Array & Functions

Created By :- The EasyLearn Academy

What is an Array?

- **An array is a special variable, which can store multiple values in one single variable.**
- **Each variable's value can be accessed by unique number known as element which is always unsigned integer.**
- If we need to store/process/display so many related values like list of student, list of car, list of customer, then array can be useful data structure.
- We can process/display array's element value one by one using loops.
- Array reduce overall code length in program.

Array in PHP is different from array in other language IIke C/C++/JAVA

- Array in PHP can store any type values of in it. For example first value in array can be string, second value can be integer third value can be Boolean. **It means there is no specific data type of array in PHP**.
- Array in PHP is dynamic means **we can add any number of values in it at any time**.
- Array don't have to store all values in sequence. It means first element of array is not always has 0 index as well as next value can be at any index. It means indexes many not be adjacent.
- We can also create array which has no index. We can access such array and it's value using keys. Keys are always string

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 and its index/key.
- Each element in the array has its own index/jet so that it can be easily accessed/changed.
- 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 has value then can be accessed/changed/removed using index.
- Index is always positive integer number.
- There are two methods to create a numeric array.

1st method

```
• $cars=array(“MARUTI",“AUDI","BMW","Toyota");
• $number=array(11,25,100,200,500,2500);
```

2nd method

```
• $student[0]=" Kilosnikov";
  $student [1]=" vijay@gmail.com";
  $student [2]=21;
  $student [10]=true;
```

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.
- We can not use for, while, do while loop on associative array.
- To work on associative array we have to use foreach loop.
- **Example 1**
- In this example we use an array to store various attributes of single person.
- $player = array(“name"=>”mohan”, “age"=>30, “gender"=>true);
- Echo $player[“name”];

FOREACH LOOP (NEW LOOP)

- Php has one special type of loop known as foreach loop.
- This loop is designed to work on the elements of **associative arrays**.
- *It can not be used on other array types.*
- **It will return an error when you try to use it on a variable with a different data type or an uninitialized variable.**
- There are two syntaxes for foreach loop.
- Let us see them in detail.

foreach (associative array as $value)

- This loop work on associative array only.
- It will start automatically and finish also automatically.
- It will execute no of times array has the elements. Means if associative array has 5 elements then foreach loop will execute 5 times.
- In first execution $value variable has first elements value, in second execution $value has second value of array and so on.
- We can add our code inside braces of foreach loop and it will for each and every element of associative array.
- If we change $value variable's value if will not effect original value of array.

foreach (**associative array** as $**key** => $**value**)

- It will execute no of times array has the elements. Means if associative array has 5 elements then foreach loop will execute 5 times.
- In first execution $value variable has first elements value and $key variable has first key, in second execution $value has second value of array and $key variable has second key of array and so on.
- So it means this loop is used to work upon both keys and value of array.
- If we change $value variable's value or $key variable's value if will not effect original value of array

Multidimensional Arrays

- PHP programming language also support multidimensional array.
- in a multidimensional array, each element in the main array can also be an array.
- And each element in the sub-array can be an array, and so on.

Example of MULTIDIMENSIONAL ARRAY

```
$families = array
    (
    “father"=>array
    (
    “anil",
    “patel",
    true,
    ),
    “mother"=>array
    (
    “sushila“,
    “kakadiya”,
     45.
    ),
    “child1”=>array
    (
    rahul,
    rahul@gmail.com,
    15-aug-1999
    ));
```

THE ARRAY ABOVE WOULD LOOK LIKE THIS IF WRITTEN TO THE OUTPUT:

```
Array
(
[father] => Array
  (
  [0] => anil
  [1] => patel
  [2] => true
  )
mother] => Array
  (
  [0] => shusila, [1] = kakadiya [2] = 45
  )
[child1] => Array
  (
  [0] => rahul
  [1] => rahul@gmail.com
  [2] => 15-aug-1999
  )
)
```

