



PHP 5

*Prepared By: Eng. Jack Fayed
National Telecommunication Institute - NTI*



What is PHP



PHP is a programming language that allows web developers to create dynamic content that interacts with databases. PHP is basically used for developing web based software applications.

- *PHP is an acronym for "PHP Hypertext Preprocessor".*
- *PHP is a server-side scripting language and it is executed on the server.*
- *PHP is free and open-source and works with all major operating systems.*
- *PHP Creates a customized user experience for visitors based on information gathered from them.*
- *It is integrated with a number of popular databases, including MySQL, PostgreSQL, Oracle, Sybase, Informix, and Microsoft SQL Server.*

- *PHP generates dynamic webpage contents.*
- *PHP can create, open, read, write, delete, and close files on the server.*
- *PHP can collect form data.*
- *PHP can send and receive cookies data.*
- *PHP can add, delete, and modify data in database.*
- *PHP can restrict users to access some pages on a website.*
- *PHP can encrypt data.*

- *PHP files can contain text, HTML, CSS, JavaScript, and PHP code.*
- *PHP code are executed on the server, and the result is returned to the browser as HTML.*
- *PHP files have ".php" extension.*

PHP - Environment Setup

In order to develop and run PHP Web pages three vital components needed to be installed on your system.

Web Server

- *PHP needs a Web Server software, including Microsoft's Internet Information Server (IIS), but then most often used is Apache Server.*

Database

- *PHP will work with many databases, including Oracle and Sybase, but the most commonly used is MySQL database.*

The PHP engine

- *In order to process PHP script instructions a PHP Engine must be installed to generate HTML output that can be sent to the Web Browser.*



Escaping to PHP



The PHP engine needs a way to differentiate PHP code from other elements in the page. The mechanism for doing so is known as 'escaping to PHP'.

Canonical PHP tags

The most universally effective PHP tag style:

Syntax

<?php

php code here

?>

Same as

<?php

php code here

?>



Adding PHP to HTML pages



Case Sensitivity

- *The PHP keywords are NOT case-sensitive but variable names are case-sensitive.*

The Semicolon

- *The semicolon after the line of PHP code indicates the end of a PHP statement and should never be forgotten.*

White Space

- *Whitespaces, tab indents, and blank lines are ignored between PHP statements.*

Syntax

```
<html>
  <head>
    <title>My First PHP Page</title></head>
  <body>
    <?php
      PHP code goes here.....
    ?>
  </body>
</html>
```



Commenting PHP Code



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

Single-line comments

They are generally used for short explanations

Syntax

```
<?php  
    # This is a Single-line comment, and  
    # This is the second line of the comment  
  
    // This is a Single-line comment too.  
?>
```

Multiple lines comment

They are generally used to provide pseudocode algorithms and more detailed explanations when necessary.

Syntax

```
<?php  
/* This is Multiple lines comment  
Author : -----  
Purpose: -----  
Subject: PHP 5  
*/  
?>
```

PHP Variables

The variable is the main way to store information in the middle of a PHP program.

About PHP variables.

- All variables in PHP are denoted with a leading dollar sign (\$), followed by the variable name.
- Variable name must start with a letter or the underscore character and cannot start with a number.
- Variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _).
- Variable names are case sensitive.
- Variables are assigned with the “=” operator, with the variable on the left-hand side and the value on the right-hand side.

PHP Variables Creating (Declaring).

- PHP has no syntax to declare a variable, it will be created when a value is assigned to it.
- PHP automatically converts the variable to the correct data type, depending on its value.

Example

```
<?php  
$x="Hello world";      ➔ x is a string variable  
$y=300;                ➔ y is a number variable  
$z=15.3;               ➔ z is a number variable  
?>
```

“echo” and “print” Keywords

“echo” and “print” commands are used to print the output on an HTML page.

PHP “echo” Statement.

- “echo” statement are used to print the output on an HTML page.
- “echo” statement can display anything that can be displayed to the browser, such as strings, numbers, variables values.
- “echo” is a language construct not a function, so it can be used without parentheses “echo” or “echo()”.
- “echo” can pass multiple string separated as (,).

Example

```
<?php  
$x=1983;  
echo "<h2>NTI was established in </h2>";  
echo $x,"<br>";  
echo " NTI ", " was ", " established ", " in ", $x;  
echo "<h2>NTI was established in $x</h2>";  
?>
```

NTI was established in

1983

NTI was established in 1983

NTI was established in 1983

PHP “print” Statement.

- *The print statement (an alternative to echo) used to display output to the browser.*
- *“print” is also a language construct not function, so it can be used without parentheses “print” or “print()”.*
- *“print” can print only one string at a single statement*
- *“print” always returns 1. That's why the echo statement considered marginally faster as it doesn't return any value.*

Example

```
<?php  
$x=1983;  
print "<h2> NTI was established in </h2>";  
print $x;  
print "<br>";  
print "NTI was established in $x.";  
?>
```

NTI was established in

1983

NTI was established in 1983

NTI was established in 1983

Single or Double Quotes

PHP allows to use single or double quotes: with double quotes you can easily concatenate variables, while with single quotes you cannot.

Example

```
<?php  
$x=1983;  
echo "<h3>NTI was established in $x</h3>";  
?>
```

NTI was established in 1983

Example

```
<?php  
$x=1983;  
echo '<h3>NTI was established in $x</h3>';  
?>
```

NTI was established in \$x

PHP Concatenation

PHP Concatenation is linking strings or variables together. For this the dot (.) is needed to be used; the dot is the concatenation operator. The concatenations works with both the “echo” and the “print” statements.

Example

```
<?php  
$x=1983;  
  
echo "<h3> NTI was established in ". $x . "</h3>";  
  
print "<h3> NTI was established in ". $x . "</h3>";  
  
?>
```

NTI was established in 1983

NTI was established in 1983



The Escape Character

Sometimes it is necessary to escape some characters, such as: printing the single or the double quotes itself in your document, in this case you must stop the php from interpreting this character as a part of the code and display it as it is.

In this case you need to use the backslash (\) as an escape character.

Example

```
<?php  
    echo 'That\'s the world!!!!';  
?>
```

That's the world!!!!

var_dump() function

The “var_dump” function is a function used to display structured information about the variables including its type, value and length.

var_dump() function

Syntax

```
var_dump(variablename);
```

Example

```
<?php  
$abc = 123;  
var_dump($abc);  
$xyz = "NTI";  
var_dump($xyz);  
?>
```

```
int 123
```

```
string 'NTI' (Length=3)
```



PHP Data Types

PHP data types are used to hold different types of data or values. PHP supports 8 primitive data types that can be divided into 3 categories:

- *Scalar Data Types.*
- *Compound Data Types.*
- *Special Data Types.*

Scalar Data Types:

- *Integer Data Type.*
- *Floating Point Numbers Data Type.*
- *String Data Type.*
- *Boolean Data Type.*

Compound Data Types:

- *Array Data Type.*
- *Object Data Type.*

Special Data Types.

- *Resource Data Type.*
- *NULL Data Type.*



PHP Scalar Data Types

A variable is a scalar when it holds a single value. PHP has four scalar data types including integer, float, string and Boolean.



PHP Integers Data Type



PHP Integers are numbers, without fractional part or decimal point. it can be specified in decimal (base 10), hexadecimal (base 16) or octal (base 8) notation, and optionally preceded by a sign (- or +).

Decimal (base 10) Integer

The decimal Integer is the integer that has a base of 10. Therefore, the 10 digits 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9 are used.

Example

```
<?php  
    // Decimal base integers  
    $dec1 = 654;  
    Var_dump($dec1);  
?>
```

int 654

Octal (base 8) Integer

The octal Integer is the integer that has a base of 8. Therefore, the 8 digits 0, 1, 2, 3, 4, 5, 6 and 7 are used.

In PHP, the octal integers is declared with leading 0.

Example

```
<?php  
    // octal base integers  
    $oct1 = 0205;  
    var_dump($oct1);  
?>
```

int 133

Hexadecimal (base 16) Integer

The decimal Integer is the integer that has a base of 16. The 10 digits from “0” to “9” and the 6 letters from “A” to “F” are used. The Hexadecimal digit “A” is equivalent to the decimal number “10” and the Hexadecimal digit “F” is equivalent to the decimal number “15”.

*In PHP, The hexadecimal integer is declared with leading **0x**.*

Example

```
<?php  
    // hexadecimal base integers  
    $hex = 0x45;  
    var_dump($hex);  
?>
```

int 69

PHP Floating Point Numbers Data Type

Floating point numbers (also known as "floats", "doubles", or "real numbers") are decimal or fractional numbers.

PHP Floating Point Numbers Data Type

Example

```
<?php  
    $a = 1.234;  
    var_dump($a);  
  
    $b = 10.2e3;  
    var_dump($b);  
  
    $c = 4E-10;  
    var_dump($c);  
?>
```

```
float 1.234  
float 10200  
float 4.0E-10
```

PHP Strings Data Type

Strings are sequences of characters, A string can hold letters, numbers, and special characters. The simplest way to specify a string is to enclose it in single or double quotes.

Example

```
<?php  
$a = 'MCIT';  
var_dump($a);  
  
$b = "#NTI#";  
var_dump($b);  
  
$c = 'That\'s Egypt.';  
var_dump($c);  
?>
```

```
string 'MCIT' (Length=4)  
string '#NTI#' (Length=5)  
string 'That's Egypt.' (Length=13)
```

PHP Booleans Data Type

PHP Booleans hold only one of two values, either TRUE or FALSE.

Example

```
<?php  
  
    // Assign the value TRUE to a variable  
  
    $x = true;  
  
    var_dump($x);  
  
?>
```

boolean true

PHP Compound Data Types

Compound data is the data that can contain more than one value. PHP provides two compound data types including arrays and objects.

PHP Arrays Data Type

An array is a variable that can hold more than one value at a time. It is useful to aggregate a series of related items together.

Example

```
<?php  
  
$colors = array("Red", "Green", "Blue");  
  
var_dump($colors);  
  
?>
```

```
array (size=3)  
  0 => string 'Red' (Length=3)  
  1 => string 'Green' (Length=5)  
  2 => string 'Blue' (Length=4)
```



PHP Objects Data Type



An object is a data type that not only allows storing data but also information about it. An object is a specific instance of a class which serve as templates for objects.

Example

```
<?php  
class greeting{  
}  
$message = new greeting;  
var_dump($message);  
?>
```

object(greeting)[1]

PHP Special Data Types

There are two special data types in PHP that have special meanings: Resource and Null.



PHP Resources Data Type

A resource is a special variable, holding a reference to an external resource.

Resource variables typically hold special handlers to opened files and database connections.

Example

```
<?php  
    $handle = fopen("note.txt", "r");  
    var_dump($handle);  
  
    $link = mysql_connect("localhost", "root", "pass");  
    var_dump($link);  
?>
```

resource(3, stream)
resource(5, mysql Link)

PHP NULL Data Type

The NULL value is used to represent empty variables in PHP. A variable of type NULL is a variable without any data.

Example

```
<?php  
  
$a = NULL;  
  
var_dump($a);  
  
?>
```

null

Note that both \$X = NULL; and \$X = "" are not the same.



PHP Operators



Operators are symbols that tell the PHP processor to perform certain actions. For example, the addition (+) symbol is an operator that tells PHP to add two variables or values, while the greater-than (>) symbol is an operator that tells PHP to compare two values.

PHP Arithmetic Operators

The arithmetic operators are used to perform common arithmetical operations, such as addition, subtraction, multiplication and division.

<i>Operator</i>	<i>Description</i>	<i>Example</i>	<i>Result</i>
+	Addition	$$x + y	Sum of $$x$ and $$y$
-	Subtraction	$$x - y	Difference of $$x$ and $$y$.
*	Multiplication	$$x * y	Product of $$x$ and $$y$.
/	Division	$$x / y	division of $$x$ on $$y$
%	Modulus	$$x \% y	Remainder of $$x$ divided by $$y$

Example

```
<?php  
$x = 10;  
$y = 4;  
echo($x + $y);           // Outputs: 14  
echo($x - $y);           // Outputs: 6  
echo($x * $y);           // Outputs: 40  
echo($x / $y);           // Outputs: 2.5  
echo($x % $y);           // Outputs: 2  
?>
```

PHP Assignment Operators

The assignment operators are used to assign values to variables.

PHP Assignment Operators

<i>Operator</i>	<i>Description</i>	<i>Example</i>	<i>Same As</i>
=	<i>Assign</i>	$\$x = \y	$\$x = \y
+=	<i>Add and assign</i>	$\$x += \y	$\$x = \$x + \$y$
-=	<i>Subtract and assign</i>	$\$x -= \y	$\$x = \$x - \$y$
*=	<i>Multiply and assign</i>	$\$x *= \y	$\$x = \$x * \$y$
/=	<i>Divide and assign result</i>	$\$x /= \y	$\$x = \$x / \$y$
%=	<i>Divide and assign modulus</i>	$\$x %= \y	$\$x = \$x \% \$y$

Example

assume that each line x starts by “\$x = 100;”

```
$x += 30;          // x=130  
$x -= 20;          // x=80  
$x *= 25;          // x=2500  
$x /= 10;          // x=10  
$x %= 15;          // x= 10
```



PHP Comparison Operators



The comparison operators are used to compare two values in a Boolean fashion.

PHP Comparison Operators

Operator	Name	Example	Result
<code>==</code>	<i>Equal</i>	<code>\$x == \$y</code>	<i>True if \$x is equal to \$y</i>
<code>===</code>	<i>Identical</i>	<code>\$x === \$y</code>	<i>True if \$x and \$y are equal and have the same type</i>
<code>!=</code>	<i>Not equal</i>	<code>\$x != \$y</code>	<i>True if \$x is not equal to \$y</i>
<code><></code>	<i>Not equal</i>	<code>\$x <> \$y</code>	<i>True if \$x is not equal to \$y</i>
<code>!==</code>	<i>Not identical</i>	<code>\$x !== \$y</code>	<i>True if \$x is not equal to \$y, or they are not of the same type</i>
<code><</code>	<i>Less than</i>	<code>\$x < \$y</code>	<i>True if \$x is less than \$y</i>
<code>></code>	<i>Greater than</i>	<code>\$x > \$y</code>	<i>True if \$x is greater than \$y</i>
<code>>=</code>	<i>Greater than or equal to</i>	<code>\$x >= \$y</code>	<i>True if \$x is greater than or equal to \$y</i>
<code><=</code>	<i>Less than or equal to</i>	<code>\$x <= \$y</code>	<i>True if \$x is less than or equal to \$y</i>

Example

```
$x = 25;  
$y = 35;  
$z = "25";  
  
var_dump($x == $z);           // Outputs: boolean true  
var_dump($x === $z);          // Outputs: boolean false  
var_dump($x != $y);           // Outputs: boolean true  
var_dump($x !== $z);          // Outputs: boolean true  
var_dump($x < $y);            // Outputs: boolean true  
var_dump($x > $y);            // Outputs: boolean false  
var_dump($x <= $y);           // Outputs: boolean true  
var_dump($x >= $y);           // Outputs: boolean false
```



PHP Incrementing and Decrementing Operators



The increment/decrement operators are used to increment/decrement a variable's value.

PHP Incrementing and Decrementing Operators

<i>Operator</i>	<i>Name</i>	<i>Effect</i>
<code>++\$x</code>	<i>Pre-increment</i>	<i>Increments \$x by one, then returns \$x</i>
<code>\$x++</code>	<i>Post-increment</i>	<i>Returns \$x, then increments \$x by one</i>
<code>--\$x</code>	<i>Pre-decrement</i>	<i>Decrements \$x by one, then returns \$x</i>
<code>\$x--</code>	<i>Post-decrement</i>	<i>Returns \$x, then decrements \$x by one</i>

PHP Incrementing and Decrementing Operators

Example

```
$x = 10;  
    echo ++$x;      // Outputs: 11  
    echo $x;        // Outputs: 11  
  
$x = 10;  
    echo $x++;      // Outputs: 10  
    echo $x;        // Outputs: 11  
  
$x = 10;  
    echo --$x;      // Outputs: 9  
    echo $x;        // Outputs: 9  
  
$x = 10;  
    echo $x--;      // Outputs: 10  
    echo $x;        // Outputs: 9
```

PHP Logical Operators

The logical operators are typically used to combine conditional statements.

<i>Operator</i>	<i>Name</i>	<i>Example</i>	<i>Result</i>
<i>and</i>	<i>And</i>	$\$x$ and $\$y$	<i>True if both \$x and \$y are true</i>
<i>or</i>	<i>Or</i>	$\$x$ or $\$y$	<i>True if either \$x or \$y is true</i>
<i>xor</i>	<i>Xor</i>	$\$x$ xor $\$y$	<i>True if either \$x or \$y is true, but not both</i>
<i>&&</i>	<i>And</i>	$\$x \&& \y	<i>True if both \$x and \$y are true</i>
<i> </i>	<i>Or</i>	$\$x \y	<i>True if either \$x or \$y is true</i>
<i>!</i>	<i>Not</i>	$!\$x$	<i>True if \$x is not true</i>

PHP String Operators

There are two operators which are specifically designed for strings.

