CHAPTER–2

ARRAYS AND FUNCTIONS IN PHP

2.1 INTRODUCTION TO PHP ARRAYS

Arrays are complex variables that allow us to store more than one value or a group of values under a single variable name. Let's suppose you want to store colors in your PHP script. Storing the colors one by one in a variable could look something like this:

Example

```
<?php
$color1 = "Red";
$color2 = "Green";
$color3 = "Blue";
?>
```

But what, if you want to store the states or city names of a country in variables and this time this not just three may be hundred. It is quite hard, boring, and bad idea to store each city name in a separate variable. And here array comes into play.

Characteristics of an array in PHP

- An array is a collection of 0 or more elements
- Each array value is accessed using array index.
- Each element can be of any PHP type (including another array)
- In PHP, arrays can have unlimited elements and unlimited depth (subject to server memory limit)
- Every element in an array has a corresponding key. The key is used to refer to a specific element
- Unlike other languages, you don't need to specify the size of an array: you can add or remove elements at any time.

2.1.1 CREATING, MANIPULATING AND TRAVERSING DIFFERENT TYPES OF ARRAYS

There are three types of arrays that you can create. These are:

- **Indexed array** – An array with a numeric key.
- **Associative array** – An array where each key has its own specific value.
- **Multidimensional array** – An array containing one or more arrays within itself.

Indexed/Numeric Array

These arrays can store numbers, strings and any object but their index will be represented by numbers. By default array index starts from 0.

Creating Numeric Array

An array is created using the array() keyword.

Creating an array:

```
<?php
$myArray = array();
?>
```

Initializing an array:

```
<?php
$myArray = array(10, 20, 30);
?>
```

Example: To create and access numeric arrays.

```
<html>
<body>
<?php
/* First method to create array. */
$numbers = array( 1, 2, 3, 4, 5);
foreach( $numbers as $value )
{
    echo "Value is $value <br />";
}
```

```
/* Second method to create array. */
$numbers[0] = "one";
$numbers[1] = "two";
$numbers[2] = "three";
$numbers[3] = "four";
$numbers[4] = "five";
foreach( $numbers as $value )
{
    echo "Value is $value <br />";
}
?>
</body>
</html>
```

Output:

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

Associative arrays

The associative arrays are very similar to numeric arrays in term of functionality but they are different in terms of their index method.

Key-value pair

Associative array will have their index as string so that you can establish a strong association between key and values.

For example, to store the salaries of employees in an array, a numerically indexed array would not be the best choice. Instead, we could use the employees names as the keys in our associative array, and the value would be their respective salary.

Associative elements are passed in the format 'key' => 'value':

NOTE : Don't keep associative array inside double quote while printing otherwise it would not return any value.

Example: To create Associative Array

```
<?php
$myArray = array
(
    'a' => 'my string',
    'b' => array()
);
?>
```

You can combine these when creating an array. The first element will have index of 0:

```
<?php
$myArray = array(1234, 'wish' => 'Good Morning');
?>
```

You can manually specify the index of values. PHP will automatically determine the next index. In the following example, the bar element will have an index of 11:

```
<?php
$myArray = array(10 => 'Good', 'Morning');
?>
```

Example: Creating an associative array

```
<html>
<body>
<?php
/* First method to associate create array. */
$salaries = array
(
    "jitendra" => 2000,
    "dipak" => 1000,
    "sara" => 500
);

echo "Salary of jitendra is ". $salaries['jitendra'] . "<br />";
echo "Salary of dipak is ".  $salaries['dipak']. "<br />";
echo "Salary of sara is ".  $salaries['sara']. "<br />";

/* Second method to create array. */
$salaries['jitendra'] = "high";
$salaries['dipak'] = "medium";
$salaries['sara'] = "low";

echo "Salary of jitendra is ". $salaries['jitendra'] . "<br />";
echo "Salary of dipak is ".  $salaries['dipak']. "<br />";
echo "Salary of sara is ".  $salaries['sara']. "<br />";
?>
</body>
</html>
```

Output

```
Salary of jitendra is 2000
Salary of dipak is 1000
Salary of sara is 500
Salary of jitendra is high
Salary of dipak is medium
Salary of sara is low
```

Multidimensional Arrays

A multi-dimensional 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. Values in the multi-dimensional array are accessed using multiple indexes.

Example: Using multidimensional arrays

```
<html>
<body>
<?php
$Marks  = array
(
    "jitendra" => array
    (
        "physics" => 35,
        "maths" => 30,
        "chemistry" => 39
    ),
    "dipak" => array
    (
        "physics" => 30,
        "maths" => 32,
        "chemistry" => 29
    ),
    "sara" => array
    (
        "physics" => 31,
        "maths" => 22,
        "chemistry" => 39
    )
);
/* Accessing multi-dimensional array values */
echo "Marks  for jitendra in physics : " ;
echo $Marks ['jitendra']['physics'] . "<br />";
echo "Marks  for dipak in maths : ";
echo $Marks ['dipak']['maths'] . "<br />";
```

```
echo "Marks  for sara in chamistry : " ;
echo $Marks ['sara']['chemistry'] . "<br />";
?>
</body>
</html>
```

Output:

```
Marks for jitendra in physics : 35
Marks for dipak in maths : 32
Marks for sara in chamistry : 39
```

2.2 PHP STRINGS

In PHP strings are series of characters.PHP only supports a 256-character set.PHP does not support Unicodes. PHP string can be as large as 2GB.

There are 4 types of Strings in PHP:

1. Single quoted
2. Double quoted
3. Heredoc syntax
4. Nowdoc syntax (since PHP 5.3.0)

Single quoted Strings in PHP

Single quoted strings are enclosed with ' (single quote character)

Variables and escape sequences for special characters are not expanded in Single quoted strings.

If we have a single quote within the string we escape it using /

If there is a \ (forward-slash) it can be escaped by // (double slash)

Escape sequences like \r \n are not expanded in Single quotes strings.

Examples :

```
<?php
$var = "Om";
echo 'Hello, <br>';
echo '"Double quotes here have no issues."<br>';
echo 'We can
    have
        multi-line
    Single quote strings.<br>';
echo 'Newline sequence \n
    wont expand.<br>';
echo 'I\'m here!<br>';
echo 'My Name is $var, err!!, variables won\'t expand.<br>';
?>
```

Output:

```
Hello,
"Double quotes here have no issues."
We can have multi-line Single quote strings.
Newline sequence \n wont expand.
I'm here!
My Name is $var, err!!, variables won't expand.
```

Double quoted Strings in PHP

Double quoted String literals are enclosed with double quote characters (").e.g. :

```
$var = "Double quoted String";
```

Escape sequences are expanded in Double quoted Strings

Variable names are expanded in Double quoted Strings

Double quoted Strings interprets more escape sequences then Single quoted strings.

\n – Newline or Linefeed

\r – Carriage return

\t – Horizontal tab

\v – Vertical tab

\e – Escape

\f – Form feed

**** – Backslash

\$ – Dollar sign

\" – Double-quote

```
<?php
$var = "Om";
echo "Hello, <br>";
echo "'Single quotes here have no issues.'<br>";
echo "We can
    have
        multi-line
    Double quote strings.<br>";
echo "Newline sequence \n
    is expand in Double quoted strings.<br>";
echo "He is the \"Man\"!<br>";
echo "My Name is $var, Wow!!, variable names do get
    expanded in Double quoted strings. <br>";
?>
```

Output:

```
Hello,
'Single quotes here have no issues.'
We can have multi-line Double quote strings.
Newline sequence
    is expand in Double quoted strings.
He is the "Man"!
My Name is Om, Wow!!, variable names do get expanded in Double quoted strings.
```

Heredoc Strings in PHP

Heredoc is just like Double quoted strings, but it does not start and end with a double quotes, hence there is no need to escape double quotes, while we can use other escape codes listed above.

Syntax:

The Heredoc String starts with <<< followed by by and **identifier**, then a **newline**, then the **string** and finally ends with the **same identifier** and a semicolon.

Example:

```
<?php
$str = <<<EOD
This is an example
of heredoc spanning
multiple lines.
EOD;
echo $str;
?>
```

Output:

```
This is an example of heredoc spanning multiple lines.
```

Notes : Here EOD is used as an Identifier.

