PHP: **H**ypertext **P**reprocessor

PHP

- PHP is a server-side scripting language, like ASP
- PHP scripts are executed on the server
- PHP supports many databases
- PHP is an open source software
- PHP is free to download and use

- PHP files can contain text, HTML tags and scripts
- PHP files are returned to the browser as plain HTML
- PHP files have a file extension of ".php", ".php3", or ".phtml"

Server-Side Scripting

- Server-side web scripting is mostly about connecting web sites to backend servers, processing data and controlling the behavior of higher layers such as HTML and CSS
- Two-way communication:
- ■**Server to client**: Web pages can be assembled from backend-server output
- ■**Client to server**: Customer-entered information can be acted upon

Learning PHP Syntax and Variables

- PHP Is Forgiving
- PHP is whitespace insensitive
- PHP is sometimes case sensitive

```
<?php
$capital = 67;
print("Variable capital is $capital<BR>");
print("Variable CaPiTaL is $CaPiTaL<BR>");
?>
```

  - O/p
  - Variable capital is 67
  - Variable CaPiTaL is

- Function names are *not* case sensitive, and neither are the basic language constructs (if, then, else, while, and the like)

Learning PHP Syntax and Variables

- Statements are expressions terminated by semicolons
- Expressions are combinations of tokens
  - The smallest building blocks of PHP are the *indivisible tokens*, such as numbers (3.14159), strings("two"), variables ($two), constants (TRUE), and the special words that make up the syntax of PHP itself (if, else, and so forth
- Precedence, associativity, and evaluation order
- Expressions and types
  - 2 + 2 * "nonsense" + TRUE

Comments in PHP

```
<html>
<body>
<?php
//This is a comment
#This is a comment


/*
This is
a comment
block
*/
?>

</body>
```

Variables in PHP

```
$var_name = value;
```

```
<?php
$txt="Hello World!";
$x=16;
?>
```

PHP is a Loosely Typed Language

Variables

All variables in PHP are denoted with a leading dollar sign ($).

■■The value of a variable is the value of its most recent assignment.

■■Variables are assigned with the = operator, with the variable on the left-hand side and the

expression to be evaluated on the right.

■■Variables can, but do not need, to be declared before assignment.

■■Variables have no intrinsic type other than the type of their current value.

■■Variables used before they are assigned have default values

- A variable name must start with a letter or an underscore "_"
- A variable name can only contain alpha-numeric characters and underscores (a-z, A-Z, 0-9, and _)
- A variable name should not contain spaces. If a variable name is more than one word, it should be separated with an underscore ($my_string), or with capitalization ($myString)

Variables

- Checking assignment with isset
  - isset that tests a variable to see whether it has been assigned a value
  - unset($set_var); will make $set_var into an unbound variable

```
$set_var = 0;
if (isset($set_var))
print("set_var is set.<BR>");
else
print("set_var is not set.<BR>");
```

Variable scope

A variable declared outside a function has a GLOBAL SCOPE and can only be accessed outside a function:

```
<?php
$x = 5; // global scope
function myTest() {
  // using x inside this function will generate an error
  echo "<p>Variable x inside function is: $x</p>";
}
myTest();
echo "<p>Variable x outside function is: $x</p>";
?>
```

A variable declared within a function has a LOCAL SCOPE and can only be accessed within that function:

```
<?php
function myTest() {
  $x = 5; // local scope
  echo "<p>Variable x inside function is: $x</p>";
}
myTest();
// using x outside the function will generate an error
echo "<p>Variable x outside function is: $x</p>";
?>
```

Variable scope

The global keyword is used to access a global variable from within a function.

```
<?php
$x = 5;
$y = 10;
function myTest() {
  global $x, $y;
  $y = $x + $y;
}
myTest();
echo $y; // outputs 15
?>
```

Each time the function is called, that variable will still have the information it contained from the last time the function was called.

