❖ Introduction to PHP:

- The full form of PHP is “Hypertext Preprocessor”. Its original name was “Personal Home Page”
- Rasmus Lerdorf software engineer, Apache team member is the creator and original driving force behind PHP. The first part of PHP was developed for his personal use in late 1994.
- By the middle of 1997, PHP was being used on approximately 50,000 sites worldwide.
- PHP is server-side scripting language, which can be embedded in HTML or used as a stand-alone.
- PHP doesn’t do anything about what a page looks and sounds like. In fact, most of what PHP does is invisible to the end user.
- Someone looking at a PHP page will not necessarily be able to tell that it was not written purely in HTML, because usually the result of PHP is HTML.
- PHP is an official module of Apache HTTP Server.
- PHP is fully cross-platform, meaning it runs native on several flavors of Unix, as well as on Windows and now on Mac OS X.

❖ Advantages of PHP

- **Cost:** PHP costs you nothing. It is open source software and doesn’t need to purchase it for development.
- **Ease of Use:** PHP is easy to learn, compared to the others. A lot of Ready-made PHP scripts are freely available in market so, you can use them in your project or get some help from them.
- **HTML- Support:** PHP is embedded within HTML; In other words, PHP pages are ordinary HTML pages that escape into PHP mode only when necessary. When a client requests this page, the web server preprocesses it. This means it goes through the page from top to bottom, looking for sections of PHP, which it will try to resolve.
- **Cross-platform compatibility:** PHP and MySQL run native on every popular flavor of Unix and windows. A huge percentage of the world’s HTTP servers run on one of these two classes of operating system.
- **PHP is compatible with the three leading Web servers:** Apache HTTP Server for Unix and Windows, Microsoft Internet Information Server, and Netscape Enterprise Server. It also works with several lesser-known servers, including Alex Blits’ fhttpd, Microsoft’s Personal Web Server, AOL Server and Omnicentrix’s Omniserver application server.
- **Stability:** The word stable means two different things in this context:
    - The server doesn’t need to be rebooted often
    - The software doesn’t change radically and incompatibly from release to release.

  To our advantage, both of these apply to both MySQL and PHP.
- **Speed:** PHP is pleasingly zippy in its execution, especially when compiled as and Apache module on the Unix side. Although it takes a slight performance hit by being interpreted rather than compiled, this is far outweighed by the benefits PHP drives from its status as a Web server module.

❖ Basic PHP Syntax

- A PHP file normally contains HTML tags, just like an HTML file, and some PHP scripting code.
- A PHP scripting block starts with <?php and ends with ?>.
- A PHP scripting block can be placed anywhere in the document.
- Each code line in PHP must end with a semicolon. The semicolon is a separator and is used to distinguish one set of instructions from another.
- There are two basic statements to output text with PHP: **echo** and **print**.
- In the example we have used the echo statement to output the text "Hello World".

```
<html>
<body>
        <?php
                echo "Hello World";
        ?>
</body>
</html>
```

❖ Escaping from HTML

- When PHP parses a file, it looks for opening and closing tags, which tell PHP to start and stop interpreting the code between them.
- Parsing in this manner allows php to be embedded in different documents, as the PHP parser ignores everything outside of a pair of opening and closing tags. Most of the time you will see php embedded in HTML documents, as in this example.

```
<p>This is going to be ignored.</p>
        <?php echo 'While this is going to be parsed.'; ?>
<p>This will also be ignored.</p>
```

- There are four different pairs of opening and closing tags, which can be used in php.
    - <?php ?>
    - <script language="php"> </script>
    - <? ?> (Short Tag) “short_open_tag = On”
    - <% %> (ASP Style Tag) “asp_tags = on”

- Two of those, <?php ?> and <script language="php"> </script>, are always available. The other two are short tags and ASP style tags, and can be turned on and off from the **php.ini** configuration file.
- For example :

```
<?php   echo 'Welcome To RPBC';  ?>
<br>
<script language="php">
      echo 'Welcome To RPBC';
</script>
```

❖ Some important things to know when scripting with PHP.

- **PHP is Case Sensitive**
  - PHP is case sensitive - therefore watch your capitalization closely when you create or call variables, objects and functions. For example variable $A and $a both are different.
- **White Space**
  - PHP ignores extra spaces. You can add white space to your script to make it more readable. The following lines are equivalent:

    ```
    $name="Anil"
    $name = "Anil"
    ```
- **Insert Special Characters**
  - You can also insert special characters with a backslash:

    ```
    echo " \"Happy Birthday\"";
    ```

    output:

    "Happy Birthday"
- **Comments**
  - In PHP, we use // (C style comment) or # (Shell Style Comment) to make a single-line comment And /* and */ to make a large comment block.
  - For Example:

    ```
    // This is a C style comment
    # This is a Shell style comment
    /*
        This is a comment block
    */
    ```
- **Ending Statements With a Semicolon**
  - In PHP Each statement terminate with semicolon but semicolon is optional for last statement of php block.
  - For Example:

    ```
    <?php
        echo “Hello<br>”;
        echo “How Are You<br>”;
        echo “I am Fine<br>”      // semicolon is optional for last statement
    ?>
    ```

❖ Variables in PHP

- Variables in PHP are represented by a dollar sign followed by the name of the variable.
- The variable name is case-sensitive.
- Variable naming rule is: A valid variable name starts with a letter or underscore, followed by any number of letters, numbers, or underscores.
- For Example:

```
<?php
    $var = 'Hello';
    $Var = 'World';
    echo "$var, $Var";      // outputs "Hello,World"

    $4site = 'Hello';     // invalid; starts with a number
    $_4site = 'Hello';    // valid; starts with an underscore
?>
```

❖ Variable scope

- The scope of a variable is the context within which it is defined.
- For the most part all PHP variables only have a single scope.
- With in user-defined functions a local function scope is introduced. Any variable used inside a function is by default limited to the local function scope. For example:

```
<?php
        $a = 1; /* global scope */

        function Test()
        {
                $a = 100;
                echo “A:= $a”; /* reference to local scope variable */
        }
        Test();
        echo “<br>A:= $a”; /* reference to global scope variable */

?>
```

output :

```
A:= 100
A:= 1
```

This can cause some problems in that people may inadvertently change a global variable.

❖ The global keyword

- In PHP global variables must be declared global inside a function if they are going to be used in that function using global keyword and PHP-defined *$GLOBALS* array.
- First, an example use of *global*:

```
<?php
        $a = 1;
        $b = 2;
        function Sum()
        {
          global $a, $b;
          $b = $a + $b;
        }
        Sum();
        echo $b;
?>
```

- The above script will output "3". By declaring *$a* and *$b* global within the function, all references to either variable will refer to the global version.
- A second way to access variables from the global scope is to use the special PHP-defined *$GLOBALS* array.
- The previous example can be rewritten as:

```
<?php
        $a = 1;
        $b = 2;
        function Sum()
        {
          $GLOBALS['b'] = $GLOBALS['a'] + $GLOBALS['b'];
        }
        Sum();
        echo $b;
?>
```

❖ **Using static variables**

- Another important feature of variable scope is the *static* variable.
- A static variable exists only in a local function scope, but it does not lose its value when program execution leaves this scope. Consider the following example:
- **Example demonstrating need for static variables**

```
<?php
        function Test()
        {
           $a = 0;
           echo $a;
           $a++;
        }
        Test();
        Test();
        Test();
?>
```

ouput :

0 0 0

- This function is quite useless since every time it is called it sets $a to 0 and prints "0". The $a++ which increments the variable serves no purpose since as soon as the function exits the $a variable disappears.
- To make a useful counting function which will not lose track of the current count, the *$a* variable is declared static:
- **Example use of static variables**

```
<?php
        function Test()
        {
           static $a = 0;
           echo $a;
           $a++;
        }
        Test();
        Test();
        Test();
?>
```

ouput :

0 1 2

- Now, every time the Test() function is called it will print the value of *$a* and increment it.
- **Declaring static variables**

```
<?php
        function foo()
        {
           static $int = 0;          // correct
           static $int = 1+2;        // wrong  (as it is an expression)
           static $int = sqrt(121);  // wrong  (as it is an expression too)

           $int++;
           echo $int;
        }
?>
```

❖ **Variable variables**

- Sometimes it is convenient to be able to have variable variable names.
- That is, a variable name which can be set and used dynamically.
- A variable variables takes the value of a variable and treats that as the name of a variable.
- For Example :

```
<?php
        $a = 'hello';
        $$a = 'world';  // $hello=’world’;

        echo "$a ${$a}";
        echo "$a $hello";
?>
```

- In the above example, *hello*, can be used as the name of a variable by using two-dollar signs.
- At this point two variables have been defined and stored in the PHP symbol tree: *$a* with contents "hello" and *$hello* with contents "world".
- Here both echo statement produce same output : **hello world**

❖ Variables from outside PHP (HTML Forms GET and POST)

- When a form is submitted to a PHP script, the information from that form is automatically made available to the script.
- There are many ways to access this information here explain Get & Post method of Form Object.

❖ **GET Method:**

- The GET method passes arguments from in page to the next page as a part of the URL (Uniform Resource Locator) Query String.
- When used for form handling, GET appends the indicated variable name and value to the URL designated in the ACTION attribute with a question mark separator.
- Each item submitted via GET method is accessed in the handler via the $_GET array.

Example:

```
<html>
<body>
<form action="welcome.php" method="GET">
        Enter your name: <input type="text" name="name" />
        Enter your age: <input type="text" name="age" />
        <input type="submit"value=”OK” />
</form>
</body>
</html>
```

- **URL : http://localhost/welcome.php?name=Anil&age=22&submit=OK**
- The "welcome.php" file looks like this:

```
<html>
<body>
        Welcome <?php echo $_GET["name"]; ?>.<br />
        You are <?php echo $_GET["age"]; ?> years old!
</body>
</html>
```

- **OUTPUT:**

  Welcome Anil.
  You are 22 years old!

- **Advantage Of GET Method:**
  - It construct an actual new and differentiable URL query string so user can bookmark this page.
- **Disadvantages Of GET Method:**
  - It is not suitable for login form because username & password fully visible onscreen.
  - Every GET submission is recorded in the web server log, data set included.
  - The length of URL is limited so limited data pass using GET method. **(Query string to be limited 255 characters)**

❖ **POST Method:**

- POST method is the preferred method of form submission.
- The form data set is included in the body of the form when it is forwarded to the processing agent (web server).
- No visible change to the URL will result according to the different data submitted.
- Each item submitted via POST method is accessed in the handler via the $_POST array.
- **Advantages Of POST method:**
  - It is more secure then GET because user entered information is never visible in the URL.
  - There is a much larger limit on the amount of data that can be passed (a couple of kilobytes).
- **Disadvantages Of POST method:**
  - The result at a given moment cannot be book marked.
  - The result should be expired by the browser, so that an error will result if the user employs the Back button to revisit the page.
  - This method can be incompatible with certain firewall setups.

Example :

In Above example: set form tag method POST instead of GET
And in welcome.php file replace $_GET with $_POST.

❖ PHP Operators

There are three types of operators.

- Firstly there is the unary operator which operates on only one value, for example ! (the negation operator) or ++ (the increment operator).
- The second group is termed binary operators; this group contains most of the operators that PHP supports. For example logical, conditional operator etc…
- The third group is the ternary operator: **? :** It should be used to select between two expressions depending on a third one.

❖ Arithmetic Operators

Example	Name	Result
-$a	Negation	Opposite of $a.
$a + $b	Addition	Sum of $a and $b.
$a - $b	Subtraction	Difference of $a and $b.
$a * $b	Multiplication	Product of $a and $b.
$a / $b	Division	Quotient of $a and $b.
$a % $b	Modulus	Remainder of $a divided by $b.

Note:

- The division operator ("/") returns a float value anytime, even if the two operands are integers (or strings that get converted to integers).
- Remainder $a % $b is negative for negative $a.

❖ Increment/decrement Operators

Example	Name	Effect
++$a	Pre-increment	Increments $a by one, then returns $a.
$a++	Post-increment	Returns $a, then increments $a by one.
--$a	Pre-decrement	Decrements $a by one, then returns $a.
$a--	Post-decrement	Returns $a, then decrements $a by one.

❖ Assignment Operators

Operator	Example	Is The Same As
=	$x=$y	$x=$y
+=	$x += $y	$x = $x + $y
-=	$x -= $y	$x= $x - $y
*=	$x *= $y	$x=$x * $y
/=	$x /= $y	$x=$x / $y
%=	$x%=$y	$x=$x % $y

❖ **Comparison Operators**

Example	Name	Result
$a == $b	Equal	**TRUE** if $a is equal to $b.
$a === $b	Identical	**TRUE** if $a is equal to $b, and they are of the same type. (introduced in PHP 4)
$a != $b	Not equal	**TRUE** if $a is not equal to $b.
$a <> $b	Not equal	**TRUE** if $a is not equal to $b.
$a !== $b	Not identical	**TRUE** if $a is not equal to $b, or they are not of the same type. (introduced in PHP 4)
$a < $b	Less than	**TRUE** if $a is strictly less than $b.
$a > $b	Greater than	**TRUE** if $a is strictly greater than $b.
$a <= $b	Less than or equal to	**TRUE** if $a is less than or equal to $b.
$a >= $b	Greater than or equal to	**TRUE** if $a is greater than or equal to $b.

❖ **Logical Operators**

Example	Name	Result
$a and $b	And	**TRUE** if both $a and $b are **TRUE**.
$a or $b	Or	**TRUE** if either $a or $b is **TRUE**.
$a xor $b	Xor	**TRUE** if either $a or $b is **TRUE**, but not both.
! $a	Not	**TRUE** if $a is not **TRUE**.
$a && $b	And	**TRUE** if both $a and $b are **TRUE**.
$a || $b	Or	**TRUE** if either $a or $b is **TRUE**.

❖ **Ternary Operator**

- One especial useful operator in ternary conditional operator
- Its job is to takes three expression and use truth value of the first expression to decide which of the other two expression to evaluate and return.

Syntax:

Test-expression ? yes-expression : no-expression

- The value of this expression is the result of yes-expression if test-expression is true; otherwise no-expression.

❖ **String Operators**

- There are two string operators.
- The first is the concatenation operator ('.'), which returns the concatenation of its right and left arguments.
- The second is the concatenating assignment operator ('.='), which appends the argument on the right side to the argument on the left side.

```
<?php
        $a = "Hello ";
        $b = $a . "World!"; // now $b contains "Hello World!"

        $a = "Hello ";
        $a .= "World!";     // now $a contains "Hello World!"
 ?>
```

❖ Control Structures:

❖ if Statment

- The if construct is one of the most important features of many languages, PHP included. It allows for conditional execution of code fragments.
- PHP features an *if* structure that is similar to that of C:

```
if (expression)
{
   statements
}
```

- As described in the section about expressions, *expression* is evaluated to its Boolean value.
- If *expression* evaluates to TRUE, PHP will execute *statements*, and if it evaluates to FALSE - it'll ignore it.
- The following example would display a is bigger than b if *$a* is bigger than *$b*:

```
<?php
        if ($a > $b)
        {
                echo "a is bigger than b";
        }
?>
```

❖ If…else

- Often you'd want to execute a statement if a certain condition is met, and a different statement if the condition is not met. This is what *else* is for.
- else extends an *if* statement to execute a statement in case the expression in the *if* statement evaluates to FALSE.

```
if (expression)
{
   block-1
}
else
{
   block-2
}
```

- If *expression* evaluates to TRUE then block –1 is executed Else block-2 is executed.
- For example, the following code would display a is bigger than b if *$a* is bigger than *$b*, and a is NOT bigger than b otherwise:

```
<?php
        if ($a > $b)
        {    echo "a is bigger than b";  }
        else
        {    echo "a is NOT bigger than b";     }
?>
```

❖ Elseif

- elseif, as its name suggests, is a combination of if and else.
- Like else, it extends an if statement to execute a different statement in case the original if expression evaluates to FALSE. However, unlike else, it will execute that alternative expression only if the elseif conditional expression evaluates to TRUE.

- For example, the following code would display a is bigger than b, a equal to b or a is smaller than b:

```
<?php
        if ($a > $b)
        {
           echo "a is bigger than b";
        } elseif ($a == $b)
        {
                echo "a is equal to b";
        }
        else
        {
                echo "a is smaller than b";
        }
?>
```

❖ **switch**

- The *switch* statement is similar to a series of IF statements on the same expression.
- In many occasions, you may want to compare the same variable (or expression) with many different values, and execute a different piece of code depending on which value it equals to. This is exactly what the *switch* statement is for.
- **Syntax:**

```
switch (expression)
{
        case label1:
          code to be executed if expression = label1;
          break;
        case label2:
          code to be executed if expression = label2;
          break;
        default:
          code to be executed
          if expression is different
          from both label1 and label2;
}
```

- **Example:**

```
<?php
        switch ($x)
        {
                case 1:
                  echo "Number 1";
                  break;
                case 2:
                  echo "Number 2";
                  break;
                case 3:
                  echo "Number 3";
                  break;
                default:
                  echo "No number between 1 and 3";
        }
?>
```

❖ **while**

- *while* loops are the simplest type of loop in PHP. They behave just like their C counterparts.
- The basic form of a *while* statement is:

```
while (expr)
{
   statements
}
```

- The meaning of a *while* statement is simple. It tells PHP to execute the nested statement(s) repeatedly, as long as the *while* expression evaluates to TRUE.
- The value of the expression is checked each time at the beginning of the loop, so even if this value changes during the execution of the nested statement(s), execution will not stop until the end of the iteration (each time PHP runs the statements in the loop is one iteration).
- Sometimes, if the *while* expression evaluates to FALSE from the very beginning, the nested statement(s) won't even be run once.
- **Example:**

```
<?php
        $i = 1;
        while ($i <= 5)
        {
           echo $i++;
        }
?>
```

❖ **do-while**

- do-while loops are very similar to while loops, except the truth expression is checked at the end of each iteration instead of in the beginning.
- The main difference from regular while loops is that the first iteration of a do-while loop is guaranteed to run (the truth expression is only checked at the end of the iteration), whereas it's may not necessarily run with a regular while loop (the truth expression is checked at the beginning of each iteration, if it evaluates to FALSE right from the beginning, the loop execution would end immediately).
- The basic form of a *while* statement is:

```
do
{
   statements
} while (expr);
```

- **Example:**

```
<?php
        $i = 5;
        do
        {
           echo $i;
        } while ($i >0 );
?>
```

- The above loop would run one time exactly, since after the first iteration, when truth expression is checked, it evaluates to FALSE ($i is not bigger than 0) and the loop execution ends.

❖ **for**

- for loops are the most complex loops in PHP.
- They behave like their C counterparts. The syntax of a for loop is:

```
for (expr1; expr2; expr3)
{
   statements
}
```

- The first expression (expr1) is evaluated (executed) once unconditionally at the beginning of the loop.
- In the beginning of each iteration, expr2 is evaluated. If it evaluates to TRUE, the loop continues and the nested statement(s) are executed. If it evaluates to FALSE, the execution of the loop ends.
- At the end of each iteration, expr3 is evaluated (executed).
- Each of the expressions can be empty. *expr2* being empty means the loop should be run indefinitely (PHP implicitly considers it as TRUE, like C). This may not be as useless as you might think, since often you'd want to end the loop using a conditional *break* statement instead of using the *for* truth expression.
- Consider the following examples. All of them display numbers from 1 to 10:

```
<?php
        /* example 1 */

        for ($i = 1; $i <= 10; $i++)
        {
           echo $i;
        }

        /* example 2 */

        for ($i = 1; ; $i++) {
           if ($i > 10)
                {
                        break;
                }
           echo $i;
        }

        /* example 3 */

        $i = 1;
        for (; ; )
        {
           if ($i > 10) {       break;    }
           echo $i;
           $i++;
        }

        /* example 4 */

        for ($i = 1; $i <= 10; print $i, $i++);
?>
```

- Of course, the first example appears to be the nicest one (or perhaps the fourth), but you may find that being able to use empty expressions in for loops comes in handy in many occasions.