The identifier at the end of herdoc (EOD; in above example) should not have and leading or trailing spaces.

Nowdoc Strings in PHP

Nowdoc's are similar to heredoc's, except variables are not expanded in nowdoc's (just like Single quoted Strings)

String identifier is surrounded it single quote.

Only difference between heredoc's and nowdoc's syntax is that nowdoc's starting identifier is surrounded by single quotes.

```
<?php
$name = "Om";
//Heredoc example
echo <<<EOT
```

```
My name is $name
I love PHP.
EOT;
echo "<br>";
//Nowdoc example
echo <<<'EOT'
My name is $name
I love PHP.
EOT;
?>
```

Output:

```
My name is Om I love PHP.
My name is $name I love PHP.
```

2.3 USER DEFINED FUNCTION

A function is a self-contained block of code that performs a specific task.

User-Defined Functions

In addition to the built-in functions, PHP also allows you to define your own functions. It is a way to create reusable code packages that perform specific tasks and can be kept and maintained separately form main program. Here are some advantages of using functions:

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

Creating and Invoking Functions

The basic syntax of creating a custom function can be give with:

```
function functionName()
{
    // Code to be executed
}
```

The declaration of a user-defined function start with the word function, followed by the name of the function you want to create followed by parentheses i.e. () and finally place your function's code between curly brackets {}.

This is a simple example of an user-defined function, that display today's date:

Example

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

Output:

```
Today is Sunday
```

Note : A function name must start with a letter or underscore character not with a number, optionally followed by the more letters, numbers, or underscore characters. Function names are case-insensitive.

Functions with Parameters

You can specify parameters when you define your function to accept input values at run time. The parameters work like placeholder variables within a function; they're replaced at run time by the values (known as argument) provided to the function at the time of invocation.

```
function myFunc($oneParameter, $anotherParameter)
{
    // Code to be executed
}
```

You can define as many parameters as you like. However for each parameter you specify, a corresponding argument needs to be passed to the function when it is called.

The getSum() function in following example takes two integer values as arguments, simply add them together and then display the result in the browser.

Example

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

Output:

```
Sum of the two numbers 10 and 20 is : 30
```

Tip : An argument is a value that you pass to a function, and a parameter is the variable within the function that receives the argument.

Functions with Optional Parameters and Default Values

You can also create functions with optional parameters – just insert the parameter name, followed by an equals (=) sign, followed by a default value, like this.

Example

```
<?php
// Defining function
function customFont($font, $size=1.5)
{
    echo "<p style=\"font-family: $font; font-size: {$size}em;\">Hello, world!</p>";
}
// Calling function
customFont("Arial", 2);
customFont("Times", 3);
customFont("Courier");
?>
```

Output:

Hello, world!

Hello, world!

Hello, world!

As you can see the third call to custom Font() doesn't include the second argument. This causes PHP engine to use the default value for the $size parameter which is 1.5.

Returning Values from a Function

A function can return a value back to the script that called the function using the return statement. The value may be of any type, including arrays and objects.

Example

```
<?php
// Defining function
function getSum($num1, $num2)
```

```
{
    $total = $num1 + $num2;
    return $total;
}
// Printing returned value
echo getSum(5, 10); // Outputs: 15
?>
```

Output:

```
15
```

A function can not return multiple values. However, you can obtain similar results by returning an array, as demonstrated in the following example.

Example

```
<?php
// Defining function
function divideNumbers($dividend, $divisor)
{
    $quotient = $dividend / $divisor;
    $array = array($dividend, $divisor, $quotient);
    return $array;
}
// Assign variables as if they were an array
list($dividend, $divisor, $quotient) = divideNumbers(10, 2);
echo $dividend;  // Outputs: 10
echo $divisor;   // Outputs: 2
echo $quotient;  // Outputs: 5
?>
```

Output:

```
10
2
5
```

Passing Arguments to a Function

In PHP we can pass argument to the function by two way Call By Value and Call By Reference.

Call-by-Value

The default behavior for user-defined functions in PHP is call-by-value. This means that when you pass variables to a function call, PHP makes copies of the variable values to pass on to the function. So, whatever the function does, it is not able to change the actual variables that appear in the function call. This behavior can be good or bad. It's a nice reassurance if you only want to use a function for its returned value, but it can also be a source of confusion and frustration if changing the passed variable is actually your goal.

Let's demonstrate call-by-value with a fragile and extremely inefficient implementation of subtraction:

```
function my_subtract ($num1, $num2)
{
    if ($num1 < $num2)
    echo "Negative numbers are imaginary";
    $return_result = 0;
    while($num1 > $num2)
    {
        $num1 = $num1 - 1;
        $return_result = $return_result + 1;
    }
    return($return_result);
}
$first_op = 493;
$second_op = 355;
$result1 = my_subtract($first_op, $second_op);
print("result1 is $result1<BR>");
$result2 = my_subtract($first_op, $second_op);
print("result2 is $result2<BR>");
```

Reassuringly, we find that our result is the same both times we perform the same subtraction:

```
result1 is 138
result2 is 138
```

This is true even though my_subtract changes the value of its formal parameter $num1 – that variable only holds a copy of the value that was in the actual parameter $first_op, and so $first_op cannot be affected.

Call-by-Reference

PHP offers two different ways to have functions actually modify their arguments: in the function definition and in the function call.

If you want to define a function to operate directly on a passed variable, simply put an ampersand in front of the formal parameter in the definition, like so:

```
function my_subtract_ref (&$num1, &$num2)
{
    if ($num1 < $num2)
    die("Negative numbers are imaginary");
    $return_result = 0;
    while($num1 > $num2)
```

```
        {
            $num1 = $num1 - 1;
            $return_result = $return_result + 1;
        }
        return($return_result);
    }
    $first_op = 493;
    $second_op = 355;
    $result1 = my_subtract_ref($first_op, $second_op);
    print("result1 is $result1<BR>");
    $result2 = my_subtract_ref($first_op, $second_op);
    print("result2 is $result2<BR>");
```

Now, if we perform exactly the same subtraction calls as we did the first time, we get theoutput:

```
result1 is 138
result1 is 0
```

2.3.1 VARIABLE FUNCTION

One of the neater tricks you can do in PHP is to use variables in place of the names of user defined functions. That is, rather than typing a literal function name into your code, you type a dollar-sign variable – the function that is actually called at runtime will depend on the string that that variable has been assigned to. In some sense, this allows us to use functions as data.

For example, the following two function calls are exactly equivalent:

```
function customized_greeting ()
{
    print("You are being greeted in a customized way!<BR>");
}
customized_greeting();
$my_greeting = 'customized_greeting';
$my_greeting();
```

Output:

```
You are being greeted in a customized way!
You are being greeted in a customized way!
```

Because function names are just strings, they can also be used as arguments to functions or be returned as a function's result.

2.3.2 VARIABLE LENGTH ARGUMENT

1. **func_get_arg(position)** - It returns the value of the specified argument provided in the function call operation. "position" is the position index of the specified argument. The position index of the first argument is 0. For example, if "func_get_arg(2)" is used in a function, it will return the value of the 3rd argument.

2. **func_num_args()** - It returns the total number of arguments provided in the function call operation. For example, if "func_num_args()" returns 1, you know that there is only 1 argument provided by calling code.

3. **func_get_args()** - It creates and returns an array which contains values of all arguments provided in the function call operation. For example, if "$args = func_get_args()" is used in a function, $args will be array containing all arguments.

With all these 3 functions, we say that PHP supports variable-length argument lists with some basic rules:

- The function call can provide more arguments than the number of argument variables defined in the function statement.
- You can pass arguments to a function that has no argument variable defined.