Function in PHP

Array related function

ARRAY()

- It is used to create an any type of 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
    $myarray = array(10,12, 8 => 80,  4 => 40, 19, 3 => 30);
    print_r($myarray);
    ?>
```

SORT()

- This function sorts an numeric array.
- This function arrange array's elements will arrange from lowest to highest.
- 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()

- Sort an **associative array** in given order and maintain key value pair association.

- This function sorts an array in way that key value pair both moved to given position to sort an array.
- It has following syntax

bool **asort** (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

COUNT()

- Count how many elements are 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.
- It has following syntax.

  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()

- Creates variables from associative array, name of the variables will be the key of element and value of variables will be the value of the key.
- Mostly used with $_REQUEST AND $_POST.
- It has following syntax.
- int extract (array var_array [, int extract_type [, string prefix]]).
- <?php
  - $teacher = array(“name”=>”ankit”,”surname”=>”patel”,”email”=>”xyz@gmail.com”)
  - Extract($teacher);
  - Echo “$name $surname $email”;
- ?>

LIST()

- Create variable from numeric array passed as an argument.
- One must give variable name in list function equal to no of element in array given on the right side of list function.
- Like **array()**, this is not really a function, but a language construct.
- **list()** is used to assign a list of variables in one operation.
- **It** 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 at the right of given 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");
print_r($stack);
?>
```

ARRAY_POP

- Pop (remove) the element from the right 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() // array to string

- 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
  $CarList = array(‘Maruti‘,’Audi’,’BMW’);
  $comma_separated = implode(",", $CarList);
  echo $comma_separated; // Maruti,Audi,BMW
?>
```

EXPLODE() //STRING TO ARRAY

- 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
```

ARRAY_CHANGE_KEY_CASE

- Returns an array with all keys from array lowercased or uppercased.
- Numbered indices are left as is.
- **array_change_key_case** (array $array [, int $case = CASE_LOWER]) : array
- Either **CASE_UPPER** or **CASE_LOWER** (default) can be used as second argument
-
```
<?php
$input_array = array("FirSt" => 1, "SecOnd" => 4);
print_r(array_change_key_case($input_array, CASE_
UPPER));
?>
```
- Output
- Array ([FIRST] => 1 [SECOND] => 4)

ARRAY_KEY_EXISTS

- Checks if the given key or index exists in the array
- **array_key_exists()** returns **TRUE** if the given key is set in the array.
- key can be any value possible for an array index.
- Syntax
- **array_key_exists** (mixed $key , array $array) : bool
-
```
<?php
$search_array = array('first' => 1, 'second' => 4);
if (array_key_exists('first', $search_array)) {
    echo "The 'first' element is in the array";
}
?>
```

ARRAY_SEARCH

- Searches the array for a given value and returns the first corresponding key if successful
- **array_search** (mixed $needle , array $haystack [, bool $strict = **FALSE**]) : mixed
- needle means the searched value.
- Haystack means array in which search operation should be performed.
- Returns the key for needle if it is found in the array, **FALSE** otherwise.

-
  ```
  <?php
  $array = array(0 => 'blue', 1 => 'red', 2 => 'green', 3 =
  > 'red');

  $key = array_search('green', $array); // $key = 2;
  $key = array_search('red', $array);   // $key = 1;
  ?>
  ```

ARRAY_UNSHIFT

- Prepend one or more elements to the beginning of an array.
- **array_unshift** (array &$array [, mixed $...]) : int

-
  ```
  <?php
  $queue = array("orange", "banana");
  array_unshift($queue, "apple", "raspberry");
  print_r($queue);
  ?>
  ```
- The above example will output:

-
  ```
  Array ( [0] => apple [1] => raspberry [2] =>
  orange [3] => banana )
  ```

Maths Related 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)
   ?>
```

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(1241757.12345, -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);
   ?>
```

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 Function

ECHO()

- It is used to display output as strings.
- It has following syntax
  - 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 give output as a formatted string
- It has following syntax
-         int **printf** (string format [, mixed args [, mixed ...]])