<i>Operator</i>	<i>Description</i>	<i>Example</i>	<i>Result</i>
.	<i>Concatenation</i>	<code>\$str1 . \$str2</code>	<i>Concatenation of \$str1 and \$str2</i>
.=	<i>Concatenation assignment</i>	<code>\$str1 .= \$str2</code>	<i>Appends the \$str2 to the \$str1</i>

Example

```
<?php
```

```
$x = "Hello";
```

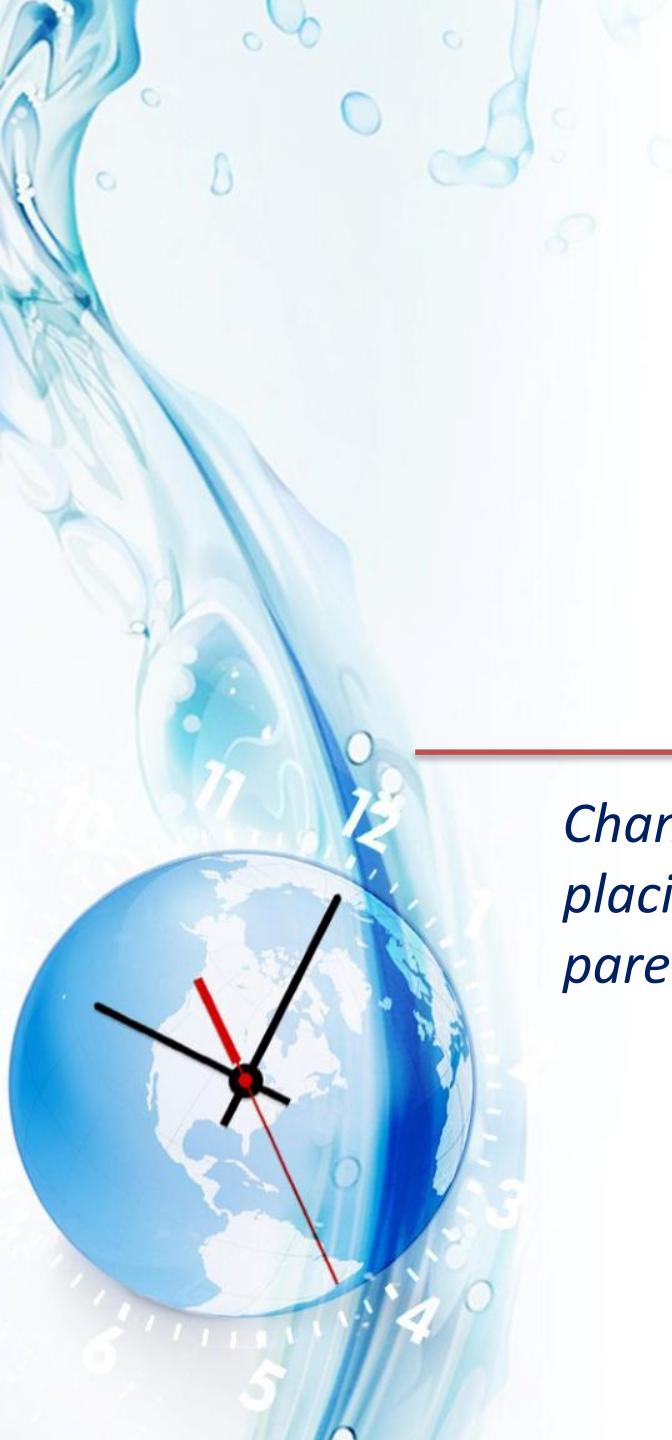
```
$y = " World!";
```

```
echo $x . $y; // Outputs: Hello World!
```

```
$x .= $y;
```

```
echo $x; // Outputs: Hello World!
```

?



PHP Converting Data Types



Changing the data type of a variable can occurs by placing the name of the desired data type in parentheses before the variable's name.

Example

```
<?php  
    $test= 8.23;  
    echo $test;                      // Displays "8.23"  
    echo (string)$test;              // Displays "8.23"  
    echo (int)$test;                 // Displays "8"  
    echo (float)$test;               // Displays "8.23"  
    echo (boolean)$test;             // Displays "1"  
    var_dump((array)$test);          // array (size=1) 0 => float 8.23  
    var_dump((object)$test)          // object(stdClass)[1] public  
                                    'scalar' => float 8.23  
?  
?
```

PHP Automatic Data Type Conversion

PHP automatically converts one type of variable to another whenever possible.

Example

```
<?php  
    $x = "12";           //String Variable  
    $y = 20;             //Integer Variable  
    $z = $x + $y;  
    var_dump($x);  
    var_dump($y);  
    var_dump($z);  
?>
```

```
string '12' (Length=2)  
int 20  
int 32
```

PHP \$\$ Variables

The PHP \$\$ Variables (reference variables) uses the value of a created variable as a new variable name and assign new value to it.

Example

```
<?php
```

`$x = "abc";` → creates var \$x and assigns value “abc” to it.

`$$x = 200;` → creates var \$abc and assigns value “200” to it.

`echo $x;` → prints “abc”.

`echo $$x;` → prints “200”.

`echo $abc;` → prints “200”.

```
?>
```

PHP Data Type Functions

There are many functions can be used to determine or check or change the type of a variable.

<i>Function Name</i>	<i>Description</i>	<i>Output</i>
<i>gettype(var)</i>	<i>Returns the variable type</i>	<i>Integer string Boolean double NULL</i>
<i>is_numeric(var)</i>	<i>Checks for number variable.</i>	<i>True or false</i>
<i>is_int(var)</i>	<i>Checks for Integer variable.</i>	<i>True or false</i>
<i>is_float(var)</i>	<i>Checks for Float variable.</i>	<i>True or false</i>
<i>is_string(var)</i>	<i>Checks for String variable.</i>	<i>True or false</i>
<i>is_bool(var)</i>	<i>Checks for Boolean variable.</i>	<i>True or false</i>
<i>is_null(var)</i>	<i>Checks for null variable.</i>	<i>True or false</i>

<i>Function Name</i>	<i>Description</i>	<i>Output</i>
<i>is_array(var)</i>	<i>Checks for array variable.</i>	<i>True or false</i>
<i>is_object(var)</i>	<i>Checks for object variable.</i>	<i>True or false</i>
<i>empty(var)</i>	<i>check whether a variable is empty or not.</i>	<i>True or false</i>
<i>isset(var)</i>	<i>check whether a variable is set or not.</i>	<i>True or false</i>
<i>settype(var,type)</i>	<i>set the type of a variable.</i>	<i>True or false</i>
<i>unset(var1 ,var2)</i>	<i>Destroys a given variable/s.</i>	--

gettype() function

The gettype() function is used to get the type of a variable.



Example 1

```
<?php  
    $a = "12";  
    echo gettext($a); ➔ STRING  
?>
```

Example 2

```
<?php  
    $a = 12;  
    echo gettext($a); ➔ INTEGER  
?>
```

Example 3

```
<?php  
    $a = 0.815;  
    echo gettext($a); ➔ DOUBLE  
?>
```

Example 4

```
<?php  
    $a = null;  
    echo gettext($a); ➔ NULL  
?>
```

Example 5

```
<?php  
    $a = true;  
    echo gettype($a); → BOOLEAN  
?>
```

Example 6

```
<?php  
    $a = array('A','B','C');  
    echo gettype($a); → array  
?>
```

is_numeric() function

The `is_numeric()` function is used to check whether a variable is numeric or not.

is_numeric() function

Example

```
<?php  
    $a = "12";  
    echo is_numeric($a); ➔ true  
?>
```

Example

```
<?php  
    $a = 12;  
    echo is_numeric($a); ➔ true  
?>
```

is_int() function

The `is_int()` function is used to test whether the type of the specified variable is an integer or not.

is_int() function

Example

```
$a = "12";  
echo is_int($a); ➔ false
```

Example

```
$a = 12;  
echo is_int($a); ➔ true
```

Example

```
$a = 12.5;  
echo is_int($a); ➔ false
```



is_float() function



The `is_float ()` function is used to find whether a variable is a float or not.

is_float() function

Example

```
$a = "12.5";  
echo is_float($a); ➔ false
```

Example

```
$a = 12.5;  
echo is_float($a); ➔ true
```

Example

```
$a = 12;  
echo is_float($a); ➔ false
```

is_string() function

The `is_string()` function is used to find whether a variable is a string or not.

is_string() function

Example

```
$a = "12.5";  
echo is_string($a); ➔ true
```

Example

```
$a = 12.5;  
echo is_string($a); ➔ false
```

Example

```
$a = "hello";  
echo is_string($a); ➔ true
```

is_bool() function

The `is_bool()` function is used to find whether a variable is an a boolean or not.

is_bool() function

Example

```
$a = true;  
echo is_bool($a); ➔ true
```

Example

```
$a = "true";  
echo is_bool($a); ➔ false
```

Example

```
$a = "";  
echo is_bool($a); ➔ false
```



is_null() function

The `is_null ()` function is used to test whether a variable is NULL or not.

is_null() function

Example

```
$a = null;  
echo is_null($a); ➔ true
```

Example

```
$a = "null";  
echo is_null($a); ➔ false
```

Example

```
$a = "";  
echo is_null($a); ➔ false
```

is_array() function

The `is_array()` function is used to find whether a variable is an array or not.

Example

```
<?php  
    $a = array('A','B','C');  
    echo is_array($a); ➔ true  
?>
```

Example

```
<?php  
    $a = "Hello";  
    echo is_array($a); ➔ false  
?>
```

is_object() function

The `is_object()` function is used to find whether a variable is an object or not.

Example

```
<?php  
  
class greeting{  
  
}  
  
$a = new greeting;  
  
echo is_object($a); ➔ true  
?>
```

empty() function

The empty() function is used to check whether a variable is empty or not.

empty() function

Example

```
$a = null;  
echo empty($a); → true
```

Example

```
$a = "";  
echo empty($a); → true
```

Example

```
<?php  
echo empty($a); → true  
?>
```

isset() function

The isset() function is used to check whether a variable is set or not.

Example

```
$a = null;  
echo isset($a); → false
```

Example

```
$a = "";  
echo isset($a); → true
```

Example

```
$a = "hello";  
echo isset($a); → true
```

settype() function

The settype() function is used to set (change) the type of a variable.

Syntax

```
settype(var, datatype);
```

Example

```
$a = "5";  
settype($a, "integer");  
var_dump($a);
```

```
int 5
```

Example

```
$a = "hello";  
settype($a, "integer");  
var_dump($a);
```

```
int 0
```

unset() function

The unset() function destroys a given variable or variables.

Syntax

```
unset(var1, var2);
```

Example

```
$a = "5";  
unset($a);  
var_dump($a); ➔ null
```

Example

```
unset($a,$b);
```

printf() Function

The printf() function outputs a formatted string to the screen (on screen only), it takes a string argument, followed by one or more additional arguments.

Syntax

```
printf(text,argument)
```

Example

```
$x=3.141593;
```

```
printf( "The actual value for PI is: %f <br>", $x );
```

```
printf( "The rounded value for PI is: %d", $x );
```

```
The actual value for PI is: 3.141593  
The rounded value for PI is: 3
```

Arguments

- **%d** → Positive integer; as decimal (base10).
- **%f** → float.
- **%o** → Integer; as octal (base8).
- **%s** → String.
- **%x** → Integer; hexadecimal with lowercase letters.
- **%X** → Integer; hexadecimal with uppercase letters.

Example 2

```
$x=255;
```

```
printf("The decimal value for $x is: %d <br>", $x );
```

```
printf("The Hex-decimal value is: %X <br>The octal value is: %o",  
$x , $x);
```

The decimal value for 255 is: 255
The Hex-decimal value is: FF
The octal value is: 377

Example 3

```
<?php  
    $x=255;  
    $y=printf("The hex value for is: %x", $x);  
    var_dump($y);  
?>
```

The hex value for is: ff

int 24

Note that:

- *The returned value from the function “printf()” to the variable “y” is the length of the generated string and not the string itself.*

sprintf() Function

The sprintf() function writes a formatted string to a variable (to a variable only), it takes a string argument, followed by one or more additional arguments.

Syntax

sprintf(text,argument)

Example

```
$x=3.141593;
```

```
$y=sprintf( "The actual value for PI is: %f", $x );
```

```
var_dump($y);
```

```
string 'The actual value for PI is: 3.141593' (Length=36)
```



PHP String Functions

PHP provides many built-in functions can used to manipulate strings.

strlen() function

The strlen() function takes a string and returns the number of characters in it.

Syntax

strlen(varname);

Example

```
<?php  
$a = "hello";  
echo strlen($a);  
?>
```

5



Substr() function



The substr() function is used to return certain parts of a string. It takes 3 parameters: the variable name, the starting index and the length. Ending index is optional, if not specified, the whole string will be printed out.

Syntax

```
substr(variablename, starting_index, length);
```

Example

```
$x = "Welcome to nti!";  
echo substr($x, 0, 7);
```

Welcome

Example

```
$x = "Welcome to nti!";  
echo substr($x, 3);
```

come to nti!



strtoupper() function

The “strtoupper()” converts the case of a string to uppercase. It takes one parameter: the string.

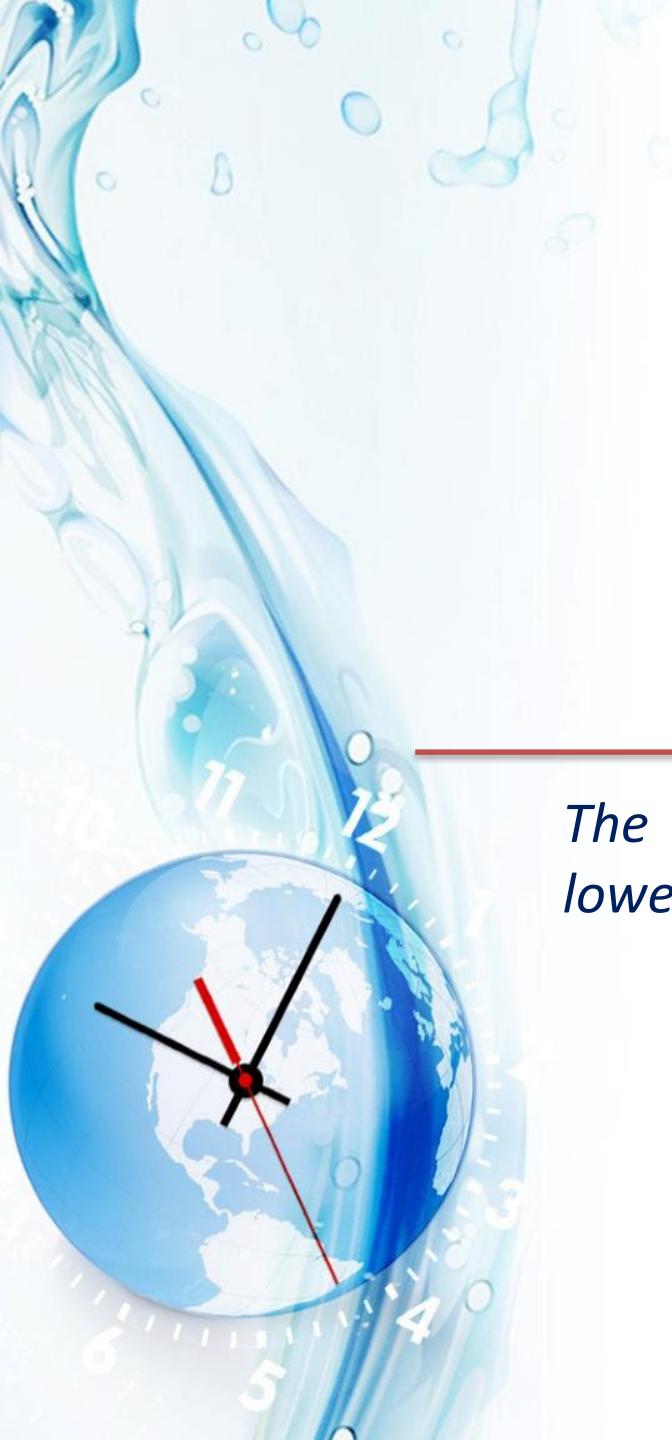
Syntax

strtoupper(variablename);

Example

```
<?php  
$x = "Welcome to nti!";  
echo strtoupper($x);  
?>
```

WELCOME TO NTI!



strtolower() function

The “strtolower()” converts the case of a string to lowercase. It takes one parameter: the string.