To help us understand how use variable-length argument list feature, I wrote this tutorial example:

```
<?php
function f2c($fahrenheit)
{
    $celsius = (func_get_arg(0) - 32.0) / 1.8;
    return $celsius;
}
print("\n Calling a function that uses func_get_arg():\n");
if (f2c(20.0)<0.0) print(" It's cold here!\n");
function myMax()
{
    $max = NULL;
    if (func_num_args()>0) $max = func_get_arg(0);
    for ($i=1; $i<func_num_args(); $i++)
    {
        if ($max < func_get_arg($i)) $max = func_get_arg($i);
    }
    return $max;
}
print("\n Calling a function that uses func_num_args():\n");
print("      ".myMax()."\n");
print("      ".myMax(3)."\n");
```

```
print("      ".myMax(3, 1, 4)."\n");
print("      ".myMax(3, 1, 4, 1, 5, 9, 2, 6, 5)."\n");
function myMin()
{
    $list = func_get_args();
    $min = NULL;
    if (count($list)>0) $min = $list[0];
    for ($i=1; $i<count($list); $i++)
    {
        if ($min > $list[$i]) $min = $list[$i];
    }
    return $min;
}
print("\n Calling a function that uses func_get_args():\n");
print("      ".myMin().."\n");
print("      ".myMin(5)."\n");
print("      ".myMin(5, 6, 2)."\n");
print("      ".myMin(5, 6, 2, 9, 5, 1, 4, 1, 3)."\n");
?>
```

Output:

```
Calling a function that uses func_get_arg():
    It's cold here!
Calling a function that uses func_num_args():
    3
    4
    9
Calling a function that uses func_get_args():
    5
    2
    1
```

2.4 VARIABLE SCOPE, ACCESSING GLOBAL VARIABLES INSIDE A FUNCTION

The location of variable declaration determines the extent of a variable's visibility within the PHP program i.e. where the variable can be used or accessed. This accessibility is known as variable scope.

By default, variables declared within a function are local and they cannot be viewed or manipulated from outside of that function, as demonstrated in the example below:

Example

```
<?php
// Defining function
function test()
{
    $greet = "Hello World!";
    echo $greet;
}

test(); // Outputs: Hello World!

echo $greet; // Generate undefined variable error
?>
```

Similarly, if you try to access or import an outside variable inside the function, you'll get an undefined variable error, as shown in the following example:

Example

```
<?php
$greet = "Hello World!";

// Defining function
function test()
{
    echo $greet;
}

test();  // Generate undefined variable error

echo $greet; // Outputs: Hello World!
?>
```

As you can see in the above examples the variable declared inside the function is not accessible from outside, likewise the variable declared outside of the function is not accessible inside of the function. This separation reduces the chances of variables within a function getting affected by the variables in the main program.

Tip : It is possible to reuse the same name for a variable in different functions, since local variables are only recognized by the function in which they are declared.

The global Keyword

There may be a situation when you need to import a variable from the main program into a function, or vice versa. In such cases, you can use the global keyword before the variables inside a function. This keyword turns the variable into a global variable, making it visible or accessible both inside and outside the function, as show in the example below:

Example

```
<?php
$greet = "Hello World!";

// Defining function
function test()
{
    global $greet;
    echo $greet;
}

test(); // Outpus: Hello World!
echo $greet; // Outpus: Hello World!

// Assign a new value to variable
$greet = "Goodbye";

test(); // Outputs: Goodbye
echo $greet; // Outputs: Goodbye
?>
```

Output:

```
$greet inside function is: Hello World!
$greet outside of function is: Hello World!
$greet inside function is: Goodbye
$greet outside of function is: Goodbye
```

2.5. USING PHP BUILT-IN FUNCTIONS

PHP provides a wide range of built-in functions that can be used to perform various tasks such as string manipulation, array manipulation, file handling, math operations, and more. Some of the most commonly used built-in functions in PHP include:

2.5.1 STRING PROCESSING FUNCTIONS

PHP provides a wide range of built-in functions that can be used for string processing, which makes it easy to manipulate and modify strings in various ways.

CHR

Returns a character from a specified ASCII value

Syntax

```
chr(ascii)
```

Parameter	Description
ascii	Required. An ASCII value

Return Values

Returns the specified character.

Example: Using chr function

```
<?php
echo chr(52)."<br />";
echo chr(052)."<br />";
echo chr(0x52)."<br />";
?>
```

Output:

```
4
*
R
```

ORD

Returns the ASCII value of the first character of a string

Syntax

```
ord(string)
```

Parameter	Description
string	Required. The string to get an ASCII value from

Example: Using ord function

```
<?php
echo ord("h")."<br />";
echo ord("hello")."<br />";
?>
```

Output:

```
104
104
```

STRTOLOWER

The strtolower() function converts a string to lowercase.

Syntax

```
strtolower(string)
```

Parameter	Description
string	Required. Specifies the string to convert

Example: Using strtolower function

```
<?php
echo strtolower("Hello WORLD.");
?>
```

Output:

```
hello world.
```

STRTOUPPER

Converts a string to uppercase letters

Definition and Usage

The strtoupper() function converts a string to uppercase.

Syntax

```
strtoupper(string)
```

Parameter	Description
string	Required. Specifies the string to convert

Example: Using strtoupper function

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

Output:

```
HELLO WORLD!
```

STRLEN

Returns the length of a string

Definition and Usage

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

Syntax

```
strlen(string)
```

Parameter	Description
string	Required. Specifies the string to check

Example: Using strlen function

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

Output:

```
12
```

LTRIM

Strips whitespace from the left side of a string

Syntax

```
ltrim(string,charlist)
```

Parameter	Description
string	Required. Specifies the string to check

charlist Optional. Specifies which characters to remove from the string. If omitted, all of the following characters are removed:

```
"\0" - NULL
"\t" - tab
"\n" - new line
"\x0B" - vertical tab
"\r" - carriage return
" " - ordinary white space
```

Example-1: Using ltrin function

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

Output:

```
Without ltrim:      Hello World!
With ltrim: Hello World!
```

If you select "View source" in the browser window, you will see the following HTML:

```
<html>
<body>
Without ltrim:    Hello World!<br />With ltrim: Hello World!
</body>
</html>
```

Example-2: Using ltrin function

```
<?php
$str = "\r\nHello World!";
echo "Without ltrim: " . $str;
echo "<br />";
echo "With ltrim: " . ltrim($str);
?>
```

Output:

```
Without ltrim: Hello World!
With ltrim: Hello World!
```

If you select "View source" in the browser window, you will see the following HTML:

```
<html>
<body>
Without ltrim:
Hello World!<br />With ltrim: Hello World!
</body>
</html>
```

RTRIM

Strips whitespace from the right side of a string

Syntax

```
rtrim(string,charlist)
```

Parameter	Description
string	Required. Specifies the string to check
charlist	Optional. Specifies which characters to remove from the string.

Example-1: Using rtrin function

```
<html>
<body>
<?php
```

```
$str = "Hello World!      ";
echo "Without rtrim: " . $str;
echo "<br />";
echo "With rtrim: " . rtrim($str);
?>
<body>
<html>
```

Output:

```
Without rtrim: Hello World!
With rtrim: Hello World!
```

If you select "View source" in the browser window, you will see the following HTML:

```
<html>
<body>
Without rtrim: Hello World!      <br />With rtrim: Hello World!
</body>
</html>
```

Example-2: Using rtrin function

```
<?php
$str = "Hello World!\r\n";
echo "Without rtrim: " . $str;
echo "<br />";
echo "With rtrim: " . rtrim($str);
?>
```

Output:

```
Without rtrim: Hello World!
With rtrim: Hello World!
```

If you select "View source" in the browser window, you will see the following HTML:

```
<html>
<body>
Without rtrim: Hello World!
<br />With rtrim: Hello World!
</body>
</html>
```

TRIM

Strips whitespace from both sides of a string

Syntax

```
trim(string,charlist)
```

Parameter	Description
string	Required. Specifies the string to check

charlist Optional.