- Produces output according to *format*
- *Example*
- ***<?php***
  $s = '12500';
  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 characte
  r '#'
  ?>

JOIN()

- Join function convert array into string which contain all the arrays element with the separator string between each element.
- Join() is an alias of **implode()** function.
- <?php
  - $fruits = array(“apple”,”banana”,”mango”,”cherry”);
  - $fruit_names = join(“ ”,$fruits)
  - echo $fruit_names; // apple banana mango cherry

 ?>

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.
- syntax
- int **print** (string arg)

FPRINTF()

- fprintf() writes a formatted string to a into file whose reference will be passed via file pointer variable.
- syntax.
- int **fprintf** (resource handle, string format [, mixed args [, mixed ...]])
  Example

-
  ```
  <?php
  $fp = fopen('date.txt', 'w‘)

  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 into a variable.
- 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 (%).

POSSIBLE FORMAT VALUES

- *%% - Returns a percent sign*
- *%c - The character according to the ASCII value*
- *%d - Signed decimal number*
- *%u - Unsigned decimal number*
- *%f - Floating-point number (local settings aware)*
- *%F - Floating-point number (not local settings aware)*
- *%s - String*

Example 1

```
<?php
  $str = “Mr Ankit Patel";
  $number = 12;
  $txt = sprintf(“Hello %s how are you. Your lucky
  number is  %u",$str,$number);
  echo $txt;
  ?>
```

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 = "    The EasyLearn Academy!";
    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)
- 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)

Example

```
<?php
   $str = “      The EasyLearn Academy!           ";
   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.
- Syntax
- mixed **str_replace** (mixed $ *needle*, mixed $ *haystack*, mixed $subject [, int &$count])

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
   $alphabets =
   str_split(“abcdefghijklmnopqrstuvwxyz01234567
   89");
   ?>
```

The output of the code above will be:

```
Array
   (
   [0] => a
   [1] => b
   [2] => c
   [3] => d
   [4] => e
   )
```

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(“banana",“apple")==0)
   {
          echo “Both string are same”
   }
    else
    {
          echo “Both string are not 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(haystack,niddle,position_for_search)

Parameter Description

- haystack
  - Specifies the string to search
- niddle
  - Specifies the string to find
- Position_for_search
  - Specifies where to begin the search

EXAMPLE

```
<?php
   echo strpos("Hello easylearn",“learn");
   ?>
```

The output of the code above will be:

```
10
```

STRLEN()

- The strlen() function returns the length of a string.
- **Syntax**
  - strlen(string)
- **Example**
- <?php
  echo strlen("Hello student!");
  ?> The output of the code above will be:
- 13

STRTOLOWER()

- The strtolower() function converts a string to lowercase.
- **Syntax**
  - strtolower(string)
- <?php
  echo strtolower(“THE EASYLEARN.");
  ?>

```
<?php
	echo strtolower(“THE
EASYLEARN.");
?>
```

- The output of the code above will be:
  - the easylearn.

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!

UCWORDS

- returns a string with the first character of each word in str capitalized, if that character is alphabetic.
- **syntax**
- :**string** (string $str [, string $delimiters])

```
<?php
   $name = 'hello world!';
   echo ucwords($name);          // Hello World!
   $name = 'HELLO WORLD!';
   echo ucwords($name);          // 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 jumps over t
  he lazy dog.";
  $newtext = wordwrap($text, 20, "<br />");
  echo $newtext;
  ?>
```

The above example will output:

```
The quick brown fox<br /> jumped over the
  lazy<br /> 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.

```
<?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
?>
```

STRSTR()