Syntax

```
strtolower(variablename);
```

Example

```
<?php  
$x = "WELCOME to NTI!";  
echo strtolower($x);  
?>
```

welcome to nti!

ucfirst() function

The “ucfirst()” function Converts the first character of a string to uppercase.

Syntax

```
ucfirst(variablename);
```

Example

```
<?php  
$x = "welcome to nti";  
echo ucfirst($x);  
?>
```

Welcome to nti

ucwords() function

The “ucwords()” function Converts the first character of each word of a string to uppercase.

Syntax

ucwords(variablename);

Example

```
<?php  
$x = "welcome to nti";  
echo ucwords($x);  
?>
```

Welcome To Nti

explode() function

“The explode()” function breaks a string into an array using a separator. (the separator cannot be an empty string).

Syntax

explode(separator,variablename);

Example

```
<?php  
    $x = "WELCOME to nti!";  
    $y= explode(" ",$x);  
    var_dump($y);  
?>
```

```
array (size=3)  
  0 => string 'WELCOME' (Length=7)  
  1 => string 'to' (Length=2)  
  2 => string 'nti!' (Length=4)
```

Itrim() function

The “Itrim()” function Removes whitespace or other predefined characters from the left side of a string

Syntax: removing whitespace only from the left side

Itrim(variablename);

Example

```
<?php  
    $x = "      welcome to nti!";  
    echo Itrim($x);  
?>
```

welcome to nti!

Syntax: removing predefined characters from the left side

Itrim(variablename, characters);

Example

```
<?php  
    $x = "welcome to nti!";  
    echo Itrim($x, "welcome ");  
?>
```

: to nti!

rtrim() function

The “rtrim()” function Removes whitespace or other predefined characters from the right side of a string

Syntax: removing whitespace only from the right side

rtrim(variablename);

Example

```
<?php  
    $x = "welcome to nti!      ";  
    echo rtrim($x);  
?>
```

welcome to nti!

Syntax: removing predefined characters from the right side

rtrim(variablename, characters);

Example

```
<?php  
    $x = "welcome to nti!";  
    echo rtrim($x, " nti!");  
?>
```

welcome to



trim() function

The “trim()” function Removes whitespace from both sides of a string.

Syntax

```
trim(variablename);
```

Example

```
<?php  
$x = "      NTI      ";  
echo trim($x);  
?>
```

NTI

md5() function

The “md5()” function returns the MD5 hash of a string.

Syntax

```
md5(variablename);
```

Example

```
<?php  
$x = "NTI";  
echo md5($x);  
?>
```

```
e3982ea40d45bb5891e7eb379abd223e
```

str_ireplace() function

The “*str_ireplace()*” function replaces some characters in a string with another characters. (case-insensitive)

Syntax

```
str_ireplace(char-find, char-replace, variablename);
```

Example

```
<?php  
$x = "welcome to NTI";  
$y=str_ireplace("nti","MCIT",$x);  
echo $y;  
?>
```

welcome to MCIT

str_replace() function

The “`str_ireplace()`” function replaces some characters in a string with another characters.
(case-sensitive)

Syntax

```
str_replace(char-find, char-replace, variablename);
```

Example

```
<?php  
$x = "welcome to nti";  
$y=str_replace("nti","MCIT",$x);  
echo $y;  
?>
```

welcome to MCIT

str_shuffle() function

The “str_shuffle()” function shuffles all characters of a string Randomly.

Syntax

```
str_shuffle(variablename);
```

Example

```
<?php  
$x = "welcome to nti";  
$y=str_shuffle($x);  
echo $y;  
?>
```

mteleo owcitr

str_split() function

The “str_split()” function splits a string into an array using a defined length.

Syntax

str_split(variablename, length);

Example

```
<?php  
    $x = "welcome to nti";  
    $y=str_split($x,5);  
    var_dump($y);  
?>
```

```
array (size=3)  
  0 => string 'welco' (Length=5)  
  1 => string 'me to' (Length=5)  
  2 => string ' nti' (Length=4)
```

str_word_count() function

The “`str_word_count()`” function counts the number of words *inside* a string.

str_word_count() function

Syntax

```
str_word_count(variablename);
```

Example

```
<?php  
    $x = "welcome to nti";  
    $y= str_word_count($x);  
    echo $y;  
?>
```

3

strip_tags() function

The “strip_tags()” function removes any html or php tags from the string

Syntax

```
strip_tags(variablename);
```

Example

```
<?php  
$x = "<p>welcome to nti!</p>";  
$y= strip_tags($x);  
echo $y;  
?>
```

welcome to nti!



strpos() function

The “strpos()” function searches for a string in another string and returns the first position.

*The “strpos()” function is case sensitive for case insensitive use “**stripos()**” function.*



Syntax

```
strpos(variablename, text);
```

Example

```
<?php  
$x = "welcome to nti, nti is a non-profit organization";  
$y= strpos($x,"nti");  
echo $y;  
?>
```

11



strrpos() function



The “strrpos()” function searches for a string in another string and returns the last position.

*The “strrpos()” function is case sensitive for case insensitive use “**stripos()**” function.*

Syntax

```
strrpos(variablename, text);
```

Example

```
<?php  
$x = "welcome to nti, nti is a non-profit organization";  
$y= strpos($x,"nti");  
echo $y;  
?>
```

16

strrev() function

The “strrev()” function reverses the order of the characters inside a string.

Syntax

```
strrev(variablename);
```

Example

```
<?php  
$x = "welcome to nti";  
$y= strrev($x);  
echo $y;  
?>
```

itn ot emoclew



PHP Math Functions

PHP has several built-in functions that help to perform anything from simple additions or subtraction to advanced calculations.



abs() function



The “abs()” function is used to get the absolute value of an integer or a float.

Syntax

```
abs(number);
```

Example

```
<?php  
    echo abs(5); // Outputs: 5 (integer)  
    echo abs(-5); // Outputs: 5 (integer)  
    echo abs(4.2); // Outputs: 4.2 (double/float)  
    echo abs(-4.2); // Outputs: 4.2 (double/float)  
    echo abs($x);  
?>
```

ceil() function

The ceil() function is used to round a fractional value up to the next highest integer value.



Syntax

```
ceil(number);
```

Example

```
<?php  
echo ceil(4.2); // Outputs: 5  
echo ceil(9.99); // Outputs: 10  
echo ceil(-5.18); // Outputs: -5  
?>
```

floor() function

The floor() function is used to round a fractional value down to the next lowest integer value.



Syntax

```
floor(number);
```

Example

```
<?php  
echo floor(4.2); // Outputs: 4  
echo floor(9.99); // Outputs: 9  
echo floor(-5.18); // Outputs: -6  
?>
```

sqrt() function

The sqrt() function is used to get the square root of a positive number. This function returns a special value NAN for negative numbers.

Syntax

```
sqrt(number);
```

Example

```
<?php  
    echo sqrt(9); // Outputs: 3  
    echo sqrt(25); // Outputs: 5  
    echo sqrt(10); // Outputs: 3.1622776601684  
    echo sqrt(-16); // Outputs: NAN  
?>
```



cos() and sin() and tan() functions



The “cos()”, “sin()” and “tan()” functions are used to returns the cosine and the sine and the tangent for numbers.

cos() and sin() and tan() functions

Syntax

```
cos(number);    sin(number);    tan(number);
```

Example

```
<?php  
$x=3;  
echo (cos($x)); // Outputs: -0.98999249660045  
echo (sin($x)); // Outputs: 0.14112000805987  
echo (tan($x)); // Outputs: -0.14254654307428  
?>
```



mod() function



The “mod()” function is used to find the remainder after division of one number by another (modulus).

Syntax

mod(num1,num2);

Example

```
<?php  
$x=9;  
$y=2;  
echo (mod($x,$y));      // Outputs: 1  
?>
```



rand() function

The “rand()” function is used to generate a random number (between 0 and 32767).

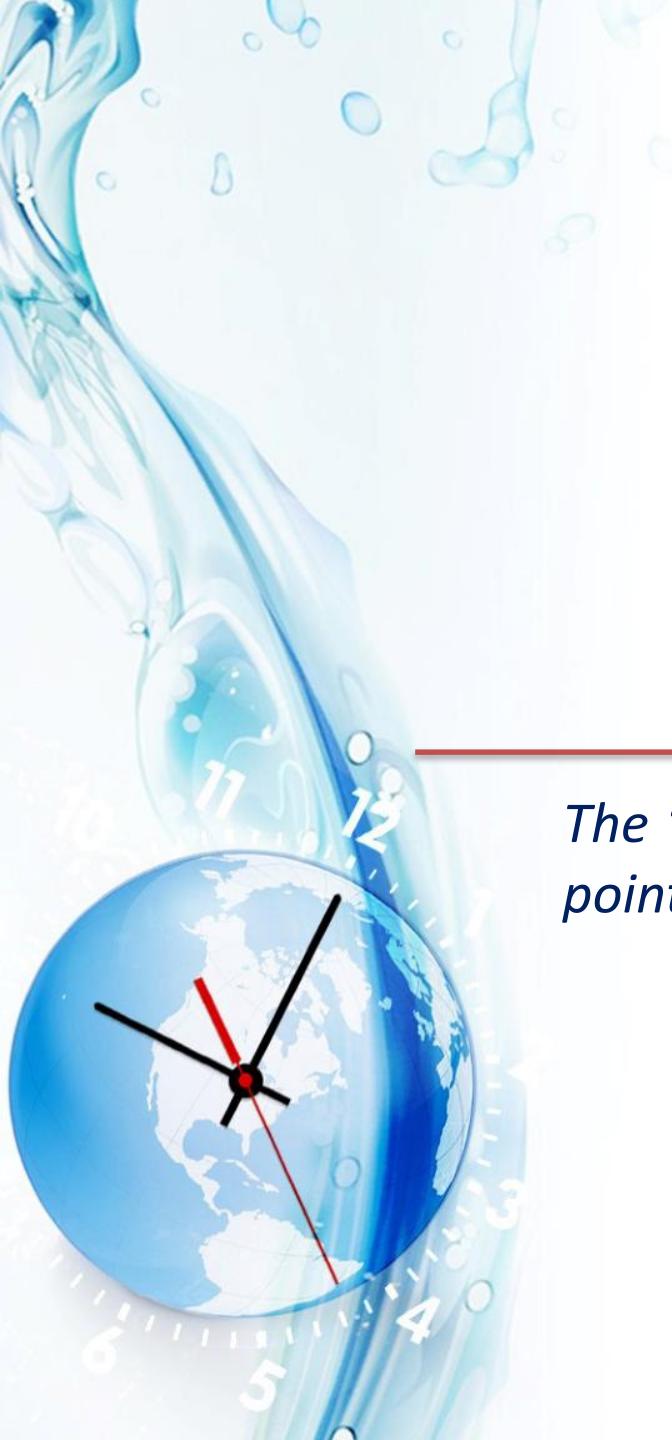
if a range larger than 32767 is required, min and max arguments can be added to the function.

Syntax

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

Example

```
<?php  
echo rand();           // Outputs: number between (0-32767)  
echo rand(1, 10);     // Outputs: number between (1-10)  
echo rand(100, 200);  // Outputs: number between (100-200)  
?>
```



round() function



The “round()” function is used to round a floating-point number.

Syntax

round(number);

Example

```
<?php  
$x=5.6;  
$y=5.1;  
echo(round($x)); // Outputs: 6  
echo(round($y)); // Outputs: 5  
?>
```



decbin() function

The “decbin()” function is used to convert a decimal number into binary number.

Syntax

```
decbin(decimal-number);
```

Example

```
<?php  
    echo decbin(2); // Outputs: 10  
    echo decbin(5); // Outputs: 101  
    echo decbin(99); // Outputs: 1100011  
?>
```



bindec() function

The “bindec()” function converts a number from binary to decimal.

Syntax

```
bindec(binary-number);
```

Example

```
<?php  
    echo bindec(10);    // Outputs: 2  
    echo bindec(101);   // Outputs: 5  
    echo bindec(1100011); // Outputs: 99  
?>
```



dechex() function



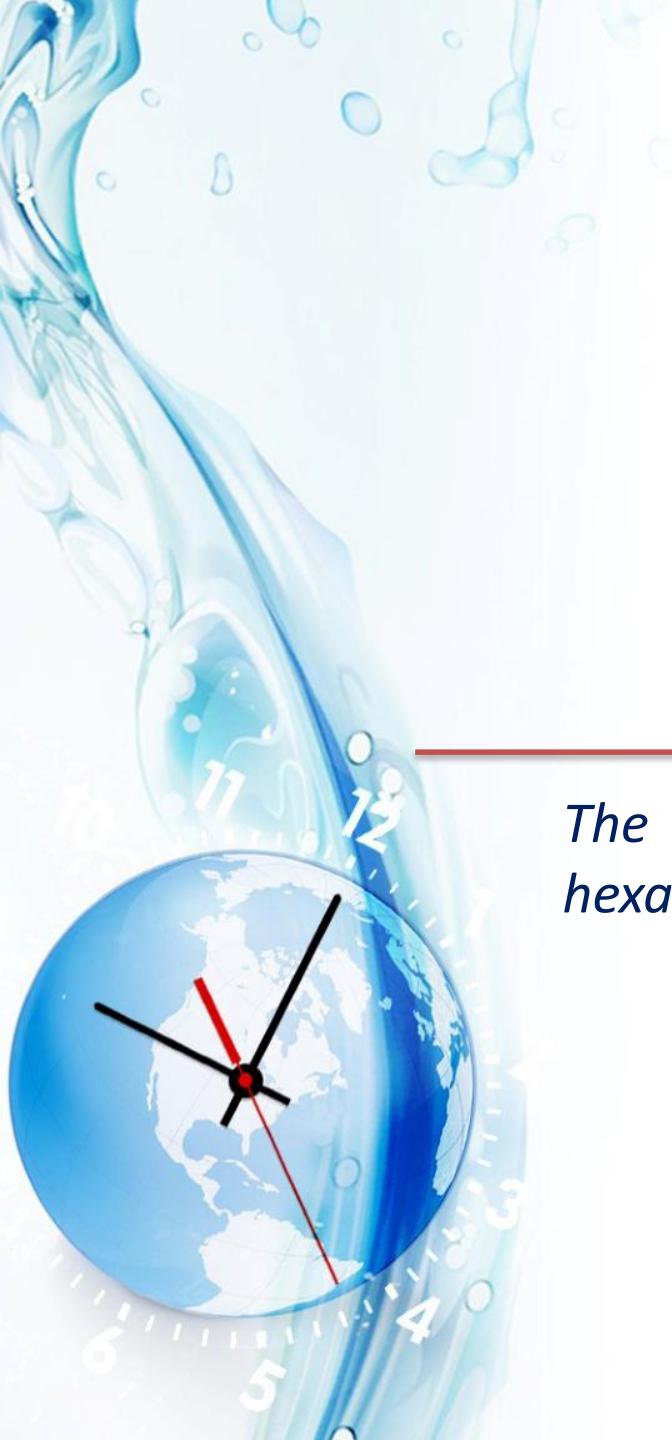
The “`dechex()`” function is used to convert a decimal number into hexadecimal number.

Syntax

```
dechex(decimal-number);
```

Example

```
<?php  
    echo dechex(255); // Outputs: ff  
    echo dechex(196); // Outputs: c4  
    echo dechex(0);  // Outputs: 0  
?>
```



hexdec() function



The “hexdec()” function is used to converts a hexadecimal number to a decimal number.

Syntax

hexdec(hex-number);

Example

```
<?php  
    echo hexdec(ff); // Outputs: 255  
    echo hexdec(c4); // Outputs: 196  
    echo hexdec(0);  // Outputs: 0  
?>
```



decoct() function



The “decoct()” function is used to convert a decimal number to octal number.

Syntax

```
decoct(decimal-number);
```

Example

```
<?php  
    echo decoct(12); // Outputs: 14  
    echo decoct(256); // Outputs: 400  
    echo decoct(77); // Outputs: 115  
?>
```



octdec() function



The “octdec()” function is used to convert an octal number to decimal number.

Syntax

```
octdec(octal-number);
```

Example

```
<?php  
    echo octdec(14); // Outputs: 12  
    echo octdec(400); // Outputs: 256  
    echo octdec(115); // Outputs: 77  
?>
```

base_convert() function

The “`base_convert()`” function can be used to convert a number from one base system to other.