Example-1: Using trim function

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

Output:

```
Without trim: Hello World!
With trim: Hello World!
```

If you select "View source" in the browser window, you will see the following HTML:

```
<html>
<body>
Without trim:      Hello World!      <br />With trim: Hello World!
</body>
</html>
```

Example-2: Using trim function

```
<?php
$str = "\r\nHello World!\r\n";
echo "Without trim: " . $str;
echo "<br />";
echo "With trim: " . trim($str);
?>
```

Output:

```
Without trim: Hello World!
With trim: Hello World!
```

If you select "View source" in the browser window, you will see the following HTML:

```
<html>
<body>
Without trim:
Hello World!
<br />With trim: Hello World!
</body>
</html>
```

SUBSTR

Returns a part of a stringDescription

Syntax

```
substr(string,start,length)
```

Parameter	Description
string	Required. Specifies the string to return a part of
start	Required. 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

Example-1: Using substr function

```
<?php
echo substr("Hello world!",6);
?>
```

Output:

```
world!
```

Example-2: Using substr function

```
<?php
echo substr("Hello world!",6,5);
?>
```

Output:

```
world
```

STRCMP

Compares two strings (case-sensitive)

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)
```

Parameter	Description
string1	Required. Specifies the first string to compare
string2	Required. Specifies the second string to compare

Example: Using strcmp function

```
<?php
echo strcmp("Hello world!","Hello world!");
?>
```

Output:

```
0
```

STRCASECMP

Compares two strings (case-insensitive)

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

```
strcasecmp(string1,string2)
```

Parameter	Description
string1	Required. Specifies the first string to compare
string2	Required. Specifies the second string to compare

Example: Using strcasecmp function

```
<?php
echo strcasecmp("Hello world!","HELLO WORLD!");
?>
```

Output:

```
0
```

STRPOS

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

Syntax

```
strpos(string,find,start)
```

Parameter	Description
string	Required. Specifies the string to search
find	Required. Specifies the string to find
start	Optional. Specifies where to begin the search

Example: Using strpos function

```
<?php
echo strpos("Hello world!","wo");
?>
```

Output:

```
6
```

STRRPOS

Finds the position of the last occurrence of a string inside another string (case-sensitive) This function returns the position on success, otherwise it returns FALSE.

Syntax

```
nbstrrpos(string,find,start)
```

Parameter	Description
string	Required. Specifies the string to search
find	Required. Specifies the string to find
start	Optional. Specifies where to begin the search

Example: Using strrpos function

```
<?php
echo strrpos("Hello world!","wo");
?>
```

Output:

```
6
```

STRSTR

Finds the first occurrence of a string inside another string (case-sensitive)

This 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)
```

Parameter	Description
string	Required. Specifies the string to search
search	Required. 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

Example-1: Using strstr function

```
<?php
echo strstr("Hello world!","world");
?>
```

Output:

```
world!
```

Example-2: Using strstr function

In this example we will search a string for the ASCII value of "o":

```
<?php
echo strstr("Hello world!",111);
?>
```

Output:

```
o world!
```

STRISTR

Finds the first occurrence of a string inside another string (case-insensitive)

This function 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)
```

Parameter	Description
string	Required. Specifies the string to search
search	Required. 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

Example-1: Using stristr function

```
<?php
echo stristr("Hello world!","WORLD");
?>
```

Output:

```
world!
```

Example-2: Using stristr function

In this example we will search a string for the ASCII value of "o":

```
<?php
echo stristr("Hello world!",111);
?>
```

Output:

```
o world!
```

STR_REPLACE

Replaces some characters in a string (case-sensitive)

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

```
str_replace(find,replace,string,count)
```

Parameter	Description
find	Required. Specifies the value to find
replace	Required. Specifies the value to replace the value in find
string	Required. Specifies the string to be searched
count	Optional. A variable that counts the number of replacements

Example-1: Using str_replace function

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

Output:

```
Hello Peter!
```

Example-2: Using str_replace function

In this example we will demonstrate str_replace() with an array and a count variable:

```
<?php
$arr = array("blue","red","green","yellow");
print_r(str_replace("red","pink",$arr,$i));
echo "Replacements: $i";
?>
```

Output:

```
Array
(
    [0] => blue
    [1] => pink
    [2] => green
    [3] => yellow
)
Replacements: 1
```

Example-3: Using str_replace function

In this example we will demonstrate str_replace() with less elements in replace than find:

```
<?php
$find = array("Hello","world");
$replace = array("B");
$arr = array("Hello","world","!");
print_r(str_replace($find,$replace,$arr));
?>
```

Output:

```
Array
(
    [0] => B
    [1] =>
    [2] => !
)
```

STRREV

Reverses a string

Syntax

```
strrev(string)
```

Parameter	Description
string	Required. Specifies the string to reverse

Example: Using strrev function

```
<?php
echo strrev("Hello World!");
?>
```

Output:

```
!dlroW olleH
```

ECHO

The echo() function outputs one or more strings.

Syntax

```
echo(strings)
```

Parameter	Description
Strings	Required. One or more strings to be sent to the output

Example: Using echo function

```
<?php
$str = "Who's K D P?";
echo $str;
echo "<br />";
echo $str."<br />I don't know!";
?>
```

Output:

```
Who's K D P?
Who's K D P?
I don't know!
```

Example-2: Using echo function

Difference of single and double quotes. Single quotes will print the variable name, not the value:

```
<?php
$color = "red";
echo "Roses are $color";
echo "<br />";
echo 'Roses are $color';
?>
```

Output:

```
Roses are red
Roses are $color
```

PRINT

Outputs a string

Syntax

```
print(strings)
```

Parameter	Description
strings	Required. One or more strings to be sent to the output

Example: Using print function

```
<?php
$str = "Who's K D P?";
print $str;
print "<br />";
print $str."<br />I don't know!";
?>
```

Output:

```
Who's K D P?
Who's K D P?
I don't know!
```

Example-2: Using print function

Difference of single and double quotes. Single quotes will print the variable name, not the value:

```
<?php
$color = "red";
print "Roses are $color";
print "<br />";
print 'Roses are $color';
?>
```

Output:

```
Roses are red
Roses are $color
```

2.5.2 MATHEMATICAL FUNCTIONS

PHP provides a variety of mathematical functions that can be used to perform calculations and operations on numeric data.

ABS

Returns the absolute value of a number

Syntax

```
abs(x)
```

Parameter	Description
x	Required. A number. If the number is of type float, the return type is also float, otherwise it is integer

Example: Using abs function

```
<?php
echo(abs(6.7) . "<br />");
echo(abs(-3) . "<br />");
echo(abs(3));
?>
```

Output:

```
6.7
3
3
```

CEIL

Returns the value of a number rounded upwards to the nearest integer

Syntax

```
ceil(x)
```

Parameter	Description
x	Required. A number

Example: Using ceil function

```
<?php
echo(ceil(0.60) . "<br />");
echo(ceil(0.40) . "<br />");
echo(ceil(5) . "<br />");
echo(ceil(5.1) . "<br />");
echo(ceil(-5.1) . "<br />");
echo(ceil(-5.9))
?>
```

Output:

```
1
1
5
6
-5
-5
```

FLOOR

Returns the value of a number rounded downwards to the nearest integer

Syntax

```
floor(x)
```

Parameter	Description
x	Required. A number

Example: Using floor function

```
<?php
echo(floor(0.60) . "<br />");
echo(floor(0.40) . "<br />");
echo(floor(5) . "<br />");
echo(floor(5.1) . "<br />");
echo(floor(-5.1) . "<br />");
echo(floor(-5.9))
?>
```

Output:

```
0
0
5
5
-6
-6
```

ROUND

Rounds a number to the nearest integer

Syntax

```
round(x,prec)
```

Parameter	Description
x	Required. The number to be round
prec	Optional. The number of digits after the decimal point

Example: Using floor function

```
<?php
echo(round(0.60) . "<br />");
echo(round(0.50) . "<br />");
echo(round(0.49) . "<br />");
echo(round(-4.40) . "<br />");
echo(round(-4.60))
?>
```

Output:

```
1
1
0
-4
-5
```

FMOD

Returns the remainder (modulo) of the division of the arguments

Syntax

```
fmod(x,y)
```

Parameter	Description
x	Required. A number
y	Required.

Example: Using fmod function

```
<?php
$r = fmod(5,2);
echo $r
?>
```

Output:

```
1
```

MIN

Returns the number with the lowest value of two specified numbers

Syntax

```
min(x,y)
```

Parameter	Description
x	Required. A number
y	Required. A number

Example: Using min function