- Returns part of haystack string starting from and including the first occurrence of **needle** to the end of **haystack**.
- string **strstr** (string $haystack , mixed $needle [, bool $before_needle = **FALSE**])
- before_needle means If **TRUE**, **strstr()** returns the part of the haystack before the first occurrence of the needle (excluding the needle).

```
<?php
        $email  = 'theeasylearn@gmail.com';
        $domain = strstr($email, '@');
        echo $domain; // prints @gmail.com
?>
```

Date & time related 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 following format parameter in 1st argument

FORMAT PARAMETER …

format character	Description	Example returned values
	Day	
d	**Day of the month, 2 digits with leading zeros**	***01*** **to** ***31***
D	**A textual representation of a day, three letters**	***Mon*** **through** ***Sun***
j	**Day of the month without leading zeros**	***1*** **to** ***31***
l **(lowercase 'L')**	**A full textual representation of the day of the week**	***Sunday*** **through** ***Saturday***
S	**English ordinal suffix for the day of the month, 2 characters**	***st***, ***nd***, ***rd*** **or** ***th***. **Works well with** ***j***
z	**The day of the year (starting from 0)**	***0*** **through** ***365***
	Week	
W	**ISO-8601 week number of year, weeks starting on Monday (added in PHP 4.1.0)**	**Example:** ***42*** **(the 42nd week in the year)**

FORMAT PARAMETER …

Month		
F	A full textual representation of a month, such as January or March	*January* through *December*
m	Numeric representation of a month, with leading zeros	*01* through *12*
M	A short textual representation of a month, three letters	*Jan* through *Dec*
n	Numeric representation of a month, without leading zeros	*1* through *12*
t	Number of days in the given month	*28* through *31*

FORMAT PARAMETER …

Year		
L	Whether it's a leap year	*1* if it is a leap year, *0* otherwise.
Y	A full numeric representation of a year, 4 digits	Examples: *1999* or *2003*
y	A two digit representation of a year	Examples: *99* or *03*
Time		
a	Lowercase Ante meridiem and Post meridiem	*am* or *pm*
A	Uppercase Ante meridiem and Post meridiem	*AM* or *PM*
g	12-hour format of an hour without leading zeros	*1* through *12*
G	24-hour format of an hour without leading zeros	*0* through *23*
h	12-hour format of an hour with leading zeros	*01* through *12*
H	24-hour format of an hour with leading zeros	*00* through *23*
i	Minutes with leading zeros	*00* to *59*
s	Seconds, with leading zeros	*00* through *59*

FORMAT PARAMETER …

Timezone		
e	Timezone identifier (added in PHP 5.1.0)	Examples: *UTC*, *GMT*, *Atlantic/Azores*
I (capital i)	Whether or not the date is in daylights savings time	*1* if Daylight Savings Time, *0* otherwise.
T	Timezone setting of this machine	Examples: *EST*, *MDT* ...
Z	Timezone offset in seconds. The offset for timezones west of UTC is always negative, and for those east of UTC is always positive.	*-43200* through *43200*

TIME()

- time() returns Return current Unix timestamp.
- 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).
- 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)

Example

```
<?php
  echo(date("M-d-
  Y",mktime(0,0,0,12,05,2005))."<br />");
  echo(date("M-d-Y",mktime(0,0,0,02,1,2013))."<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:

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

Other useful function

DEFINE()

- Defines a named constant.
- Constant means a variable whose value can not be changed during execution of program.
- Do remember that constant name do not start with $ sign and so can not be directly used in double quote.
- We can not use constant in string. To use it with string we have to give contant name after string preceded by . Opertor
- syntax
- **define** (string name , mixed $value)
- Example
- <?php

```
  define("ERROR_FILE","error.log");
 Echo ERROR_FILE; // error.log
```

- ?>

EXIT() & DIE()

- Output a given message and terminate the current script.
- **Exit and die functions are same. So one can use any function from both.**
- syntax
- **exit** ([string $status])
- Die([string $status])
- Example
- <?php

```
        $filename = ”myfile.txt”;
       $file = fopen($filename, 'r')
          or exit("unable to open file ($filename)");
```

- ?>
- Exit function will print message if file can not be created.
- This function is often used on right side of other function so if function failed php script can be terminated with error message.

HEADER()

- **header()** is used to send a raw HTTP header.
- **It is mainly used to send user forcefully from one page to another page.**
- Remember that **header()** must be called before any actual output is sent, either by normal HTML tags, blank lines in a file, or from PHP.

