(0, 0); // 1
echo pow(-1, 5.5); // error
?>
```

rand()

- It is used to return random number between given minimum and maximum value.

•If called without the optional *min*, *max* arguments **rand()** returns a pseudo-random integer between 0 and **RAND_MAX**.

- It has following syntax
- int **rand** ([int min, int max])
  now let us see an example

```
<?php
echo rand() . "\n";
echo rand() . "\n";
echo rand(5, 15);
?>
```

sqrt()

- It is used to return squre root of the given number.
- It has following syntax
- float **sqrt** (float arg)
  example

```
<?php
echo sqrt(9); // 3
?>
```

min()

- It is used to findout minimum number from the list of value.
- It has following syntax.
- mixed **min** (number arg1, number arg2 [, number ...])
  example

```
<?php
echo min(2, 3, 1, 6, 7);  // 1
echo min(array(2, 4, 5)); // 2
```

```
?>
```

max()

- It is used to findout maximum value from the list of the supplied value as argument.
- It has following syntax.
- mixed **max** (number arg1, number arg2 [, number ...])
  example

```
<?php
echo max(1, 3, 5, 6, 7);  // 7
echo max(array(2, 4, 5)); // 5
?>
```

♦ String Related inbuilt Function

echo()

- It is used to output one or more strings.
- It has following
- void **echo** (string arg1 [, string ...])
- **echo ()** is not actually a function (it is a language construct), so you are not required to use parentheses with it.

printf()

- It is used to Output a formatted string

•It has following syntax

- int **printf** (string format [, mixed args [, mixed ...]])
- Produces output according to *format*

Example

```
<?php
$s = 'monkey';
$t = 'many monkeys';
printf("[%s]\n",      $s); // standard string output
printf("[%10s]\n",    $s); // right-justification with spaces
printf("[%-10s]\n",   $s); // left-justification with spaces
printf("[%010s]\n",   $s); // zero-padding works on strings too
printf("[%'#10s]\n",  $s); // use the custom padding character '#'
?>
```

join()

- Join() is an alias of **implode()** function.
- So whatever we can do with implode can also be done using join() function.
- It means returns a string containing a string representation of all the array elements in the same order, with the separator string between each element.

print()

- It is used to output string.

- **print()** is not actually a real function (it is a language construct) so you are not required to use parentheses with its argument list.
- It has following syntax
- int **print** (string arg)

fprintf()

- fprintf() writes a formatted string to a stream.
- Write a string produced according to *format* to the stream resource specified by *handle*.
- It has following syntax.
- int **fprintf** (resource handle, string format [, mixed args [, mixed ...]])

Example

```
<?php
if (!($fp = fopen('date.txt', 'w')))
   return;

fprintf($fp, "%04d-%02d-%02d", $year, $month, $day);
// will write the formatted ISO date to date.txt
?>
```

sprintf()

- The sprintf() function writes a formatted string to a variable.
- It has following syntax.
- sprintf(format,arg1,arg2,arg++)
- The arg1, arg2, ++ parameters will be inserted at percent (%) signs in the main string.
- This function works "step-by-step". At the first % sign, arg1 is inserted, at the second % sign, arg2 is inserted, etc.
- each conversion specification consists of a percent sign (*%*).
- It may be followed by type specifier.

Possible format values

- %% - Returns a percent sign
- %b - Binary number
- %c - The character according to the ASCII value
- %d - Signed decimal number
- %e - Scientific notation (e.g. 1.2e+2)
- %u - Unsigned decimal number
- %f - Floating-point number (local settings aware)
- %F - Floating-point number (not local settings aware)
- %o - Octal number
- %s - String
- %x - Hexadecimal number (lowercase letters)
- %X - Hexadecimal number (uppercase letters)

Example 1

```
?php
$str = “Mr Anand shah";
$number = 30;
```

```
$txt = sprintf("Hello %s how are you. Your lucky number is  %u",$str,$number);
echo $txt;
?>
Example 2
<?php
$pivalue = 3.14;
$value = sprintf("%f",$ pivalue );
echo $value;
?>
```

ltrim()

- The ltrim() function will remove whitespaces or other predefined character from the left side of a string.
- The list of predefined character can be any of the following.
- "\0" - NULL
- "\t" - tab
- "\n" - new line
- "\x0B" - vertical tab
- "\r" - carriage return
- " " - ordinary white space

ltrim()

- **It has the following Syntax**
- ltrim(string,charlist)

Example