```
<?php
echo(min(5,7) . "<br />");
echo(min(-3,5) . "<br />");
echo(min(-3,-5) . "<br />");
echo(min(7.25,7.30))
?>
```

Output:

```
5
-3
-5
7.25
```

MAX

Returns the number with the highest value of two specified numbers

Syntax

```
max(x,y)
```

Parameter	Description
X	Required. A number
y	Required. A number

Example: Using max function

```
<?php
echo(max(5,7) . "<br />");
echo(max(-3,5) . "<br />");
echo(max(-3,-5) . "<br />");
echo(max(7.25,7.30))
?>
```

Output:

```
7
5
-3
7.3
```

POW

Returns the value of x to the power of y

Syntax

```
pow(x,y)
```

Parameter	Description
x	Required. Specifies the number to be raised
y	Required. The power to which to raise the number

Example: Using pow function

```
<?php
echo pow(4,2) . "<br />";
echo pow(6,2) . "<br />";
echo pow(-6,2) . "<br />";
echo pow(-6,-2) . "<br />";
echo pow(-6,5.5);
?>
```

Output:

```
16
36
36
0.0277777777778
-1.#IND
```

SQRT

Returns the square root of a number

Syntax

```
sqrt(x)
```

Parameter	Description
x	Required. A number

Example: Using sqrt function

```
<?php
echo(sqrt(0) . "<br />");
echo(sqrt(1) . "<br />");
echo(sqrt(9) . "<br />");
echo(sqrt(0.64) . "<br />");
echo(sqrt(-9))
?>
```

Output:

```
0
1
3
0.8
-1.#IND
```

RAND

Returns a random integer

If this function is called without parameters, it returns a random integer between 0 and RAND_MAX.

If you want a random number between 10 and 100 (inclusive), use rand (10,100).

Syntax

```
rand(min,max)
```

Parameter	Description
min,max	Optional. Specifies the range the random number should lie within

Example: Using rand function

```
<?php
echo(rand() . "<br />");
echo(rand() . "<br />");
echo(rand(10,100))
?>
```

Output:

```
17757
3794
97
```

RAD2DEG()

The rad2deg() function converts radians to degrees. Radians are calculated as being $degrees multiplied by the mathematical constant pi, then divided by 180.

Syntax

PHP rad2deg() Function has the following syntax.

```
float rad2deg ( float number )
```

Parameter

Parameter	Is Required	Description
number	Required.	A radian value to convert

Return

Item	Description
Return Value	The equivalent of number in degrees
Return Type	Float

Example

Convert radians to degrees:

```
<?php
echo rad2deg(pi()) . "\n";
echo rad2deg(pi()/4) . "\n";
```

```
$atan_deg = rad2deg(atan(0.46));
echo $atan_deg;
?>
```

Output:

```
180
45
24.702430227771
```

PI()

The pi() function returns the value of PI.

The named constant M_PI is identical to pi().

Syntax

PHP pi() Function has the following syntax.

```
pi();
```

Parameter

None

Return

Item	Description
Return Value	3.1415926535898
Return Type	Float

Example

Return the value of PI:

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

Output:

```
3.1415926535898
```

EXP()

The exp() function returns e raised to the power of x (ex).

'e' is the base of the natural system of logarithms (approximately 2.718282) and x is the number passed to it.

Syntax

PHP exp() Function has the following syntax.

```
exp(num);
```

Parameter

Parameter	Description
num	Required. Specifies the exponent

Return

Item	Description
Return Value	'e' raised to the power of x
Return Type	Float

Example

Return 'e' raised to the power of different numbers:

```
<?php
echo(exp(0) . "\n");
echo(exp(1) . "\n");
echo(exp(10) . "\n");
echo(exp(4.8));
?>
```

Output:

```
1
2.718281828459
22026.465794807
121.51041751873
```

LOG()

The log() function returns the natural logarithm of a number, or the logarithm of number to base.

Syntax

PHP log() Function has the following syntax.

```
log(number,base);
```

Parameter

Parameter	Is Required	Description
number	Required.	Value to calculate the logarithm for
base	Optional.	The logarithmic base to use. Default is 'e'

Return

Item	Description
Return Value	The natural logarithm of a number, or the logarithm of number to base
Return Type	Float

Example

Return the natural logarithm of different numbers:

```
<?php
echo log(2.7183) . "\n";
echo log(2) . "\n";
echo log(1) . "\n";
echo log(0);
?>
```

Output:

```
1.000006684914
0.69314718055995
0
-INF
```

DECBIN()

The decbin() function converts a decimal number into a binary number.

Syntax

PHP decbin() Function has the following format.

```
string decbin ( int num )
```

Parameter

Parameter	Is Required	Description
num	Required.	Decimal value to convert

Return

Item	Description
Return Value	A string that contains the binary number of the decimal value
Return Type	String

Example

Converts a decimal number into a binary number

```
<?PHP
print decbin(16); // "10000"
echo decbin("3") . "\n";
echo decbin("1") . "\n";
echo decbin("15") . "\n";
echo decbin("7");
?>
```

Output:

```
1000011
1
1111
111
```

DECOCT()

The decoct() function converts a decimal number into an octal number. It takes just one parameter, which is the number to convert.

Syntax

PHP decoct() Function has the following format.

```
string decoct ( int num )
```

Parameter

Parameter	Is Required	Description
num	Required.	Decimal value to convert

Return

Item	Description
Return Value	A string that contains the octal number of the decimal value
Return Type	String

Note : To convert octal to decimal, use the octdec() function.

Example

Convert decimal to octal:

```
<?php
echo decoct("30") . "\n";
echo decoct("10") . "\n";
echo decoct("17") . "\n";
echo decoct("70") . "\n";
print decoct(19); // "23"
?>
```

Output:

```
36
12
21
106
23
```

DECHEX()

The dechex() function converts a decimal number into a binary number.

Syntax

PHP dechex() Function has the following format.

```
string dechex ( int num )
```

Parameter

Parameter	Is Required	Description
number	Required.	Decimal value to convert

Return

Item	Description
Return Value	A string that contains the hexadecimal number of the decimal value
Return Type	String

Note : To convert hexadecimal to decimal, use the hexdec() function.

Example

Convert decimal hexadecimal:

```
<?php
print dechex(232); // "e8"
print "\n";
echo dechex("30") . "\n";
echo dechex("10") . "\n";
echo dechex("1587") . "\n";
echo dechex("70");
?>
```

Output:

```
e8
1e
a
633
46
```

()

The sin() function calculates the sine value.

yntax

P sin() Function has the following syntax.

```
float sin ( float num )
```

Parameter

Parameter	Description
number	Required. Specifies a value in radians

The parameter should be passed as radians. We should use deg2rad() to convert degrees to radians.

Return

PHP sin() Function returns the sine value.

Example

Return the sine of different numbers

```
<?php
echo(sin(3) . "\n");
echo(sin(-3) . "\n");
echo(sin(0) . "\n");
echo(sin(M_PI) . "\n");
echo(sin(M_PI_2) . "\n");

$sin1 = sin(10);
print($sin1);
print "\n";
$sin2 = sin(deg2rad(80));
print($sin2);
?>
```

Output:

```
0.14112000805987
-0.14112000805987
0
1.2246467991474E-16
1
-0.54402111088937
0.98480775301221
```

cos()

cos() calculates the cosine value.

Syntax

PHP cos() Function has the following format.

```
float cos ( float num )
```

Parameter

Parameter	Is Required	Description
number	Required.	A number

The parameter should be passed as radians. We should use deg2rad() to convert degrees to radians.

Return

Item	Description
Return Value	The cosine of number
Return Type	Float

The cos() function returns a numeric value between -1 and 1, which represents the cosine of the angle.

Example

Return the cosine of different numbers:

```
<?php
    $cos1 = cos(10);
    print($cos1);
    echo("\n");
    $cos2 = cos(deg2rad(80));
    print($cos1);
    echo("\n");
    echo(cos(3) . "\n");
    echo(cos(-3) . "\n");
    echo(cos(0) . "\n");
    echo(cos(M_PI) . "\n");
    echo(cos(2*M_PI));
?>
```

Output:

```
-0.83907152907645
-0.83907152907645
-0.98999249660045
-0.98999249660045
1
-1
1
```

TAN()

Calculates the tangent value of the number provided as its only parameter.

Syntax

PHP tan() Function has the following syntax.