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

Constants

- No $ sign in name
- Uppercase letters
- Global scope

- error_reporting(E_ALL)

- define(MY_ANSWER, 42)

Types assigned by context

```
$sub = substr(12345, 2, 2);
print("sub is $sub<BR>");
```

34

Type Summary

- *Integers* are whole numbers, without a decimal point, like 495.
- *Doubles* are floating-point numbers, like 3.14159 or 49.0.
- *Booleans* have only two possible values: TRUE and FALSE.
- *NULL* is a special type that only has one value: NULL.
- *Strings* are sequences of characters, like 'PHP 4.0 supports string operations.'
- *Arrays* are named and indexed collections of other values.
- *Objects* are instances of programmer-defined classes, which can package up both other kinds of values and functions that are specific to the class.
- *Resources* are special variables that hold references to resources external to PHP (such as database connections).

```
$literally = 'My $variable will not print!\\n';
print($literally);
```

produces the browser output:

```
My $variable will not print!\n
```

To embed a single quotation mark (such as an apostrophe) in a singly quoted string, escape it with a backslash, as in the following:

```
$singly_quoted = 'This quote mark\'s no big deal either';
```

Although in most contexts backslashes are interpreted literally in singly quoted strings, you may also use two backslashes (\\) as an escape sequence for a single (nonescaping) backslash. This is useful when you want a backslash as the final character in a string, as in:

```
$win_path = 'C:\\InetPub\\PHP\\';
print("A Windows-style pathname: $win_path<BR>");
```

which is displayed as:

```
A Windows-style pathname: C:\InetPub\PHP\
```

Just as with singly quoted strings, quotes of the opposite type can be freely included without an escape character:

```
$has_apostrophe = "There's no problem here";
```

```
$animal = "antelope"; // first assignment
$saved_string = "The animal is $animal<BR>";
$animal = "zebra"; // reassignment
print("The animal is $animal<BR>"); //first display line
print($saved_string); //second display line
```

```
The animal is zebra
The animal is antelope
```

```
echo "This will print in the user's browser window.";
```

Or equivalently:

```
echo("This will print in the user's browser window.");
```

```
echo "This will print in the ", "user's browser window.";
```

The parenthesized version, however, will not accept multiple arguments:

```
echo ("This will produce a ", "PARSE ERROR!");
```

echo "Hello" ,"World"; ----HelloWorld
echo ("Hello" ,"World")----Error
Print "Hello" ,"World";----Error
Print ("Hello" ,"World")----Error

The command `print` is very similar to `echo`, with two important differences:

- Unlike `echo`, `print` can accept only one argument.
- Unlike `echo`, `print` returns a value, which represents whether or not the `print` statement succeeded.

Operators,Conditional Statements, Loops

Boolean constants

```
if (TRUE)
print("This will always print<BR>");
else
print("This will never print<BR>");
```

Logical operators

Logical operators combine other logical values to produce new Boolean values

Logical Operators

Operator	Behavior
and	Is true if and only if both of its arguments are true.
or	Is true if either (or both) of its arguments are true.
!	Is true if its single argument (to the right) is false and false if its argument is true.
xor	Is true if either (but not both) of its arguments are true.
&&	Same as and but binds to its arguments more tightly. (See the discussion of precedence later in the chapter.)
||	Same as or but binds to its arguments more tightly.

Logical operators

- order of precedence are: !, &&, ||, and, xor
- Logical operators short-circuit
  - if ($denom != 0 && $numer / $denom> 2)
    print(“More than twice as much!”);

The ternary operator

testExpression ?yesExpression :noExpression

```
$max_num = $first_num> $second_num ? $first_num : $second_num;
```

Branching

- **if statement - use this statement to execute some code only if a specified condition is true**
- **if...else statement - use this statement to execute some code if a condition is true and another code if the condition is false**
- **if...elseif....else statement - use this statement to select one of several blocks of code to be executed**
- **switch statement - use this statement to select one of many blocks of code to be executed**

The if Statement

- **if (condition) code to be executed if condition is true;**

- <html><body><?php
- $d=date("D");
- if ($d=="Fri") echo "Have a nice weekend!";
- ?></body></html>

The if...else Statement

- if (condition)
- code to be executed if condition is true;
- else
- code to be executed if condition is false;

- <html><body><?php
- $d=date("D");
- if ($d=="Fri")
- echo "Have a nice weekend!";
- else
- echo "Have a nice day!";
- ?></body></html>

The if Statement

```
<html><body><?php
$d=date("D");
if ($d=="Fri")
  {
  echo "Hello!<br />";
  echo "Have a nice weekend!";
  echo "See you on Monday!";
  }
?></body></html>
```

The if...elseif....else Statement

```
if (condition)
  code to be executed if condition is true;
elseif (condition)
  code to be executed if condition is true;
else
  code to be executed if condition is false;
```

```
<html><body><?php
$d=date("D");
if ($d=="Fri")
  echo "Have a nice weekend!";
elseif ($d=="Sun")
  echo "Have a nice Sunday!";
else
  echo "Have a nice day!";
?></body></html>
```