```
<?php
    header('location:index.php'); //send user on default page of current directory
    exit;
?>
```

- Download file

```
<?php
    // We force  a PDF to be downloaded
    header('Content-Type: application/pdf');
    // force browser to show dialog box and name willl be downloaded.pdf
    header('Content-Disposition: attachment; filename="downloaded.pdf"');
    // read file test.pdf from current directory using libarary function readfile
    readfile('test.pdf');
?>
```

ISSET()

- It is used to check whether variable or element of array exist in memory or not. Return true if exists otherwise return false.
- **isset** (mixed $var [, mixed $...]) : bool
- One can pass multiple variable name in this function
- <?php
  - If(isset($name)==true)
    - Echo “name variable exists”;
  - Else
    - Echo “name variable does not exists”;
- ?>

EMPTY()

- Determine whether a variable is considered to be empty.
- A variable is considered empty if it does not exist or if its value equals **FALSE**.
- **empty()** does not generate a warning if the variable does not exist. So should be used to check whether variable is empty or not.
- <?php
- $var='';
- if (empty($var)) { // Evaluates to true because $var is empty
-     echo '$var is either 0, empty, or not set at all';
- }
- ?>

IS_NUMERIC()

- Finds whether the given variable is numeric. Numeric strings consist of optional sign, any number of digits, optional decimal part and optional exponential part.
- Thus +*0123.45e6* is a valid numeric value.
- Hexadecimal (e.g. *0xf4c3b00c*) and binary (e.g. *0b10100111001*) notation is not allowed.
- Syntax
- **is_numeric** (mixed $var) : bool
- Example
- <?php
- $num='123';
- if (is_numeric($num)==true) {
-     echo '$num is number';
- }
- ?>

INCLUDE()

- Include() function is used to add existing php/html files into current files.
- If file does not exists then this php function give warning but keep executing code till the end of current php file.
- One can include same file multiple time using include function.
- One has to pass relative path of the file along with filename to include the file
- Syntax
- Include(filename)
- Example
- <?php
  - Include("function.php")
- ?>

INCLUDE_ONCE()

- Include_once() function is used to add existing php/html files into current files.
- If file does not exists then this php function give warning but keep executing code till the end of current php file.
- One can not include same file multiple time using include function.
- One has to pass relative path of the file along with filename to include the file
- Syntax
- Include(filename)
- Example
- <?php
  - Include_once("function.php")
- ?>

REQUIRE()

- require() function is also used to add existing php/html files into current files.
- If file does not exists then this php function give error and abort current php script execution.
- One can include same file multiple time using require function.
- One has to pass relative path of the file along with filename to include the file
- Syntax
- Require (filename)
- Example
- <?php
  - require("function.php")
- ?>

REQUIRE_ONCE()

- require_once() function is also used to add existing php/html files into current files.
- If file does not exists then this php function give error and abort current php script execution
- One can not include same file multiple time using require function.
- One has to pass relative path of the file along with filename to include the file
- Syntax
- require_once(filename)
- Example
- <?php
  - require_once(“function.php”)
- ?>

$_SERVER[]

- $_SERVER is an ready only 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.
- $_SERVER is created automatically by server.

$_SERVER[‘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[‘SERVER_NAME’]

- The name of the server host under which the current script is executing.

'SERVER_SOFTWARE'

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

'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

'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.

REMOTE_ADDR'

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

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*'.