```
float tan ( float num )
```

Parameter

Parameter	Description
number	Required. Specifies a value in radians

The parameter should be passed as radians. We should use deg2rad() to convert degrees to radians.

Return

Item	Value
Return Value	The tangent of number
Return Type	Float

Example

Return the tangent of different numbers

```
<?php
$tan1 = tan(10);
print($tan1);
print "\n";
$tan2 = tan(deg2rad(80));
print($tan2);
print "\n";

echo(tan(M_PI_4) . "\n");
echo(tan(0.50) . "\n");
echo(tan(-0.50) . "\n");
echo(tan(5) . "\n");
echo(tan(1) . "\n");
echo(tan(-5) . "\n");
echo(tan(-1));
?>
```

Output:

```
0.64836082745909
5.6712818196177
1
0.54630248984379
-0.54630248984379
-3.3805150062466
1.5574077246549
3.3805150062466
-1.5574077246549
```

DEG2RAD()

The deg2rad() function converts degrees to radians.

Syntax

PHP deg2rad() Function has the following format.

```
float deg2rad ( float num )
```

Parameter

Parameter	Is Required	Description
num	Required.	Degree to convert

Return

Item	Description
Return Value	The radian equivalent of number
Return Type	Float

Note : To convert a radian value to a degree value, look at the rad2deg() function.

Example

Radians are calculated as being $degrees multiplied by the mathematical constant pi, then divided by 180.

```
<?php
echo deg2rad("45") . "\n";
echo deg2rad("90") . "\n";
echo deg2rad("360"). "\n";

$deg = 180;
$rad = deg2rad($deg);
echo "$deg degrees is equal to $rad radians.\n";

$sin1 = sin(deg2rad(80));
print $sin1;
?>
```

Output:

```
0.78539816339745
1.5707963267949
6.2831853071796
180 degrees is equal to 3.1415926535898 radians.
0.98480775301221
```

2.5.3 DATE/TIME FUNCTION

PHP has a rich set of built-in functions that can be used to work with date and time.

DATE

Formats a local time/date

Syntax

```
date(format,timestamp)
```

Parameter	Description
format	Required. Specifies how to return the result:

- d – The day of the month (from 01 to 31)
- D – A textual representation of a day (three letters)
- j – The day of the month without leading zeros (1 to 31)
- l (lowercase 'L') – A full textual representation of a day
- N – The ISO-8601 numeric representation of a day (1 for Monday through 7 for Sunday)
- S – The English ordinal suffix for the day of the month (2 characters st, nd, rd or th. Works well with j)
- w – A numeric representation of the day (0 for Sunday through 6 for Saturday)
- z – The day of the year (from 0 through 365)
- W – The ISO-8601 week number of year (weeks starting on Monday)
- F – A full textual representation of a month (January through December)
- m – A numeric representation of a month (from 01 to 12)
- M – A short textual representation of a month (three letters)
- n – A numeric representation of a month, without leading zeros (1 to 12)
- t – The number of days in the given month
- L – Whether it's a leap year (1 if it is a leap year, 0 otherwise)
- The ISO-8601 year number
- Y – A four digit representation of a year
- y – A two digit representation of a year
- a – Lowercase am or pm
- A – Uppercase AM or PM
- B – Swatch Internet time (000 to 999)
- g – 12-hour format of an hour (1 to 12)
- G – 24-hour format of an hour (0 to 23)
- h – 12-hour format of an hour (01 to 12)
- H – 24-hour format of an hour (00 to 23)
- i – Minutes with leading zeros (00 to 59)

- s – Seconds, with leading zeros (00 to 59)
- e – The timezone identifier (Examples: UTC, Atlantic/Azores)
- I (capital i) – Whether the date is in daylights savings time (1 if Daylight Savings Time, 0 otherwise)
- Difference to Greenwich time (GMT) in hours (Example: +0100)
- T – Timezone setting of the PHP machine (Examples: EST, MDT)
- Z – Timezone offset in seconds. The offset west of UTC is negative, and the offset east of UTC is positive (-43200 to 43200)
- c – The ISO-8601 date (e.g. 2004-02-12T15:19:21+00:00)
- r – The RFC 2822 formatted date (e.g. Thu, 21 Dec 2000 16:01:07 +0200)
- U – The seconds since the Unix Epoch (January 1 1970 00:00:00 GMT)
- timestamp Optional.

Example: Using date function

```
<?php
// Assuming today is March 10th, 2001, 5:16:18 pm, and that we are in the
// Mountain Standard Time (MST) Time Zone
$today = date("F j, Y, g:i a");                    // March 10, 2001, 5:16 pm
$today = date("m.d.y");                            // 03.10.01
$today = date("j, n, Y");                          // 10, 3, 2001
$today = date("Ymd");                              // 20010310
$today = date('h-i-s, j-m-y, it is w Day');        // 05-16-18, 10-03-01, 1631 1618 6 Satpm01
$today = date('\i\t \i\s \t\h\e jS \d\a\y.');      // it is the 10th day.
$today = date("D M j G:i:s T Y");                  // Sat Mar 10 17:16:18 MST 2001
$today = date('H:m:s \m \i\s\ \m\o\n\t\h');        // 17:03:18 m is month
$today = date("H:i:s");                            // 17:16:18
?>
```

GETDATE

Returns an array that contains date and time information for a Unix timestamp\

The returning array contains ten elements with relevant information needed when formatting a date string:

[seconds] - seconds

[minutes] - minutes

[hours] - hours

[mday] - day of the month

[wday] - day of the week

[year] - year

[yday] - day of the year

[weekday] - name of the weekday

[month] - name of the month

Syntax

```
getdate(timestamp)
```

Parameter	Description
timestamp	Optional. Specifies the time in Unix time format

Example-1: Using getdate function

```
<?php
print_r(getdate());
?>
```

Output:

```
Array
{
    [seconds] => 45
    [minutes] => 52
    [hours] => 14
    [mday] => 24
    [wday] => 2
    [mon] => 1
    [year] => 2006
    [yday] => 23
    [weekday] => Tuesday
    [month] => January
    [0] => 1138110765
}
```

Example-2: Using getdate function

```
<?php
$my_t=getdate(date("U"));
print("$my_t[weekday], $my_t[month] $my_t[mday], $my_t[year]");
?>
```

Output:

```
Wednesday, January 25, 2006
```

SETDATE

Sets the date

This function set the year, month and day as per passing parameters from setdate() function.

Parameter	Description
Month	Required. Specifies the month
day	Required. Specifies the day
year	Required. Specifies the year

Syntax

```
setdate ( year ,month ,day )
```

Example: Using setdate function

```
<?php
$date = date_create();
date_date_set($date, 2001, 2, 3);
echo date_format($date, 'Y-m-d');
?>
```

Output:

```
2001-02-03
```

CHECKDATE

Validates a Gregorian date

A date is valid if:

- month is between 1 and 12 inclusive
- day is within the allowed number of days for the particular month
- year is between 1 and 32767 inclusive

Syntax

```
checkdate(month,day,year)
```

Parameter	Description
Month	Required. Specifies the month
day	Required. Specifies the day
year	Required. Specifies the year

Example: Using checkdate function

```
<?php
var_dump(checkdate(12,31,2000));
var_dump(checkdate(2,29,2003));
var_dump(checkdate(2,29,2004));
?>
```

Output:

```
bool(true)
bool(false)
bool(true)
```

TIME

The time() function returns the current time as a Unix timestamp (the number of seconds since January 1 1970 00:00:00 GMT).

Syntax

```
time(void)
```

Parameter	Description
Void	Optional.

Example: Using time function

```
<?php
$t=time();
echo($t . "<br />");
echo(date("D F d Y",$t));
?>
```

Output:

```
1138618081
Mon January 30 2006
```

MKTIME

Returns the 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)
```

Parameter	Description
hour	Optional. Specifies the hour
minute	Optional. Specifies the minute
second	Optional. Specifies the second
month	Optional. Specifies the numerical month
day	Optional. Specifies the day
year	Optional. Specifies the year. The valid range for year is on some systems between 1901 and 2038.
is_dst	Optional. Set this parameter to 1 if the time is during daylight savings time (DST), 0 if it is not, or -1 (the default) if it is unknown.