Syntax

```
base_convert(number, frombase, tobase);
```

Example: (Binary ⇔ Decimal)

```
<?php  
    // Convert decimal to binary  
    echo base_convert('12', 10, 2); // Outputs: 1100  
    // Convert binary to decimal  
    echo base_convert('1100', 2, 10); // Outputs: 12  
?>
```

Example: (Decimal \leftrightarrow Hex)

```
<?php  
  
    // Convert decimal to hexadecimal  
  
    echo base_convert('10889592', 10, 16); // Outputs: a62978  
  
    // Convert hexadecimal to decimal  
  
    echo base_convert('a62978', 16, 10); // Outputs: 10889592  
  
?>
```

Example: (Decimal ⇔ Octal)

```
<?php  
    // Convert decimal to octal  
    echo base_convert('82', 10, 8); // Outputs: 122  
    // Convert octal to decimal  
    echo base_convert('122', 8, 10); // Outputs: 82  
?>
```

Example: (Hex ⇔ Binary)

```
<?php  
  
    // Convert hexadecimal to binary  
  
    echo base_convert('abc', 16, 2); // Outputs: 101010111100  
  
    // Convert binary to hexadecimal  
  
    echo base_convert('101010111100', 2, 16); // Outputs: abc  
  
?>
```



PHP Constants



A constant is a name or an identifier for a fixed value. Constant are like variables except that once they are defined, they cannot be undefined or changed (except magic constants).

Same as variables except:

- *The value of the constant is defined once and cannot be changed during the script.*
- *The constant name must start with letter or underscore.*
- *The \$ is not used with the constant name.*
- *Constants are global for all the code.*

PHP Constant Creating (Defining)

- To declare a constant, the “`define()`” syntax is used with three parameters:
 1. Name of the constant.
 2. Value of the constant.
 3. Whether the constant name should be case-insensitive or not (optional and the default is false).

Example

```
<?php  
    define(constant-name, value, case-sensitivity);  
?>
```

Example

```
<?php  
    define("x", "Welcome to NTI");    → case sensitive  
    echo x;  
?>
```

Example

```
<?php  
    define("x", "Welcome to NTI", true);    → case in-sensitive  
    echo x;  
?>
```

defined() function

The defined() function returns true if the constant exists, otherwise it returns false.

Syntax

defined(constant-name);

Example

```
<?php  
define('x1', 2018);  
echo defined('x1');    ➔ returns true  
echo defined('x2');    ➔ returns false  
?>
```



PHP Magic Constant

PHP provides a large number of predefined constants to any script named “Magic Constants”.

Returns the current line number of the file.

```
<?php  
    echo "The Line number : ". __LINE__;  
?>
```

The Line number : 2

Returns the full path and filename of the file.

```
<?php  
    echo "Your file name :". __FILE__;  
?>
```

Your file name :C:\wamp\www\test\index.php

Returns the current PHP version.

```
<?php  
echo "Current PHP Version is: ".PHP_VERSION;  
?>
```

Current PHP Version is: 5.5.12

PHP Magic Constant - PHP_INT_MAX

Returns the PHP integer value limit.

```
<?php  
echo "Integer Maximum Value: ".PHP_INT_MAX;  
?>
```

Integer Maximum Value : 2147483647

PHP Magic Constant - __FUNCTION__

Returns the function name as it was declared.

```
<?php  
function test() {  
    echo "Function name is: ". __FUNCTION__ ;  
}  
test();  
?>
```

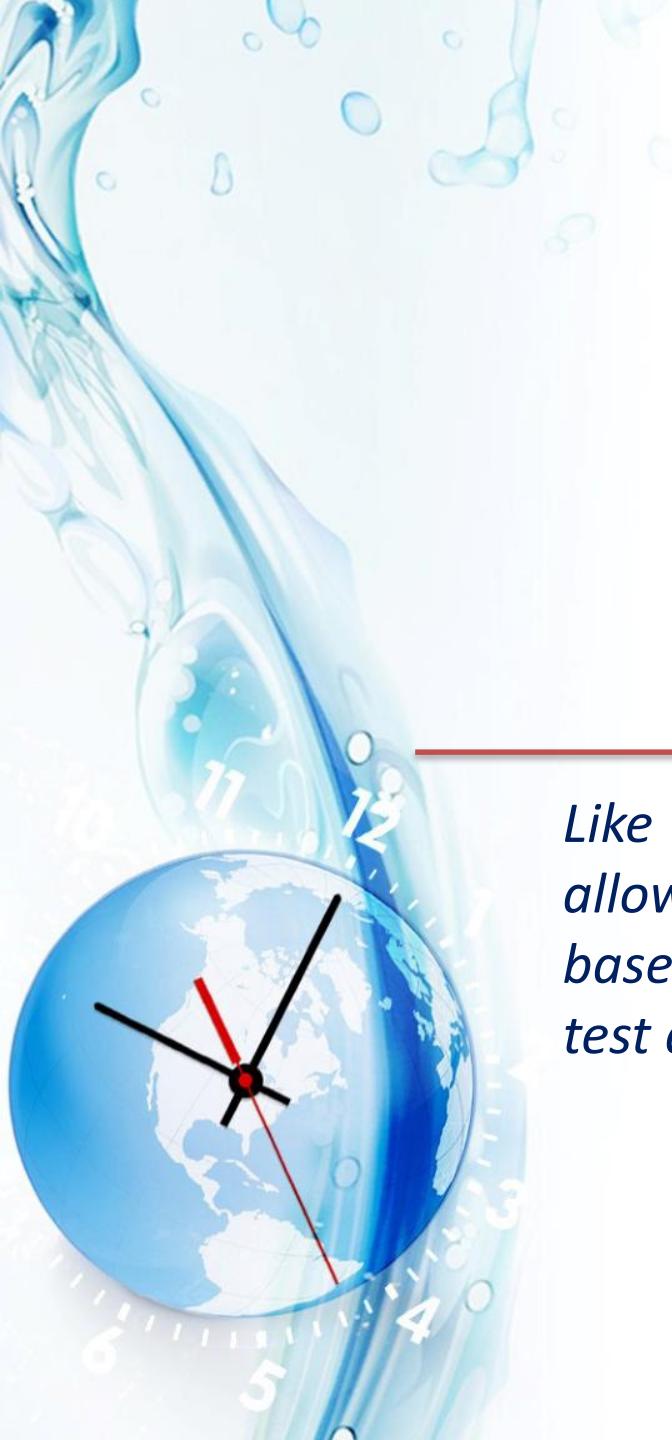
Function name is: test

PHP Magic Constant - __CLASS__

Returns the class name as it was declared.

```
<?php  
class demo{  
    function test(){  
        echo "class name: ". __CLASS__;  
    }  
}  
  
$object=new demo();  
$object->test();  
?>
```

class name: demo



Decision Statements



Like most programming languages, PHP also allows to write code that perform different actions based on the results of a logical or comparative test conditions at run time.



PHP if Statement



The if statement is used to execute a block of code only if the specified condition returns true. This is the simplest PHP's conditional statements.

Syntax

```
if(condition)
{
    // Code to be executed
}
```

Example

```
$x =150;
if($x >100)
{
    echo "number is larger than 100";
}
```

number is larger than 100



The if...else Statement



The if...else statement allows to execute one block of code if the specified condition returns true and another block of code if it is evaluates to false.

Syntax

```
if(condition)
```

```
{
```

```
// Code to be executed if condition is true
```

```
}
```

```
else
```

```
{
```

```
// Code to be executed if condition is false
```

```
}
```

Example

```
<?php  
$x=50;  
if($x>100)  
{  
    echo "number is larger than 100";  
}  
else  
{  
    echo "number is smaller than 100";  
}  
?>
```

number is smaller than 100



The if...elseif...else Statement



The if...elseif...else a special statement that is used to combine multiple if...else statements.

Syntax

```
if(condition1){  
    // Code to be executed if condition1 is true  
}  
elseif(condition2){  
    // Code to be executed if the condition1 is false and  
    // condition2 is true  
}  
else{  
    // Code to be executed if both conditions are false  
}
```

Syntax

```
$x=50;  
if($x>100){  
    echo "number is larger than 100";  
}  
elseif($x==100){  
    echo "number is equal 100";  
}  
else{  
    echo "number is smaller than 100";  
}
```

number is smaller than 100



The Switch...Case Statement



The switch-case statement tests a variable against a series of values until it finds a match, and then executes the block of code corresponding to that match.

Syntax

```
switch(variable){  
    case value1:  
        // Code to be executed if variable=value1  
        break;  
    case value2:  
        // Code to be executed if variable=value2  
        break;  
    ...  
    default:  
        // Code to be executed if variable is not equal to any  
        // previous value  
}
```

Example

```
$x="Saturday";  
switch($x){  
    case "Friday":  
        echo "Today is off";  
        break;  
    case "Saturday":  
        echo "Today is off";  
        break;  
    default:  
        echo "Today is working day";  
}
```

Today is off

The switch statement will be executed line by line, and will not stop even if the true case statement is found, all the next case statements will be executed till the end of the switch block.

To prevent this a break statement is added to the end of each case block.



PHP Loops



Loops are used to execute the same block of code several times, until a certain condition is met. loops are used to automate the repetitive tasks within a program to save the time and effort. PHP supports four different types of loops.



PHP for Loop

The for loop repeats a block of code until a certain condition is met. It is typically used to execute a block of code for certain number of times.

Syntax

```
for(initialization; condition; increment){  
    // Code to be executed  
}
```

initialization

- *it is used to initialize the counter variables, and evaluated once unconditionally before the first execution of the body of the loop.*

condition

- *in the beginning of each iteration, condition is evaluated. If it evaluates to true, the loop continues and the nested statements are executed. If it evaluates to false, the execution of the loop ends.*

increment

- *it updates the loop counter with a new value. It is evaluate at the end of each iteration.*

Example

```
<?php  
for($i=1; $i<=3; $i++){  
    echo "The number is $i <br>";  
}  
?>
```

The number is 1
The number is 2
The number is 3



PHP while Loop

The “while” loop executes the statement while a specified condition is true. the “while” loop first check the condition then execute the statement.

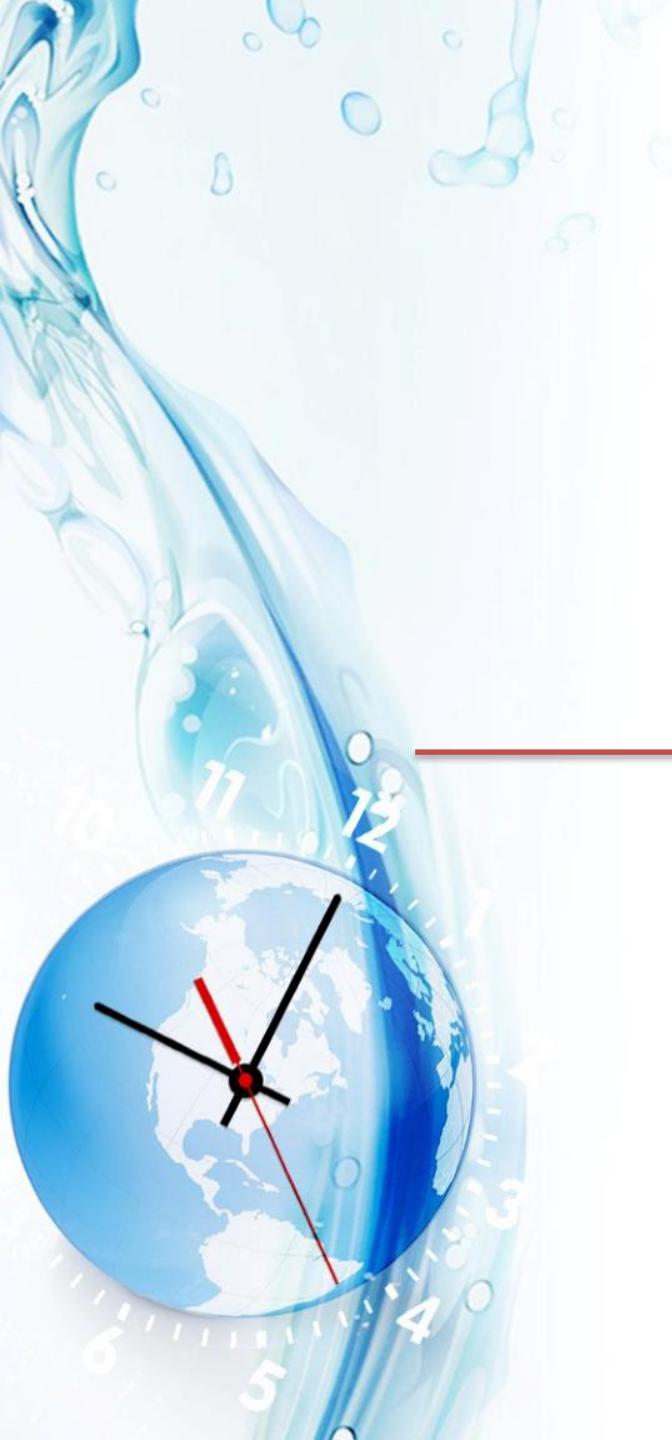
Syntax

```
<?php  
    while(condition){  
        // Code to be executed  
    }  
?>
```

Example

```
<?php  
    $i = 1;  
    while($i <= 3){  
        echo "The number is $i <br>";  
        $i++;  
    }  
?>
```

The number is 1
The number is 2
The number is 3



PHP do...while

The “do – while” loop first execute the statement then check the condition, it means if the condition is false the statement will be executed at least one time.

Syntax

```
<?php  
    do{  
        // Code to be executed  
    }  
    while(condition);  
?>
```

Example

```
<?php  
$i = 1;  
do{  
    echo "The number is $i <br>";  
    $i++;  
}  
while($i <= 3);  
?>
```

The number is 1
The number is 2
The number is 3

PHP foreach Loop

The foreach loop is used to iterate over arrays.

Syntax

```
<?php  
    foreach(array-name as variable-name){  
        // Code to be executed  
    }  
?>
```

Example

```
<?php  
    $colors = array("Red", "Green", "Blue");  
    $colors[10] = "black";  
  
    // Loop through colors array  
    foreach($colors as $x){  
        echo "$x <br>";  
    }  
?>
```

Red
Green
Blue
black

Example: printing index and values

```
<?php  
    $colors = array("Red", "Green", "Blue");  
    $colors[10] = "black";  
    // Loop through colors array  
    foreach($colors as $index =>$x){  
        echo "location $index Contains: $x<br>";  
    }  
?>
```

location 0 Contains: Red
location 1 Contains: Green
location 2 Contains: Blue
location 10 Contains: black



PHP Break

PHP break statement breaks the execution of current for, while, do-while, switch and for-each loop. If you use break inside inner loop, it breaks the execution of inner loop only.

Syntax

```
<?php  
    for($i=1;$i<=10;$i++){  
        echo "$i <br/>";  
        if($i==5){ break; }  
    }  
?>
```

1
2
3
4
5



PHP Functions

PHP also allows to define functions. It is a way to create reusable code packages that perform specific tasks and can be kept and maintained separately from main program.

Syntax

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

Example

```
<?php  
function hello() {  
    echo "hello world"; }  
// Calling function  
hello();  
?>
```



PHP Functions with Parameters



Functions that 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.

Syntax

```
function functionName($Parameter1, $Parameter2){  
    // Code to be executed  
}
```

Example

```
<?php  
function getsum($num1, $num2) {  
    echo $num1+$num2; }  
getsum(10,15);  
?>
```



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.

Functions with Optional Parameters and Default Values

Syntax

```
function functionName($Parameter1, $Parameter2=value){  
    // Code to be executed  
}
```

Example

```
<?php  
function getsum($num1, $num2=50) {  
    echo $num1+$num2; }  
getsum(10,15);  
getsum(10);  
?>
```

PHP Functions with Returned Values

A function can return a value back to the script that called the function using the return statement.

Example

```
<?php  
function getsum($num1, $num2) {  
    $total=$num1+$num2;  
    Return $total;  
}  
echo getsum(10,15);  
?>
```



PHP Variables Scope



The location of the declaration of a variable determines the 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.*
- *Similarly, if you try to access or import an outside variable inside the function, you'll get an undefined variable error.*



PHP The global Keyword



To access a variable from the main program inside a function, or vice versa the “global” keyword is used inside the function. This keyword turns the variable into a global variable, making it visible or accessible both inside and outside the function.

Example: Accessing Global Variable from function

```
<?php  
  
$x=50;  
  
function test(){  
  
    global $x; // allows the function to access the  
    // global variable x.  
  
    echo $x;  
  
}  
  
test();  
  
?>
```

Example: Creating Global Variable by function

```
<?php  
  
function test(){  
  
    global $x; // Creates a global variable x.  
  
    $x=50;  
  
}  
  
test();  
  
echo $x;  
  
?>
```



PHP The Static Variable



When a function is completed, all its variables are destroyed. But the static variable will not lose its value when the function exits and will still hold its value when the function is called again.

Syntax

```
<?php  
function test(){  
    static $x=50;  
    $x++;  
    echo $x;  
}  
  
test();      //prints 51  
test();      //prints 52  
test();      //prints 53  
?>
```

PHP Arrays

Arrays are complex variables that allow to store more than one value using single variable name.