```
<html>
<body>
<?php
$str = "    SSCCS!";
echo "Without ltrim: " . $str;
echo "<br />";
echo "With ltrim: " . ltrim($str);
?>
<body>
<html>
```

rtrim()

- The rtrim() function will remove whitespaces or other predefined character from the right side of a string.
- **It has the following Syntax**
- rtrim(string,charlist)

Example

```
<?php
$str = “123456!    ";
echo "Without rtrim: " . $str . “*”;
echo "<br />";
echo "With rtrim: " . rtrim($str) . “*”;
?
```

trim()

- The trim() function removes whitespaces and other predefined characters from both sides of a string.
- **It has following Syntax**
- trim(string,charlist)

Example

```
<?php
$str = " ssccs! ";
echo "Without trim: " . $str;
echo "<br />";
echo "With trim: " . trim($str);
?>
```

str_pad()

- The str_pad() function pads a string to a new length.
- Syntax
- str_pad(string,length,[pad_string,pad_type])
- First argument is string to pad,
- second argument is length of padding,
- third argument is padding string
- and last argument is type of padding which can be any one of the following.

–**STR_PAD_BOTH –**

- Pad to both sides of the string.
- 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

Example 1

```
<?php
$str = "Hello World";
echo str_pad($str,20,".");
?>
```

The output of the code above will be:

Hello World.....

Example 2

```
<?php
$str = "Hello World";
echo str_pad($str,20,".",STR_PAD_LEFT);
?> The output of the code above will be:
.....Hello World
```

str_repeat()

- The str_repeat() function repeats a string a specified number of times.
- **Syntax**
- str_repeat(string,repeat)

Example

```
<?php
echo str_repeat("*",15);
?>
```

The output of the code above will be:

****

str_replace()

- The str_replace() function replaces some characters with some other characters in a string.
- This function works by the following rules:
- If the string to be searched is an array, it returns an array
- If the string to be searched is an array, find and replace is performed with every array element.
- If both find and replace are arrays, and replace has fewer elements than find, an empty string will be used as replace.
- If find is an array and replace is a string, the replace string will be used for every find value.

str_replace()

- Syntax
  str_replace (find , replace , string , [count])
- Parameter Description
  - Find
    Specifies the value to find
  - Replace
    Specifies the value to replace the value in *find*
  - String
    Specifies the string to be searched
  - Count
    A variable that counts the number of replacements

Example

```
<?php
echo str_replace("hi Earth","Earth","Hello world!");
?>
```

The output of the code above will be:

Hi Hello World!

str_split()

- The str_split() function splits a string into an array.
- Syntax
- str_split(string,length)
- Parameter Description
- String
- Specifies the string to split
- Length
- Specifies the length of each array element. Default is 1

Example

```
<?php
print_r(str_split("ssccs"));
?>
```

The output of the code above will be:

```
Array
(
[0] => s
[1] => v
[2] => c
[3] => c
[4] => s
)
```

strcmp()

- The strcmp() function compares two strings.
- This function returns:
- 0 - if the two strings are equal
- <0 - if string1 is less than string2
- >0 - if string1 is greater than string2
- **Syntax**
- strcmp(string1,string2)
- String1 and string2 are the two string to be compared.

Example

```
<?php
if(strcmp("ssccs","ssccs")==0)
        {
                echo "Both string are same
        }
?>
```

strpos()

- The strpos() function returns the position of the first occurrence of a string inside another string.
- If the string is not found, this function returns FALSE.

- **Syntax**
- strpos(string,find,start)
- **Parameter Description**
- String
- Specifies the string to search
- find
- Specifies the string to find
- start
- Specifies where to begin the search

Example