Example: Using mktime function

```
<?php
echo(date("M-d-Y",mktime(0,0,0,12,36,2001))."<br />");
echo(date("M-d-Y",mktime(0,0,0,14,1,2001))."<br />");
echo(date("M-d-Y",mktime(0,0,0,1,1,2001))."<br />");
echo(date("M-d-Y",mktime(0,0,0,1,1,99))."<br />");
?>
```

Output:

```
Jan-05-2002
Feb-01-2002
Jan-01-2001
Jan-01-1999
```

GETTIMEOFDAY()

The gettimeofday() function returns the current time.

Syntax

PHP gettimeofday() Function has the following syntax.

```
gettimeofday(return_float);
```

Parameter

PHP gettimeofday() Function has the following syntax.

- return_float - Optional. When set to TRUE, a float instead of an array is returned. Default is FALSE

Return

PHP gettimeofday() Function returns an associative array by default, with the following array keys:

- [sec] - seconds since the Unix Epoch
- [usec] - microseconds
- [minuteswest] - minutes west of Greenwich
- [dsttime] - type of dst correction

If the return_float parameter is set to true, a float is returned

Example

Return the current time:

```
<?php
// Print the array from gettimeofday()
print_r(gettimeofday());

// Print the float from gettimeofday()
echo gettimeofday(true);
?>
```

Output:

```
Array
{
    [sec] => 1677409087
    [usec] => 217982
    [minuteswest] => 0
    [dsttime] => 0
}
1677409087.2487
```

DATE_CREATE()

The date_create() function returns a new DateTime object.

Syntax

PHP date_create() Function has the following syntax.

```
date_create(time,timezone);
```

Parameter

PHP date_create() Function has the following parameters.

Parameter	Is Required	Description
time	Optional.	A date/time string. NULL indicates the current time.
timezone	Optional.	Timezone of time. Default is the current timezone.

Return

PHP date_create() Function returns a new DateTime object on success. FALSE on failure.

Example-1:

Return a new DateTime object, and then format the date:

```
<?php
$date=date_create("2013-01-15");
echo  date_format($date,"Y/m/d");
?>
```

Output:

```
2023/01/25
```

Example-2:

Return a new DateTime object (with a given timezone), and then format the date and time:

```
<?php
$date=date_create("2013-02-15  21:40:00",timezone_open("Europe/Oslo"));
echo  date_format($date,"Y/m/d  H:iP");
?>
```

Output:

```
2023/02/15  21:40+01:00
```

DATE_FORMAT()

The date_format() function formats a date for the specified format.

Syntax

PHP date_format() Function has the following syntax.

```
date_format(object,format);
```

Parameter

Parameter	Is Required	Description
object	Required.	DateTime object returned by date_create()
format	Required.	Format for the date

Format

Format character	Description	Example
a	Lowercase am/pm	am or pm
A	Uppercase am/pm	AM or PM
B	Swatch Internet Time	000 to 999
c	ISO 8601 date, time, and time zone	2004-06-18T09:26:55+01:00
d	2-digit day of month, leading zeros	01 to 31
D	Day string, three letters	Mon, Thu, Sat
F	Month string, full	January, August
g	12-hour clock hour, no leading zeros	1 to 12
G	24-hour clock hour, no leading zeros	0 to 23
h	12-hour clock hour, leading zeros	01 to 12
H	24-hour clock hour, leading zeros	00 to 23
i	Minutes with leading zeros	00 to 59
I	Is daylight savings time active?	1 if yes, 0 if no
j	Day of month, no leading zeros	1 to 31
l	Day string, full	Monday, Saturday
L	Is it a leap year?	1 if yes, 0 if no
m	Numeric month, leading zeros	01 to 12
M	Short month string	Jan, Aug
n	Numeric month, no leading zeros	1 to 12
O	Difference from GMT	200
r	RFC-822 formatted date	Sat, 22 Dec 1979 17:30 +0000
s	Seconds, with leading zeros	00 to 59
S	English ordinal suffix for day number	st, nd, rd, or th
t	Number of days in month	28 to 31
T	Time zone for server	GMT, CET, EST
U	Unix Timestamp	1056150334
w	Numeric day of week	0 (Sunday), 6 (Saturday)
W	ISO-8601 week number of year	30 (30th week of the year)
y	Two-digit representation of year	97, 02
Y	Four-digit representation of year	1997, 2002
z	Day of year	0 to 366
Z	Time zone offset in seconds	-43200 to 43200

DATE_ATOM	Atom	2013-04-12T15:52:01+00:00
DATE_COOKIE	HTTP Cookies	Friday, 12-Apr-13 15:52:01 UTC
DATE_ISO8601	ISO-8601	2013-04-12T15:52:01+0000
DATE_RFC822	RFC 822	Fri, 12 Apr 13 15:52:01 +0000
DATE_RFC850	RFC 850	Friday, 12-Apr-13 15:52:01 UTC
DATE_RFC1036	RFC 1036	Fri, 12 Apr 13 15:52:01 +0000
DATE_RFC1123	RFC 1123	Fri, 12 Apr 2013 15:52:01 +0000
DATE_RFC2822	RFC 2822	Fri, 12 Apr 2013 15:52:01 +0000
DATE_RFC3339	Same as DATE_ATOM (since PHP 5.1.3)	2013-04-12T15:52:01+00:00
DATE_RSS	RSS	Fri, 12 Aug 2013 15:52:01 +0000
DATE_W3C	World Wide Web Consortium	2013-04-12T15:52:01+00:00

Return

PHP date_format() Function returns the formatted date as a string. FALSE on failure.

Example

Return a new DateTime object, and then format the date:

```
<?php
$date=date_create("2013-03-15");
echo  date_format($date,"Y/m/d  H:i:s");
echo  "\n";
$date=date_create("2013-03-15");
echo   date_format($date,DATE_ATOM);
?>
```

Output:

```
2023/03/15  00:00:00
2023-03-15T00:00:00+00:00
```

DATE_DIFF()

The date_diff() function returns the difference between two DateTime objects.

Syntax

PHP date_diff() Function has the following syntax.

```
date_diff(datetime1,datetime2,absolute);
```

Parameter

Parameter	Is Required	Description
datetime1	Required.	DateTime object
datetime2	Required.	DateTime object
absolute	Optional.	TRUE indicates that the interval/difference MUST be positive. Default is FALSE

Return

PHP date_diff() Function returns a DateInterval object on success that represents the difference between the two dates.

PHP date_diff() Function returns FALSE on failure.

Example

Calculate the difference between two dates

```
<?php
$date1=date_create("2011-03-15");
$date2=date_create("2013-12-12");
$diff=date_diff($date1,$date2);
?>
```

IMPORTANT QUESTIONS

Short Questions 2 Marks

1. Explain foreach statement with example

Ans.

- The Foreach loop is used to loop over all the elements of an array.
- **Syntax:**

```
foreach (array as value)
{
    code to be executed;
}
foreach (array as key => value)
{
    code to be executed;
}
```

- An alternative form of For each loop gives you access to the current key.

```
<?php
$person = array("Name" => "Andrew", "Age" => 21);
foreach ($person as $key => $value)
{
    echo $key . " is " . $value . "<br />";
}
?>
```

- **Output:**

```
Name is Andrew
Age is 21
```

2. Explain any two string functions.

Ans.

- **chr:**
  - Returns a one-character string containing the character specified byASCII value.
  - Syntax is string chr (ASCII – VALUE)
- **ord:**
  - Returns the ASCII value of the first character of string.
  - Syntax is int ord (string)

3. Explain any two mathematical functions.

Ans.

- **abs**
  - The abs() function returns the absolute value of a number.
  - Syntax is abs(x)
- **ceil**
  - The ceil() function returns the value of a number rounded UPWARDS to the nearest integer.
  - Syntax: float ceil(x)

4. Explain strpos () function.

Ans.