The PHP Switch Statement

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

The PHP Switch Statement

```
<html><body><?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";
}
?></body></html>
```

Looping

- while - loops through a block of code while a specified condition is true
- do...while - loops through a block of code once, and then repeats the loop as long as a specified condition is true
- for - loops through a block of code a specified number of times
- foreach - loops through a block of code for each element in an array

The while Loop

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

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

The do...while Statement

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

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

The for Loop

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

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

The foreach Loop

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

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

Break and continue

```
for ($x = 1; $x < 10; $x++)
{
// if $x is odd, break out
if ($x % 2 != 0)
break;
print("$x ");
}

for ($x = 1; $x < 10; $x++)
{
// if $x is odd, skip this loop
if ($x % 2 != 0)
continue;
print("$x ");
}
```

Using Functions

- function_name(expression_1, expression_2, ..., expression_n)

- sqrt(9);
- rand(10, 10 + 10);
- strlen("This has 22 characters");
- pi();

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

Using Functions

```
Function better_deal ($amount_1, $price_1,$amount_2, $price_2)
{
$per_amount_1 = $price_1 / $amount_1;
$per_amount_2 = $price_2 / $amount_2;
return($per_amount_1 < $per_amount_2);
}
$liters_1 = 1.0;
$price_1 = 1.59;
$liters_2 = 1.5;
$price_2 = 2.09;
if (better_deal($liters_1, $price_1,$liters_2, $price_2))
print("The first deal is better!<BR>");
else
print("The second deal is better!<BR>");
```

Functions and Variable Scope

```
Function SayMyABCs ()
{
$count = 0;
while ($count < 10)
{
print(chr(ord('A') + $count));
$count = $count + 1;
}
print("<BR>Now I know $count letters<BR>");
}
$count = 0;
SayMyABCs();                                ABCDEFGHIJ
$count = $count + 1;
print("Now I've made $count function call(s).<BR>");     1
SayMyABCs();                                ABCDEFGHIJ
$count = $count + 1;
print("Now I've made $count function call(s).<BR>");     2
```

Global versus local

```
function SayMyABCs2 ()
{
global $count;
while ($count < 10)
{
print(chr(ord('A') + $count));
$count = $count + 1;
}
print("<BR>Now I know $count letters<BR>");          10
}
$count = 0;
SayMyABCs2();                          ABCDEFGHIJ
$count = $count + 1;                   11
print("Now I've made $count function call(s).<BR>");     11
SayMyABCs2();                                            ----
$count = $count + 1;
print("Now I've made $count function call(s).<BR>");     12
```

Static variables

```
function SayMyABCs3 ()
{
static $count = 0; //assignment only if first time called
$limit = $count + 10;
while ($count < $limit)
{
print(chr(ord('A') + $count));
$count = $count + 1;
}
print("<BR>Now I know $count letters<BR>");
}
$count = 0;
SayMyABCs3();
$count = $count + 1;
print("Now I've made $count function call(s).<BR>");
SayMyABCs3();
$count = $count + 1;
print("Now I've made $count function call(s).<BR>");
```

ABCDEFGHIJ ,KLMNOPQRST

10 20

1

2

Recursion

```
function countdown ($num_arg)
{
if ($num_arg> 0)
{
print("Counting down from $num_arg<BR>");
countdown($num_arg - 1);
}
}
countdown(5);
```

PHP Form Handling

```
<html><body>
<form action="welcome.php" method="post">
Name: <input type="text" name="fname" />
Age: <input type="text" name="age" />
<input type="submit" />
</form></body></html>
```

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

The $_GET Function

```
<form action="welcome.php" method="get">
Name: <input type="text" name="fname" />
Age: <input type="text" name="age" />
<input type="submit" />
</form>
```

When the user clicks the "Submit" button, the URL sent to the server could look something like this:

http://www.w3schools.com/welcome.php?fname=Peter&age=37

The "welcome.php" file can now use the $_GET function to collect form data (the names of the form fields will automatically be the keys in the $_GET array):

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

The $_POST Function

```
<form action="welcome.php" method="post">
Name: <input type="text" name="fname" />
Age: <input type="text" name="age" />
<input type="submit" />
</form>
```