There are three types of arrays in PHP:

➤ *Indexed array*

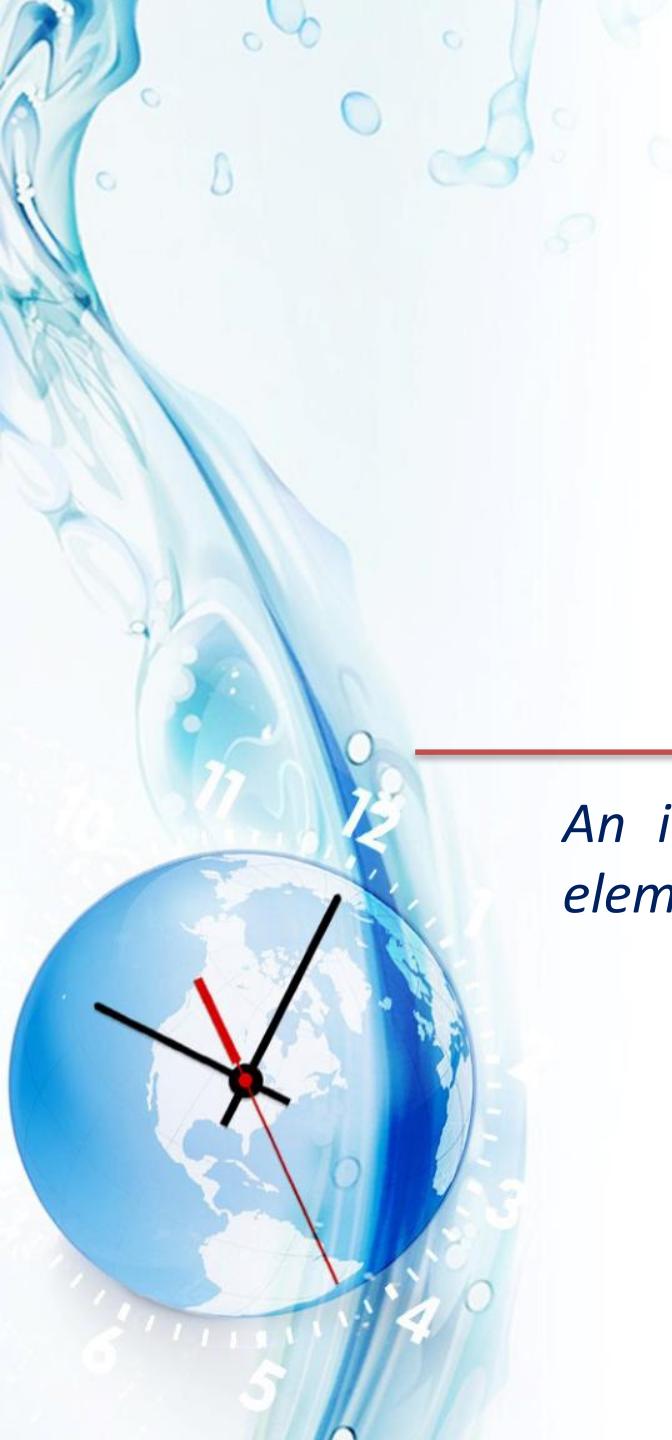
- ✓ An array with a numeric key.

➤ *Associative array*

- ✓ An array where each key has its own specific value.

➤ *Multidimensional array*

- ✓ An array containing one or more arrays.



PHP Indexed Arrays

An indexed or numeric array stores each array element with a numeric index.

Syntax

```
<?php  
    $array_name = array("val0", "val1", "val2");  
?>
```

Example

```
<?php  
    $colors = array("Red", "Green", "Blue");  
?>
```

Note: In an indexed array, the indexes are automatically assigned and start with 0, and the values can be any data type.

Syntax: Manual assigned indexes

```
<?php  
    $array_name[index] = "value";  
?>
```

Example

```
<?php  
    $colors[0] = "Red";  
    $colors[1] = "Green";  
    $colors[2] = "Blue";  
?>
```



Printing values of indexed array

Printing values of indexed array

Syntax

```
<?php  
    echo $array_name[index];  
?>
```

Example

```
<?php  
    $colors = array("Red", "Green", "Blue");  
    echo $colors[1];  
?>
```

Green

Getting the length of an array

The built in function “count()” is used to return the length of the array (the number of elements inside the array).

Getting the length of an array

Syntax

```
<?php  
    count($arrname);  
?>
```

Example

```
<?php  
    $colors = array("Red", "Green", "Blue");  
    count($colors);  
?>
```

3

Looping through Indexed Array

To loop through an indexed array, we can use the “for” loop or the “while” loop or the “foreach” loop to loop through and print the variables.

Looping through Indexed Array

Example: using for loop

```
<?php  
    $colors = array("Red", "Green", "Blue");  
    $len=count($colors);  
  
    for($i=0;$i<$len;$i++)  
    {  
        echo $colors[$i];  
        echo "<br>";  
    }  
?>
```

Red
Green
Blue

Looping through Indexed Array

Example: using while loop

```
<?php  
$colors = array("Red", "Green", "Blue");  
$len=count($colors);  
$i=0;  
while($i<$len)  
{  
    echo $colors[$i];  
    echo "<br>";  
    $i++;  
}  
?>
```

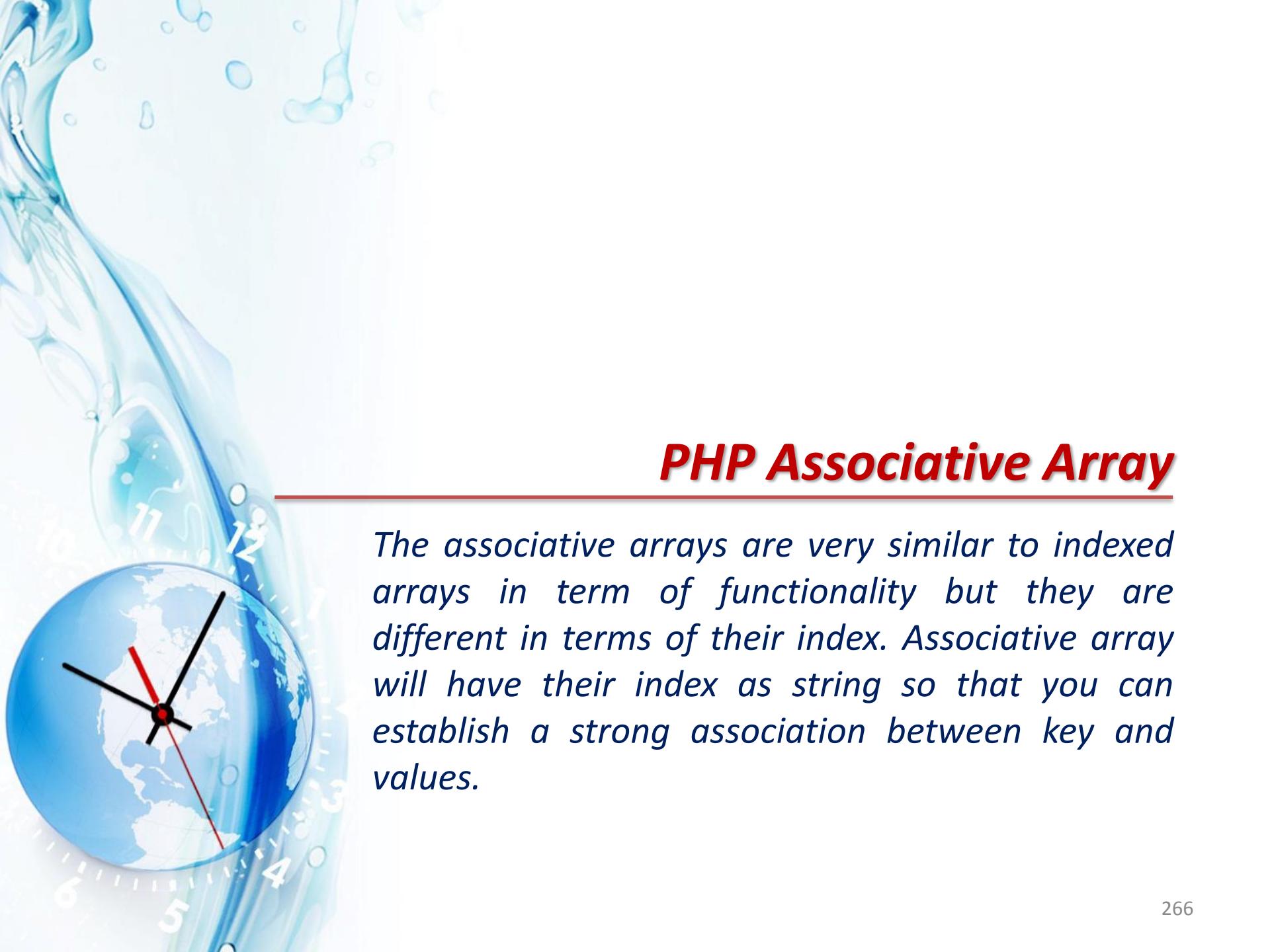
Red
Green
Blue

Looping through Indexed Array

Example: using *foreach* loop

```
<?php  
  
$colors = array("Red", "Green", "Blue");  
  
foreach($colors as $x){  
  
    echo "$x <br>";  
  
}  
  
?>
```

Red
Green
Blue



PHP Associative Array

The associative arrays are very similar to indexed arrays in term of functionality but they are different in terms of their index. Associative array will have their index as string so that you can establish a strong association between key and values.

Syntax

```
<?php  
$array_name = array("key0"=>val0, "key1"=>val1);  
?>
```

Example

```
<?php  
$ages = array("Peter"=>22, "Clark"=>32, "John"=>28);  
?>
```

Syntax 2

```
<?php  
    $array_name["key"] = "value";  
?>
```

Example

```
<?php  
    $ages["Peter"] = "22";  
    $ages["Clark"] = "32";  
    $ages["John"] = "28";  
?>
```

Printing Values of Associative Array

Printing Values of Associative Array

Syntax

```
<?php  
    echo $array_name["key"];  
?>
```

Example

```
<?php  
    $ages = array("Peter"=>22, "Clark"=>32, "John"=>28);  
    echo $ages["Peter"] ;  
?>
```

22

Looping through Associative Array

To loop through an Associative array, the “foreach” loop can be used to print out the keys and values.

Looping through Associative Array

Example: using foreach loop to print values only

```
<?php  
$ages = array("Peter"=>22, "Clark"=>32, "John"=>28);  
  
foreach($ages as $x){  
    echo "$x <br>";  
}  
?>
```

22
32
28

Example: using foreach loop to print keys and values only

```
<?php  
$ages = array("Peter"=>22, "Clark"=>32, "John"=>28);  
  
foreach($ages as $x=>$y){  
    echo "Name is $x and Age is $y <br>";  
}  
?>
```

Name is Peter and Age is 22
Name is Clark and Age is 32
Name is John and Age is 28



PHP Multidimensional Arrays

The multidimensional array is an array in which each element is also an array.

Example

```
<?php  
// Creating array  
$contacts = array(  
    array("name" => "Peter Parker", "age" => "30"),  
    array("name" => "Clark Kent", "age" =>"20"),  
    array("name" => "Harry Potter", "age" => "35")  
);  
// Access nested value  
echo $contacts[0]["name"] ." Age is: " . $contacts[0]["age"];  
?>
```

Peter Parker Age is: 30

print_r() function

The “`print_r()`” function is used to print all the variables and all the values inside the array (for testing purposes). (also “`var_dump()`” function can be used to provide more details.)

Syntax

```
print_r($arrayname);
```

Example

```
<?php  
  
$ages = array("Peter"=>22, "Clark"=>32, "John"=>28);  
  
print_r($ages);  
  
?>
```

```
Array ( [Peter] => 22 [Clark] => 32 [John] => 28 )
```



Array functions



implode() or join() function

*“The `implode()`” or the “`join()`” functions used to join an array “**values**” in a string using a specified joiner. (used with indexed and associative arrays only).*

implode() or join() function

Syntax

implode(joiner, \$arrayname);

Example

```
<?php  
  
$ages = array("Peter"=>22, "Clark"=>32, "John"=>28);  
  
$x=implode(" ",$ages);  
  
echo $x;  
  
?>
```

22 32 28



Sorting Arrays



Elements in an array can be sorted in alphabetical or numerical order, descending or ascending using one of the following built in functions.

sort() function

The “sort()” function sorts indexed arrays (only) in ascending order (according to its values).

Syntax

sort(\$arrname)

Example

```
<?php  
    $ages = array("Peter", "Clark", "John");  
    sort($ages);  
    var_dump($ages);  
?>
```

```
array (size=3)  
  0 => string 'Clark' (Length=5)  
  1 => string 'John' (Length=4)  
  2 => string 'Peter' (Length=5)
```



rsort() function

The “rsort()” function sorts indexed arrays (only) in descending order (according to its values).

Syntax

```
rsort($arrname)
```

Example

```
<?php  
    $ages = array("Peter", "Clark", "John");  
    rsort($ages);  
    var_dump($ages);  
?>
```

```
array (size=3)  
  0 => string 'Peter' (Length=5)  
  1 => string 'John' (Length=4)  
  2 => string 'Clark' (Length=5)
```

asort() function

The “asort()” function sorts associative arrays (only) in ascending order (according to its values).

Syntax

asort(\$arrname)

Example

```
<?php  
$ages = array("Peter"=>30, "Clark"=>20, "John"=>25);  
asort($ages);  
var_dump($ages);  
?>
```

```
array (size=3)  
  'Clark'  => int 20  
  'John'   => int 25  
  'Peter'  => int 30
```

ksort() function

The “ksort()” function sorts associative arrays (only) in ascending order (according to its keys).

Syntax

```
ksort($arrname)
```

Example

```
<?php  
$ages = array("Peter"=>30, "Clark"=>20, "John"=>25);  
ksort($ages);  
var_dump($ages);  
?>
```

```
array (size=3)  
  'Clark'  => int 20  
  'John'   => int 25  
  'Peter'  => int 30
```

arsort() function

The “arsort()” function sorts associative arrays (only) in descending order (according to its values).

Syntax

arsort(\$arrname)

Example

```
<?php  
$ages = array("Peter"=>30, "Clark"=>20, "John"=>25);  
arsort($ages);  
var_dump($ages);  
?>
```

```
array (size=3)  
  'Peter'  => int 30  
  'John'   => int 25  
  'Clark'  => int 20
```

krsort() function

The “krsort()” function sorts associative arrays (only) in descending order (according to its keys).

Syntax

krsort(\$arrname)

Example

```
<?php  
    $ages = array("Peter"=>30, "Clark"=>20, "John"=>25);  
    krsort($ages);  
    var_dump($ages);  
?>
```

```
array (size=3)  
  'Peter' => int 30  
  'John'  => int 25  
  'Clark' => int 20
```



list() function

The `list()` function is used to assign indexed array values to a list of variables in one operation.

Syntax

```
list($var1,$var2,$var3)=$arrayname
```

Example

```
<?php  
  
$names = array("Peter", "Clark", "John");  
  
list($x,$y,$z)=$names;  
  
var_dump($z);  
  
?>
```

string 'John' (Length=4)

array_combine() function

Create an associative array by combining two indexed arrays.

Syntax

array_combine(\$arr1, \$arr2)

Example

```
<?php  
    $names = array("Peter", "Clark", "John");  
    $ages = array("30", "20", "25");  
    $new_arr=array_combine($names, $ages);  
    var_dump($new_arr);  
?>
```

```
array (size=3)  
  'Peter' => string '30' (Length=2)  
  'Clark' => string '20' (Length=2)  
  'John' => string '25' (Length=2)
```

array_diff() function

“array_diff()” function search for the values (only) of an array in other arrays, and creates a new array with the missing elements.

Syntax

array_diff(\$arr1, \$arr2, \$arr3)

Example

```
<?php  
    $a1 = array("30", "20", "25");  
    $a2 = array("31", "25");  
    $a3 = array("10", "20");  
    $x=array_diff($a1,$a2,$a3);  
    var_dump($x);  
?>
```

array (size=2)
0 => string '30' (Length=2)

array_diff_key() function

“array_diff()” function search for the Keys (only) of an array in other arrays, and creates a new array with the missing elements.