❖ **foreach**

- Loops over the array given by the parameter.
- On each loop, the value of the current element is assigned to $value and the array pointer is advanced by one - so on the next loop, you'll be looking at the next element.
- **Syntax**

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

- **Example**

  The following example demonstrates a loop that will print the values of the given array:

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

❖ **break**

- *break* ends execution of the current *for*, *foreach*, *while*, *do-while* or *switch* structure.
- *break* accepts an optional numeric argument which tells it how many nested enclosing structures are to be broken out of.
- For Example :

```
<?php
        $i = 0;
        while (++$i)
        {
           switch ($i)
           {
                case 5:
                   echo "At 5<br />\n";
                   break 1;  /* Exit only the switch. */
                case 10:
                   echo "At 10; quitting<br />\n";
                   break 2;  /* Exit the switch and the while. */
                default:
                   break;
            }
        }
?>
```

❖ **continue**

- continue is used within looping structures to skip the rest of the current loop iteration and continue execution at the condition evaluation and then the beginning of the next iteration.
- continue accepts an optional numeric argument which tells it how many levels of enclosing loops it should skip to the end of.

- For Example

```
<?php
        for ($i = 0; $i < 5; ++$i) {
            if ($i == 2)
                continue;
            echo "   $i";
        }
?>
```

output :

```
0  1  3  4
```

❖ **Alternative syntax for control structures**

- PHP offers an alternative syntax for some of its control structures; namely, if, while, for, foreach, and switch. In each case, the basic form of the alternate syntax is to change the opening brace to a colon (:) and the closing brace to endif;, endwhile;, endfor;, endforeach;, or endswitch;, respectively.
- The alternative syntax applies to else and elseif. The following is an if structure with elseif and else in the alternative format:

```
<?php
        if ($a == 5):
                echo "a equals 5";
        elseif ($a == 6):
            echo "a equals 6";
        else:
             echo "a is neither 5 nor 6";
        endif;
?>
```

❖ **Arrays**

- An array in PHP is actually an ordered map.
- A map is a type that maps *values* to *keys*.
- This type is optimized in several ways; so you can use it as a real array, or a list (vector), hash table (which is an implementation of a map), dictionary, collection, stack, queue and probably more. Because you can have another PHP array as a value, you can also quite easily simulate trees.
- **Specifying with array()**

- An array can be created by the array() language-construct. It takes a certain number of comma-separated key => value pairs.

```
array( [key =>] value
    , ...
    )
```

- *key* may be an integer or string
- *value* may be any value
- For Example:

```
<?php
        $arr = array("foo" => "bar", 12 => true);

        echo $arr["foo"]; // bar
        echo $arr[12];    // 1
?>
```

- A *key* may be either an *integer* or a string. If a key is the standard representation of an integer, it will be interpreted as such (i.e. *"8"* will be interpreted as *8*, while *"08"* will be interpreted as *"08"*). Floats in *key* are truncated to integer. There are no different indexed and associative array types in PHP; there is only one array type, which can both contain integer and string indices.
- A value can be of any PHP type.

```
<?php
        $arr = array("somearray" => array(6 => 5, 13 => 9, "a" => 42));

        echo $arr["somearray"][6];    // 5
        echo $arr["somearray"][13];   // 9
        echo $arr["somearray"]["a"];  // 42
?>
```

- If you do not specify a key for a given value, then the maximum of the integer indices is taken, and the new key will be that maximum value + 1. If you specify a key that already has a value assigned to it, that value will be overwritten.

```
<?php
        // This array is the same as ...
        array(5 => 43, 32, 56, "b" => 12);

        // ...this array
        array(5 => 43, 6 => 32, 7 => 56, "b" => 12);
?>
```

- Using TRUE, as a key will evaluate to integer 1 as key. Using FALSE as a key will evaluate to integer 0 as key. Using NULL as a key will evaluate to the empty string. Using the empty string, as key will create (or overwrite) a key with the empty string and its value it is not the same as using empty brackets.

- **Creating/modifying with square-bracket syntax**

- You can also modify an existing array by explicitly setting values in it.
- This is done by assigning values to the array while specifying the key in brackets. You can also omit the key, add an empty pair of brackets ("*[]*") to the variable name in that case.

```
$arr[key] = value;
$arr[] = value;
```

- key may be an integer or string. value may be any value
- If *$arr* doesn't exist yet, it will be created. So this is also an alternative way to specify an array.
- To change a certain value, just assign a new value to an element specified with its key.
- If you want to remove a key/value pair, you need to unset() it.
- For Example :

```
<?php
        $arr = array(5 => 1, 12 => 2);
        $arr[] = 56;    // This is the same as $arr[13] = 56;
        $arr["x"] = 42; // This adds a new element to the array with key "x"

        unset($arr[5]); // This removes the element from the array

        unset($arr);    // This deletes the whole array
?>
```

- If you provide the brackets with no key specified, then the maximum of the existing integer indices is taken, and the new key will be that maximum value + 1. If no integer indices exist yet, the key will be *0* (zero). If you specify a key that already has a value assigned to it, that value will be overwritten.
- Note that the maximum integer key used for this need not currently exist in the array. It simply must have existed in the array at some time since the last time the array was re-indexed.
- The following example illustrates:

```
<?php
    $array = array(1, 2, 3, 4, 5);  // Create a simple array.
    print_r($array);
    echo "<br>";
    // Now delete every item, but leave the array itself intact:
    foreach ($array as $i => $value)
    {
       unset($array[$i]);
    }
    print_r($array);
    echo "<br>";
    // Append an item ( the new key is 5, instead of 0 as you might expect).
    $array[] = 6;
    print_r($array);
    echo "<br>";
    // Re-index:
    $array = array_values($array);
    $array[] = 7;
    print_r($array);
?>
```

output:

```
Array(   [0] => 1    [1] => 2    [2] => 3    [3] => 4    [4] => 5 )
Array(  )
Array(    [5] => 6 )
Array(    [0] => 6    [1] => 7 )
```

❖ **User Define Functions**

User-defined functions are not requirement in PHP. You can produce interesting and useful Web sites simply with the basic language constructs and the large body of built-in functions. If you find that your code files are getting longer, harder to understand, and more difficult to manage, however, it may be indication that you should start wrapping some of your code up into functions.

What is function?

A function us a way of wrapping up a chunk of code and giving that chunk a name, so that you can use that chunk later in just one line of code. Functions are most useful when you will be using the code in more than one place, but they can helpful even in one-use situation, because they can make your code much more readable,

Function definition syntax

```
Function function-name ($argument-1 ,$argument-2 ..)
{
    statement -1;
    statement – 2;
     …..
 }
```

function definition have four parts:

- The special word function
- The name that you want to give your function
- The function's parameter list-dollar-sign variable separate by commas
- The function body – a brace-enclosed set of statements

Function naming rule : A valid function name starts with a letter or underscore, followed by any number of letters, numbers, or underscores.

Example :

```
<?php
        echo "WelCome To AIT";
        function sum()
        {
                $a=10;
                $b=20;
                $ans=$a+$b;
                echo "<br>Sum := $ans";
        }
        sum();
?>
```

❖ **Function arguments**

- Information may be passed to functions via the argument list, which is a comma-delimited list of expressions.
- PHP supports passing arguments by value (the default), passing by reference, and default argument values. Variable-length argument lists are supported only in PHP 4 and later;

Example:

```
<?php
        echo "WelCome To AIT";
        function sum($a,$b)
        {
                $ans=$a+$b;
                echo "<br>Sum := $ans";
    }
        sum(10,100);
?>
```

❖ **Making arguments be passed by reference**

- By default, function arguments are passed by value (so that if you change the value of the argument within the function, it does not get changed outside of the function). If you wish to allow a function to modify its arguments, you must pass them by reference.
- If you want an argument to a function to always be passed by reference, you can prepend an ampersand (&) to the argument name in the function definition:

```
<?php
        function add_some_extra(&$string)
        {
           $string .= 'and something extra.';
        }

        $str = 'This is a string, ';
        add_some_extra($str);
        echo $str;    // outputs 'This is a string, and something extra.'
?>
```

❖ **Default argument values**

- A function may define C++ style default values for scalar arguments. You can write your function to have default values. For example you can define an argument as having a default value, which the parameters for that arguments will adopt when the function call, provided you don't pass a value in for that argument.
- But you must define any arguments with default value to the right of any arguments without default value.
- For Example :

```
<?php
        function makecoffee($type = "cappuccino")
        {
           return "Making a cup of $type.\n";
        }
        echo makecoffee();
        echo makecoffee("espresso");
?>
```

output

```
Making a cup of cappuccino.
Making a cup of espresso.
```

- **default arguments function declaration**

```
function make($name = "anil", $age)  // Incorrect default arguments
function make( $age ,$name = "anil")  // Correct default arguments
```

❖ **Variable-length argument lists**

- PHP 4 and above has support for variable-length argument lists in user-defined functions. This is really quite easy, using the func_num_args(), func_get_arg(), and func_get_args() functions.
- No special syntax is required, and argument lists may still be explicitly provided with function definitions and will behave as normal.

- **func_num_args**
  - func_num_args -- Returns the number of arguments passed to the function

  Syntax:

  ```
  int func_num_args ( void )
  ```

  - Returns the number of arguments passed into the current user-defined function.
  - func_num_args() will generate a warning if called from outside of a user-defined function.
  - This function cannot be used directly as a function parameter. Instead, its result may be assigned to a variable, which can then be passed to the function.
  - Because this function depends on the current scope to determine parameter details, it cannot be used as a function parameter. If you must pass this value, assign the results to a variable, and pass the variable.

For Example:

```
<?php
        function foo()
        {
           $numargs = func_num_args();
           echo "Number of arguments: $numargs\n";
        }

        foo(1, 2, 3);    // Prints 'Number of arguments: 3'
?>
```

- **func_get_arg**
  - func_get_arg -- Return an item from the argument list

Syntax:

```
mixed func_get_arg ( int arg_num )
```

- Returns the argument which is at the *arg_num*'th offset into a user-defined function's argument list.
- Function arguments are counted starting from zero.
- func_get_arg() will generate a warning if called from outside of a function definition. This function cannot be used directly as a function parameter. Instead, its result may be assigned to a variable, which can then be passed to the function.
- If *arg_num* is greater than the number of arguments actually passed, a warning will be generated and func_get_arg() will return FALSE.

```
<?php
        function foo()
        {
            $numargs = func_num_args();
            echo "Number of arguments: $numargs<br>\n";
            if ($numargs >= 2)
            {
                echo "Second argument is: " . func_get_arg(1) . "<br>\n";
            }
        }

        foo (1, 2, 3);
?>
```

- **func_get_args**
  - func_get_args -- Returns an array comprising a function's argument list

Syntax

```
array func_get_args ( void )
```

- Returns an array in which each element is a copy of the corresponding member of the current user-defined function's argument list.
- func_get_args() will generate a warning if called from outside of a function definition. This function cannot be used directly as a function parameter. Instead, its result may be assigned to a variable, which can then be passed to the function.
- This function returns a copy of the passed arguments only, and does not account for default (non-passed) arguments.

For Example :

```
<?php
        function foo()
        {
           $numargs = func_num_args();
           echo "Number of arguments: $numargs<br />\n";
           if ($numargs >= 2) {
              echo "Second argument is: " . func_get_arg(1) . "<br >";
           }
           $arg_list = func_get_args();
           for ($i = 0; $i < $numargs; $i++) {
              echo "Argument $i is: " . $arg_list[$i] . "<br />\n";
           }
        }

        foo(1, 2, 3);
?>
```

❖ **Returning values**

- Values are returned by using the optional return statement. Any type may be returned, including lists and objects.
- This causes the function to end its execution immediately and pass control back to the line from which it was called.

```
<?php
        function square($num)
        {
           return $num * $num;
        }
        echo square(4);   // outputs '16'.
?>
```

- You can't return multiple values from a function, but similar results can be obtained by returning a list.
- Returning an array to get multiple values

```
<?php
        function small_numbers()
        {
           return array (0, 1, 2);
        }
        list ($zero, $one, $two) = small_numbers();
?>
```

❖ **Variable functions**

- PHP supports the concept of variable functions.
- This means that if a variable name has parentheses appended to it, PHP will look for a function with the same name as whatever the variable evaluates to, and will attempt to execute it. Among other things, this can be used to implement callbacks, function tables, and so forth.
- Variable functions won't work with language constructs such as echo(), print(), unset(), isset(), empty(), include(), require() and the like.
- You need to use your own wrapper function to utilize any of these constructs as variable functions.

Variable function example

```
<?php
        function foo()
        {
                echo "In foo()<br />\n";
        }

        function bar($arg = '')
        {
                echo "In bar(); argument was '$arg'.<br />\n";
        }

        // This is a wrapper function around echo
        function echoit($string)
        {
                echo $string;
        }

        $func = 'foo';
        $func();        // This calls foo()

        $func = 'bar';
        $func('test');  // This calls bar()

        $func = 'echoit';
        $func('test');  // This calls echoit()
?>
```

Variable Handling Functions

❖ **gettype**

- gettype -- Get the type of a variable

Syntax:

string gettype (mixed var)

- Possibles values for the returned string are: "boolean" (since PHP 4),"integer", "double" (for historical reasons "double" is returned in case of a float, and not simply "float"),"string", "array", "object", "resource" (since PHP 4), "NULL" (since PHP 4), "user function" (PHP 3 only, deprecated), "unknown type"

❖ **settype**

- settype -- Set the type of a variable

Syntax:

bool settype (mixed &var, string type)

- Possibles values for the returned string are: "boolean" (since PHP 4),"integer", "double" (for historical reasons "double" is returned in case of a float, and not simply "float"),"string", "array", "object", "resource" (since PHP 4), "NULL" (since PHP 4), "user function" (PHP 3 only, deprecated), "unknown type"
- Returns TRUE on success or FALSE on failure.

Example:

```
<?php
        $foo = "5bar"; // string
        $bar = true;   // boolean
        settype($foo, "integer"); // $foo is now 5   (integer)
        settype($bar, "string");  // $bar is now "1" (string)
?>
```

❖ **strval**

- strval -- Get string value of a variable

Syntax:

string strval (mixed var)

- Returns the string value of *var*.
- *var* may be any scalar type. You cannot use strval() on arrays or objects.

❖ **floatval**

- floatval -- Get float value of a variable
- **Syntax:**

float floatval (mixed var)

❖ **intval**

- intval -- Get the integer value of a variable

Syntax:

int intval (mixed var [, int base])

Example:

```
<?php
        $var = '122.34343The';
        echo "Var:=". $var;
        echo "<br>String Value:". strval($var);
        echo "<br>Integer Value:". intval($var);
        echo "<br>Float Value:". floatval($var);
?>
```

❖ **is_string , is_int , is_float , is_null , is_bool , is_array**

- Finds whether a variable is a string, integer, float, null, boolean and array respectively.
- Return value is Boolean.

Syntax:

bool ***abovefunction*** (mixed var)

❖ **isset**

- isset -- Determine whether a variable is set

Syntax:

bool isset (mixed var [, mixed var [, ...]])

- Returns TRUE if *var* exists; FALSE otherwise.

❖ **unset**

- unset -- Unset a given variable

Synatx:

void unset (mixed var [, mixed var [, mixed ...]])

- unset() destroys the specified variables.

Example:

```
<?php
    $a = "test";
    $b = "anothertest";
    var_dump(isset($a));      // TRUE
    var_dump(isset($a, $b)); // TRUE
    unset ($a);
    var_dump(isset($a));     // FALSE
    var_dump(isset($a, $b)); // FALSE
    $foo = NULL;
    var_dump(isset($foo));   // FALSE
?>
```

❖ **print_r**

- print_r -- Prints human-readable information about a variable

Syntax:

bool print_r (mixed expression [, bool return])

- print_r() displays information about a variable in a way that's readable by humans. If given a string, integer or float, the value itself will be printed.
- If given an array, values will be presented in a format that shows keys and elements.

Example:

```
<?php
    $a = array ('a' => 'apple', 'c' => array ('x'));
    print_r ($a);
?>
```

Output:

```
Array
(
    [a] => apple
    [c] => Array
        (
            [0] => x
        )
)
```

❖ **Strings**

- A string is series of characters.
- In PHP, a character is the same as a byte, that is, there are exactly 256 different characters possible.
- This also implies that PHP has no native support of Unicode.
- It is no problem for a string to become very large. There is no practical bound to the size of strings imposed by PHP, so there is no reason at all to worry about long strings.
- A string literal can be specified in three different ways.
    - single quoted
    - double quoted
    - heredoc syntax

- **Single quoted**
    - The easiest way to specify a simple string is to enclose it in single quotes (the character ').
    - To specify a literal single quote, you will need to escape it with a backslash (\), like in many other languages.
    - If a backslash needs to occur before a single quote or at the end of the string, you need to double it.
    - Note that if you try to escape any other character, the backslash will also be printed! So usually there is no need to escape the backslash itself.
    - For Example:

```
<?php
        echo 'this is a simple string';

        // Outputs: Arnold once said: "I'll be back"
        echo 'Arnold once said: "I\'ll be back"';

        // Outputs: You deleted C:\*.*?
        echo 'You deleted C:\\*.*?';

        // Outputs: You deleted C:\*.*?
        echo 'You deleted C:\*.*?';

        // Outputs: This will not expand: \n a newline
        echo 'This will not expand: \n a newline';

        // Outputs: Variables do not $expand $either
        echo 'Variables do not $expand $either';
?>
```

- **Double quoted**
    - If the string is enclosed in double-quotes (").

- **Heredoc**
    - Another way to delimit strings is by using heredoc syntax ("<<<"). One should provide an identifier after <<<, then the string, and then the same identifier to close the quotation.
    - The closing identifier must begin in the first column of the line. Also, the identifier used must follow the same naming rules as any other label in PHP: it must contain only alphanumeric characters and underscores, and must start with a non-digit character or underscore.
    - It especially useful in creating pages that contains HTML forms.

- For Example :

```
<?php
        $sing='Welcome To RPBC \n';
        $db="Welcome To RPBC \n";
        $str = <<<EOD
                Example of string
                spanning multiple lines
                using heredoc syntax.
                EOD;

        echo $db;
        echo $sing;
        echo $str;
?>
```

- **Escaped characters**

sequence	meaning
\n	linefeed (LF or 0x0A (10) in ASCII)
\r	carriage return (CR or 0x0D (13) in ASCII)
\t	horizontal tab (HT or 0x09 (9) in ASCII)
\\	backslash
\$	dollar sign
\"	double-quote
\[0-7]{1,3}	the sequence of characters matching the regular expression is a character in octal notation
\x[0-9A-Fa-f]{1,2}	the sequence of characters matching the regular expression is a character in hexadecimal notation

String Functions

❖ **chr**

- Returns a one-character string containing the character specified by *ascii*.

Syntax :

string chr (int ascii)

Example :

```
echo chr(65);   // A
echo chr(97);   // a
```

❖ **ord**

- Returns the ASCII value of the first character of *string*.
- This function complements chr().

Syntax :

int **ord** (string string)

Example :

```
echo ord("A");  //65
echo ord("anil");  //97
```

❖ **strtolower**

- Returns *string* with all alphabetic characters converted to lowercase.

Syntax :

string **strtolower** (string str)

Example :

```
echo strtolower("WelCome To RPBC");  //welcome to rpbc
```

❖ **strtoupper**

- Returns *string* with all alphabetic characters converted to uppercase.

Syntax :

string **strtoupper** (string str)

Example :

```
echo strtoupper("WelCome To RPBC");  //WELCOME TO RPBC
```

❖ **ucfirst**

- ucfirst -- Make a string's first character uppercase

Syntax :

string **ucfirst** (string str)

- Returns a string with the first character of *str* capitalized, if that character is alphabetic.