When the user clicks the "Submit" button, the URL sent to the server could look something like this:

http://www.w3schools.com/welcome.php

The "welcome.php" file can now use the $_POST function to collect form data (the names of the form fields will automatically be the keys in the $_POST array):

```
Welcome <?php echo $_POST["fname"]; ?>.<br />
You are <?php echo $_POST["age"]; ?> years old!
```

The PHP $_REQUEST Function

The $_REQUEST function can be used to collect form data sent with both the GET and POST methods.

```
Welcome <?php echo $_REQUEST["fname"]; ?>!<br />
You are <?php echo $_REQUEST["age"]; ?> years old.
```

String Variables in PHP

```
<?php
$txt="Hello World";
echo $txt;
?>
```

The Concatenation Operator (.)

```
<?php
$txt1="Hello World!";
$txt2="What a nice day!";
echo $txt1 . " " . $txt2;
?>
```

The strlen() function

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

The strpos() function

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

O/p 6

Array

In PHP, there are three kind of arrays:

- **Numeric array** - An array with a numeric index
- **Associative array** - An array where each ID key is associated with a value
- **Multidimensional array** - An array containing one or more arrays

Numeric Arrays

A numeric array stores each array element with a numeric index.

There are two methods to create a numeric array.

In the following example the index are automatically assigned (the index starts at 0):

```
$cars=array("Saab","Volvo","BMW","Toyota");
```

2. In the following example we assign the index manually:

```
$cars[0]="Saab";
$cars[1]="Volvo";
$cars[2]="BMW";
$cars[3]="Toyota";
```

```
<?php
$cars[0]="Saab";
$cars[1]="Volvo";
$cars[2]="BMW";
$cars[3]="Toyota";
echo $cars[0] . " and " . $cars[1] . " are Swedish cars.";
?>
```

Associative Arrays

An associative array, each ID key is associated with a value.

With associative arrays we can use the values as keys and assign values to them

```
$ages = array("Peter"=>32, "Quagmire"=>30, "Joe"=>34);
```

OR

```
$ages['Peter'] = "32";
$ages['Quagmire'] = "30";
$ages['Joe'] = "34";
```

```
<?php
$ages['Peter'] = "32";
$ages['Quagmire'] = "30";
$ages['Joe'] = "34";
echo "Peter is " . $ages['Peter'] . " years old.";
?>
```

Multidimensional Arrays

```
<html>
<body>
<?php
$cars = array
  (
  array("Volvo",22,18),
  array("BMW",15,13),
  array("Saab",5,2),
  array("Land Rover",17,15)
  );
```

Volvo: In stock: 22, sold: 18.
BMW: In stock: 15, sold: 13.
Saab: In stock: 5, sold: 2.
Land Rover: In stock: 17, sold: 15.

Volvo	22	18
BMW	15	13
Saab	5	2
Land Rover	17	15

```
echo $cars[0][0].": In stock: ".$cars[0][1].", sold: ".$cars[0][2].".<br>";
echo $cars[1][0].": In stock: ".$cars[1][1].", sold: ".$cars[1][2].".<br>";
echo $cars[2][0].": In stock: ".$cars[2][1].", sold: ".$cars[2][2].".<br>";
echo $cars[3][0].": In stock: ".$cars[3][1].", sold: ".$cars[3][2].".<br>";
?>
</body>
</html>
```

Multidimensional Arrays

```
<html><body><?php
$cars = array
  (
  array("Volvo",22,18),
  array("BMW",15,13),
  array("Saab",5,2),
  array("Land Rover",17,15)
  );
 for ($row = 0; $row < 4; $row++) {
  echo "<p><b>Row number $row</b></p>";
  echo "<ul>";
  for ($col = 0; $col < 3; $col++) {
    echo "<li>".$cars[$row][$col]."</li>";
  }
  echo "</ul>";
}
?></body></html>
```

Row number 0

- Volvo
- 22
- 18

Row number 1

- BMW
- 15
- 13

Row number 2

- Saab
- 5
- 2

Row number 3

- Land Rover
- 17
- 15

Multidimensional Arrays

```
<html>
<body>
<?php
$families = array
  (
  "Griffin"=>array("Peter","Lois","Megan"),
  "Quagmire"=>array("Glenn"),
  "Brown"=>array("Cleveland","Loretta","Junior")
  );

echo $families["Griffin"][2]."<br>";
echo $families['Brown'][0]."<br>";
?>
</body>
</html>
```