Syntax

array_diff_key(\$arr1, \$arr2, \$arr3)

Example

```
<?php  
    $a1=array("Peter"=>"30","Ali"=>"25","Ahmed"=>"15");  
    $a2=array("Peter"=>"30","Ali"=>"20");  
    $x=array_diff_key($a1,$a2);  
    var_dump($x);  
?  
?
```

```
array (size=1)  
    'Ahmed' => string '15' (Length=2)
```

array_diff_assoc() function

*“array_diff()” function search for the **keys** and **values** of an array in other arrays, and creates a new array with the missing elements.*

Syntax

array_diff_assoc(\$arr1, \$arr2, \$arr3)

Example

```
<?php  
    $a1=array("Peter"=>"30","Ali"=>"25","Ahmed"=>"15");  
    $a2=array("Peter"=>"30","Ali"=>"20");  
    $x=array_diff_assoc($a1,$a2);  
    var_dump($x);  
?>
```

```
array (size=2)  
  'Ali' => string '25' (Length=2)  
  'Ahmed' => string '15' (Length=2)
```

array_flip() function

The “array_flip()” function flips-switches all keys with their associated values in an array and return the new elements in a new array.

Syntax

array_flip(\$arrname)

Example

```
<?php  
    $a1=array("Peter"=>"30","Ali"=>"25","Ahmed"=>"15");  
    $a2=array_flip($a1);  
    var_dump($a2);  
?>
```

```
array (size=3)  
  30 => string 'Peter' (Length=5)  
  25 => string 'Ali' (Length=3)  
  15 => string 'Ahmed' (Length=5)
```

array_reverse() function

PHP “array_reverse()” function creates a new array containing values of another array in reversed order.

Syntax

array_reverse(\$arrname)

Example

```
<?php  
    $a1=array("summer","winter","spring","autumn");  
    $a2=array_reverse($a1);  
    var_dump($a2);  
?>
```

```
array (size=4)  
  0 => string 'autumn' (Length=6)  
  1 => string 'spring' (Length=6)  
  2 => string 'winter' (Length=6)  
  3 => string 'summer' (Length=6)
```

in_array() function

PHP “in_array()” function Checks if a value exists in an array and returns true or false.

Syntax

in_array(string, \$arrayname)

Example

```
<?php
    $arr = array("ahmed", "ali", "nader");
    $x="ali";
    if(in_array($x, $arr)) {
        echo "value is found";
    };
?>
```

array_key_exists() function

PHP “array_key_exists()” Checks if the specified key exists in the array and returns true or false.

Syntax

array_key_exists(string,\$arrayname)

Example

```
<?php  
$ages=array("ahmed"=>"20","ali"=>"20");  
$x="ahmed";  
if (array_key_exists($x,$ages))  
{  
    echo "Key Found";  
}  
?>
```



PHP Date and Time



PHP Date() Function

The PHP date() function convert a timestamp to a more readable date and time.

- *The computer stores dates and times in a format called UNIX Timestamp.*
- *It measures time as a number of seconds since the beginning of the Unix epoch (January 1, 1970 00:00:00 GMT).*
- *PHP “date()” function is used to converts a timestamp to a format that is readable to humans.*

Syntax

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

<i>format</i>	<i>Required, specifies the format of returned date and time</i>
<i>timestamp</i>	<i>optional, if not included current date and time will be used.</i>

Example: returns today's date

```
$x = date("d/m/Y");  
echo $x;
```

01/01/2018

Example: returns a defined date

```
$x = date("d/m/Y", 9990505000);  
echo $x;
```

20/05/2014

These are some the date-related formatting characters that are commonly used in format string:

- **d** - Represent day of the month (01 to 31).
- **j** - Represent day of the month (1 to 31)
- **D** - Represent day of the week (Mon to Sun).
- **I** - Represent day of the week (Monday to Sunday).
- **S** - English suffix for the day (st, nd, rd or th).
- **m** - Represent month (01 or 12).
- **n** - Represent month (1 or 12).
- **M** - Represent month (Jan to Dec).
- **F** - Represent month (January through December).
- **y** - Represent year in two digits (08 or 14).
- **Y** - Represent year in four digits (2008 or 2014).

- *h* - Represent hour in 12-hour format (01 to 12).
- *H* - Represent hour in 24-hour format (00 to 23).
- *i* - Represent minutes (00 to 59).
- *s* - Represent Seconds (00 to 59).
- *a* - Represent lowercase (am or pm).
- *A* - Represent uppercase (AM or PM).
- *e* - Timezone identifier (Europe/Paris or Africa/Cairo).

Example

```
echo date("d/m/Y") . "<br>";  
echo date("d-m-Y") . "<br>";  
echo date("d.m.Y");
```

06/08/2018
06-08-2018
06.08.2018

Example

```
echo date("h:i:s") . "<br>";  
echo date("F d, Y h:i:s A") . "<br>";  
echo date("h:i a");
```

10:22:19
August 06, 2018 10:22:19 PM
10:22 pm



PHP time() Function

The time() function is used to get the current time as a Unix timestamp (the number of seconds since the beginning of the Unix epoch: January 1 1970 00:00:00 GMT).

Example

```
<?php  
$x = time();  
echo($x);  
?>
```

1533587359

PHP mktime() Function

The mktime() function returns the Unix timestamp for a given date.



Syntax

mktme(hour,minute,second,month,day,year)

Example

```
<?php  
$a1= mktme(10,50,30,11,1,1979);  
echo $a1;  
echo "<br>";  
echo date("d/m/Y",$a1);  
?>
```

310297830
01/11/1979



Changing PHP Time Zone

The PHP host server is in another country or set up for a different time zone, this means that you need to update the PHP server time zone to match your time.

Syntax

```
date_default_timezone_set(timezone);
```

Example: Changing time zone to Cairo Egypt

```
<?php  
    date_default_timezone_set("Africa/Cairo");  
?>
```

Example: to show the current server time zone

```
<?php  
    echo date("e");  
?>
```

Europe/Paris



PHP Superglobals

Superglobals

- *PHP Superglobals are predefined variables that are always accessible from any function, regardless of the scope, without having to do anything special.*
- *The PHP superglobal variables are:*
 - ✓ *\$GLOBALS*
 - ✓ *\$_FILES*
 - ✓ *\$_SERVER*
 - ✓ *\$_COOKIE*
 - ✓ *\$_REQUEST*
 - ✓ *\$_SESSION*
 - ✓ *\$_POST*
 - ✓ *\$_GET*

\$GLOBALS

PHP stores all global variables in an associative array called “\$GLOBALS”. The array index (key) represents the name of the variable.

Example

```
<?php  
    $a = 100;  
    $b = 200;  
    function Myfun() {  
        echo $GLOBALS['a'];  
        $GLOBALS['c']=$GLOBALS['b']+50;  
    }  
    Myfun();  
    echo "<br>".$c;  
?>
```

100
250

\$_SERVER

`$_SERVER` is an associative array created by the web server, it contains information about headers, paths, and script locations.

`$_SERVER['PHP_SELF']`

Returns the filename the currently executing page.

Ex ➔ /nti/index.php

`$_SERVER['SERVER_NAME']`

Returns Hosting Server Name.

Ex ➔ www.nti.sci.eg

`$_SERVER['SCRIPT_FILENAME']`

Returns the absolute pathname of the current file.

Ex ➔ C:/wamp/www/nti/index.php

`$_SERVER['SERVER_PORT']`

Returns the hosting service port number.

Ex ➔ 80

\$_SERVER['SERVER_SIGNATURE']

Returns the server version and virtual host name.

*Ex → Apache/2.4.9 (Win64) PHP/5.5.12 Server at
www.nti.sci.eg Port 80*

\$_SERVER['REQUEST_METHOD']

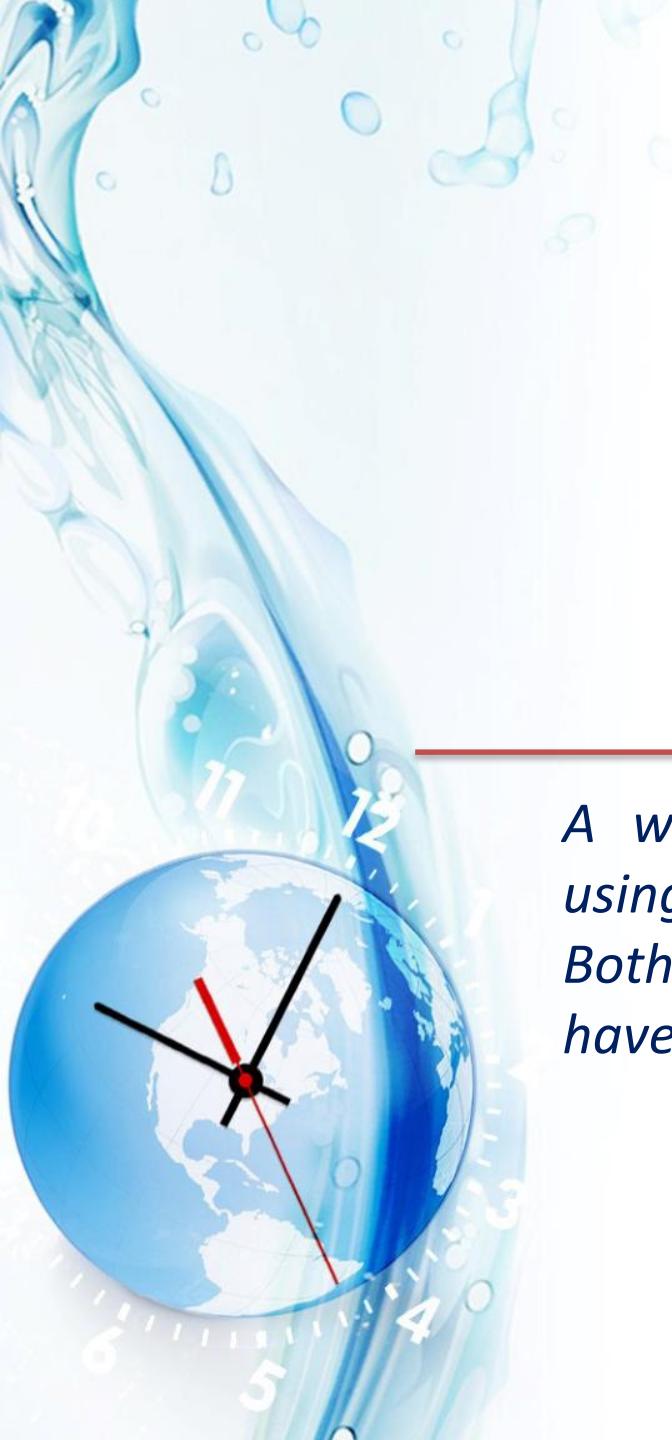
Returns the request method used to access the page.

Ex → POST

\$_SERVER['REMOTE_ADDR']

Returns the IP address from where the user is accessing the current page.

Ex → 10.90.10.5



PHP GET and POST

A web browser communicates with the server using one of the two HTTP methods, GET or POST. Both methods pass the information differently and have different advantages and disadvantages.

The GET Method

Using “GET” method the data is sent as *URL parameters* that are usually strings of *name* and *value pairs* separated by ampersands (&). In general, a URL with GET data will look like this:

http://www.nti.sci.eg/data.php?id=100&user=ahmed

➤ ?id=100&user=Ahmed

- ✓ Sends parameters using “GET” method
- ✓ Id ➔ variable name.
- ✓ 100 ➔ value for the first variable
- ✓ & ➔ concatenates a new variable to the string.
- ✓ user ➔ second variable name
- ✓ Ahmed ➔ value for the second variable

Advantages and Disadvantages of Using the GET Method

- Since the data sent by the GET method are displayed in the URL, it is possible to bookmark the page with specific query string values.
- The GET method is not suitable for passing sensitive information such as the username and password, because these are fully visible in the URL query string as well as potentially stored in the client browser's memory as a visited page.
- Because the GET method assigns data to a server environment variable, the length of the URL is limited. So, there is a limitation for the total data to be sent.

\$_GET

PHP provides the superglobal variable `$_GET` to access all the information sent either through the URL or submitted through an HTML form using the `method="get"`.

The form in HTML Page

```
<body><form method="get" action="data.php">
    <label>id:</label><input type="text" name="id">
    <label>user:</label><input type="text" name="user">
    <input type="submit" value="Submit">
</form></body>
```

Data.php

```
<?php
    echo "<p>Hi, " .  $\$_GET["user"]$  . "</p>";
    echo "<p>your ID is " .  $\$_GET["id"]$  . "</p>";
?>
```

The POST Method

Using POST method the data is sent to the server as a package in a separate communication. The data will not visible in the URL, i.e. remains as is:

<http://www.nti.sci.eg/data.php>

Advantages and Disadvantages of Using the POST Method

- *It is more secure than GET because user-entered information is never visible in the URL query string.*
- *There is a much larger limit on the amount of data that can be passed, text data and binary data (uploading a file) is sent using POST.*
- *Since the data sent by the POST method is not visible in the URL, so it is not possible to bookmark the page with specific query.*

\$_POST

*PHP provide another superglobal variable ***\$_POST*** to access all the information sent via post method or submitted through an HTML form using the ***method="post"***.*

The form in HTML Page

```
<body><form method="post" action="data.php">
    <label>id:</label><input type="text" name="id">
    <label>user:</label><input type="text" name="user">
    <input type="submit" value="Submit">
</form></body>
```

Data.php

```
<?php
    echo "<p>Hi, " .  $\$_POST["user"]$  . "</p>";
    echo "<p>your ID is " .  $\$_POST["id"]$  . "</p>";
?>
```

\$_REQUEST

PHP provides another superglobal variable `$_REQUEST` that contains the values of both the `$_GET` and `$_POST`.

The form in HTML Page

```
<body><form method="post" action="data.php">
    <label>id:</label><input type="text" name="id">
    <label>user:</label><input type="text" name="user">
    <input type="submit" value="Submit">
</form></body>
```

Data.php

```
<?php
    echo "<p>Hi, " . $_REQUEST["user"] . "</p>";
    echo "<p>your ID is " . $_REQUEST["id"] . "</p>";
?>
```

The form in HTML Page

```
<body><form method="get" action="data.php">
    <label>id:</label><input type="text" name="id">
    <label>user:</label><input type="text" name="user">
    <input type="submit" value="Submit">
</form></body>
```

Data.php

```
<?php
    echo "<p>Hi, " .  $\$_REQUEST["user"]$  . "</p>";
    echo "<p>your ID is " .  $\$_REQUEST["id"]$  . "</p>";
?>
```



PHP Cookies

A cookie is a small text file that lets you store a small amount of data (nearly 4KB) on the user's computer. Each time the browser requests a page to the server, all the data in the cookie is automatically sent to the server within the request.

To set a cookie in PHP The “setcookie()” function is used. Make sure you call the setcookie() function before any output generated by your script otherwise cookie will not set.

Syntax

```
setcookie(name, value, expire, path, domain, secure);
```

<i>name</i>	<i>The given name of the cookie</i>
<i>value</i>	<i>The given value of the cookie.</i>
<i>expires</i>	<i>The expiry date in UNIX timestamp format. (default is 0 i.e. the cookie will expire cookie will expire at the end of the session)</i>
<i>Path</i>	<i>Specify the path on the server for which the cookie will be available. (default is /, i.e. available to all the domain.</i>
<i>Domain</i>	<i>Specify the domain for which the cookie is available to e.g www.example.com.</i>
<i>secure</i>	<i>This field, if present, indicates that the cookie should be sent only if a secure HTTPS connection exists.</i>

The PHP “\$_COOKIE” superglobal variable is used to retrieve a cookie value. It typically an associative array that contains a list of all the cookies values sent by the browser in the current request, keyed by cookie name.

Example: Setting a Cookie

```
setcookie("uname", "ahmed", time() + 30 * 24 * 60 * 60);
```

Example: Accessing a Cookie

```
echo $_COOKIE["uname"];
```

You can delete a cookie by calling the same `setcookie()` function with the cookie name and any value (such as an empty string) however this time you need to set the expiration date in the past.

Example: Removing a Cookie

```
setcookie("uname", "", time()-3600);
```



PHP Sessions



PHP Session is an alternative way to make the data accessible across the various pages of an entire website.

Why using Session rather than Cookies?

Cookies have some security issues.

- *Cookies are stored on user's computer it is possible for an attacker to easily modify a cookie.*

Cookies automatically sent to the server.

- *Five cookies are stored on user's system, each having 4KB in size, the browser needs to upload 20KB of data each time the user views a page, which can affect the site's performance.*

How session based environment works?

- *PHP stores data on the server rather than user's computer.*
- *Every user is identified by a unique number called session identifier or SID.*
- *This unique session ID is used to link each user with his own information on the server.*
- *The PHP creates a file in a temporary directory on the server where registered session variables and their values are stored.*

How session based environment works?

- *This data will be available to all pages on the site during that visit.*
- *The location of the temporary file is determined by a setting in the “php.ini” file called “session.save_path”.*
- *The php.ini file is located inside the apache server folder: “C:\wamp\bin\apache\apache2.4.9\bin”*



Starting a PHP Session

Before storing any information in session variables, session must be started. The PHP function “session_start()” is used to create a new session and generate a unique session ID for the user.