Example :

```
echo ucfirst("welCome To RPBC");  // WelCome To RPBC
```

❖ **ucwords**

- ucwords -- Uppercase the first character of each word in a string

Syntax :

string **ucwords** (string str)

- Returns a string with the first character of each word in *str* capitalized, if that character is alphabetic.
- The definition of a word is any string of characters that is immediately after a whitespace (These are: space, form-feed, newline, carriage return, horizontal tab, and vertical tab).

Example :

```
echo ucwords("welCome tO RPBC");  // WelCome TO RPBC
```

❖ **strlen**

- Returns the length of the given *string*.

Syntax:

```
int strlen ( string string )
```

Example :

```
echo strlen("RPBC");  // 4
```

❖ **ltrim**

- ltrim -- Strip whitespace (or other characters) from the beginning of a string

Syntax:

string **ltrim** (string str [, string charlist])

- This function returns a string with whitespace stripped from the beginning of *str*. Without the second parameter, ltrim() will strip these characters:

- " " (ASCII *32* (*0x20*)), an ordinary space.
- "\t" (ASCII *9* (*0x09*)), a tab.
- "\n" (ASCII *10* (*0x0A*)), a new line (line feed).
- "\r" (ASCII *13* (*0x0D*)), a carriage return.
- "\0" (ASCII *0* (*0x00*)), the *NUL*-byte.
- "\x0B" (ASCII *11* (*0x0B*)), a vertical tab.

- You can also specify the characters you want to strip, by means of the *charlist* parameter. Simply list all characters that you want to be stripped. With .. you can specify a range of characters.

Example :

```
$str = "  ...Welcome..";
echo ($str."Length :=". strlen($str)."<br>"); // 14
$str=ltrim($str);
echo ($str."Length :=". strlen($str)."<br>"); //12
$str=ltrim($str," .");
echo ($str."Length :=". strlen($str)); //9
```

❖ **rtrim**

- rtrim is same as ltrim function but rtrim -- Strip whitespace (or other characters) from the end of a string

Syntax:

string **rtrim** (string str [, string charlist])

Example :

```
$str = "...Welcome..  ";
echo ($str."Length :=". strlen($str)."<br>"); // 14
$str=rtrim($str);
echo ($str."Length :=". strlen($str)."<br>"); //12
$str=rtrim($str," .");
echo ($str."Length :=". strlen($str)); //10
```

❖ **trim**

- trim is same as ltrim & rtrim function but trim -- Strip whitespace (or other characters) from the beginning & end of a string

Syntax:

string **rtrim** (string str [, string charlist])

Example :

```
$str = "  ...Welcome..  ";
echo ($str."Length :=". strlen($str)."<br>"); //  16
$str=trim($str);
echo ($str."Length :=". strlen($str)."<br>"); // 12
$str=trim($str," .");
echo ($str."Length :=". strlen($str)); // 7
```

❖ **substr**

- substr -- Return part of a string

Syntax:

string substr (string string, int start [, int length])

- substr() returns the portion of *string* specified by the *start* and *length* parameters.
- If *start* is non-negative, the returned string will start at the *start*'th position in *string*, counting from zero.
- For instance, in the string '*abcdef*', the character at position *0* is '*a*', the character at position *2* is '*c*', and so forth.

Example :

```
echo substr('abcdef', 1);     // bcdef
echo substr('abcdef', 1, 3);  // bcd
echo substr('abcdef', 0, 4);  // abcd
// Accessing single characters in a string
can also be achived using "curly braces"
$string = 'abcdef';
echo $string{0};              // a
echo $string{3};              // d
```

- If *start* is negative, the returned string will start at the *start*'th character from the end of *string*.

```
$rest = substr("abcdef", -1);    // returns "f"
$rest = substr("abcdef", -2);    // returns "ef"
$rest = substr("abcdef", -3, 1); // returns "d"
```

- If *length* is given and is positive, the string returned will contain at most *length* characters beginning from *start* (depending on the length of *string*). If *string* is less than or equal to *start* characters long, FALSE will be returned.
- If *length* is given and is negative, then that many characters will be omitted from the end of *string* (after the start position has been calculated when a *start* is negative). If *start* denotes a position beyond this truncation, an empty string will be returned.

```
$rest = substr("abcdef", 0, -1);  // returns "abcde"
$rest = substr("abcdef", 2, -1);  // returns "cde"
$rest = substr("abcdef", 4, -4);  // returns ""
$rest = substr("abcdef", -3, -1); // returns "de"
```

❖ **strcmp**

- strcmp -- Binary safe string comparison

Syntax:

int strcmp (string str1, string str2)

- Returns < 0 if *str1* is less than *str2*; > 0 if *str1* is greater than *str2*, and 0 if they are equal.
- This comparison is case sensitive.

Example:

```
echo strcmp(“Hello”,”hello”); // -1
```

❖ **strcasecmp**

- strcasecmp -- Binary safe case-insensitive string comparison

Syntax:

int strcasecmp (string str1, string str2)

- Returns < 0 if *str1* is less than *str2*; > 0 if *str1* is greater than *str2*, and 0 if they are equal.

Example:

```
$var1 = "Hello";
$var2 = "hello";
if (strcasecmp($var1, $var2) == 0) {
   echo '$var1 is equal to $var2 in a case-insensitive string comparison';
}
```

❖ **strncasecmp**

- strncasecmp -- Binary safe case-insensitive string comparison of the first n characters

Syntax:

```
int strncasecmp ( string str1, string str2, int len )
```

- This function is similar to strcasecmp(), with the difference that you can specify the (upper limit of the) number of characters (*len*) from each string to be used in the comparison.
- Returns < 0 if *str1* is less than *str2*; > 0 if *str1* is greater than *str2*, and 0 if they are equal.

❖ **strpos**

- strpos -- Find position of first occurrence of a string

Syntax:

```
int strpos ( string haystack, mixed needle [, int offset] )
```

- Returns the numeric position of the first occurrence of *needle* in the *haystack* string. Unlike the strrpos(), this function can take a full string as the *needle* parameter and the entire string will be used.
- If *needle* is not found, strpos() will return boolean FALSE.
- The optional *offset* parameter allows you to specify which character in *haystack* to start searching. The position returned is still relative to the beginning of *haystack*.

Example:

```
$newstring = 'abcdef abcdef';
$pos = strpos($newstring, 'a'); // $pos = 0
$pos = strpos($newstring, 'a', 1); // $pos = 7, not 0
```

❖ **substr_count**

- substr_count -- Count the number of substring occurrences

Syntax:

```
int substr_count ( string haystack, string needle [, int offset [, int length]] )
```

- substr_count() returns the number of times the *needle* substring occurs in the *haystack* string.
- *needle* is case sensitive.

Parameters

- *haystack* : The string to search in
- *needle* : The substring to search for
- *offset* : The offset where to start counting
- *length* : The maximum length after the specified offset to search for the substring. It outputs a warning if the offset plus the length is greater than the *haystack* length.
- **Return Values :**This functions returns an integer.

- **Examples**

```
<?php
        $text = 'This is a test';
        echo strlen($text); // 14

        echo substr_count($text, 'is'); // 2

        // the string is reduced to 's is a test', so it prints 1
        echo substr_count($text, 'is', 3);
?>
```

❖ **strrpos**

- strrpos -- Find position of last occurrence of a char in a string

Syntax:

```
int strrpos ( string haystack, string needle [, int offset] )
```

- Returns the numeric position of the last occurrence of *needle* in the *haystack* string. Note that the needle in this case can only be a single character in PHP 4. If a string is passed as the needle, then only the first character of that string will be used.
- If *needle* is not found, returns FALSE.

Example:

```
$newstring = 'abcdef abcdef';
$pos = strrpos($newstring, 'a'); // $pos = 7
```

❖ **strstr**

Syntax:

```
string strstr ( string haystack, string needle )
```

- Returns part of *haystack* string from the first occurrence of *needle* to the end of *haystack*.
- If *needle* is not found, returns FALSE.
- If *needle* is not a string, it is converted to an integer and applied as the ordinal value of a character.
- This function is case-sensitive.

Example:

```
$email = 'user@example.com';
$domain = strstr($email, '@');
echo $domain; // prints @example.com
```

❖ **stristr**

- stristr -- Case-insensitive strstr()

Syntax:

```
string stristr ( string haystack, string needle )
```

- Returns all of *haystack* from the first occurrence of *needle* to the end. *needle* and *haystack* are examined in a case-insensitive manner.
- If *needle* is not found, returns FALSE.
- If *needle* is not a string, it is converted to an integer and applied as the ordinal value of a character.

Example

```
 $email = 'USER@EXAMPLE.com';
 echo stristr($email, 'e');
// outputs ER@EXAMPLE.com
```

❖ **strrchr**

- strrchr -- Find the last occurrence of a character in a string

Syntax:

string strrchr (string haystack, string needle)

- This function returns the portion of *haystack* which starts at the last occurrence of *needle* and goes until the end of *haystack*.
- Returns FALSE if *needle* is not found.
- If *needle* contains more than one character, only the first is used in PHP 4. This behavior is different from that of strchr().
- If *needle* is not a string, it is converted to an integer and applied as the ordinal value of a character.

Example

```
echo strrchr("WelCome To RPBC",'o'); // o RPBC
```

❖ **str_replace**

- str_replace -- Replace all occurrences of the search string with the replacement string

Syntax:

mixed str_replace (mixed search, mixed replace, mixed subject [, int &count])

- This function returns a string or an array with all occurrences of *search* in *subject* replaced with the given *replace* value.
- Every parameter in str_replace() can be an array.
- If *subject* is an array, then the search and replace is performed with every entry of *subject*, and the return value is an array as well.
- If *search* and *replace* are arrays, then str_replace() takes a value from each array and uses them to do search and replace on *subject*.
- If *replace* has fewer values than *search*, then an empty string is used for the rest of replacement values.
- If *search* is an array and *replace* is a string, then this replacement string is used for every value of *search*.

Example:

```
<?php
    // Provides: <body text='black'>
    $bodytag = str_replace("%body%", "black", "<body text='%body%'>");

    // Provides: Hll Wrld f PHP
    $vowels = array("a", "e", "i", "o", "u", "A", "E", "I", "O", "U");
    $onlyconsonants = str_replace($vowels, "", "Hello World of PHP");

    // Provides: You should eat pizza, beer, and ice cream every day
    $phrase  = "You should eat fruits, vegetables, and fiber every day.";
    $healthy = array("fruits", "vegetables", "fiber");
    $yummy   = array("pizza", "beer", "ice cream");

    $newphrase = str_replace($healthy, $yummy, $phrase);

    // Use of the count parameter is available as of PHP 5.0.0
    $str = str_replace("ll", "", "good golly miss molly!", $count);
    echo $count; // 2
?>
```

❖ **str_ireplace**

- str_ireplace -- Case-insensitive version of str_replace().

❖ **substr_replace**

- substr_replace -- Replace text within a portion of a string

Syntax:

mixed substr_replace (mixed string, string replacement, int start [, int length])

- substr_replace() replaces a copy of *string* delimited by the *start* and (optionally) *length* parameters with the string given in *replacement*. The result string is returned. If *string* is an array then array is returned.
- If *start* is positive, the replacing will begin at the *start*'th offset into *string*.
- If *start* is negative, the replacing will begin at the *start*'th character from the end of *string*.
- If *length* is given and is positive, it represents the length of the portion of *string* which is to be replaced. If it is negative, it represents the number of characters from the end of *string* at which to stop replacing. If it is not given, then it will default to strlen(*string*); i.e. end the replacing at the end of *string*.

Example:

```
<?php
    $var = 'ABCDEFGH:/MNRPQR/';
    echo "Original: $var<hr />\n";

    /* These two examples replace all of $var with 'bob'. */
    echo substr_replace($var, 'bob', 0) . "<br />\n";
    echo substr_replace($var, 'bob', 0, strlen($var)) . "<br />\n";

    /* Insert 'bob' right at the beginning of $var. */
    echo substr_replace($var, 'bob', 0, 0) . "<br />\n";

    /* These next two replace 'MNRPQR' in $var with 'bob'. */
    echo substr_replace($var, 'bob', 10, -1) . "<br />\n";
    echo substr_replace($var, 'bob', -7, -1) . "<br />\n";

    /* Delete 'MNRPQR' from $var. */
    echo substr_replace($var, '', 10, -1) . "<br />\n";
?>
```

❖ **strrev**

- strrev -- Reverse a string

Syntax:

string strrev (string string)

Example:

```
echo strrev("Hello world!"); // outputs "!dlrow olleH"
```

❖ **strval**

- strval -- Get string value of a variable

Syntax:

string strval (mixed var)

- Returns the string value of *var*. See the documentation on string for more information on converting to string.
- *var* may be any scalar type. You cannot use strval() on arrays or objects.

❖ **addslashes**

- addslashes -- Quote string with slashes

Syntax:

string addslashes (string str)

- Returns a string with backslashes before characters that need to be quoted in database queries etc. These characters are single quote ('), double quote ("), backslash (\) and NUL (the NULL byte).

Example:

```
$str = "Is your name O'reilly?";
// Outputs: Is your name O\'reilly?
echo addslashes($str);
```

❖ **quotemeta**

- quotemeta -- Quote meta characters

Synatx:

string quotemeta (string str)

- Returns a version of str with a backslash character (\) before every character that is among these:

. \ + * ? [^] ($)

Example:

```
$str = "this characters ( $ , * )  are very special to me \n”
// Outputs: this characters \( \$ , \* \)  are very special to me \\n
echo addslashes($str);
```

❖ **stripslashes**

- stripslashes -- Un-quote string quoted with addslashes()

Syntax:

string stripslashes (string str)

- Returns a string with backslashes stripped off. (\' becomes ' and so on.) Double backslashes (\\) are made into a single backslash (\).

Example:

```
$str = "Is your name O\'reilly?";
// Outputs: Is your name O'reilly?
echo stripslashes($str);
```

❖ **echo**

- echo -- Output one or more strings

Syntax:

void echo (string arg1 [, string ...])

- Outputs all parameters.
- echo() is not actually a function (it is a language construct), so you are not required to use parentheses with it.
- echo() (unlike some other language constructs) does not behave like a function, so it cannot always be used in the context of a function.
- Additionally, if you want to pass more than one parameter to echo(), the parameters must not be enclosed within parentheses.

Example :

```
echo "Hello World";
echo "This spans
multiple lines. The newlines will be
output as well";
echo "This spans\nmultiple lines. The newlines will be\noutput as well.";
```

- echo() also has a shortcut syntax, where you can immediately follow the opening tag with an equals sign. This short syntax only works with the **short_open_tag** configuration setting enabled.

```
I have <?=$foo?> foo.
```

❖ **print**

- print -- Output a string

Syntax:

```
int print ( string arg )
```

- Outputs *arg*. Returns *1*, always.
- print() is not actually a real function (it is a language construct) so you are not required to use parentheses with its argument list.

Example:

```
print("Hello World");
print "print() also works without parentheses.";
print "This spans
multiple lines. The newlines will be
output as well";

print "This spans\nmultiple lines. The newlines will be\noutput as well.";
print "escaping characters is done \"Like this\".";
```

❖ **Difference Between PRINT & ECHO**

- Unlike echo, print can accept only one argument.

```
echo "Hello","How Are You?";        // valid
print "Hello","How Are You?";        // invalid
```

- Unlike echo, print return a value, which represents whether the print statement succeeded. The value return 1 if the printing was successfully and 0 if unsuccessfully.

```
$a=print ("Hello");
print $a; // display 1
```

MATH Function

❖ **abs**

- abs -- Absolute value

Syntax:

```
number abs ( mixed number )
```

- Returns the absolute value of *number*.
- If the argument *number* is of type float, the return type is also float, otherwise it is integer (as float usually has a bigger value range than integer).

Example:

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

❖ **ceil**

- ceil -- Round fractions up

Syntax:

```
float ceil ( float value )
```

- Returns the next highest integer value by rounding up *value* if necessary.
- The return value of ceil() is still of type float as the value range of float is usually bigger than that of integer.

Example:

```
echo ceil(4.3);    // 5
echo ceil(9.999);  // 10
```

❖ **floor**

- floor -- Round fractions down

Syntax:

```
float floor ( float value )
```

- Returns the next lowest integer value by rounding down *value* if necessary.
- The return value of floor() is still of type float because the value range of float is usually bigger than that of integer.

Example:

```
echo floor(4.3);   // 4
echo floor(9.999); // 9
```

❖ **round**

- round -- Rounds a float

Syntax:

```
float round ( float val [, int precision] )
```

- Returns the rounded value of *val* to specified *precision* (number of digits after the decimal point).
- *precision* can also be negative or zero (default).

Example:

```
echo round(3.4);         // 3
echo round(3.5);         // 4
echo round(3.6);         // 4
echo round(3.6, 0);      // 4
echo round(1.95583, 2);  // 1.96
echo round(5.045, 2);    // 5.05
echo round(1241757, -3); // 1242000
```

❖ **fmod**

- fmod -- Returns the floating point remainder (modulo) of the division of the arguments

Syntax:

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

- Returns the floating point remainder of dividing the dividend (*x*) by the divisor (*y*). The reminder (*r*) is defined as: x = i * y + r, for some integer *i*. If *y* is non-zero, *r* has the same sign as *x* and a magnitude less than the magnitude of *y*.

Example:

```
$x = 5.7;
$y = 1.3;
$r = fmod($x, $y);
// $r equals 0.5, because 4 * 1.3 + 0.5 = 5.7
```

❖ **min**

- min -- Find lowest value

Syntax:

```
mixed min ( number arg1, number arg2 [, number ...] )
mixed min ( array numbers )
```

- min() returns the numerically lowest of the parameter values.

- If the first and only parameter is an array, min() returns the lowest value in that array.
- If the first parameter is an integer, string or float, you need at least two parameters and min() returns the smallest of these values.
- You can compare an unlimited number of values.
- PHP will evaluate a non-numeric string as *0*, but still return the string if it's seen as the numerically lowest value. If multiple arguments evaluate to *0*, min() will use the first one it sees (the leftmost value).