$families[“Griffin”][0]
$families[“Brown”][2]

Megan
Cleveland

Multidimensional Arrays

```
<?php
// Multidimensional array
$superheroes = array(
    "spider-man" => array(
        "name" => "Peter Parker",
        "email" => "peterparker@mail.com",
    ),
    "super-man" => array(
        "name" => "Clark Kent",
        "email" => "clarkkent@mail.com",
    ),
    "iron-man" => array(
        "name" => "Harry Potter",
        "email" => "harrypotter@mail.com",
    )
);
 // Printing all the keys and values one by one
$keys = array_keys($superheroes);
for($i = 0; $i < count($superheroes); $i++) {
    echo $keys[$i] . "{<br>";
    foreach($superheroes[$keys[$i]] as $key => $value) {
        echo $key . " : " . $value . "<br>";
    }
    echo "}<br>";
```

```
spider-man{
name : Peter Parker
email : peterparker@mail.com
}
super-man{
name : Clark Kent
email : clarkkent@mail.com
}
iron-man{
name : Harry Potter
email : harrypotter@mail.com
}
```

PHP Array Functions

Function	Description
array()	Creates an array
array_change_key_case()	Returns an array with all keys in lowercase or uppercase
array_chunk()	Splits an array into chunks of arrays
array_combine()	Creates an array by using one array for keys and another for its values
array_count_values()	Returns an array with the number of occurrences for each value
array_diff()	Compares array values, and returns the differences
array_diff_assoc()	Compares array keys and values, and returns the differences
array_diff_key()	Compares array keys, and returns the differences
array_diff_uassoc()	Compares array keys and values, with an additional user-made function check, and returns the differences
array_diff_ukey()	Compares array keys, with an additional user-made function check, and returns the differences
array_fill()	Fills an array with values
array_filter()	Filters elements of an array using a user-made function
array_flip()	Exchanges all keys with their associated values in an array
array_intersect()	Compares array values, and returns the matches
array_intersect_assoc()	Compares array keys and values, and returns the matches
array_intersect_key()	Compares array keys, and returns the matches
array_intersect_uassoc()	Compares array keys and values, with an additional user-made function check, and returns the matches
array_intersect_ukey()	Compares array keys, with an additional user-made function check, and returns the matches

PHP Array Functions

array_key_exists()	Checks if the specified key exists in the array
array_keys()	Returns all the keys of an array
array_map()	Sends each value of an array to a user-made function, which returns new values
array_merge()	Merges one or more arrays into one array
array_merge_recursive()	Merges one or more arrays into one array
array_multisort()	Sorts multiple or multi-dimensional arrays
array_pad()	Inserts a specified number of items, with a specified value, to an array
array_pop()	Deletes the last element of an array
array_product()	Calculates the product of the values in an array
array_push()	Inserts one or more elements to the end of an array
array_rand()	Returns one or more random keys from an array
array_reduce()	Returns an array as a string, using a user-defined function
array_reverse()	Returns an array in the reverse order
array_search()	Searches an array for a given value and returns the key
array_shift()	Removes the first element from an array, and returns the value of the removed element
array_slice()	Returns selected parts of an array
array_splice()	Removes and replaces specified elements of an array
array_sum()	Returns the sum of the values in an array
array_udiff()	Compares array values in a user-made function and returns an array
array_udiff_assoc()	Compares array keys, and compares array values in a user-made function, and returns an array

PHP Array Functions

array_udiff_uassoc()	Compares array keys and array values in user-made functions, and returns an array
array_uintersect()	Compares array values in a user-made function and returns an array
array_uintersect_assoc()	Compares array keys, and compares array values in a user-made function, and returns an array
array_uintersect_uassoc()	Compares array keys and array values in user-made functions, and returns an array
array_unique()	Removes duplicate values from an array
array_unshift()	Adds one or more elements to the beginning of an array
array_values()	Returns all the values of an array
array_walk()	Applies a user function to every member of an array
array_walk_recursive()	Applies a user function recursively to every member of an array
arsort()	Sorts an array in reverse order and maintain index association
asort()	Sorts an array and maintain index association
compact()	Create array containing variables and their values
count()	Counts elements in an array, or properties in an object
current()	Returns the current element in an array
each()	Returns the current key and value pair from an array
end()	Sets the internal pointer of an array to its last element
extract()	Imports variables into the current symbol table from an array
in_array()	Checks if a specified value exists in an array
key()	Fetches a key from an array