Starting a new PHP Session

```
<?php  
    // Starting session  
    session_start();  
?>
```

Note: The `session_start()` function first checks to see if a session already exists by looking for the presence of a session ID, if the session is already started, it sets up the session variables and if doesn't, it starts a new session by creating a new session ID.

Turning on Auto Session Start

If you don't want to call the PHP "start_session()" function on every page you can turn on the auto session start by settin "session.auto_start" variable to "1" in the "php.ini" file.

The “\$_SESSION” superglobal array is used to store all the session’s data as key-value pairs. The stored data can be accessed during lifetime of a session.

Example: Storing Session Data

```
<?php  
    session_start();  
  
    $_SESSION["username"] = "Ali";  
    $_SESSION["password"] = "12345";  
?>
```

Example: Accessing Session Data

```
<?php  
    session_start();  
  
    echo 'Welcome, ' . $_SESSION["username"] . ' your  
    current password is ' . $_SESSION["password"];  
?>
```

Welcome, Ali your current password is 12345

Example: Getting Session Array Structure

```
<?php  
    session_start();  
    var_dump($_SESSION);  
?>
```

```
array (size=2)  
  'username' => string 'Ali' (Length=3)  
  'password' => int 12345
```



Destroying a PHP Session

To remove certain single session data, unset the corresponding key of the `$_SESSION` array. But to destroy all the session data completely, simply call the “`session_destroy()`” function.

Example: remove certain single session data

```
<?php  
    session_start();  
    unset($_SESSION["password"]);  
?>
```

Example: Enhanced code

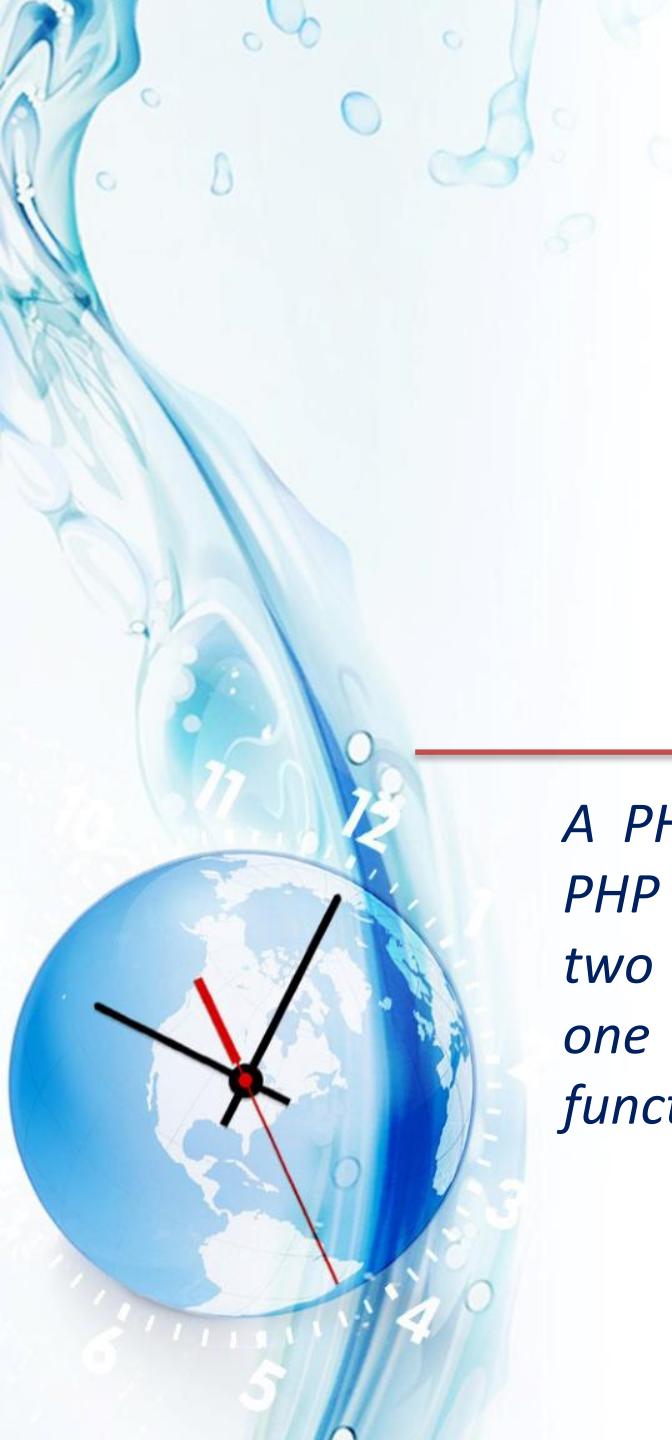
```
<?php  
    session_start();  
    if(isset($_SESSION["password"])){  
        unset($_SESSION["password"]);  
    }  
?>
```

Example: To destroy all the session data completely

```
<?php  
    // Starting session  
    session_start();  
  
    // Destroying session  
    session_destroy();  
?>
```

Note that:

- Every PHP session has a timeout value (duration).
- It is measured in seconds
- It determines how long a session should remain alive in the absence of any user activity.
- To change the timeout duration:
 - ✓ Open the “php.ini” file.
 - ✓ Change the value of “session.gc_maxlifetime” variable to the required time out value (in seconds).



PHP File Inclusion



A PHP file content can be included into another PHP file before the server executes it. There are two PHP functions which can be used to include one PHP file into another PHP file; the “include()” function and the require() function;

why Inclusion?

- *This is a strong point of PHP which helps in creating functions, headers, footers, or elements that can be reused on multiple pages.*
- *This will help developers to make it easy to change the layout of complete website with minimal effort. If there is any change required then instead of changing thousand of files just change included file.*

The include() and the require() Functions:

- Both functions take all the text in a specified file and copies it into the file that uses the include function.
- If there is any problem in loading the included file:
 - ✓ The “include()” function: generates a warning but the script will continue execution.
 - ✓ The require() function: generates a fatal error and halt the execution of the script.

Example: include()

```
<html>  
  
<body>  
  
    <?php include("menu.php"); ?>  
  
    <p>This is an example to show how to include PHP  
    file!</p>  
  
</body>  
  
</html>
```

Example: require()

```
<html>  
  
<body>  
  
    <?php require("menu.php"); ?>  
  
    <p>This is an example to show how to include PHP  
    file!</p>  
  
</body>  
  
</html>
```

The include_once and require_once Statements

- PHP provides *include_once* and *require_once* statements. These statements behave in the same way as *include* and *require* statements with one exception.
- The *include_once* and *require_once* statements will only include the file once even if asked to include it a second time.

exit() and die() functions

The exit() or die() functions are used to print a message and exits the current program.

Example

```
<?php
session_start();
if (isset($_SESSION['user'])){
    echo "Access Granted";
}
else {
    exit("<h1>Access Denied</h1>");
    echo "hello"; // this line will never runs
}
?>
```



PHP File System

PHP allows you to work with files and directories stored on the web server.

Getting the file size

The “filesize()” function returns the size of the file in bytes.

Syntax

filesize(file_name)

Example

```
<?php  
$x=filesize("data.txt");  
echo $x;  
?>
```



Getting file info

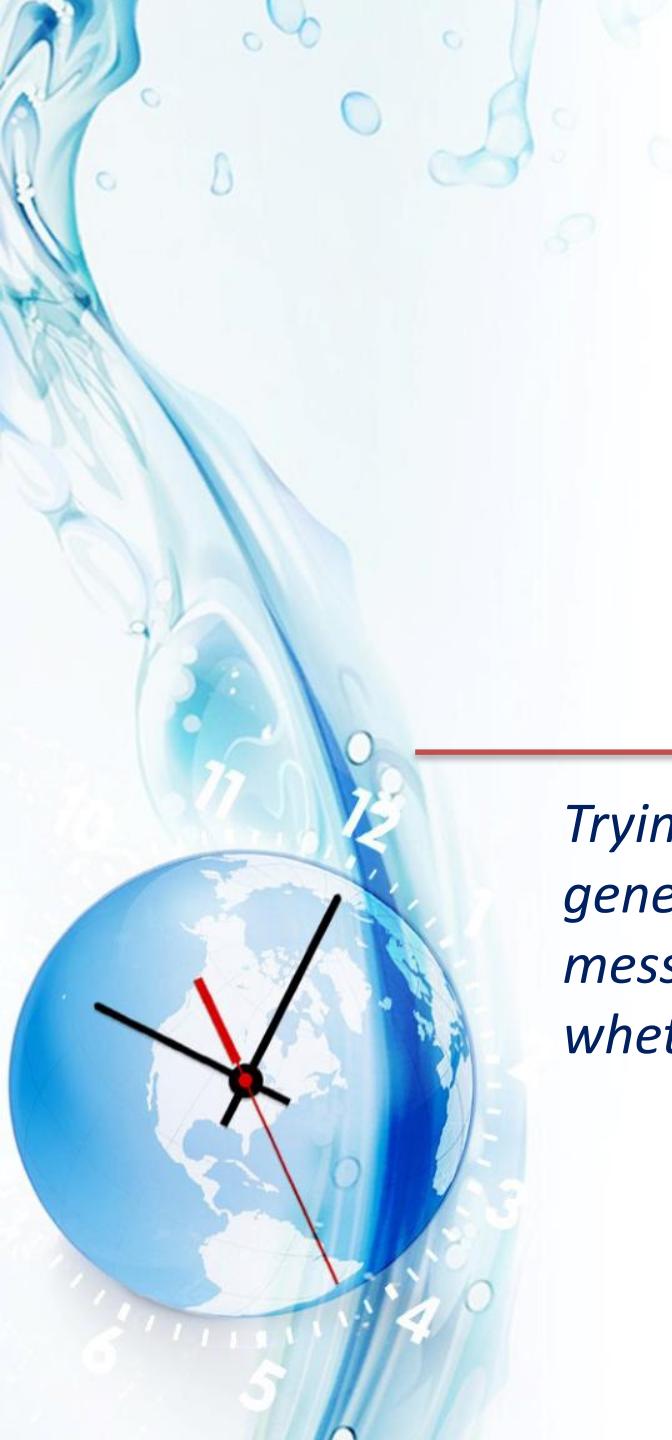
PHP uses the “pathinfo()” function to get other information about the selected file such as the directory name, the complete file name (name and ext) and the file extension.

Syntax

pathinfo(file_name)

Example

```
<?php  
$x=pathinfo("data.txt"); // returned values in $x array  
echo $x['dirname'];      // prints the directory name  
echo $x['basename'];     // prints data.txt  
echo $x['filename'];     // prints data  
echo $x['extension'];    // prints txt  
?>
```



Checking for File Existence



Trying open a file that doesn't exist, PHP will generate a warning message. To avoid these error messages use the "file_exists()" function to check whether a file or directory exists or not.

Opening file

```
<?php  
if(file_exists("data.txt"))  
{  
    $conn = fopen("data.txt", "r");  
} else {  
    echo "File does not exist.";  
}  
?>
```



Opening a File with PHP

*To work with a file you first need to open the file.
The PHP “fopen()” function is used to open a file.*

Syntax

```
fopen(filename, mode);
```

- *filename*: specifies the name of the file to be opened.
- *mode*: The file may be opened in one of 8 modes.

File opening modes:

- **r** ➔ *Opens the file for reading only. (file must be exist and contain data).*
- **r+** ➔ *Opens a file for reading and writing. (file must be exist and contain data).*
- **w** ➔ *Opens the file for writing only and clears the contents of file. If the file does not exist, PHP will attempt to create it.*
- **w+** ➔ *Open the file for reading and writing and clears the contents of file. If the file does not exist, PHP will attempt to create it.*

File opening modes:

- **a** ➔ Append. Opens the file for writing only. Preserves file content by writing to the end of the file. If the file does not exist, PHP will attempt to create it.
- **a+** ➔ Read/Append. Opens the file for reading and writing. Preserves file content by writing to the end of the file. If the file does not exist, PHP will attempt to create it.
- **x** ➔ creates new file for writing only. generates an error if the file already exists.
- **x+** ➔ creates new file for reading and writing. generates an error if the file already exists.

Example

```
$conn=fopen("data.txt", "a");
```



Closing a File with PHP

Once you've finished working with a file, it needs to be closed. The "fclose()" function is used to close the file.

Syntax

```
fclose($file_resource)
```

Example

```
$conn=fopen("data.txt", "r") or die("Cannot open file.");  
.....  
fclose($conn);
```

Reading Files Data

The `fread()` function is used to read a specified number of characters from a file.

Syntax

fread(file_resource, file_length_inbytes)

Example

```
$conn=fopen("data.txt", "r") or die("Cannot open file.");  
  
$content = fread($conn, "20");  
  
fclose($conn);  
  
echo $content;
```

Using `file_size()` with `fread()` to read the whole file data

```
<?php  
$size= filesize("data.txt");  
$conn=fopen("data.txt", "r") or die("Cannot open file.");  
$content = fread($conn, $size);  
fclose($conn);  
echo $content;  
?>
```



PHP readfile() function

Using the “readfile()” function is the easiest way to read the entire contents of a file in PHP. This function accepts the name of a file, and prints the entire file directly to the browser.

Syntax

```
readfile(file_name)
```

Example

```
<?php  
    readfile("data.txt") or die("Cannot open the file.");  
?>
```

PHP file_get_contents() function

The “file_get_contents()” function accepts the name and path to a file, and reads the entire file into a string variable.

PHP file_get_contents function

Syntax

```
file_get_contents(file_name)
```

Example

```
<?php  
$x=file_get_contents("data.txt") or die("Cannot open");  
echo $x;  
?>
```



PHP file() function

The PHP “file()” function does a similar job to file_get_contents() function, but it returns the file contents as an array of lines, rather than a single string. Each element of the returned array corresponds to a line in the file

Syntax

file(file_name)

Example

```
<?php  
$arr = file("data.txt") or die("Cannot open the file.");  
foreach($array1 as $x){  
    echo $x. "<br>";  
}  
?>
```



Writing Data to Files

The `fwrite()` function is used to write or append data to a file.

Syntax

```
fwrite(file_resource, string)
```

Example

```
<?php  
$conn = fopen("data.txt", "w") or die("Cannot open file.");  
fwrite($conn, "MCIT \r\nNTI") or die ("Cannot write the file.");  
fclose($conn);  
?>
```

MCIT
NTI

Note: if the “Data.txt” file doesn’t exist PHP will automatically create it and write the data. But, if the file already exist, PHP will erase the contents of this file before writing the new data.

file_put_contents() function

“file_put_contents()” function provides an easy method of writing the data to a file without needing to open it. This function accepts the name of the file and the data to be written to the file.

file_put_contents()

Syntax

```
file_put_contents(file_resource, string)
```

Example

```
<?php  
    file_put_contents("data.txt", "MCIT \r\nNTI")  
    or die ("Cannot write the file.");  
?>
```

MCIT
NTI

Note: if the “Data.txt” file already exists, PHP will overwrite it by default.

Example: using file_put_contents() to append data to a file

```
<?php  
    file_put_contents("data.txt", "MCIT \r\nNTI", FILE_APPEND)  
    or die ("Cannot write the file.");  
?  
12
```

Note: To preserve the file's contents a third parameter named “FILE_APPEND” is added to the function as to append the new data to the file instead of overwriting it.



PHP Renaming Files

Files and directories can be renamed using the PHP “rename()” function.

Syntax

```
rename(old_name, new_name);
```

Example

```
<?php  
    rename("data.txt", "data_2.txt"); // renames a file.  
    rename("win", "windows"); // renames a directory.  
    rename("data.txt", " data1\data_2.txt");  
        // renames and moves the file to the directory "data1".  
    rename("data1", "data2\data1");  
        // renames and moves the file to the directory "data2".  
?>
```

PHP Deleting Files

Files and directories can be deleted using the PHP “unlink()” function.

Syntax

```
unlink(file_name);
```

Example

```
<?php  
    unlink("data.txt"); // deletes a file.  
    unlink("win"); // deletes a directory.  
?>
```

Working with Directories

PHP also allows to work with directories on the file system, you can open a directory, create or delete a directory, list all files in the directory, and so on...

Creating a New Directory

PHP uses the “mkdir()” function to create a new empty directory. “it returns true if the directory is created successfully.”

Syntax

```
mkdir(dir_name);
```

Example

```
<?php  
    mkdir("hello"); // line1  
    mkdir("win/upload"); // line2  
?>
```

Line 1: creates folder named hello.

Line 2: creates folder named upload inside win folder “win folder must be pre-exists before running this code”.

Example: Check the existence of directory before creating

```
<?php  
  
if(!file_exists("hello")){  
  
    mkdir("hello");  
  
    echo "Directory created.";  
  
} else {  
  
    echo "Directory already exists.";  
  
}  
  
?>
```

Copying Files

PHP uses the “copy()” function to copy a file from one location to another. “it returns true if the file is copied successfully.”