```
<?php
echo strpos("Hello ssccs",“sv");
?>
```

The output of the code above will be:

6

strlen()

- The strlen() function returns the length of a string.
- **Syntax**
- strlen(string)

Example

```
<?php
echo strlen("Hello ssccs!");
?>
```

- The output of the code above will be:
- 12

strrev()

- the strrev() function reverses a string.

- **Syntax**
  - strrev(string)

Example

```
<?php
echo strrev(“ssccs");
?>
```

- The output of the code above will be:
- sccvs

strtolower()

- The strtolower() function converts a string to lowercase.
- **Syntax**

- strtolower(string)

```
<?php
        echo strtolower("SWAMI VIVEK.");
?>
```

- The output of the code above will be:
- swami vivek.

strtoupper()

- The strtoupper() function converts a string to uppercase.
- Syntax
- strtoupper(string)

```
<?php
echo strtoupper("Hello WORLD!");
?>
```

- The output of the code above will be:
- HELLO WORLD!

wordwrap()

- The wordwrap() function wraps a string into new lines when it reaches a specific length.
- This function returns the string broken into lines on success, or FALSE on failure.
- **Syntax**
  wordwrap(string,[width,break,cut])
- Parameters
- String
  Specifies the string to break up into.
- lineswidth.
  Specifies the maximum line width. Default is 75
- Break
  Specifies the characters to use as break. Default is "\n"
- cut.
  Specifies whether words longer than the specified width should be wrapped. Default is FALSE (no-wrap)

Example

```
<?php
$text = "The quick brown fox jumped over the lazy dog.";
$newtext = wordwrap($text, 20, "<br />");
echo $newtext;
?>
```

The above example will output:

The quick brown fox
 jumped over the lazy
 dog.

substr()

- it is used to return part of the string.
- It has following syntax
- string **substr** (string string, int start [, int length])

- **substr()** returns the portion of *string* specified by the *start* and *length* parameters.

Example

```
<?php
echo substr('abcdef', 1);     // bcdef
echo substr('abcdef', 1, 3);  // bcd
echo substr('abcdef', 0, 4);  // abcd
echo substr('abcdef', 0, 8);  // abcdef
echo substr('abcdef', -1, 1); // f
?>
```

Date & time relation Library function

date()

- It is used to return current date and time.
- The return value of this function depends upon the parameter passed to it.
- By default it returns only date if the second argument is not supplied to it.
- It has following syntax.
- string **date** (string format [, int timestamp])
  one can pass one or more format parameter in 1st argument

Time()

- time() returns Return current Unix timestamp.
- It has following syntax.
- **int time (void)**
  actually it returns the current time measured in the number of seconds since the Unix Epoch (January 1 1970 00:00:00 GMT).

Example

```
<?php
echo time();
?>
```

Strtotime()

- The strtotime() function parses an English textual date or time into a Unix timestamp (the number of seconds since January 1 1970 00:00:00 GMT).
- It has following syntax
- strtotime(time,[now])

Example

```
<?php
echo(strtotime("now") . "<br />");
echo(strtotime("3 October 2005") . "<br />");
echo(strtotime("+5 hours") . "<br />");
echo(strtotime("+1 week") . "<br />");
echo(strtotime("+1 week 3 days 7 hours 5 seconds") . "<br />");
echo(strtotime("next Monday") . "<br />");
```

```
echo(strtotime("last Sunday"));
?>
```

Mktime()

- Mktime() function is used to get Unix timestamp for a date.
- This timestamp contains the number of seconds between the Unix Epoch (January 1 1970 00:00:00 GMT) and the time specified.
- **Syntax**
- mktime(hour,minute,second,month,day,year,is_dst)

Mktime() ...

Example

```
<?php
echo(date("M-d-Y",mktime(0,0,0,12,36,2005))."<br />");
echo(date("M-d-Y",mktime(0,0,0,14,1,2013))."<br />");
echo(date("M-d-Y",mktime(0,0,0,1,1,2007))."<br />");
echo(date("M-d-Y",mktime(0,0,0,1,1,97))."<br />");
?>
```

The output of the code above would be:

Jan-05-2005
Feb-01-2013
Jan-01-2007
Jan-01-1997