Example:

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

echo min(0, 'hello');     // 0
echo min('hello', 0);     // hello
echo min('hello', -1);    // -1

// With multiple arrays, min compares from left to right
// so in our example: 2 == 2, but 4 < 5
$val = min(array(2, 4, 8), array(2, 5, 1)); // array(2, 4, 8)

// If both an array and non-array are given, the array is  never  returned  as
it's  considered the largest
$val = min('string', array(2, 5, 7), 42);   // string
?>
```

❖ **max**

- max -- Find highest value

Syntax:

```
mixed max ( number arg1, number arg2 [, number ...] )
mixed max ( array numbers )
```

- max() returns the numerically highest of the parameter values.
- If the first and only parameter is an array, max() returns the highest value in that array.
- If the first parameter is an integer, string or float, you need at least two parameters and max() returns the biggest of these values.
- You can compare an unlimited number of values.
- PHP will evaluate a non-numeric string as *0*, but still return the string if it's seen as the numerically highest value. If multiple arguments evaluate to *0*, max() will use the first one it sees (the leftmost value).

Example:

```
echo max(1, 3, 5, 6, 7);  // 7
echo max(array(2, 4, 5)); // 5
echo max(0, 'hello');     // 0
echo max('hello', 0);     // hello
echo max(-1, 'hello');    // hello

// With multiple arrays, max compares from left to right
// so in our example: 2 == 2, but 4 < 5
$val = max(array(2, 4, 8), array(2, 5, 7)); // array(2, 5, 7)

// If both an array and non-array are given, the array is always returned as
it's seen as the largest
$val = max('string', array(2, 5, 7), 42);   // array(2, 5, 7)
```

❖ **pow**

- pow -- Exponential expression

Syntax:

number pow (number base, number exp)

- Returns *base* raised to the power of *exp*. If possible, this function will return an integer.
- If the power cannot be computed, a warning will be issued, and pow() will return FALSE.
- PHP cannot handle negative *base*s.

Example:

```
echo pow(-1, 20); // 1
echo pow(0, 0); // 1
echo pow(-1, 5.5); // error
```

❖ **sqrt**

- sqrt -- Square root

Syntax:

float sqrt (float arg)

- Returns the square root of *arg*.

Example:

```
echo sqrt(9); // 3
echo sqrt(10); // 3.16227766 ...
```

❖ **exp**

- exp -- Calculates the exponent of e (the Neperian or Natural logarithm base)

Syntax:

float exp (float arg)

- Returns e raised to the power of *arg*.
- 'e' is the base of the natural system of logarithms, or approximately 2.718282.

Example:

```
echo exp(12) ;  // 1.6275E+005
echo exp(5.7);  //  298.87
```

❖ **rand**

- rand -- Generate a random integer

Syntax:

int rand ([int min, int max])

- If called without the optional *min*, *max* arguments rand() returns a pseudo-random integer between 0 and RAND_MAX. If you want a random number between 5 and 15 (inclusive), for example, use *rand (5, 15).*

Example:

```
echo rand() ;
echo rand() ;
echo rand(5, 15);
```

The above example will output something similar to:

```
7771
22264
11
```

❖ **base_convert**

- base_convert -- Convert a number between arbitrary bases

Syntax:

string base_convert (string number, int frombase, int tobase)

- Returns a string containing *number* represented in base *tobase*. The base in which *number* is given is specified in *frombase*.
- Both *frombase* and *tobase* have to be between 2 and 36, inclusive.
- Digits in numbers with a base higher than 10 will be represented with the letters a-z, with a meaning 10, b meaning 11 and z meaning 35.

Example:

```
$hexadecimal = 'A37334';
echo base_convert($hexadecimal, 16, 2);  // 101000110111001100110100
```

❖ **bindec**

- bindec -- Binary to decimal

Syntax:

number bindec (string binary_string)

- Returns the decimal equivalent of the binary number represented by the *binary_string* argument.

Example:

```
echo bindec('110011') ; // 51
echo bindec('000110011') ; // 51
```

❖ **octdec**

- octdec -- Octal to decimal

❖ **hexdec**

- hexdec – Hexa-Decimal to decimal

❖ **decbin**

- decbin – Decimal to Binary

❖ **decoct**

- decoct – Decimal to Octal

❖ **dechex**

- dechex – Decimal to Hexa-Decimal

Date and Time Functions

❖ **date**

- date -- Format a local time/date

Syntax:

string date (string format [, int timestamp])

- Returns a string formatted according to the given format string using the given integer *timestamp* or the current local time if no timestamp is given.
- In other words, *timestamp* is optional and defaults to the value of time().
- The valid range of a timestamp is typically from Fri, 13 Dec 1901 20:45:54 GMT to Tue, 19 Jan 2038 03:14:07 GMT. (These are the dates that correspond to the minimum and maximum values for a 32-bit signed integer).
- The following characters are recognized in the *format* parameter string

format character	Description	Example returned values
Day	---	---
d	Day of the month, 2 digits with leading zeros	01 to 31
D	A textual representation of a day, three letters	Mon through Sun
j	Day of the month without leading zeros	1 to 31
l (lowercase 'L')	A full textual representation of the day of the week	Sunday through Saturday
N	ISO-8601 numeric representation of the day of the week (added in PHP 5.1.0)	1 (for Monday) through 7 (for Sunday)
S	English ordinal suffix for the day of the month, 2 characters	st, nd, rd or th. Works well with j
w	Numeric representation of the day of the week	0 (for Sunday) through 6 (for Saturday)
z	The day of the year (starting from 0)	0 through 365
Week	---	---
W	ISO-8601 week number of year, weeks starting on Monday (added in PHP 4.1.0)	Example: 42 (the 42nd week in the year)
Month	---	---
F	A full textual representation of a month, such as January or March	January through December
m	Numeric representation of a month, with leading zeros	01 through 12
M	A short textual representation of a month, three letters	Jan through Dec
n	Numeric representation of a month, without leading zeros	1 through 12
T	Number of days in the given month	28 through 31
Year	---	---
L	Whether it's a leap year	1 if it is a leap year, 0 otherwise.
O	ISO-8601 year number. This has the same value as Y, except that if the ISO week number (W) belongs to the previous or next year, that year is used instead. (added in PHP 5.1.0)	Examples: 1999 or 2003

format character	Description	Example returned values
Y	A full numeric representation of a year, 4 digits	Examples: 1999 or 2003
Y	A two digit representation of a year	Examples: 99 or 03
Time	---	---
A	Lowercase Ante meridiem and Post meridiem	am or pm
A	Uppercase Ante meridiem and Post meridiem	AM or PM
B	Swatch Internet time	000 through 999
G	12-hour format of an hour without leading zeros	1 through 12
G	24-hour format of an hour without leading zeros	0 through 23
H	12-hour format of an hour with leading zeros	01 through 12
H	24-hour format of an hour with leading zeros	00 through 23
I	Minutes with leading zeros	00 to 59
S	Seconds, with leading zeros	00 through 59
Timezone	---	---
E	Timezone identifier (added in PHP 5.1.0)	Examples: UTC, GMT, Atlantic/Azores
I (capital i)	Whether or not the date is in daylights savings time	1 if Daylight Savings Time, 0 otherwise.
O	Difference to Greenwich time (GMT) in hours	Example: +0200
T	Timezone setting of this machine	Examples: EST, MDT ...
Z	Timezone offset in seconds. The offset for timezones west of UTC is always negative, and for those east of UTC is always positive.	-43200 through 43200
Full Date/Time	---	---
C	ISO 8601 date (added in PHP 5)	2004-02-12T15:19:21+00:00
R	RFC 2822 formatted date	Example: Thu, 21 Dec 2000 16:01:07 +0200
U	Seconds since the Unix Epoch (January 1 1970 00:00:00 GMT)	See also time()

Example:

```
// set the default timezone to use. Available since PHP 5.1
date_default_timezone_set('UTC');
// Prints something like: Monday
echo date("l");
// Prints something like: Monday 15th of August 2005 03:12:46 PM
echo date('l dS \of F Y h:i:s A');
```

- You can prevent a recognized character in the format string from being expanded by escaping it with a preceding backslash. If the character with a backslash is already a special sequence, you may need to also escape the backslash.

  Example:

```
// prints something like: Wednesday the 15th
echo date("l \\t\h\e jS");
```

- Some examples of date() formatting. Note that you should escape any other characters, as any which currently have a special meaning will produce undesirable results, and other characters may be assigned meaning in future PHP versions. When escaping, be sure to use single quotes to prevent characters like \n from becoming newlines.
- **Example:**

```
<?php
    // Assuming today is: March 10th, 2001, 5:16:18 pm
    $today = date("F j, Y, g:i a");                 // March 10, 2001, 5:16 pm
    $today = date("m.d.y");                         // 03.10.01
    $today = date("j, n, Y");                       // 10, 3, 2001
    $today = date("Ymd");                           // 20010310
    $today = date('h-i-s, j-m-y, it is w Day z ');  // 05-16-17, 10-03-
    01, 1631 1618 6 Fripm01
    $today = date('\i\t \i\s \t\h\e jS \d\a\y.');   // It is the 10th day.
    $today = date("D M j G:i:s T Y");               // Sat Mar 10 15:16:08 MS
    T 2001
    $today = date('H:m:s \m \i\s\ \m\o\n\t\h');     // 17:03:17 m is month
    $today = date("H:i:s");                         // 17:16:17
?>
```

❖ **getdate**

- getdate -- Get date/time information

Syntax:

```
array getdate ( [int timestamp] )
```

- Returns an associative array containing the date information of the *timestamp*, or the current local time if no *timestamp* is given, as the following associative array elements:
- Key elements of the returned associative array

Key	Description	Example returned values
"seconds"	Numeric representation of seconds	*0* to *59*
"minutes"	Numeric representation of minutes	*0* to *59*
"hours"	Numeric representation of hours	*0* to *23*
"mday"	Numeric representation of the day of the month	*1* to *31*
"wday"	Numeric representation of the day of the week	*0* (for Sunday) through *6* (for Saturday)
"mon"	Numeric representation of a month	*1* through *12*
"year"	A full numeric representation of a year, 4 digits	Examples: *1999* or *2003*
"yday"	Numeric representation of the day of the year	*0* through *365*
"weekday"	A full textual representation of the day of the week	*Sunday* through *Saturday*
"month"	A full textual representation of a month, such as January or March	*January* through *December*
0	Seconds since the Unix Epoch, similar to the values returned by time() and used by date().	System Dependent, typically -*2147483648* through *2147483647*.

Example:

```
<?php
        $today = getdate();
        print_r($today);
?>
```

Output:

```
Array
(
   [seconds] => 40
   [minutes] => 58
   [hours]   => 21
   [mday]    => 17
   [wday]    => 2
   [mon]     => 6
   [year]    => 2003
   [yday]    => 167
   [weekday] => Tuesday
   [month]   => June
   [0]       => 1055901520
)
```

❖ **checkdate**

- checkdate -- Validate a Gregorian date

Syntax:

bool checkdate (int month, int day, int year)

- Returns TRUE if the date given is valid; otherwise returns FALSE. Checks the validity of the date formed by the arguments.
- A date is considered valid if:

  year is between 1 and 32767 inclusive

  month is between 1 and 12 inclusive

  Day is within the allowed number of days for the given *month*.

  Leap *year*s are taken into consideration.

Example:

```
<?php
        var_dump(checkdate(12, 31, 2000));  // bool(true)
        var_dump(checkdate(2, 29, 2001));    // bool(false)
?>
```

❖ **gmdate**

- gmdate -- Format a GMT/UTC date/time

Syntax:

string gmdate (string format [, int timestamp])

- Identical to the date() function except that the time returned is Greenwich Mean Time (GMT).
- For example, when run in Finland (GMT +0200), the first line below prints "Jan 01 1998 00:00:00", while the second prints "Dec 31 1997 22:00:00".

Example:

```
<?php
    echo date("M d Y H:i:s", mktime(0, 0, 0, 1, 1, 1998));
    echo gmdate("M d Y H:i:s", mktime(0, 0, 0, 1, 1, 1998));
?>
```

❖ **time**

- time -- Return current Unix timestamp

Syntax:

int time (void)

- Returns the current time measured in the number of seconds since the Unix Epoch (January 1 1970 00:00:00 GMT).

Example:

```
<?php
        $nextWeek = time() + (7 * 24 * 60 * 60);
          // 7 days; 24 hours; 60 mins; 60secs
        echo 'Now:       '. date('Y-m-d') ."\n";
        echo 'Next Week: '. date('Y-m-d', $nextWeek) ."\n";
?>
```

Output:

```
Now:       2005-03-30
Next Week: 2005-04-07
```

❖ **microtime**

- microtime -- Return current Unix timestamp with microseconds

Syntax:

mixed microtime ([bool get_as_float])

- microtime() returns the current Unix timestamp with microseconds.
- This function is only available on operating systems that support the gettimeofday() system call.
- When called without the optional argument, this function returns the string "msec sec" where sec is the current time measured in the number of seconds since the Unix Epoch (0:00:00 January 1, 1970 GMT), and msec is the microseconds part. Both portions of the string are returned in units of seconds.
- When *get_as_float* is given, and evaluates to TRUE, microtime() will return a float.

Example:

```
<?php
        $time_start = microtime(true);
        // Sleep for a while
        usleep(100);

        $time_end = microtime(true);
        $time = $time_end - $time_start;

        echo "Did nothing in $time seconds\n";
?>
```

❖ **localtime**

- localtime -- Get the local time

Syntax:

array localtime ([int timestamp [, bool is_associative]])

- The localtime() function returns an array identical to that of the structure returned by the C function call.
- The first argument to localtime() is the timestamp, if this is not given the current time as returned from time() is used.

- The second argument to the localtime() is the *is_associative*, if this is set to FALSE or not supplied than the array is returned as a regular, numerically indexed array. If the argument is set to TRUE then localtime() is an associative array containing all the different elements of the structure returned by the C function call to localtime.
- The names of the different keys of the associative array are as follows:
    - "tm_sec" - seconds
    - "tm_min" - minutes
    - "tm_hour" - hour
    - "tm_mday" - day of the month
    - "tm_mon" - month of the year, starting with 0 for January
    - "tm_year" - Years since 1900
    - "tm_wday" - Day of the week
    - "tm_yday" - Day of the year
    - "tm_isdst" - Is daylight savings time in effect
- Months are from 0 (Jan) to 11 (Dec) and days of the week are from 0 (Sun) to 6 (Sat).

Example:

```
<?php
        $localtime = localtime();
        $localtime_assoc = localtime(time(), true);
        print_r($localtime);
        print_r($localtime_assoc);
?>
```

Output:

```
Array
(
   [0] => 24
   [1] => 3
   [2] => 19
   [3] => 3
   [4] => 3
   [5] => 105
   [6] => 0
   [7] => 92
   [9] => 1
)

Array
(
   [tm_sec] => 24
   [tm_min] => 3
   [tm_hour] => 19
   [tm_mday] => 3
   [tm_mon] => 3
   [tm_year] => 105
   [tm_wday] => 0
   [tm_yday] => 92
   [tm_isdst] => 1
)
```

❖ **mktime**

- mktime -- Get Unix timestamp for a date

Syntax:

int mktime ([int hour [, int minute [, int second [, int month [, int day [, int year [, int is_dst]]]]]]])

- Returns the Unix timestamp corresponding to the arguments given. This timestamp is a long integer containing the number of seconds between the Unix Epoch (January 1 1970 00:00:00 GMT) and the time specified.
- Arguments may be left out in order from right to left; any arguments thus omitted will be set to the current value according to the local date and time.
- **Parameters**
- *hour* : The number of the hour.
- *Minute :* The number of the minute.
- *second* :The number of seconds past the minute.
- *month* :The number of the month.
- *day* :The number of the day.
- *year* :

  The number of the year, may be a two or four digit value, with values between 0-69 mapping to 2000-2069 and 70-100 to 1970-2000. On systems where time_t is a 32bit signed integer, as most common today, the valid range for *year* is somewhere between 1901 and 2038, although this limitation is overcome as of PHP 5.1.0.
- *is_dst* :

  This parameter can be set to 1 if the time is during daylight savings time (DST), 0 if it is not, or -1 (the default) if it is unknown whether the time is within daylight savings time or not. If it's unknown, PHP tries to figure it out itself. This can cause unexpected (but not incorrect) results. Some times are invalid if DST is enabled on the system PHP is running on or *is_dst* is set to 1. If DST is enabled in e.g. 2:00, all times between 2:00 and 3:00 are invalid and mktime() returns an undefined (usually negative) value. Some systems (e.g. Solaris 8) enable DST at midnight so time 0:30 of the day when DST is enabled is evaluated as 23:30 of the previous day.
- **Return Values**

  mktime() returns the Unix timestamp of the arguments given. If the arguments are invalid (eg. if the year, month and day are all 0), the function returns FALSE (before PHP 5.1 it returned *-1*).

Examples

- each of the following lines produces the string "Jan-01-1998".

```
<?php
        echo date("M-d-Y", mktime(0, 0, 0, 12, 32, 1997));
        echo date("M-d-Y", mktime(0, 0, 0, 13, 1, 1997));
        echo date("M-d-Y", mktime(0, 0, 0, 1, 1, 1998));
        echo date("M-d-Y", mktime(0, 0, 0, 1, 1, 98));
?>
```

ARRAY FUNCTIONS

❖ **list**

- list -- Assign variables as if they were an array

Syntax:

void list (mixed varname, mixed ...)

- Like array(), this is not really a function, but a language construct.
- list() is used to assign a list of variables in one operation.
- list() only works on numerical arrays and assumes the numerical indices start at 0.

Example:

```
<?php
    $info = array('coffee', 'brown', 'caffeine');
    // Listing all the variables
    list($drink, $color, $power) = $info;
    echo "$drink is $color and $power makes it special.\n";
    // Listing some of them
    list($drink, , $power) = $info;
    echo "$drink has $power.\n";
    // Or let's skip to only the third one
    list( , , $power) = $info;
    echo "I need $power!\n";
?>
```

❖ **is_array**

- is_array -- Finds whether a variable is an array

Syntax:

bool is_array (mixed var)

- **Parameters**
- *var* : The variable being evaluated.
- **Return Values**
- Returns TRUE if *var* is an array, FALSE otherwise.

Examples

```
<?php
    $yes = array('this', 'is', 'an array');
    echo is_array($yes) ? 'Array' : 'not an Array';
    $no = 'this is a string';
    echo is_array($no) ? 'Array' : 'not an Array';
?>
```

❖ **count**

- count -- Count elements in an array, or properties in an object

Syntax:

int count (mixed var [, int mode])

- Returns the number of elements in *var*, which is typically an array, since anything else will have one element.
- If *var* is not an array or an object with implemented *Countable* interface, *1* will be returned. There is one exception, if *var* is NULL, *0* will be returned.
- If the optional *mode* parameter is set to COUNT_RECURSIVE (or 1), count() will recursively count the array. This is particularly useful for counting all the elements of a multidimensional array. The default value for *mode* is *0*. count() does not detect infinite recursion.

- Please see the Array section of the manual for a detailed explanation of how arrays are implemented and used in PHP.

Example:

```
<?php
        $a[0] = 1;
        $a[1] = 3;
        $a[2] = 5;
        $result = count($a); // $result == 3
        $b[0]  = 7;
        $b[5]  = 9;
        $b[10] = 11;
        $result = count($b);  // $result == 3
        $result = count(null); // $result == 0
        $result = count(false); // $result == 1
?>
```

❖ **sizeof**

- sizeof -- Alias of count()

❖ **in_array**

- in_array -- Checks if a value exists in an array

Syntax:

```
bool in_array ( mixed needle, array haystack [, bool strict] )
```

- Searches *haystack* for *needle* and returns TRUE if it is found in the array, FALSE otherwise.
- If the third parameter *strict* is set to TRUE then the in_array() function will also check the types of the *needle* in the *haystack*.
- If *needle* is a string, the comparison is done in a case-sensitive manner.

Example:

```
<?php
        $os = array("Mac", "NT", "Irix", "Linux");
        if (in_array("Irix", $os)) {
           echo "Got Irix";
        }
        if (in_array("mac", $os)) {
           echo "Got mac";
        }
        $a = array('1.10', 12.4, 1.13);
        if (in_array('12.4', $a, true)) {
           echo "'12.4' found with strict check\n";
        }
        if (in_array(1.13, $a, true)) {
           echo "1.13 found with strict check\n";
        }
?>
```

Output:

```
Got Irix
1.13 found with strict check
```

❖ **unset**

- unset -- Unset a given variable

Syntax:

void unset (mixed var [, mixed var [, mixed ...]])

- unset() destroys the specified variables.

Example:

```
<?php
    // destroy a single variable
    unset($foo);
    // destroy a single element of an array
    unset($bar['quux']);
    // destroy more than one variable
    unset($foo1, $foo2, $foo3);
?>
```

❖ **current**

- current -- Return the current element in an array

Syntax:

mixed current (array &array)

- Every array has an internal pointer to its "current" element, which is initialized to the first element inserted into the array.
- The current() function simply returns the value of the array element that's currently being pointed to by the internal pointer. It does not move the pointer in any way.
- If the internal pointer points beyond the end of the elements list, current() returns FALSE.

❖ **pos**

- pos -- Alias of current()

❖ **next**

- next -- Advance the internal array pointer of an array

Syntax:

mixed next (array &array)

- Returns the array value in the next place that's pointed to by the internal array pointer, or FALSE if there are no more elements.

❖ **prev**

- prev -- Rewind the internal array pointer

Syntax:

mixed prev (array &array)

- Returns the array value in the previous place that's pointed to by the internal array pointer, or FALSE if there are no more elements.

❖ **reset**

- reset -- Set the internal pointer of an array to its first element

Syntax:

mixed reset (array &array)

- reset() rewinds *array*'s internal pointer to the first element and returns the value of the first array element, or FALSE if the array is empty.

❖ **end**

- end -- Set the internal pointer of an array to its last element

Syntax:

mixed end (array &array)

- end() advances *array*'s internal pointer to the last element, and returns its value.

Example:

```
<?php
    $transport = array('foot', 'bike', 'car', 'plane');
    $mode = current($transport); // $mode = 'foot';
    $mode = next($transport);    // $mode = 'bike';
    $mode = next($transport);    // $mode = 'car';
    $mode = prev($transport);    // $mode = 'bike';
    $mode = end($transport);     // $mode = 'plane';
    $mode = reset($transport);     // $mode = 'foot';
?>
```

❖ **each**

- each -- Return the current key and value pair from an array and advance the array cursor

Syntax:

array each (array &array)

- Returns the current key and value pair from the array *array* and advances the array cursor. This pair is returned in a four-element array, with the keys *0*, *1*, *key*, and *value*. Elements *0* and *key* contain the key name of the array element, and *1* and *value* contain the data.
- If the internal pointer for the array points past the end of the array contents, each() returns FALSE.

Example:

```
$foo = array("bob", "fred", "jussi", "jouni", "egon", "marliese");
$bar = each($foo);
print_r($bar);
```

Output:

```
Array
(
   [1] => bob
   [value] => bob
   [0] => 0
   [key] => 0
)
```

Example:

```
$foo = array("Robert" => "Bob", "Seppo" => "Sepi");
$bar = each($foo);
print_r($bar);
```

Output:

```
Array
(
   [1] => Bob
   [value] => Bob
   [0] => Robert
   [key] => Robert
)
```

Example:

```
$fruit = array('a' => 'apple', 'b' => 'banana', 'c' => 'cranberry');
reset($fruit);
while (list($key, $val) = each($fruit)) {
   echo "$key => $val\n";
}
```

Output:

```
a => apple
b => banana
c => cranberry
```

❖ **array_walk**

- array_walk -- Apply a user function to every member of an array

Syntax:

```
bool array_walk ( array &array, callback funcname [, mixed userdata] )
```

- Returns TRUE on success or FALSE on failure.
- Applies the user-defined function *funcname* to each element of the *array* array. Typically, *funcname* takes on two parameters. The *array* parameter's value being the first, and the key/index second. If the optional *userdata* parameter is supplied, it will be passed as the third parameter to the callback *funcname*.
- If *funcname* needs to be working with the actual values of the array, specify the first parameter of *funcname* as a reference. Then, any changes made to those elements will be made in the original array itself.
- array_walk() is not affected by the internal array pointer of *array*. array_walk() will walk through the entire array regardless of pointer position.
- Users may not change the array itself from the callback function. e.g. Add/delete elements, unset elements, etc.

Example:

```
$fruits = array("d" => "lemon", "a" => "orange", "b" => "banana", "c" => "apple");
function test_alter(&$item1, $key, $prefix)
{
   $item1 = "$prefix: $item1";
}
function test_print($item2, $key)
{
   echo "$key. $item2<br />\n";
}
echo "Before ...:\n";
array_walk($fruits, 'test_print');

array_walk($fruits, 'test_alter', 'fruit');
echo "... and after:\n";

array_walk($fruits, 'test_print');
```

Output :

```
Before ...:
d. lemon
a. orange
b. banana
c. apple
... and after:
d. fruit: lemon
a. fruit: orange
b. fruit: banana
c. fruit: apple
```

❖ **sort**

- sort -- Sort an array

Syntax:

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

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

Example:

```
<?php
    $fruits = array("lemon", "orange", "banana", "apple");
    sort($fruits);
    foreach ($fruits as $key => $val) {
       echo "fruits[" . $key . "] = " . $val . "\n";
    }
?>
```

Output:

```
fruits[0] = apple
fruits[1] = banana
fruits[2] = lemon
fruits[3] = orange
```

- The fruits have been sorted in alphabetical order.
- The optional second parameter *sort_flags* may be used to modify the sorting behavior using these values:
- **Sorting type flags:**
    - SORT_REGULAR - compare items normally (don't change types)
    - SORT_NUMERIC - compare items numerically
    - SORT_STRING - compare items as strings
    - SORT_LOCALE_STRING - compare items as strings, based on the current locale.

❖ **rsort**

- rsort -- Sort an array in reverse order

Syntax:

bool rsort (array &array [, int sort_flags])

- This function sorts an array in reverse order (highest to lowest).
- This function assigns new keys for the elements in *array*. It will remove any existing keys you may have assigned, rather than just reordering the keys.
- Returns TRUE on success or FALSE on failure.

Example:

```
<?php
    $fruits = array("lemon", "orange", "banana", "apple");
    rsort($fruits);
    foreach ($fruits as $key => $val) {
       echo "$key = $val\n";
    }
?>
```

Output:

```
0 = orange
1 = lemon
2 = banana
3 = apple
```

❖ **asort**

- asort -- Sort an array and maintain index association

Syntax:

bool asort (array &array [, int sort_flags])

- This function sorts an array such that array indices maintain their correlation with the array elements they are associated with.
- This is used mainly when sorting associative arrays where the actual element order is significant.
- Returns TRUE on success or FALSE on failure.

Example:

```
$fruits = array("d" =>"lemon", "a" =>"orange", "b" => "banana", "c" => "apple");
asort($fruits);
foreach ($fruits as $key => $val) {
   echo "$key = $val\n";
}
```

Output:

```
c = apple
b = banana
d = lemon
a = orange
```

❖ **arsort**

- arsort -- Sort an array in reverse order and maintain index association

❖ **key**

- key -- Fetch a key from an associative array

Syntax:

mixed key (array &array)

- key() returns the index element of the current array position.

Example:

```
<?php
        $array = array('fruit1' => 'apple', 'fruit2' => 'orange','fruit3' => 'grape',
                'fruit4' => 'apple','fruit5' => 'apple');
        // this cycle echoes all associative array key where value equals "apple"
        while ($fruit_name = current($array)) {
                if ($fruit_name == 'apple') {
                        echo key($array).'<br />';
                }
                next($array);
        }
?>
```

❖ **array_merge**

- array_merge -- Merge one or more arrays

Syntax:

array array_merge (array array1 [, array array2 [, array ...]])

- array_merge() merges the elements of one or more arrays together so that the values of one are appended to the end of the previous one. It returns the resulting array.

- If the input arrays have the same string keys, then the later value for that key will overwrite the previous one. If, however, the arrays contain numeric keys, the later value will **not** overwrite the original value, but will be appended.
- If only one array is given and the array is numerically indexed, the keys get reindexed in a continuous way.

Example:

```
<?php
        $array1 = array("color" => "red", 2, 4);
        $array2 = array("a", "b", "color" => "green", "shape" => "trapezoid", 4);
        $result = array_merge($array1, $array2);
        print_r($result);
?>
```

- **Output:**

```
Array
(
   [color] => green
   [0] => 2
   [1] => 4
   [2] => a
   [3] => b
   [shape] => trapezoid
   [4] => 4
)
```

❖ **array_combine**

- array_combine -- Creates an array by using one array for keys and another for its values

Syntax:

array array_combine (array keys, array values)

- Returns an array by using the values from the *keys* array as keys and the values from the *values* array as the corresponding values.
- Returns FALSE if the number of elements for each array isn't equal or if the arrays are empty.

Example:

```
<?php
        $a = array('green', 'red', 'yellow');
        $b = array('avocado', 'apple', 'banana');
        $c = array_combine($a, $b);
        print_r($c);
?>
```

Output:

```
Array
(
   [green]  => avocado
   [red]    => apple
   [yellow] => banana
)
```

❖ **array_values**

- array_values -- Return all the values of an array

Syntax:

array array_values (array input)

- array_values() returns all the values from the *input* array and indexes numerically the array.

Example:

```
<?php
    $array = array("size" => "XL", "color" => "gold");
    print_r(array_values($array));
?>
```

Output:

```
Array
(
   [0] => XL
   [1] => gold
)
```

❖ **array_keys**

- array_keys -- Return all the keys of an array

Syntax:

array array_keys (array input [, mixed search_value [, bool strict]])

- array_keys() returns the keys, numeric and string, from the *input* array.
- If the optional *search_value* is specified, then only the keys for that value are returned. Otherwise, all the keys from the *input* are returned. As of PHP 5, you can use *strict* parameter for comparison including type (===).

Example:

```
<?php
    $array = array(0 => 100, "color" => "red");
    print_r(array_keys($array));
?>
```

Output:

```
Array
(
   [0] => 0
   [1] => color
)
```

❖ **array_key_exists**

- array_key_exists -- Checks if the given key or index exists in the array

Syntax:

bool array_key_exists (mixed key, array search)

- array_key_exists() returns TRUE if the given *key* is set in the array. *key* can be any value possible for an array index. array_key_exists() also works on objects.

Example:

```
<?php
    $search_array = array('first' => 1, 'second' => 4);
    if (array_key_exists('first', $search_array)) {
       echo "The 'first' element is in the array";
    }
?>
```

❖ **array_reverse**

- array_reverse -- Return an array with elements in reverse order

Syntax:

```
array array_reverse ( array array [, bool preserve_keys] )
```

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

Example:

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

- This makes both *$result* and *$result_keyed* have the same elements, but note the difference between the keys. The printout of *$result* and *$result_keyed* will be:

```
Array
(
   [0] => Array
      (
         [0] => green
         [1] => red
      )

   [1] => 4
   [2] => php
)
Array
(
   [2] => Array
      (
         [0] => green
         [1] => red
      )

   [1] => 4
   [0] => php
)
```

❖ **array_push**

- array_push -- Push one or more elements onto the end of array

Syntax:

```
int array_push ( array &array, mixed var [, mixed ...] )
```

- array_push() treats *array* as a stack, and pushes the passed variables onto the end of *array*. The length of *array* increases by the number of variables pushed.
- Returns the new number of elements in the array.

Example:

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

Output:

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

❖ **array_pop**

- array_pop -- Pop the element off the end of array

Syntax:

```
mixed array_pop ( array &array )
```

- array_pop() pops and returns the last value of the *array*, shortening the *array* by one element. If *array* is empty (or is not an array), NULL will be returned.
- This function will reset() the array pointer after use.

Example:

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

Output:

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

File System Function

❖ **fopen**

- fopen -- Opens file or URL

Syntax:

resource fopen (string filename, string mode [, bool use_include_path])

- fopen() binds a named resource, specified by *filename*, to a stream. If *filename* is of the form "scheme://...", it is assumed to be a URL and PHP will search for a protocol handler (also known as a wrapper) for that scheme. If no wrappers for that protocol are registered, PHP will emit a notice to help you track potential problems in your script and then continue as though *filename* specifies a regular file.
- If PHP has decided that *filename* specifies a local file, then it will try to open a stream on that file. The file must be accessible to PHP, so you need to ensure that the file access permissions allow this access.
- The *mode* parameter specifies the type of access you require to the stream. It may be any of the following:
- **A list of possible modes for fopen() using *mode***

mode	Description
'r'	Open for reading only; place the file pointer at the beginning of the file.
'r+'	Open for reading and writing; place the file pointer at the beginning of the file.
'w'	Open for writing only; place the file pointer at the beginning of the file and truncate the file to zero length. If the file does not exist, attempt to create it.
'w+'	Open for reading and writing; place the file pointer at the beginning of the file and truncate the file to zero length. If the file does not exist, attempt to create it.
'a'	Open for writing only; place the file pointer at the end of the file. If the file does not exist, attempt to create it.
'a+'	Open for reading and writing; place the file pointer at the end of the file. If the file does not exist, attempt to create it.
'x'	Create and open for writing only; place the file pointer at the beginning of the file. If the file already exists, the fopen() call will fail by returning FALSE and generating an error of level E_WARNING. If the file does not exist, attempt to create it. This is equivalent to specifying O_EXCL|O_CREAT flags for the underlying open(2) system call. This option is supported in PHP 4.3.2 and later, and only works for local files.
'x+'	Create and open for reading and writing; place the file pointer at the beginning of the file. If the file already exists, the fopen() call will fail by returning FALSE and generating an error of level E_WARNING. If the file does not exist, attempt to create it. This is equivalent to specifying O_EXCL|O_CREAT flags for the underlying open(2) system call. This option is supported in PHP 4.3.2 and later, and only works for local files.

- The optional third *use_include_path* parameter can be set to '1' or TRUE if you want to search for the file in the include_path, too.
- If the open fails, the function returns FALSE and an error of level E_WARNING is generated. You may use @ to suppress this warning.

Example:

```
<?php
        $handle = fopen("/home/rasmus/file.txt", "r");
        $handle = fopen("/home/rasmus/file.gif", "wb");
        $handle = fopen("http://www.example.com/", "r");
        $handle = fopen("ftp://user:password@example.com/somefile.txt", "w");
?>
```

- If you are experiencing problems with reading and writing to files and you're using the server module version of PHP, remember to make sure that the files and directories you're using are accessible to the server process.
- On the Windows platform, be careful to escape any backslashes used in the path to the file, or use forward slashes.

```
<?php
        $handle = fopen("c:\\data\\info.txt", "r");
?>
```

❖ **fread**

- fread -- Binary-safe file read

Syntax:

```
string fread ( resource handle, int length )
```

- fread() reads up to *length* bytes from the file pointer referenced by *handle*. Reading stops when *length* bytes have been read, EOF (end of file) is reached, or (for network streams) when a packet becomes available, whichever comes first.

Example:

```
<?php
        // get contents of a file into a string
        $filename = "/usr/local/something.txt";
        $handle = fopen($filename, "r");
        $contents = fread($handle, filesize($filename));
        fclose($handle);
?>
```

❖ **fwrite**

- fwrite -- Binary-safe file write

Syntax:

```
int fwrite ( resource handle, string string [, int length] )
```

- fwrite() writes the contents of *string* to the file stream pointed to by *handle*. If the *length* argument is given, writing will stop after *length* bytes have been written or the end of *string* is reached, whichever comes first.
- fwrite() returns the number of bytes written, or FALSE on error.
- if the *length* argument is given, then the magic_quotes_runtime configuration option will be ignored and no slashes will be stripped from *string*.

Example:

```
<?php
        $filename = 'test.txt';
        $somecontent = "Add this to the file\n";
        $handle=fopen($filename, 'a')
        fwrite($handle, $somecontent)
        fclose($handle);
?>
```

❖ **fclose**

- fclose -- Closes an open file pointer

Syntax:

bool fclose (resource handle)

- The file pointed to by *handle* is closed.
- Returns TRUE on success or FALSE on failure.
- The file pointer must be valid, and must point to a file successfully opened by fopen().

Example:

```
<?php
    $handle = fopen('somefile.txt', 'r');
    fclose($handle);
 ?>
```

❖ **file_exists**

- file_exists -- Checks whether a file or directory exists

Syntax:

bool file_exists (string filename)

- Returns TRUE if the file or directory specified by *filename* exists; FALSE otherwise.
- On windows, use //computername/share/filename or \\computername\share\filename to check files on network shares.

Example:

```
<?php
    $filename = '/path/to/foo.txt';
    if (file_exists($filename)) {
       echo "The file $filename exists";
    } else {
       echo "The file $filename does not exist";
    }
  ?>
```

❖ **is_readable**

- is_readable -- Tells whether the filename is readable

Syntax:

bool is_readable (string filename)

- Returns TRUE if the filename exists and is readable.
- Keep in mind that PHP may be accessing the file as the user id that the web server runs as (often 'nobody'). Safe mode limitations are not taken into account.

Example:

```
<?php
    $filename = 'test.txt';
    if (is_readable($filename)) {
       echo 'The file is readable';
    } else {
       echo 'The file is not readable';
    }
?>
```

❖ **is_writable**

- is_writable -- Tells whether the filename is writable

Syntax:

bool is_writable (string filename)

- Returns TRUE if the *filename* exists and is writable. The filename argument may be a directory name allowing you to check if a directory is writeable.
- Keep in mind that PHP may be accessing the file as the user id that the web server runs as (often 'nobody'). Safe mode limitations are not taken into account.

Example:

```
<?php
    $filename = 'test.txt';
    if (is_writable($filename)) {
       echo 'The file is writable';
    } else {
       echo 'The file is not writable';
    }
?>
```

❖ **fgets**

- fgets -- Gets line from file pointer

Syntax:

string fgets (resource handle [, int length])

- Returns a string of up to *length* - 1 bytes read from the file pointed to by *handle*.
- Reading ends when *length* - 1 bytes have been read, on a newline (which is included in the return value), or on EOF (whichever comes first). If no length is specified, the length defaults to 1k, or 1024 bytes.
- If an error occurs, returns FALSE.

Example: Reading a file line by line

```
<?php
    $handle = fopen("/tmp/inputfile.txt", "r");
    while (!feof($handle)) {
       $buffer = fgets($handle, 4096);
       echo $buffer;
    }
    fclose($handle);
 ?>
```

❖ **fgetc**

- fgetc -- Gets character from file pointer

Syntax:

string fgetc (resource handle)

- Returns a string containing a single character read from the file pointed to by *handle*. Returns FALSE on EOF.

Example:

```
<?php
    $fp = fopen('somefile.txt', 'r');
    if (!$fp) {
       echo 'Could not open file somefile.txt';
    }
    while (false !== ($char = fgetc($fp))) {
       echo "$char\n";
    }
?>
```

❖ **file**

- file -- Reads entire file into an array

Syntax:

array file (string filename [, int use_include_path])

- Identical to readfile(), except that file() returns the file in an array.
- Each element of the array corresponds to a line in the file, with the newline still attached. Upon failure, file() returns FALSE.
- You can use the optional *use_include_path* parameter and set it to "1", if you want to search for the file in the include_path, too.

Example :

```
<?php
        $lines = file("test.txt");
        foreach ($lines as $line_num => $line)
        {
                echo "Line = <b>{$line_num}</b> : " . $line . "<br>";
        }
?>
```

❖ **file_get_contents**

- file_get_contents -- Reads entire file into a string

Syntax:

string file_get_contents (string filename [, bool use_include_path])

- Identical to file(), except that file_get_contents() returns the file in a string, starting at the specified *offset* up to *maxlen* bytes.
- On failure, file_get_contents() will return FALSE.
- file_get_contents() is the preferred way to read the contents of a file into a string. It will use memory mapping techniques if supported by your OS to enhance performance.

Example :

```
<?php
        $str = file_get_contents("test.txt");
        echo $str;
?>
```

❖ **file_put_contents**

- file_put_contents -- Write a string to a file
- **Syntax:**
  int file_put_contents (string filename, mixed data[, int flags [, resource context]])
- You can also specify the *data* parameter as an array (not multi-dimension arrays). This is equivalent to *file_put_contents($filename, join('', $array))*.
- As of PHP 5.1.0, you may also pass a stream resource to the *data* parameter. In result, the remaining buffer of that stream will be copied to the specified file.
- **Parameters**
- *filename* :The file name where to write the data
- *data* :The data to write. Can be either a string, an array or a stream resource *flags*
- *flags* can take FILE_USE_INCLUDE_PATH, FILE_APPEND and/or LOCK_EX (acquire an exclusive lock), however the FILE_USE_INCLUDE_PATH option should be used with caution.
- *context* : A context resource

- **Return Values:** The function returns the amount of bytes that were written to the file.

Example:

```
<?php
    $filename = 'test.txt';
    $str = file_put_contents($filename,"WELCOME TO RPBC",FILE_APPEND );
    echo $str;
?>
```

❖ **ftell**

- ftell -- Tells file pointer read/write position

Syntax:

```
int ftell ( resource handle )
```

- Returns the position of the file pointer referenced by *handle*; i.e., its offset into the file stream.
- If an error occurs, returns FALSE.
- ftell() gives undefined results for append-only streams (opened with "a" flag).

Example:

```
<?php
    // opens a file and read some data
    $fp = fopen("/etc/passwd", "r");
    $data = fgets($fp, 12);
    echo ftell($fp); // 11
    fclose($fp);
?>
```

❖ **fseek**

- fseek -- Seeks on a file pointer

Example:

```
int fseek ( resource handle, int offset [, int whence] )
```

- Sets the file position indicator for the file referenced by *handle*.
- The new position, measured in bytes from the beginning of the file, is obtained by adding *offset* to the position specified by *whence*, whose values are defined as follows:

  SEEK_SET - Set position equal to *offset* bytes.
  SEEK_CUR - Set position to current location plus *offset*.
  SEEK_END - Set position to end-of-file plus *offset*. (To move to a position before the end-of-file, you need to pass a negative value in *offset*.)
- If *whence* is not specified, it is assumed to be SEEK_SET.
- Upon success, returns 0; otherwise, returns -1. Note that seeking past EOF is not considered an error.

Example:

```
<?php
    $fp = fopen('somefile.txt');
    // read some data
    $data = fgets($fp, 4096);
    // move back to the beginning of the file same as rewind($fp);
    fseek($fp, 0);
?>
```

❖ **rewind**

- rewind -- Rewind the position of a file pointer

Example:

bool rewind (resource handle)

- Sets the file position indicator for *handle* to the beginning of the file stream.
- Returns TRUE on success or FALSE on failure.

❖ **copy**

- copy -- Copies file

Syntax:

bool copy (string source, string dest)

- Makes a copy of the file *source* to *dest*.
- Returns TRUE on success or FALSE on failure.

Example:

```
<?php
    $file = 'example.txt';
    $newfile = 'copyfile.txt';
    if (!copy($file, $newfile)) {
       echo "failed to copy $file...\n";
    }
?>
```

❖ **unlink**

- unlink -- Deletes a file

Example:

bool unlink (string filename)

- Deletes *filename*. Similar to the Unix C unlink() function.
- Returns TRUE on success or FALSE on failure.

❖ **rename**

- rename -- Renames a file or directory

Example:

bool rename (string oldname, string newname)

- Attempts to rename *oldname* to *newname*.
- Returns TRUE on success or FALSE on failure.

Example:

```
<?php
    rename("/tmp/tmp_file.txt", "/home/user/login/docs/my_file.txt");
?>
```

❖ **filesize**

- filesize -- Gets file size

Syntax:

int filesize (string filename)

- Returns the size of the file in bytes, or FALSE (and generates an error of level E_WARNING) in case of an error.

Example:

```
$filename = 'somefile.txt';
echo $filename . ': ' . filesize($filename) . ' bytes';
```

❖ **filetype**

- filetype -- Gets file type

Syntax:

string filetype (string filename)

- Returns the type of the file. Possible values are fifo, char, dir, block, link, file, and unknown.
- Returns FALSE if an error occurs.

Example:

```
<?php
        echo filetype('/etc/passwd');  // file
        echo filetype('/etc/');        // dir
?>
```

❖ **basename**

- basename -- Returns filename component of path

Syntax:

string basename (string path [, string suffix])

- Given a string containing a path to a file, this function will return the base name of the file. If the filename ends in *suffix* this will also be cut off.

Example:

```
<?php
        $path = "/home/httpd/html/index.php";
        $file = basename($path);       // $file is set to "index.php"
        $file = basename($path, ".php"); // $file is set to "index"
 ?>
```

❖ **dirname**

- dirname -- Returns directory name component of path

Syntax:

string dirname (string path)

- Given a string containing a path to a file, this function will return the name of the directory.

Example:

```
<?php
        $path = "/etc/passwd";
        $file = dirname($path); // $file is set to "/etc"
 ?>
```

❖ **pathinfo**

- pathinfo -- Returns information about a file path

Syntax:

array pathinfo (string path [, int options])

- pathinfo() returns an associative array containing information about *path*. The following array elements are returned: *dirname*, *basename* and *extension*.
- You can specify which elements are returned with optional parameter *options*. It composes from PATHINFO_DIRNAME, PATHINFO_BASENAME and PATHINFO_EXTENSION.
- It defaults to return all elements.

Example:

```
<?php
        $path_parts = pathinfo('/www/htdocs/index.html');
        echo $path_parts['dirname'], "\n";      // /www/htdocs
        echo $path_parts['basename'], "\n";   // index.html
        echo $path_parts['extension'], "\n";    // html
?>
```

❖ **fileatime**

- fileatime -- Gets last access time of file

Syntax:

```
int fileatime ( string filename )
```

- Returns the time the file was last accessed, or FALSE in case of an error. The time is returned as a Unix timestamp.

❖ **filemtime**

- filemtime -- Gets file modification time

Syntax:

```
int filemtime ( string filename )
```

- Returns the time the file was last modified, or FALSE in case of an error.
- The time is returned as a Unix timestamp, which is suitable for the date() function.

Example:

```
<?php
        $filename = 'somefile.txt';
        if (file_exists($filename)) {
           echo "last modified: " . date ("F d Y H:i:s.", filemtime($filename));
           echo "last access: " . date ("F d Y H:i:s.", fileatime($filename));
        }
?>
```

Miscellaneous Functions

❖ **define**

- define -- Defines a named constant

Example:

bool define (string name, mixed value [, bool case_insensitive])

- Defines a named constant.
- The name of the constant is given by *name*; the value is given by *value*.
- The optional third parameter *case_insensitive* is also available. If the value TRUE is given, then the constant will be defined case-insensitive.
- The default behaviour is case-sensitive; i.e. CONSTANT and Constant represent different values.

❖ **defined**

- defined -- Checks whether a given named constant exists

Syntax:

bool defined (string name)

- Returns TRUE if the named constant given by *name* has been defined, FALSE otherwise.

❖ **constant**

- constant -- Returns the value of a constant

Syntax:

mixed constant (string name)

- constant() will return the value of the constant indicated by *name*.
- constant() is useful if you need to retrieve the value of a constant, but do not know its name.

Example:

```
<?php
        define("CONSTANT", "Hello world.");
        define("GREETING", "Hello you.", true);

        echo CONSTANT; // outputs "Hello world."
        if (defined('GREETING')
        {
                echo GREETING; // outputs "Hello you."
                echo Greeting; // outputs "Hello you."
        }
        echo constant('CONSTANT');
?>
```

❖ **sleep**

- sleep -- Delay execution

Syntax:

int sleep (int seconds)

- The sleep() function delays program execution for the given number of *seconds*.

Example:

```
<?php
        echo date('h:i:s') . "\n";     \\ 10:10:10
        sleep(10); // sleep for 10 seconds
        echo date('h:i:s') . "\n";     \\ 10:10:20
?>
```

❖ **exit**

- exit -- Output a message and terminate the current script

Syntax:

void exit ([string status])

- This is not a real function, but a language construct.
- The exit() function terminates execution of the script. It prints *status* just before exiting.

Example:

```
<?php
        $filename = '/path/to/data-file';
        $file = fopen($filename, 'r')  or exit("unable to open file ($filename)");
 ?>
```

❖ **die :**

- die -- Equivalent to exit()

❖ **INCLUDEING Files**

- You can add PHP to your HTML is by putting it in a separate file and calling it by using PHP's ***include*** functions.
- There are four include functions.
    - include (' /filepath/filename')
    - require (' /filepath/filename')
    - include_once (' /filepath/filename')
    - require_once (' /filepath/filename')

- in previous version of PHP, there were significant differences in functionality and speed between the include function and require function. This is no longer true.
- *The difference only in the kind of error they throw on failure.*
    - ***include( )*** and ***include_once( )*** will generate a warning on failure.
    - ***require( )*** and ***require_once( )*** will cause a fatal error and termination of the script.
- ***include_once()*** and ***require_once( )*** differ from ***include()*** and ***require()*** in that they will allow a file to be include only once per PHP script.
- ***include_once()*** and ***require_once( )*** is extremely helpful when you are including the file that contain PHP functions, because redeclaring functions result in an automatic fatal error.
- In large PHP systems, it's quite common to include files which include other files- it can be difficult to remember whether you've included a particular function before, but with ***include_once()*** or ***require_once()*** you don't have to.

Example:

- The most common use of PHP's *include* capability to add common header nad footer to all the web pages on a site.
- **Header.inc**

```
<html>
<head>
        <title> Welcome To RPBC </title>
</head>
<body>
```

- **Footer.inc**

```
        <p> Copyright 1997 – 2006 </p>
    </body>
    </html>
```

- **Final.php**

```
<?php
    require_once(“header.inc”);
    echo “Hello World”;
    include_once(“footer.inc”);
?>
```

❖ **header**

- header -- Send a raw HTTP header

Synatx:

```
void header ( string string [, bool replace [, int http_response_code]] )
```

- header() is used to send raw HTTP headers.
- The optional *replace* parameter indicates whether the header should replace a previous similar header, or add a second header of the same type. By default it will replace, but if you pass in FALSE as the second argument you can force multiple headers of the same type.
- **For example:**

```
<?php
    header('WWW-Authenticate: Negotiate');
    header('WWW-Authenticate: NTLM', false);
?>
```

- The second optional *http_response_code* force the HTTP response code to the specified value.

- **<u>There are two special-case header calls.</u>**

- **The first is a header that starts with the string "*HTTP/*"** (case is not significant), which will be used to figure out the HTTP status code to send.

  Example:

```
<?php
    header("HTTP/1.0 404 Not Found");
?>
```

- The HTTP status header line will always be the first sent to the client, regardless of the actual header() call being the first or not. The status may be overridden by calling header() with a new status line at any time unless the HTTP headers have already been sent.
- **The second special case is the "Location:" header.** Not only does it send this header back to the browser, but it also returns a *REDIRECT* (302) status code to the browser unless some *3xx* status code has already been set.

  Example:

```
<?php
    header("Location: http://www.rpbc.edu/"); /* Redirect browser */
    /* Make sure that code below does not get executed when we redir
    ect. */
    exit;
?>
```

- PHP scripts often generate dynamic content that must not be cached by the client browser or any proxy caches between the server and the client browser. Many proxies and clients can be forced to disable caching with:

  Example:

```
<?php
        header("Cache-Control: no-cache, must-revalidate"); // HTTP/1.1
        header("Expires: Mon, 26 Jul 1997 05:00:00 GMT");
 ?>
```

- Remember that header() must be called before any actual output is sent, either by normal HTML tags, blank lines in a file, or from PHP. It is a very common error to read code with include(), or require(), functions, or another file access function, and have spaces or empty lines that are output before header() is called. The same problem exists when using a single PHP/HTML file.

  Example:

```
<html>
<?php
        /* This will give an error. Note the output above, which is before t
        he header() call */
        header('Location: http://www.example.com/');
?>
```

- If you want the user to be prompted to save the data you are sending, such as a generated PDF file, you can use the Content-Disposition header to supply a recommended filename and force the browser to display the save dialog.

  Example:

```
<?php
        // We'll be outputting a PDF
        header('Content-type: application/pdf');

        // It will be called downloaded.pdf
        header('Content-
        Disposition: attachment; filename="downloaded.pdf"');

        // The PDF source is in original.pdf
        readfile('original.pdf');
?>
```

❖ **setcookie**

- setcookie -- Send a cookie

Syntax:

bool setcookie (string name [, string value [, int expire [, string path [, string domain [, bool secure]]]]])

- setcookie() defines a cookie to be sent along with the rest of the HTTP headers.
- Like other headers, cookies must be sent *before* any output from your script (this is a protocol restriction). This requires that you place calls to this function prior to any output, including <html> and <head> tags as well as any whitespace.
- If output exists prior to calling this function, setcookie() will fail and return FALSE. If setcookie() successfully runs, it will return TRUE. This does not indicate whether the user accepted the cookie.

- All the arguments except the *name* argument are optional. You may also replace an argument with an empty string (*""*) in order to skip that argument. Because the *expire* argument is integer, it cannot be skipped with an empty string, use a zero (*0*) instead.
- The following table explains each parameter of the setcookie() function, be sure to read the Netscape cookie specification for specifics on how each setcookie() parameter works for additional information on how HTTP cookies work.

- **setcookie() parameters explained**

Parameter	Description	Examples
name	The name of the cookie.	'cookiename' is called as *$_COOKIE['cookiename']*
value	The value of the cookie. This value is stored on the clients computer; do not store sensitive information.	Assuming the *name* is 'cookiename', this value is retrieved through *$_COOKIE['cookiename']*
expire	The time the cookie expires. This is a Unix timestamp so is in number of seconds since the epoch. In other words, you'll most likely set this with the time() function plus the number of seconds before you want it to expire. Or you might use mktime().	*time()+60*60*24*30* will set the cookie to expire in 30 days. If not set, the cookie will expire at the end of the session (when the browser closes).
path	The path on the server in which the cookie will be available on.	If set to *'/'*, the cookie will be available within the entire *domain*. If set to *'/foo/'*, the cookie will only be available within the */foo/* directory and all sub-directories such as */foo/bar/* of *domain*. The default value is the current directory that the cookie is being set in.
domain	The domain that the cookie is available.	To make the cookie available on all subdomains of example.com then you'd set it to *'.example.com'*. The . is not required but makes it compatible with more browsers. Setting it to *www.example.com* will make the cookie only available in the *www* subdomain.
secure	Indicates that the cookie should only be transmitted over a secure HTTPS connection. When set to TRUE, the cookie will only be set if a secure connection exists. The default is FALSE.	*0* or *1*

- Once the cookies have been set, they can be accessed on the next page load with the $_COOKIE or *$HTTP_COOKIE_VARS* arrays. Note, autoglobals such as *$_COOKIE* became available in PHP 4.1.0. *$HTTP_COOKIE_VARS* has existed since PHP 3. Cookie values also exist in $_REQUEST.

- **Examples:**

1. setcookie() send example

```
<?php
        $value = 'something from somewhere';
        setcookie("TestCookie", $value);
        setcookie("TestCookie", $value, time()+3600);  /* expire in 1 hour */
        setcookie("TestCookie", $value, time()+3600, "/~rasmus/", ".example.com
        ", 1);
?>
```

2. read cookie example

```
        <?php
                // Print an individual cookie
                echo $_COOKIE["TestCookie"];
                echo $HTTP_COOKIE_VARS["TestCookie"];

                // Another way to debug/test is to view all cookies
                print_r($_COOKIE);
        ?>
```

3. setcookie() delete example

```
        <?php
                // set the expiration date to one hour ago
                setcookie ("TestCookie", "", time() - 3600);
                setcookie ("TestCookie", "", time() -
                 3600, "/~rasmus/", ".example.com", 1);
        ?>
```

4. setcookie() and arrays

```
        <?php
                // set the cookies
                setcookie("cookie[three]", "cookiethree");
                setcookie("cookie[two]", "cookietwo");
                setcookie("cookie[one]", "cookieone");

                // after the page reloads, print them out
                if (isset($_COOKIE['cookie']))
                {
                        foreach ($_COOKIE['cookie'] as $name => $value)
                        {
                              echo "$name : $value <br />\n";
                        }
                }
        ?>
```

Output:

```
three : cookiethree
two : cookietwo
one : cookieone
```

Predefined Variables

- **HTTP Cookies: *$_COOKIE***
  - An associative array of variables passed to the current script via HTTP cookies. Automatically global in any scope.
  - This is a 'superglobal', or automatic global, variable. This simply means that it is available in all scopes throughout a script. You don't need to do a global $_COOKIE; to access it within functions or methods, as you do with *$HTTP_COOKIE_VARS*.
  - *$HTTP_COOKIE_VARS* contains the same initial information, but is not an autoglobal. (Note that *$HTTP_COOKIE_VARS* and *$_COOKIE* are different variables and that PHP handles them as such)

- **HTTP GET variables: *$_GET***
  - An associative array of variables passed to the current script via the HTTP GET method. Automatically global in any scope.
  - This is a 'superglobal', or automatic global, variable. This simply means that it is available in all scopes throughout a script. You don't need to do a global $_GET; to access it within functions or methods, as you do with *$HTTP_GET_VARS*.
  - *$HTTP_GET_VARS* contains the same initial information, but is not an autoglobal. (Note that *$HTTP_GET_VARS* and *$_GET* are different variables and that PHP handles them as such)

- **HTTP POST variables: $_POST**
  - An associative array of variables passed to the current script via the HTTP POST method. Automatically global in any scope.
  - This is a 'superglobal', or automatic global, variable. This simply means that it is available in all scopes throughout a script. You don't need to do a global $_POST; to access it within functions or methods, as you do with *$HTTP_POST_VARS*.
  - *$HTTP_POST_VARS* contains the same initial information, but is not an autoglobal. (Note that *$HTTP_POST_VARS* and *$_POST* are different variables and that PHP handles them as such)

- **Request variables: *$_REQUEST***
  - An associative array consisting of the contents of *$_GET*, *$_POST*, and *$_COOKIE*.
  - This is a 'superglobal', or automatic global, variable. This simply means that it is available in all scopes throughout a script. You don't need to do a global $_REQUEST; to access it within functions or methods.

- **Global variables: *$GLOBALS***
  - An associative array containing references to all variables which are currently defined in the global scope of the script. The variable names are the keys of the array.
  - This is a 'superglobal', or automatic global, variable. This simply means that it is available in all scopes throughout a script. You don't need to do a global $GLOBALS; to access it within functions or methods.

❖ **Server variables: *$_SERVER***

- *$_SERVER* is an array containing information such as headers, paths, and script locations. The entries in this array are created by the webserver. There is no guarantee that every webserver will provide any of these; servers may omit some, or provide others not listed here.
- This is a 'superglobal', or automatic global, variable. This simply means that it is available in all scopes throughout a script. You don't need to do a global $_SERVER; to access it within functions or methods, as you do with *$HTTP_SERVER_VARS*.
- *$HTTP_SERVER_VARS* contains the same initial information, but is not an autoglobal. (Note that *$HTTP_SERVER_VARS* and *$_SERVER* are different variables and that PHP handles them as such)
- **You may or may not find any of the following elements in $_SERVER.**
- **'*GATEWAY_INTERFACE*'**: What revision of the CGI specification the server is using; i.e. '*CGI/1.1*'.
- **'*SERVER_NAME*' :** The name of the server host under which the current script is executing. If the script is running on a virtual host, this will be the value defined for that virtual host.
- **'*SERVER_SOFTWARE*' :** Server identification string, given in the headers when responding to requests.
- **'*SERVER_PROTOCOL*' :** Name and revision of the information protocol via which the page was requested; i.e. '*HTTP/1.0*';
- **'*REQUEST_METHOD*' :** Which request method was used to access the page; i.e. '*GET*', '*HEAD*', '*POST*', '*PUT*'.
- **'*REQUEST_TIME*' :** The timestamp of the start of the request. Available since PHP 5.1.0.
- **'*QUERY_STRING*' :** The query string, if any, via which the page was accessed.
- **'*DOCUMENT_ROOT*' :** The document root directory under which the current script is executing, as defined in the server's configuration file.
- **'*HTTP_ACCEPT*' :** Contents of the *Accept:* header from the current request, if there is one.
- **'*HTTP_ACCEPT_CHARSET*' :** Contents of the *Accept-Charset:* header from the current request, if there is one. Example: '*iso-8859-1,*,utf-8*'.
- **'*HTTP_ACCEPT_ENCODING*' :** Contents of the *Accept-Encoding:* header from the current request, if there is one. Example: '*gzip*'.
- ***'HTTP_ACCEPT_LANGUAGE'* :** Contents of the *Accept-Language:* header from the current request, if there is one. Example: '*en*'.
- **'*HTTP_CONNECTION*' :** Contents of the *Connection:* header from the current request, if there is one. Example: '*Keep-Alive*'.
- **'*HTTP_HOST*' :** Contents of the *Host:* header from the current request, if there is one.
- **'*HTTP_REFERER*' :** The address of the page (if any) which referred the user agent to the current page. This is set by the user agent. Not all user agents will set this, and some provide the ability to modify *HTTP_REFERER* as a feature. In short, it cannot really be trusted.
- **'*HTTP_USER_AGENT*' :** Contents of the *User-Agent:* header from the current request, if there is one. This is a string denoting the user agent being which is accessing the page.

- '***HTTPS***' **:** Set to a non-empty value if the script was queried through the HTTPS protocol.
- '***REMOTE_ADDR***' **:** The IP address from which the user is viewing the current page.
- '***REMOTE_HOST***' **:** The Host name from which the user is viewing the current page. The reverse dns lookup is based off the *REMOTE_ADDR* of the user.
- '***REMOTE_PORT***' **:** The port being used on the user's machine to communicate with the web server.
- '***SCRIPT_FILENAME***' **:** The absolute pathname of the currently executing script.
- '***SERVER_ADMIN***' **:** The value given to the SERVER_ADMIN (for Apache) directive in the web server configuration file. If the script is running on a virtual host, this will be the value defined for that virtual host.
- '***SERVER_PORT***' **:** The port on the server machine being used by the web server for communication. For default setups, this will be '*80*'; using SSL, for instance, will change this to whatever your defined secure HTTP port is.
- '***SERVER_SIGNATURE***' **:** String containing the server version and virtual host name which are added to server-generated pages, if enabled.
- '***PATH_TRANSLATED***' **:** Filesystem- (not document root-) based path to the current script, after the server has done any virtual-to-real mapping.
- '***SCRIPT_NAME***' **:** Contains the current script's path. This is useful for pages which need to point to themselves. The __FILE__ constant contains the full path and filename of the current (i.e. included) file.
- '***REQUEST_URI***' **:** The URI which was given in order to access this page; for instance, '*/index.html*'.
- '***PHP_AUTH_DIGEST***' **:** When running under Apache as module doing Digest HTTP authentication this variable is set to the 'Authorization' header sent by the client (which you should then use to make the appropriate validation).
- '***PHP_AUTH_USER***' **:** When running under Apache or IIS (ISAPI on PHP 5) as module doing HTTP authentication this variable is set to the username provided by the user.
- '***PHP_AUTH_PW***' **:** When running under Apache or IIS (ISAPI on PHP 5) as module doing HTTP authentication this variable is set to the password provided by the user.
- '***AUTH_TYPE***' **:**When running under Apache as module doing HTTP authenticated this variable is set to the authentication type.

❖ MySQL Database Management System

- MySQL, the most popular Open Source SQL database management system, is developed, distributed, and supported by MySQL AB.
- MySQL AB is a commercial company, founded by the MySQL developers. It is a second generation Open Source company that unites Open Source values and methodology with a successful business model.
- The MySQL Web site (http://www.mysql.com/) provides the latest information about MySQL software and MySQL AB.
- The official way to pronounce “MySQL” is “My Ess Que Ell” (not “my sequel”), but we don't mind if you pronounce it as “my sequel” or in some other localized way.
- **MySQL Features :**
- MySQL is a database management system.
- MySQL is a relational database management system.
- MySQL software is Open Source.
- The MySQL Database Server is very fast, reliable, and easy to use.
- MySQL Server works in client/server or embedded systems.
- A large amount of contributed MySQL software is available.

MySQL Function

❖ **mysql_connect**

- mysql_connect -- Open a connection to a MySQL Server

Syntax:

resource mysql_connect ([string server [, string username [, string password]]])

- **Parameters**
- ***server* :**The MySQL server. It can also include a port number. e.g. "hostname:port" . If the PHP directive mysql.default_host is undefined (default), then the default value is 'localhost:3306'
- ***username* :** The username. Default value is the name of the user that owns the server process.
- ***password* :** The password. Default value is an empty password.
- **Return Values :** Returns a MySQL link identifier on success, or FALSE on failure.

❖ **mysql_close**

- mysql_close -- Close MySQL connection

Syntax:

bool mysql_close ([resource link_identifier])

- ***link_identifier* :** The MySQL connection
- mysql_close() closes the non-persistent connection to the MySQL server that's associated with the specified link identifier. If *link_identifier* isn't specified, the last opened link is used.
- Using mysql_close() isn't usually necessary, as non-persistent open links are automatically closed at the end of the script's executionParameters

Examples