Syntax

```
copy($source, $destination);
```

Example

```
<?php  
    copy("data.txt", "win/data.txt");  
?>
```

Example

```
<?php  
  
if(copy("data.txt", "win/data.txt"))  
  
{  
  
    echo "File copied successfully.";  
  
} else {  
  
    echo "File copying fail.";  
  
}  
  
?>
```

Listing All Files in a Directory

PHP uses the “scandir()” function to list files and directories inside the specified path.

Syntax

```
scandir($directory);
```

Example

```
<?php  
  
$list=scandir("win");           // create array of listed files  
  
foreach($list as $x){  
  
    echo $x . "<br>";      //print each file name  
  
}  
  
?>
```

PHP File Upload

PHP can upload any kind of files like images, videos, ZIP files, Microsoft Office documents, PDFs, as well as executables files and a wide range of other file types.

Superglobal `$_FILES`

Once the form is submitted information about the uploaded file can be accessed via PHP superglobal array called `$_FILES`.

Syntax

```
$_FILES["attributename"]["property"];
```

Example

```
<?php  
echo $_FILES["file1"]["name"] // prints filename.ext  
echo $_FILES["file1"]["type"] // prints MIME type  
echo $_FILES["file1"]["tmp_name"] // prints temp name  
echo $_FILES["file1"]["error"] // prints error no if found  
                                "0" if no errors  
echo $_FILES["file1"]["size"] // prints file size in bytes  
?> → var_dump($_FILES["file1"]); // can be used
```

Step 1: Creating an HTML form to upload the file

```
<form action="upload.php" method="post"  
      enctype="multipart/form-data">  
  
  <h2>Upload File</h2>  
  
  <label>Filename:</label>  
  
  <input type="file" name="file1">  
  
  <input type="submit" value="Upload">  
  
</form>
```

Step 2: Creating upload.php

```
<?php  
  
move_uploaded_file($_FILES["file1"]["tmp_name"], "upload/" .  
$_FILES["file1"]["name"]);  
  
echo "file uploaded successfully";  
  
?>
```

header() function

header() function is used to send a raw HTTP header. The header() function must be called before any actual output is sent, either by normal HTML tags, blank lines in a file, or from PHP.

Example

```
header('Location: http://www.yahoo.com/');
```

This `header()` function is used to redirect the browser to “`yahoo.com`”.

Example

```
<?php  
  
header('Content-Type: application/pdf');  
  
header('Content-Disposition: attachment; filename="123.pdf"');  
  
readfile('original.pdf');  
  
?>
```

This header() function is used to enforce the browser to download a file instead of opening it.

Example: link to download file

```
$f=$_REQUEST["file_name"];
$file = ("path/$f");
$filetype=filetype($file);
$filename=basename($file);
header ("Content-Type: ".$filetype);
header ("Content-Length: ".filesize($file));
header ("Content-Disposition: attachment;
          filename=".$filename);
readfile($file);
```



PHP MySQLi

Connecting to MySQL DB Server

PHP uses the “`mysqli_connect()`” function to connect to MySQL Database Server. All communication between PHP and the MySQL database server will take place through this connection.

Connecting to MySQL Database Server

Syntax

```
mysqli_connect("hostname", "username", "pass", "db", "port");
```

Example

```
<?php  
$conn = mysqli_connect("localhost", "root", "pass", "nti");  
if(!$conn){  
    die("Connection Fails! Could not connect to DB Server.");  
} else {  
    echo "Connected to Database Server";  
}  
?>
```

Example

```
<?php  
$conn = mysqli_connect("localhost", "root", "pass");  
mysqli_select_db($conn,"nti")  
?>
```

Note: The database name parameter in the “`mysqli_connect()`” function is optional and can be added at another line using the `mysqli_select_db()` function.

```
<?php  
    $conn = mysqli_connect("localhost", "root", "pass", "nti");  
  
    echo mysqli_errno($conn);  
        // prints the error number "0" if no errors.  
  
    echo mysqli_get_host_info($conn);  
        // prints MySQL server hostname and the connection type  
  
    echo mysqli_get_server_info($conn);  
        // prints the MySQL server version  
?>
```

Closing the MySQL Server Connection

The connection to the MySQL database server will be closed automatically as soon as the execution of the script ends. However, to enforce closing it earlier the “mysqli_close()” function is used.

Closing the MySQL Server Connection

Syntax

```
mysqli_close($connname);
```

Example

```
<?php  
$conn = mysqli_connect("localhost", "root", "pass", "nti");  
if(!$conn){  
    die("Connection Fails! Could not connect to DB Server.");  
} else {  
    echo "Connected to Database Server";  
}  
mysqli_close($conn);  
?>
```

Running MySQL Queries

PHP uses the “`mysqli_query()`” function to run a MySQL query on MySQL database server.

Syntax

```
mysqli_query($connname,sql_statement);
```

Example: create new database

```
<?php  
  
$conn = mysqli_connect("localhost", "root", "pass");  
  
mysqli_query($conn, "CREATE DATABASE nti");  
  
mysqli_close($conn);  
  
?>
```

Example: create new table

```
<?php  
    $conn = mysqli_connect("localhost", "root", "pass", "nti");  
    $sql="CREATE TABLE persons(  
        id INT NOT NULL PRIMARY KEY AUTO_INCREMENT,  
        first_name VARCHAR(30) NOT NULL,  
        last_name VARCHAR(30) NOT NULL,  
        email VARCHAR(70) NOT NULL UNIQUE  
    );  
    mysqli_query($conn, $sql);  
    mysqli_close($conn);  
?>
```

Example: Inserting data into table

```
<?php  
  
$conn = mysqli_connect("localhost", "root", "pass", "nti");  
  
$sql="INSERT INTO persons (first_name, last_name,  
email) VALUES ('Ahmed', 'Ali', 'a.ali@yahoo.com')";  
  
mysqli_query($conn, $sql);  
  
mysqli_close($conn);  
  
?>
```

Example: Inserting data into table using HTML form

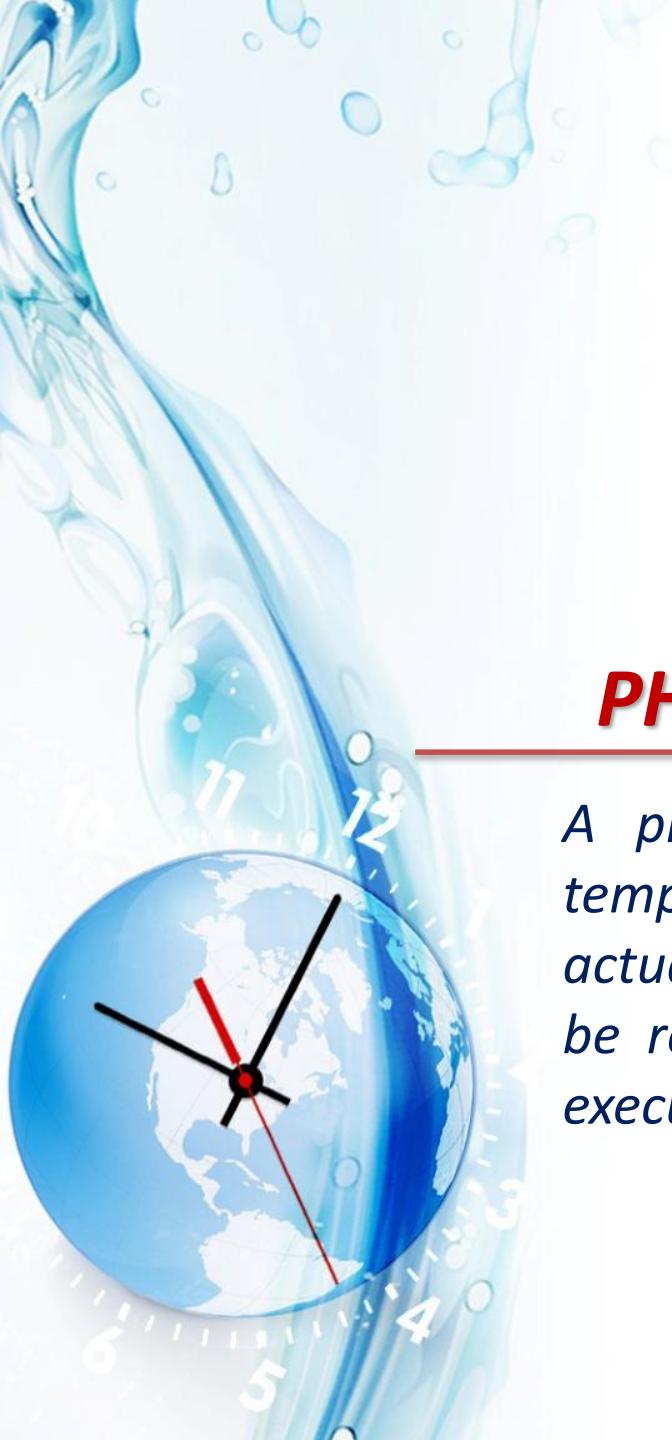
```
<?php  
$conn = mysqli_connect("localhost", "root", "pass","nti");  
$fname = $_POST['first_name'];  
$lname = $_POST['last_name'];  
$email = $_POST['email'];  
$sql="INSERT INTO persons (first_name, last_name, email)  
VALUES ('$fname', '$lname', '$email')";  
mysqli_query($conn, $sql);  
mysqli_close($conn);  
?>
```

mysqli_real_escape_string() function

PHP uses the “*mysqli_real_escape_string()*” function to escape special characters in a string for an SQL statement.

Example: `mysqli_real_escape_string()` function

```
<?php  
  
$conn = mysqli_connect("localhost", "root", "pass","nti");  
  
$fname = mysqli_real_escape_string($conn, $_POST['first_name']);  
  
$lname = mysqli_real_escape_string($conn, $_POST['last_name']);  
  
$email = mysqli_real_escape_string($conn, $_POST['email']);  
  
$sql="INSERT INTO persons (first_name, last_name, email) VALUES  
('$fname', '$lname', '$email');  
  
mysqli_query($conn, $sql);  
  
mysqli_close($conn);  
  
?>
```



PHP MySQL Prepared Statements

A prepared statement is simply a SQL query template containing placeholders instead of the actual parameters' values. These placeholders will be replaced by the actual values at the time of execution of the statement.

The prepared statement execution consists of two stages:

Prepare Stage:

- *SQL statement template is created and sent to the database server.*
- *The server performs a syntax check and query optimization, and stores it for later use.*

Execute Stage:

- *The values are sent to the server.*
- *The server creates a statement from the template and execute it with these values.*

Advantages of Using Prepared Statements

- *statement can be executed multiple times with different values, as it is prepared once, and executed multiple times.*
- *Minimize bandwidth usage, as every time the needed values only are transmitted to the server not the whole SQL command.*
- *Provides a strong protection against SQL injection, as values are not inserted directly into the SQL string, but sent separately to the server using a different protocol.*

Example: Insert New Records to Database

```
<?php
$conn = mysqli_connect("localhost", "root", "pass", "nti");
$sql = "INSERT INTO persons(first_name, last_name, email)
        VALUES (?, ?, ?)";
$stmt = mysqli_prepare($conn, $sql);
mysqli_stmt_bind_param($stmt, "sss", $fname, $lname, $email);
$fname = $_POST['first_name'];
$lname = $_POST['last_name'];
$email = $_POST['email'];
mysqli_stmt_execute($stmt);
mysqli_stmt_close($stmt);
mysqli_close($conn);
?>
```

Code Explanation

```
$sql = "INSERT INTO persons (first_name, last_name, email)  
VALUES (?, ?, ?);"
```

The question marks (?, ?, ?) is used as the placeholders for the first_name, last_name, email fields values that will be send by the form.

Code Explanation

```
mysqli_stmt_bind_param($stmt, "sss", $fname, $lname, $email);
```

- The “`mysqli_stmt_bind_param()`” function binds the variables to the placeholders `(?, ?, ?)` in the SQL statement template.

Code Explanation

```
mysqli_stmt_bind_param($stmt, "sss", $fname, $lname, $email);
```

The second argument "sss" specifies the data type of each bind variable is string. It could be one of theses characters:

- *b: binary (such as image, PDF file, etc.).*
- *d: double (floating point number).*
- *i: integer.*
- *s: string.*

Getting the ID of the Inserted Row

MySQL automatically generates an ID for the AUTO_INCREMENT column, to get this ID the “`mysqli_insert_id()`” function is used before closing the connection.

Example: Insert New Records to Database

```
$conn = mysqli_connect("localhost", "root", "pass", "nti");
$sql = "INSERT INTO persons(first_name, last_name, email)
        VALUES (?, ?, ?)";
$stmt = mysqli_prepare($conn, $sql);
mysqli_stmt_bind_param($stmt, "sss", $fname, $lname, $email);
$fname = $_POST['first_name'];
$lname = $_POST['last_name'];
$email = $_POST['email'];
mysqli_stmt_execute($stmt);
echo mysqli_insert_id($conn);
mysqli_stmt_close($stmt);
mysqli_close($conn);
```

Retrieving Data From Database

PHP also uses the “`mysqli_query()`” function to Retrieve Data From Database server and store it into a variable.

Example: Retrieve Data From Database

```
<?php  
$conn = mysqli_connect("localhost", "root", "pass", "nti");  
$sql="SELECT * FROM persons";  
$result=mysqli_query($conn, $sql);  
mysqli_close($conn);  
?>
```

The PHP “`mysqli_query()`” function executes the SQL query and returns the data into the “`result`” variable.

Counting Returned Rows

PHP uses “`mysqli_num_rows()`” function to return the number of rows returned by the query.

Example

```
<?php  
  
$conn = mysqli_connect("localhost", "root", "pass", "nti");  
  
$sql="SELECT * FROM persons";  
  
$result=mysqli_query($conn, $sql);  
  
echo mysqli_num_rows($result);  
  
mysqli_close($conn);  
  
?>
```

Retrieving Single Data Row

PHP uses “`mysqli_fetch_array()`” function to retrieve one row at a time, each time the function is invoked, it returns the next row from the result set as an array.

Example: looping through the result

```
<?php  
$conn = mysqli_connect("localhost", "root", "pass", "nti");  
$sql="SELECT * FROM persons";  
$result=mysqli_query($conn, $sql);  
for($x=0;$x<mysqli_num_rows($result);$x++){  
    $rows=mysqli_fetch_array($result);  
    echo $rows['id']."-".$rows['first_name']."<br>";  
}  
mysqli_free_result($result);  
mysqli_close($conn);  
?>
```

Code Explanation

```
for($x=0;$x<mysql_num_rows($result);$x++){  
    $rows=mysql_fetch_array($result);  
    echo $rows['id']."-".$rows['first_name']."<br>";  
}
```

The *for loop* is used to loop around the *mysql_fetch_array()* function the same number of times as the total number of records, each time the “*mysql_fetch_array*” runs returns the next record to the “\$row” array.



Sending Emails using PHP

*PHP uses “**mail()**” function for creating and sending email messages to one or more recipients dynamically from PHP application either in a plain text form or formatted HTML.*

Syntax

```
mail(to, subject, message, headers, parameters)
```

to: The recipient's email address.

subject: Subject of the email to be sent.

Message: Defines the message to be sent. each line should be separated with a line feed-LF (\n). Lines should not exceed 70 characters.

headers: Used to add extra headers such as "From", "Cc", "Bcc". The additional headers should be separated with a carriage return plus a line feed-CRLF (\r\n). (optional)

parameters: Used to pass additional parameters. (optional)



Sending Plain Text Emails

The simplest way to send an email with PHP.

Example

```
<?php  
    $to = 'ahmed@gmail.com';  
    $subject = 'Visit Egypt';  
    $message = 'Hi Ahmed, come and visit Egypt';  
    $from = 'Adel@hotmail.com';  
    if(mail($to, $subject, $message)){  
        echo 'Mail sent successfully.';  
    } else{  
        echo 'Unable to send email. Please try again.';  
    }  
?>
```

Sending HTML Formatted Emails

We're going to improve that output, and make the email into a HTML-formatted email.

Example

```
<?php
$to = 'maryjane@email.com';
$subject = 'Marriage Proposal';
$from = 'peterparker@email.com';
$headers = 'MIME-Version: 1.0' . "\r\n";
$headers .= 'Content-type: text/html; charset=iso-8859-1' . "\r\n";
$headers .= 'From: '.$from."\r\n". 'Reply-To: '.$from."\r\n".
'X-Mailer: PHP/' . phpversion();
$message = '<html><body><h1 style="color:#f40;">Hi</h1>';
$message .= '</body></html>';
mail($to, $subject, $message, $headers);
?>
```