- Find position of first occurrence of a string.
- Syntax is int strpos (string $str , mixed $needle [, int $offset = 0])
- Returns the numeric position of the first occurrence of needle in the str string.
- **Example:**

```
<?php
$mystring = 'abc';
$findme   = 'a';
$pos = strpos($mystring, $findme);

if ($pos !== false)
{
    echo "The string '$findme' was found in the string '$mystring'";
    echo " and exists at position $pos";
}
else
{
    echo "The string '$findme' was not found in the string '$mystring'";
}
?>
```

- **Output**

```
The string 'a' was found in the string 'abc' and exists at position 0
```

5. Explain fmod () function.

Ans.

- fmod() (Floating Modulo) function, modulo calculates the reminder of a division which is generally noted the '%' symbol.
- Syntax is float exp ($divided,$divisor)
- It is used to calculate the modulo of any division which may contain floats as both dividend and divisors.
- This function returns a floating-point reminder of the division.

6. Define term Defalult Argument.

Ans.

- In PHP, you can define a function having default arguments.
- So if you don'tpass the value for that argument then it will consider the default value for thatargument.
- But if you pass the explicit value for that argument then it will overwrite thedefault argument value.
- The default arguments must be specified after all non default arguments in thefunction.

■ Questions 3 Marks

1. Explain any two string functions.
2. Explain passing by reference to function.
3. Explain with example user defined function in PHP.
4. Write short note on Associative Array.
5. Explain multidimensional array.
6. Explain GLOBAL variable in PHP with example.
7. Explain user defined function with default arguments.
8. Explain user defined function with variable length arguments.
9. Explain following functions: strlen, stristr, strrev.
10. Explain following functions: ceil, floor, round.
11. Explain foreach statement with example.
12. Explain global statement with example.

■ Questions 4 Marks

1. What is Array? Explain Numeric Array with Example.
2. What is Array? Explain Associative Array with Example.
3. Explain any two Date functions with example.
4. Explain associative array with example.
5. What is user defining function? Explain how to define and call function that returns value.
6. Explain how to access variable with global statement.
7. List methods for passing arguments in function. Explain any one with example.
8. Explain foreach statement with example using concept of associative array.

   Explain function with default argument.

   Explain passing by values to function.

Questions 7 Marks

1. How to use array in PHP? Explain foreach construct in PHP with example.
2. Explain numeric and associative array with example.
3. List various string function and explain any three with example.
4. Explain with example user defined function in PHP.
5. Explain different math functions used in PHP with example.

ASSIGNMENTS

1. **Arrays Assignment:**

   (a) Write a PHP script to declare an indexed array "fruits" containing the values "apple", "banana", and "orange". Then, print the value of the first element in the array.

   (b) Write a PHP script to declare an associative array "marks" containing the keys "maths", "science", and "english" and their respective values 90, 85, and 80. Then, print the value of the "science" key.

   (c) Write a PHP script to declare a multidimensional array "students" containing the names and marks of three students. The array should have the following structure:

```
[
    [ "name" => "John", "maths" => 90, "science" => 85, "english" => 80 ],
    [ "name" => "Jane", "maths" => 80, "science" => 75, "english" => 70 ],
    [ "name" => "Bob", "maths" => 70, "science" => 65, "english" => 60 ]
]
```

   Then, print the name and marks of each student.

2. **Functions Assignment:**

   (a) Write a PHP function to calculate the sum of two numbers and return the result. The function should take two parameters "num1" and "num2".

   (b) Write a PHP function to find the maximum of three numbers and return the result. The function should take three parameters "num1", "num2", and "num3".

   (c) Write a PHP function to reverse a string and return the result. The function should take one parameter "str" containing the string to be reversed.

3. **String Functions Assignment:**

   (a) Write a PHP script to declare a string variable "str" containing the value "Hello World". Then, print the length of the string using the strlen() function.

   (b) Write a PHP script to declare a string variable "str" containing the value "Welcome to PHP". Then, convert the string to uppercase using the strtoupper() function and print the result.

   (c) Write a PHP script to declare a string variable "str" containing the value "This is a test". Then, replace the word "test" with the word "example" using the str_replace() function and print the result.

4. Mathematical Functions Assignment:

(a) Write a PHP script to declare two variables "num1" and "num2" and assign them values 10 and 3 respectively. Then, find the quotient and remainder of num1 divided by num2 using the divmod() function and print the result.

(b) Write a PHP script to declare an array "numbers" containing the values 5, 10, 15, 20, and 25. Then, find the sum of the array elements using the array_sum() function and print the result.

(c) Write a PHP script to declare a variable "num" containing the value 2. Then, find the square root of num using the sqrt() function and print the result.

5. Date Functions Assignment:

(a) Write a PHP script to print the current date and time using the date() function.

(b) Write a PHP script to declare a variable "date" containing the value "2022-01-01". Then, find the day of the week for the given date using the date() and strtotime() functions and print the result.

(c) Write a PHP script to declare a variable "date1" containing the value "2022-01-01" and a variable "date2" containing the value "2022-02-01". Then, find the number of days between the two dates using the strtotime() function and print the result.

EXERCISE

Short Question and Answers

1. What is an array in PHP?

   Ans. An array is a collection of related values of the same data type.

2. What is the syntax for creating an array in PHP?

   Ans. $array_name = array(value1, value2, value3, ...);

3. How do you access the first element of an array in PHP?

   Ans. $array_name[0];

4. How do you access the last element of an array in PHP?

   Ans. $array_name[count($array_name) - 1];

5. What is the purpose of the count() function in PHP?

   Ans. It returns the number of elements in an array.

6. What is the difference between an indexed array and an associative array in PHP?

   Ans. An indexed array uses numeric keys to access elements, while an associative array uses named keys (strings) to access elements.

7. What is the syntax for creating an associative array in PHP?

   Ans. $array_name = array("key1" => value1, "key2" => value2, "key3" => value3, ...);

8. How do you access an element in an associative array in PHP?

   Ans. $array_name["key"];

9. What is the purpose of the foreach loop in PHP?

Ans. It is used to iterate over the elements of an array.

10. What is the difference between the array_push() function and the [] operator for adding elements to an array in PHP?

Ans. The array_push() function adds one or more elements to the end of an array, while the [] operator can be used to add an element to the end of an array or to a specific index.

11. What is a string in PHP?

Ans. A string is a sequence of characters.

12. What is the syntax for creating a string in PHP?

Ans. $string_name = "string value";

13. How do you find the length of a string in PHP?

Ans. strlen($string_name);

14. What is the purpose of the substr() function in PHP?

Ans. It is used to extract a substring from a string.

15. What is the purpose of the strpos() function in PHP?

Ans. It is used to find the position of the first occurrence of a substring in a string.

16. What is the purpose of the str_replace() function in PHP?

Ans. It is used to replace all occurrences of a substring in a string with another string.

17. What is the difference between single quotes and double quotes in PHP strings?

Ans. Single quotes are used to create a string exactly as it appears, while double quotes allow for variable interpolation and escape sequences.

18. What is the purpose of the strtolower() function in PHP?

Ans. It is used to convert a string to lowercase.

19. What is the purpose of the strtoupper() function in PHP?

Ans. It is used to convert a string to uppercase.

20. What is the purpose of the abs() function in PHP?

Ans. It is used to return the absolute (positive) value of a number.

21. What is the purpose of the round() function in PHP?

Ans. It is used to round a number to the nearest integer.

22. What is the purpose of the floor() function in PHP?

Ans. It is used to round a number down to the nearest integer.

23. What is the purpose of the ceil() function in PHP?

Ans. It is used to round a number up to the nearest integer.

24. What is the purpose of the rand() function in PHP?

Ans. '. is used to generate a random integer.

25. What is the purpose of the sqrt() function in PHP?

Ans. It is used to return the square root of a number.

26. What is the purpose of the pow() function in PHP?

Ans. It is used to raise a number to a power.

27. What is the purpose of the log() function in PHP?

Ans. It is used to return the natural logarithm of a number.

28. What is the purpose of the sin() function in PHP?

Ans. It is used to return the sine of an angle (in radians).

29. What is the purpose of the cos() function in PHP?

Ans. It is used to return the cosine of an angle (in radians).

■ * ■