```
<?php
        $link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
        if (!$link) {
           die('Could not connect: ' . mysql_error());
        }
        echo 'Connected successfully';
        mysql_close($link);
?>
```

❖ **mysql_select_db**

- mysql_select_db -- Select a MySQL database
- **Syntax:**

  bool mysql_select_db (string database_name [, resource link_identifier])
- Sets the current active database on the server that's associated with the specified link identifier.
- **Parameters**
- ***database_name* :** The name of the database that is to be selected.
- ***link_identifier* :** The MySQL connection. If the link identifier is not specified, the last link opened by mysql_connect() is assumed.
- **Return Values :** Returns TRUE on success or FALSE on failure.

Examples

```
<?php
        $link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
        // make foo the current db
        $db_selected = mysql_select_db('foo', $link);
        if (!$db_selected) {
           die ('Can\'t use foo : ' . mysql_error());
        }
 ?>
```

❖ **mysql_query**

- mysql_query -- Send a MySQL query

Syntax:

resource mysql_query (string query [, resource link_identifier])

- mysql_query() sends a query (to the currently active database on the server that's associated with the specified *link_identifier*).
- **Parameters**
- ***query* :** A SQL query. The query string should not end with a semicolon.
- ***link_identifier* :** The MySQL connection. If the link identifier is not specified, the last link opened by mysql_connect() is assumed.
- **Return Values**
  - For SELECT, SHOW, DESCRIBE or EXPLAIN statements, mysql_query() returns a resource on success, or FALSE on error.
  - For other type of SQL statements, UPDATE, DELETE, DROP, etc, mysql_query() returns TRUE on success or FALSE on error.
  - mysql_query() will also fail and return FALSE if the user does not have permission to access the table(s) referenced by the query.

❖ **mysql_num_rows**

- mysql_num_rows -- Get number of rows in result

Syntax:

int mysql_num_rows (resource result)

- Retrieves the number of rows from a result set.
- This command is only valid for SELECT statements. To retrieve the number of rows affected by a INSERT, UPDATE, or DELETE query, use mysql_affected_rows().
- **Parameters**
- ***result* :** The result resource that is being evaluated. This result comes from a call to mysql_query().
- **Return Values :** The number of rows in a result set on success, or FALSE on failure.

❖ **mysql_affected_rows**

- mysql_affected_rows -- Get number of affected rows in previous MySQL operation

Syntax:

int mysql_affected_rows ([resource link_identifier])

- Get the number of affected rows by the last INSERT, UPDATE or DELETE query associated with *link_identifier*.
- **Parameters**
- ***link_identifier* :** The MySQL connection. If the link identifier is not specified, the last link opened by mysql_connect() is assumed.
- **Return Values:** Returns the number of affected rows on success, and -1 if the last query failed.

Examples

```
<?php
        $link = mysql_connect("localhost", "user", "mysql_password");
        mysql_select_db("database", $link);
        $result = mysql_query("SELECT * FROM table1", $link);
        $num_rows = mysql_num_rows($result);
        echo "$num_rows Rows\n";

        mysql_query('DELETE FROM mytable WHERE id < 10');
        echo ("Records deleted: %d\n", mysql_affected_rows());
 ?>
```

❖ **mysql_fetch_array**

- mysql_fetch_array -- Fetch a result row as an associative array, a numeric array, or both

Syntax:

array mysql_fetch_array (resource result [, int result_type])

- Returns an array that corresponds to the fetched row and moves the internal data pointer ahead.
- **Parameters**
- ***result* :** The result resource that is being evaluated. This result comes from a call to mysql_query().
- ***result_type* :** The type of array that is to be fetched. It's a constant and can take the following values: MYSQL_ASSOC, MYSQL_NUM, and the default value of MYSQL_BOTH.

- **Return Values :** Returns an array that corresponds to the fetched row, or FALSE if there are no more rows.

Examples

- **mysql_fetch_array() with MYSQL_NUM**

```
<?php
        mysql_connect("localhost", "mysql_user", "mysql_password") or
                die("Could not connect: " . mysql_error());
        mysql_select_db("mydb");
        $result = mysql_query("SELECT id, name FROM mytable");
        while ($row = mysql_fetch_array($result, MYSQL_NUM)) {
          printf("ID: %s  Name: %s", $row[0], $row[1]);
        }
        mysql_free_result($result);
?>
```

- **mysql_fetch_array() with MYSQL_ASSOC**

```
<?php
        mysql_connect("localhost", "mysql_user", "mysql_password") or
          die("Could not connect: " . mysql_error());
        mysql_select_db("mydb");
        $result = mysql_query("SELECT id, name FROM mytable");
        while ($row = mysql_fetch_array($result, MYSQL_ASSOC)) {
          printf("ID: %s  Name: %s", $row["id"], $row["name"]);
        }
        mysql_free_result($result);
?>
```

- **mysql_fetch_array() with MYSQL_BOTH**

```
<?php
        mysql_connect("localhost", "mysql_user", "mysql_password") or
                        die("Could not connect: " . mysql_error());
        mysql_select_db("mydb");
        $result = mysql_query("SELECT id, name FROM mytable");
        while ($row = mysql_fetch_array($result, MYSQL_BOTH)) {
          printf ("ID: %s  Name: %s", $row[0], $row["name"]);
        }
        mysql_free_result($result);
?>
```

❖ **mysql_fetch_row**

- mysql_fetch_row -- Get a result row as an enumerated array

Syntax:

```
array mysql_fetch_row ( resource result )
```

- Returns a numerical array that corresponds to the fetched row and moves the internal data pointer ahead.
- **Parameters**
- ***result* :** The result resource that is being evaluated. This result comes from a call to mysql_query().

- **Return Values :** Returns an numerical array that corresponds to the fetched row, or FALSE if there are no more rows.
- mysql_fetch_row() fetches one row of data from the result associated with the specified result identifier. The row is returned as an array. Each result column is stored in an array offset, starting at offset 0.
- **Examples**

```
<?php
    $result = mysql_query("SELECT id,email FROM people WHERE id = '42'");
    if (!$result) {
       echo 'Could not run query: ' . mysql_error();
       exit;
    }
    $row = mysql_fetch_row($result);
    echo $row[0]; // 42
    echo $row[1]; // the email value
?>
```

❖ **mysql_fetch_object**

- mysql_fetch_object -- Fetch a result row as an object

Syntax:

```
object mysql_fetch_object ( resource result )
```

- Returns an object with properties that correspond to the fetched row and moves the internal data pointer ahead.
- **Parameters**
- ***result* :** The result resource that is being evaluated. This result comes from a call to mysql_query().
- **Return Values:** Returns an object with properties that correspond to the fetched row, or FALSE if there are no more rows.
- **Example :**

```
<?php
    mysql_connect("hostname", "user", "password");
    mysql_select_db("mydb");
    $result = mysql_query("select * from mytable");
    while ($row = mysql_fetch_object($result)) {
       echo $row->user_id;
       echo $row->fullname;
    }
    mysql_free_result($result);
?>
```

❖ **mysql_num_fields**

- mysql_num_fields -- Get number of fields in result

Syntax:

```
int mysql_num_fields ( resource result )
```

- Retrieves the number of fields from a query.
- **Parameters**
- ***result* :** The result resource that is being evaluated. This result comes from a call to mysql_query().

- **Return Values :** Returns the number of fields in the result set resource on success, or FALSE on failure.

Example:

```
<?php
$result = mysql_query("SELECT id,email FROM people WHERE id = '42'");
 /* returns 2 because id,email === two fields */
echo mysql_num_fields($result);
?>
```

❖ **mysql_fetch_field**

- mysql_fetch_field -- Get column information from a result and return as an object

Syntax:

object mysql_fetch_field (resource result [, int field_offset])

- Returns an object containing field information. This function can be used to obtain information about fields in the provided query result.
- **Parameters**
- ***result* :** The result resource that is being evaluated. This result comes from a call to mysql_query().
- ***field_offset* :** The numerical field offset. If the field offset is not specified, the next field that was not yet retrieved by this function is retrieved. The *field_offset* starts at *0*.
- **Return Values :** Returns an object containing field information. The properties of the object are:
  - name - column name
  - table - name of the table the column belongs to
  - max_length - maximum length of the column
  - not_null - 1 if the column cannot be NULL
  - primary_key - 1 if the column is a primary key
  - unique_key - 1 if the column is a unique key
  - multiple_key - 1 if the column is a non-unique key
  - numeric - 1 if the column is numeric
  - blob - 1 if the column is a BLOB
  - type - the type of the column
  - unsigned - 1 if the column is unsigned
  - zerofill - 1 if the column is zero-filled
- **Examples**

```
<?php
    $conn = mysql_connect('localhost:3306', 'user', 'password');
    if (!$conn) {
       die('Could not connect: ' . mysql_error());
    }
     mysql_select_db('database');
    $result = mysql_query('select * from table');
     /* get column metadata */
    $i = 0;
    while ($i < mysql_num_fields($result)) {
       echo "Information for column $i:<br />\n";
       $meta = mysql_fetch_field($result, $i);
       if (!$meta) {
           echo "No information available<br />\n";
       }
```

```
    echo "<pre>
          blob:         $meta->blob
          max_length:   $meta->max_length
          multiple_key: $meta->multiple_key
          name:         $meta->name
          not_null:     $meta->not_null
          numeric:      $meta->numeric
          primary_key:  $meta->primary_key
          table:        $meta->table
          type:         $meta->type
          unique_key:   $meta->unique_key
          unsigned:     $meta->unsigned
          zerofill:     $meta->zerofill
          </pre>";
    $i++;
}
mysql_free_result($result);
?>
```

❖ **mysql_field_name**

- mysql_field_name -- Get the name of the specified field in a result

Syntax:

string mysql_field_name (resource result, int field_offset)

- **Parameters**
- ***result* :** The result resource that is being evaluated. This result comes from a call to mysql_query().
- ***field_offset* :** The numerical field offset. The *field_offset* starts at *0*. If *field_offset* does not exist, an error of level E_WARNING is also issued.
- **Return Values:** The name of the specified field index on success, or FALSE on failure.

❖ **mysql_field_type**

- mysql_field_type -- Get the type of the specified field in a result

Syntax:

string mysql_field_type (resource result, int field_offset)

- **Parameters Same as mysql_field_name**
- **Return Values:** The returned field type will be one of *"int"*, *"real"*, *"string"*, *"blob"*, and others as detailed in the MySQL documentation.

❖ **mysql_field_len**

- mysql_field_len -- Returns the length of the specified field

Syntax:

int mysql_field_len (resource result, int field_offset)

- **Parameters Same as mysql_field_name**
- **Return Values:** The name of the specified field index on success, or FALSE on failure.

❖ **mysql_field_table**

- mysql_field_table -- Get name of the table the specified field is in

Syntax:

string mysql_field_table (resource result, int field_offset)

- **Parameters Same as mysql_field_name**
- **Return Values :** The name of the table on success.

❖ **mysql_field_flags**

- mysql_field_flags -- Get the flags associated with the specified field in a result

Syntax:

string mysql_field_flags (resource result, int field_offset)

- **Parameters Same as mysql_field_name**
- **Return Values:**
  - Returns a string of flags associated with the result, or FALSE on failure.
  - The following flags are reported, if your version of MySQL is current enough to support them: "not_null", "primary_key", "unique_key", "multiple_key", "blob", "unsigned", "zerofill", "binary", "enum", "auto_increment" and "timestamp".

Example:

```
<?php
    mysql_connect("localhost", "mysql_username", "mysql_password");
    mysql_select_db("mysql");
    $result = mysql_query("SELECT * FROM func");
    $fields = mysql_num_fields($result);
    $rows   = mysql_num_rows($result);
    $table  = mysql_field_table($result, 0);
    echo "Your '" . $table . "' table has " . $fields . " fields and " . $rows . " rec
    ord(s)\n";

    echo "The table has the following fields:\n";
    for ($i=0; $i < $fields; $i++) {
       $type  = mysql_field_type($result, $i);
       $name  = mysql_field_name($result, $i);
       $len   = mysql_field_len($result, $i);
       $flags = mysql_field_flags($result, $i);
       echo $type . " " . $name . " " . $len . " " . $flags . "\n";
    }
    mysql_free_result($result);
    mysql_close();
?>
```

❖ **mysql_data_seek**

- mysql_data_seek -- Move internal result pointer

Syntax :

bool mysql_data_seek (resource result, int row_number)

- mysql_data_seek() moves the internal row pointer of the MySQL result associated with the specified result identifier to point to the specified row number. The next call to mysql_fetch_row() would return that row.
- *row_number* starts at 0. The *row_number* should be a value in the range from 0 to mysql_num_rows() - 1. However if the result set is empty (mysql_num_rows() == 0), a seek to 0 will fail with a E_WARNING and mysql_data_seek() will return FALSE.
- **Parameters**
- ***result* :** The result resource that is being evaluated. This result comes from a call to mysql_query().
- ***row_number* :** The desired row number of the new result pointer.
- **Return Values:** Returns TRUE on success or FALSE on failure.

❖ **mysql_field_seek**

- mysql_field_seek -- Set result pointer to a specified field offset

Syntax:

bool mysql_field_seek (resource result, int field_offset)

- Seeks to the specified field offset. If the next call to mysql_fetch_field() doesn't include a field offset, the field offset specified in mysql_field_seek() will be returned.
- **Parameters**
- ***result* :** The result resource that is being evaluated. This result comes from a call to mysql_query().
- ***field_offset* :** The numerical field offset. The *field_offset* starts at *0*. If *field_offset* does not exist, an error of level E_WARNING is also issued.

Example :

```
<?php
        $link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
         mysql_select_db('sample_db');
         $query = 'SELECT last_name, first_name FROM friends';
        $result = mysql_query($query);
         /* fetch rows in reverse order */
        for ($i = mysql_num_rows($result) - 1; $i >= 0; $i--)
        {
           if (!($row = mysql_fetch_assoc($result))) {
              continue;
           }
           echo $row['last_name'] . ' ' . $row['first_name'] . "<br />\n";
        }
        // mysql_field_seek
        mysql_field_seek ($result,1);
        $obj=mysql_fetch_field ($result);
        echo " Field NAME:= $obj->name   TYPE := $obj->type";

        mysql_free_result($result);
?>
```

❖ **mysql_list_dbs**

- mysql_list_dbs -- List databases available on a MySQL server
- **Syntax:**

  resource mysql_list_dbs ([resource link_identifier])
- Returns a result pointer containing the databases available from the current mysql daemon.
- **Parameters**
- ***link_identifier* :** The MySQL connection. If the link identifier is not specified, the last link opened by mysql_connect() is assumed.
- **Return Values :** Returns a result pointer resource on success, or FALSE on failure.

Examples

```
<?php
        $link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
        $db_list = mysql_list_dbs($link);
        while ($row = mysql_fetch_object($db_list)) {
            echo $row->Database . "\n";
        }
?>
```

❖ **mysql_list_tables**

- mysql_list_tables -- List tables in a MySQL database

Syntax:

resource mysql_list_tables (string database [, resource link_identifier])

- Retrieves a list of table names from a MySQL database.
- This function deprecated. It is preferable to use mysql_query() to issue a SQL *SHOW TABLES [FROM db_name] [LIKE 'pattern']* statement instead.
- **Parameters:**
- ***database* :** The name of the database
- ***link_identifier* :**The MySQL connection. If the link identifier is not specified, the last link opened by mysql_connect() is assumed.
- **Return Values:** A result pointer resource on success, or FALSE on failure.

Example:

```
<?php
        $link = mysql_connect('pdc', 'gautam', 'milan');
        $tb_list = mysql_list_tables("mysql");
        while ($row = mysql_fetch_array($tb_list))
        {
            echo $row[0];
            echo "<br>";
        }
?>
```

❖ **mysql_list_fields**

- mysql_list_fields -- List MySQL table fields

Syntax:

resource mysql_list_fields (string database_name, string table_name [, resource link_identifier])

- Retrieves information about the given table name.
- This function is deprecated. It is preferable to use mysql_query() to issue a SQL *SHOW COLUMNS FROM table [LIKE 'name']* statement instead.
- **Parameters**
- ***database_name* :** The name of the database that's being queried.
- ***table_name* :** The name of the table that's being queried.
- ***link_identifier* :** The MySQL connection. If the link identifier is not specified, the last link opened by mysql_connect() is assumed.
- **Return Values :** A result pointer resource on success, or FALSE on failure.

❖ **mysql_free_result**

- mysql_free_result -- Free result memory

Syntax:

bool mysql_free_result (resource result)

- mysql_free_result() will free all memory associated with the result identifier *result*.
- mysql_free_result() only needs to be called if you are concerned about how much memory is being used for queries that return large result sets. All associated result memory is automatically freed at the end of the script's execution.

- **Parameters**
- ***result* :** The result resource that is being evaluated. This result comes from a call to mysql_query().
- **Return Values :** Returns TRUE on success or FALSE on failure.

❖ **mysql_errno**

- mysql_errno -- Returns the numerical value of the error message from previous MySQL operation

Syntax:

int mysql_errno ([resource link_identifier])

- Returns the error number from the last MySQL function.
- Errors coming back from the MySQL database backend no longer issue warnings. Instead, use mysql_errno() to retrieve the error code.
- Note that this function only returns the error code from the most recently executed MySQL function (not including mysql_error() and mysql_errno()), so if you want to use it, make sure you check the value before calling another MySQL function.
- **Parameters**
- ***link_identifier* :** The MySQL connection. If the link identifier is not specified, the last link opened by mysql_connect() is assumed.
- **Return Values :** Returns the error number from the last MySQL function, or *0* (zero) if no error occurred.

❖ **mysql_error**

- mysql_error -- Returns the text of the error message from previous MySQL operation

Syntax:

string mysql_error ([resource link_identifier])

- Returns the error text from the last MySQL function.
- **Parameters**
- ***link_identifier* :** The MySQL connection. If the link identifier is not specified, the last link opened by mysql_connect() is assumed.
- **Return Values:** Returns the error text from the last MySQL function, or *''* (empty string) if no error occurred.

Examples

```
<?php
    $link = mysql_connect("localhost", "mysql_user", "mysql_password");
    mysql_select_db("nonexistentdb", $link);
    echo mysql_errno($link) . ": " . mysql_error($link). "\n";
?>
```

Sessions

A session can be defined as a series of related instructions between a single client and the web server, which take a place over an extended period of time. this could be a series of transactions that a user make while updating his stock or the set of requests that are made to check an e-mail account through a browser base e-mail service.

Particularly when you want to work with sensitive information , it make a lot of sense to submit it once and have it stored on the server rather then client machine.

Session support in PHP consists of a way to preserve certain data across subsequent accesses. This enables you to build more customized applications and increase the appeal of your web site.

A visitor accessing your web site is assigned an unique id, the so-called session id. This is either stored in a cookie on the user side or is propagated in the URL.

The session support allows you to register arbitrary numbers of variables to be preserved across requests. When a visitor accesses your site, PHP will check automatically (if session.auto_start is set to 1) or on your request (explicitly through session_start() or implicitly through session_register()) whether a specific session id has been sent with the request. If this is the case, the prior saved environment is recreated.

All registered variables are serialized after the request finishes. Registered variables which are undefined are marked as being not defined. On subsequent accesses, these are not defined by the session module unless the user defines them later.

When working with sessions that a record of a session is not created until a variable has been registered using the session_register() function or by adding a new key to the *$_SESSION* superglobal array. This holds true regardless of if a session has been started using the session_start() function.

❖ **session_id**

- session_id -- Get and/or set the current session id

syntax:

string session_id ([string id])

- session_id() is used to get or set the session id for the current session.
- The constant SID can also be used to retrieve the current name and session id as a string suitable for adding to URLs.
- Note that SID is only defined if the client didn't send the right cookie. See also Session handling.
- **Parameters**
- ***id* :** If *id* is specified, it will replace the current session id. session_id() needs to be called before session_start() for that purpose.
- When using session cookies, specifying an *id* for session_id() will always send a new cookie when session_start() is called, regardless if the current session id is identical to the one being set.
- **Return Values :** session_id() returns the session id for the current session or the empty string ("") if there is no current session (no current session id exists).

❖ **session_name**

- session_name -- Get and/or set the current session name

Syntax:

string session_name ([string name])

- session_name() returns the name of the current session. If *name* is specified, the name of the current session is changed to its value.
- The session name references the session id in cookies and URLs. It should contain only alphanumeric characters; it should be short and descriptive (i.e. for users with enabled cookie warnings).
- The session name is reset to the default value stored in *session.name* at request startup time. Thus, you need to call session_name() for every request (and before session_start() or session_register() are called).
- Session name can't consist only from digits, at least one letter must be present. Otherwise new session id is generated every time.

❖ **session_start**

- session_start -- Initialize session data

Syntax:

bool session_start (void)

- session_start() creates a session or resumes the current one based on the current session id that's being passed via a request, such as GET, POST, or a cookie.
- This function always returns TRUE.
- If you are using cookie-based sessions, you must call session_start() before anything is outputted to the browser.

Example:

page1.php

```
<?php
        session_start();
        echo 'Welcome to page #1';
        echo "Session ID :=" . session_id();
        echo "Session Name :=" . session_name();

        $_SESSION['favcolor'] = 'green';
        $_SESSION['animal']   = 'cat';
        $_SESSION['time']     = time();

        echo '<br /><a href="page2.php">page 2</a>';

?>
```

- After viewing page1.php, the second page page2.php will magically contain the session data.

page2.php

```
<?php
        session_start();

        echo 'Welcome to page #2<br />';

        echo $_SESSION['favcolor']; // green
        echo $_SESSION['animal'];   // cat
        echo date('Y m d H:i:s', $_SESSION['time']);

        echo '<br /><a href="page1.php">page 1</a>';
?>
```

❖ **session_unset**

- session_unset -- Free all session variables

Syntax:

void session_unset (void)

- The session_unset() function frees all session variables currently registered.
- If *$_SESSION* (or *$HTTP_SESSION_VARS* for PHP 4.0.6 or less) is used, use unset() to unregister a session variable, i.e. *unset ($_SESSION['varname']);*.
- Do NOT unset the whole *$_SESSION* with *unset($_SESSION)* as this will disable the registering of session variables through the *$_SESSION* superglobal.

❖ **session_cache_limiter**

- session_cache_limiter -- Get and/or set the current cache limiter

Syntax:

string session_cache_limiter ([string cache_limiter])

- session_cache_limiter() returns the name of the current cache limiter. If *cache_limiter* is specified, the name of the current cache limiter is changed to the new value.
- The cache limiter defines which cache control HTTP headers are sent to the client. These headers determine the rules by which the page content may be cached by the client and intermediate proxies.
- Setting the cache limiter to nocache disallows any client/proxy caching. A value of public permits caching by proxies and the client, whereas private disallows caching by proxies and permits the client to cache the contents.

❖ **session_cache_expire**

- session_cache_expire -- Return current cache expire

Syntax:

int session_cache_expire ([int new_cache_expire])

- session_cache_expire() returns the current setting of *session.cache_expire*. The value returned should be read in minutes, defaults to 180. If *new_cache_expire* is given, the current cache expire is replaced with *new_cache_expire*.
- The cache expire is reset to the default value of 180 stored in *session.cache_limiter* at request startup time. Thus, you need to call session_cache_expire() for every request (and before session_start() is called).

Example:

```
<?php
        /* set the cache limiter to 'public' */
        session_cache_limiter('public');
        $cache_limiter = session_cache_limiter();

        /* set the cache expire to 30 minutes */
        session_cache_expire(30);
        $cache_expire = session_cache_expire();

        session_start();

        echo "The cache limiter is now set to $cache_limiter<br />";
        echo "The cached session pages expire after $cache_expire minutes";
?>
```

❖ **session_destroy**

- session_destroy -- Destroys all data registered to a session

Syntax:

bool session_destroy (void)

- session_destroy() destroys all of the data associated with the current session.
- It does not unset any of the global variables associated with the session, or unset the session cookie.
- Returns TRUE on success or FALSE on failure.

Example:

```
<?php
    session_start();
    // Unset all of the session variables.
    $_SESSION = array();

    // If it's desired to kill the session, also delete the session cookie.
    // Note: This will destroy the session, and not just the session data!
    if (isset($_COOKIE[session_name()])) {
       setcookie(session_name(), '', time()-42000, '/');
    }

    // Finally, destroy the session.
    session_destroy();
?>
```

❖ PHP Debugging Basics:

- Whether you're a PHP newbie or a wizard, your programs are going to have bugs in them.
- An error message that the PHP interpreter generates falls into one of five different categories:
  - **Parse error:** A problem with the syntax of your program, such as leaving a semicolon off of the end of a statement. The interpreter stops running your program when it encounters a parse error.
  - **Fatal error:** A severe problem with the content of your program, such as calling a function that hasn't been defined. The interpreter stops running your program when it encounters a fatal error.
  - **Warning:** An advisory from the interpreter that something is fishy in your program, but the interpreter can keep going. Using the wrong number of arguments when you call a function causes a warning.
  - **Notice:** A tip from the PHP interpreter, playing the role of Miss Manners. For example, printing a variable without first initializing it to some value generates a notice.
  - **Strict notice:** An admonishment from the PHP interpreter about your coding style. Most of these have to do with esoteric features that changed between PHP 4 and PHP 5, so you're not likely to run into them too much.

- **Configuring Error Reporting**

  - First of all, you need to configure the PHP interpreter so that when an error happens, you can see information about it. The error info can be sent along with program output to the web browser.
  - To make error messages display in the browser, set the display_errors configuration directive to On.
  - To send errors to the web server error log, set log_errors to On. You can set them both to On if you want error messages in both places.
  - The error_reporting configuration directive controls which kinds of errors the PHP interpreter reports. The default value for error_reporting is E_ALL & ~E_NOTICE & ~E_STRICT, which tells the interpreter to report all errors except notices and strict notices.
  - PHP defines some constants you can use to set the value of error_reporting so that only errors of certain types get reported: E_ALL (for all errors except strict notices), E_PARSE (parse errors), E_ERROR (fatal errors), E_WARNING (warnings), E_NOTICE (notices), and E_STRICT (strict notices).

- **Inspecting Program Data**

  - If your program is acting funny, add some checkpoints that display the values of variables. That way, you can see where the program's behavior diverges from your expectations.
  - The following program incorrectly attempts to calculate the total cost of a few items:

```
<?php
	$prices = array(5.95, 3.00, 12.50);
	$total_price = 0;
	$tax_rate = 1.08; // 8% tax

	foreach ($prices as $price) {
		$total_price = $price * $tax_rate;
	}

	printf('Total price (with tax): $%.2f', $total_price);
?>
```

The program doesn't do the right thing. It prints:
Total price (with tax): $13.50

The total price of the items should be at least $20.

One way you can try to find out is to insert a line in the foreach() loop that prints the value of $total_price before and after it changes.

```
<php
	$prices = array(5.95, 3.00, 12.50);
	$total_price = 0;
	$tax_rate = 1.08; // 8% tax

	foreach ($prices as $price) {
		print "[before: $total_price]";
		$total_price = $price * $tax_rate;
		print "[after: $total_price]";
	}

	printf('Total price (with tax): $%.2f', $total_price);
?>
```

This program prints:
[before: 0][after: 6.426][before: 6.426][after: 3.24][before: 3.24][after: 13.5]Total price (with tax): $13.50

From analyzing the debugging output, you can see that $total_price isn't increasing on each trip through the foreach() loop. Scrutinizing the code further leads you to the conclusion that the line:

$total_price = $price * tax_rate;

should be

$total_price += $price * tax_rate;

❖ How HTTP Works?

- When a request for a webpages is sent to the server, it contains more then just the desired URL. There is a lot of extra information that is sent as part of the request. This is also true of the response – the server send extra information back to the web browser.
- Whether it's a client request or a server response, every HTTP message has the same format, which break down in three sections:
    - The request / response line
    - The HTTP header
    - The HTTP body
- **The HTTP Request**
    - The HTTP request that the browser sends to the Web server contains a request line, a header and a body
    - Here's an example of the request line and header:

      GET / testpage.htm HTTP/1.1
      Accept: * / *
      Accept-Language: en-us
      Connection: Keep-Alive
      Host: www.wrox.com
      User-Agent: Mozila (X11; I; Linux 2.0.32 i586)

    - **The Request Line:**
    - The first line of every request is the request line, which contain three piece of information:
        - An HTTP command known as method (GET / POST)
        - The path from the server to the resource that the client is requesting
        - The version number of HTTP

      In Above Example :

      GET / testpage.htm HTTP/1.1

    - **The HTTP Request Header:**
    - The next bit information sent is the HTTP header.
    - The HTTP request header contains information that falls into three different categories:
        - **General:** Information about either the client or server, but not specific to one or the other.
        - **Entity:** Information about datat being sent between the client and server.
        - **Request:** Information about the client configuration and different type of acceptable documents.
    - A blank line to indicate that the header information is complete.

      In Above Example :

      Accept: * / *
      Accept-Language: en-us
      Connection: Keep-Alive
      Host: www.wrox.com
      User-Agent: Mozila (X11; I; Linux 2.0.32 i586)

- **The HTTP Request Body:**
- If the POST method is used in the HTTP request line then the HTTP request body contains any data that is being sent to the server.
- Otherwise HTTP request body is empty, as it is in the example.

- **The HTTP Response:**
  - The HTTP response is sent by the server back to the client browser, and contains a response line, a header and a body.
  - Here's an example of the response line and header:

```
HTTP/1.1 200 OK                                  //the status line
Date: Fri, 31st Oct 2005 12:12:19                //the General header
Server: Apache/1.3.12 (Unix) PHP/5.0.2           //the response header
Last-modified:  Tue, 26th Oct 2005 10:13:19      //the General header
                                                 //blank line
```

  - **The Response Line:**
  - The response line contains only two bits of information:
    - The HTTP version number.
    - An HTTP request code that reports the success or failure of the request

    In Above Example:

    HTTP/1.1 200 OK

  - **The Response Header:**
  - The HTTP response header is similar to the preceding request header.
  - In the HTTP response, the header information again fall into three types:
    - **General:** Information about either the client or server, but not specific to one or the other.
    - **Entity:** contains information about datat being sent between the client and server.
    - **Response:** contains information about the server sending the response and how it can deal with the response.
  - A blank line to indicate that the header information is complete.

    In Above Example:

```
Date: Fri, 31st Oct 2005 12:12:19                //the General header
Server: Apache/1.3.12 (Unix) PHP/5.0.2           //the response header
Last-modified:  Tue, 26th Oct 2005 10:13:19      //the General header
                                                 //blank line
```

  - **The Response Body:**
  - If the request was successful, the HTTP response body contains the HTML code, ready for the browser's interpretation.
  - If unsuccessful, a failure code is sent.

❖ **IIS Installation Step**

1. Start -> Settings -> Control Panel -> Add/Remove Programs -> Add/Remove Windows Components
2. Select "Internet Information Services (IIS)" then click on "Details".
3. Some of the components you do not need, but for this example, we'll install the whole IIS package since we want the web server, FTP server, and mail capabilities.
4. Click on "Next" The dialog box will show files being copied to your hard disk.
5. After a few moments, you'll get this dialog box that ask you to insert the Windows 2000 CD into your CD-ROM drive. Put in the CD then click "OK".
6. The files will continue to be copied. This could take a while. When everything is done, you'll see this screen. Click on "Finish".

❖ **Configure PHP on IIS Web Server**

1. Go to your Windows Control Panel then click on Administrative Tools and then Click Internet Information Services icon.
2. Expand the tree and Right click on the Default Web Site and click on the Properties.
3. The Default Web Site properties appear, now click on the Home Directory
4. In the Home Directory properties click on the Configuration button.
5. In the Application Configuration properties click on the Add button.
6. Now you get the Add/Edit Application Extension Mapping window enter the executable as C:\PHP\PHP.EXE (or the appropriate location where you have installed PHP) and the extension as .php as shown in the above figure and click OK you have successfully configured PHP on your IIS Web Server!

❖ **Apache Web Server Configuration File**

The configuration directives are grouped into three basic sections:

1. Directives that control the operation of the Apache server process as a whole (the 'global environment').
2. Directives that define the parameters of the 'main' or 'default' server, which responds to requests that aren't handled by a virtual host. These directives also provide default values for the settings of all virtual hosts.
3. Settings for virtual hosts, which allow Web requests to be sent to different IP addresses or hostnames and have them handled by the same Apache server process.

Section 1: Global Environment

- **ServerType**
  - ServerType is either inetd, or standalone.
  - Inetd mode is only supported onUnix platforms.

    ServerType standalone

- **ServerRoot:**
  - The top of the directory tree under which the server's configuration, error, and log files are kept.
  - Do NOT add a slash at the end of the directory path.

    ServerRoot "D:/Program Files/Apache Group/Apache"

- **Timeout:**
  - The number of seconds before receives and sends time out.

    Timeout 300

- **KeepAlive:**
  - Whether or not to allow persistent connections (more than one request per connection).
  - Set to "Off" to deactivate.

    KeepAlive On

- **KeepAliveTimeout:**

  - Number of seconds to wait for the next request from the same client on the same connection.

    KeepAliveTimeout 15

- **AddModule**
  - Apache Modules compiled into the standard Windows build
  - This is an advanced option that may render your server inoperable! Do not use these directives without expert guidance.

    AddModule mod_php5.c

- **LoadModule [Dynamic Shared Object (DSO) Support]**
  - To be able to use the functionality of a module which was built as a DSO you have to place corresponding `LoadModule' lines at this location so the directives contained in it are actually available _before_ they are used.
  - The order in which modules are loaded is important. Don't change the order below without expert advice.

    LoadModule php5_module "D:/php/php5apache.dll"

Section 2: 'Main' server configuration

- **Port:**
  - The port to which the standalone server listens.
  - Certain firewall products must be configured before Apache can listen to a specific port.

```
Port 80
```

- **ServerName**
  - ServerName allows you to set a host name which is sent back to clients for your server if it's different than the one the program would get (i.e., use "www" instead of the host's real name).
  - You cannot just invent host names and hope they work. The name you define here must be a valid DNS name for your host. If you don't understand this, ask your network administrator.
  - If your host doesn't have a registered DNS name, enter its IP address here. You will have to access it by its address (e.g., http://123.45.67.89/) anyway, and this will make redirections work in a sensible way.
  - 127.0.0.1 is the TCP/IP local loop-back address, often named localhost. Your machine always knows itself by this address.
  - If you use Apache strictly for local testing and development, you may use 127.0.0.1 as the server name.

```
#ServerName new.host.name
```

- **DocumentRoot:**
  - The directory out of which you will serve your documents.
  - By default, all requests are taken from this directory, but symbolic links and aliases may be used to point to other locations.

```
#DocumentRoot "D:/Program Files/Apache Group/Apache/htdocs"
DocumentRoot "D:/gautam"
```

- **Directory**
  - This should be changed to whatever you set DocumentRoot to.

```
<Directory "D:/Program Files/Apache Group/Apache/htdocs">
<Directory "D:/gautam">
```

- **AddType**
  - TypesConfig describes where the mime.types file (or equivalent) is to be found.

```
<IfModule mod_mime.c>
        AddType application/x-httpd-php .php
        AddType application/x-httpd-php .php3
        AddType application/x-httpd-php-source .phps
        TypesConfig conf/mime.types
</IfModule>
```

Section 3: Virtual Hosts

- **VirtualHost:**
  - If you want to maintain multiple domains/hostnames on your machine you can setup VirtualHost containers for them.
  - Most configurations use only name-based virtual hosts so the server doesn't need to worry about IP addresses. This is indicated by the asterisks in the directives below.
  - **Use name-based virtual hosting.**

    NameVirtualHost *

VirtualHost example:

- Almost any Apache directive may go into a VirtualHost container.
- The first VirtualHost section is used for requests without a known server name.

```
<VirtualHost *>
    ServerAdmin webmaster@dummy-host.example.com
    DocumentRoot /www/docs/dummy-host.example.com
    ServerName dummy-host.example.com
    ErrorLog logs/dummy-host.example.com-error_log
    CustomLog logs/dummy-host.example.com-access_log common
</VirtualHost>
```