PHP Array Functions

krsort()	Sorts an array by key in reverse order
ksort()	Sorts an array by key
list()	Assigns variables as if they were an array
natcasesort()	Sorts an array using a case insensitive "natural order" algorithm
natsort()	Sorts an array using a "natural order" algorithm
next()	Advance the internal array pointer of an array
pos()	Alias of current()
prev()	Rewinds the internal array pointer
range()	Creates an array containing a range of elements
reset()	Sets the internal pointer of an array to its first element
rsort()	Sorts an array in reverse order
shuffle()	Shuffles an array
sizeof()	Alias of count()
sort()	Sorts an array
uasort()	Sorts an array with a user-defined function and maintain index association
uksort()	Sorts an array by keys using a user-defined function
usort()	Sorts an array by values using a user-defined function

```
<html>
<body>

<?php
$cars=array("Volvo","BMW","Toyota");
$arrlength=count($cars);

for($x=0;$x<$arrlength;$x++)
  {
  echo $cars[$x];
  echo "<br>";
  }
?>

</body>
</html>
```

```
<html>
<body>
<?php
$age=array("Peter"=>"35","Ben"=>"37","Joe"=>"43");

foreach($age as $x=>$x_value)
  {
  echo "Key=" . $x . ", Value=" . $x_value;
  echo "<br>";
  }
?>
</body>
</html>
```

O/P

Key=Peter, Value=35
Key=Ben, Value=37
Key=Joe, Value=43

PHP - Sort Functions For Arrays

sort() - sort arrays in ascending order

rsort() - sort arrays in descending order

asort() - sort associative arrays in ascending order, according to the value

ksort() - sort associative arrays in ascending order, according to the key

arsort() - sort associative arrays in descending order, according to the value

krsort() - sort associative arrays in descending order, according to the key

```
<html>
<body>
<?php
$cars=array("Volvo","BMW","Toyota");
rsort($cars);

$clength=count($cars);
for($x=0;$x<$clength;$x++)
  {
  echo $cars[$x];
  echo "<br>";
  }
?>
</body>
</html>
```

O/p

Volvo
Toyota
BMW

```
<html>
<body>

<?php
$age=array("Peter"=>"35","Ben"=>"37","Joe"=>"43");
ksort($age);

foreach($age as $x=>$x_value)
   {
   echo "Key=" . $x . ", Value=" . $x_value;
   echo "<br>";
   }
?>
</body>
</html>
```

O/P

Key=Ben, Value=37
Key=Joe, Value=43
Key=Peter, Value=35

```
<html>
<body>
<?php
$age=array("Peter"=>"35","Ben"=>"37","Joe"=>"43");
arsort($age);
foreach($age as $x=>$x_value)
   {
   echo "Key=" . $x . ", Value=" . $x_value;
   echo "<br>";
   }
?>
</body>
</html>
```

O/p

Key=Joe, Value=43
Key=Ben, Value=37
Key=Peter, Value=35

Create an array by using the elements from one "keys" array and one "values" array

```
<html>
<body>
<?php
$fname=array("Peter","Ben","Joe");
$age=array("35","37","43");
$c=array_combine($fname,$age);
print_r($c);
?>
</body>
</html>
```

O/P

Array ([Peter] => 35 [Ben] => 37 [Joe] => 43)

```
<html>
<body>

<?php
$age=array("Peter"=>"35","Ben"=>"37","Joe"=>"43");
print_r(array_change_key_case($age,CASE_UPPER));
?>

</body>
</htm
```

O/P

Array ([PETER] => 35 [BEN] => 37 [JOE] => 43)

Compare the keys of two arrays, and return the differences

```
<html>
<body>

<?php
$a1=array("red","green","blue","yellow");
$a2=array("black","pink","white");
$result=array_diff_key($a1,$a2);
print_r($result);
?>

</body>
</html>
```

O/P

Array ([3] => yellow)

Compare the keys of two arrays, and return the differences

```
<html><body>
<?php
$a1=array("a"=>"red","b"=>"green","c"=>"blue");
$a2=array("a"=>"red","c"=>"blue","d"=>"pink");
$result=array_diff_key($a1,$a2);
print_r($result);
?>
</body></html>
```

```
Array ( [b] => green )
```

array_fill(*index*,*number*,*value*);

```
<html>
<body>
<?php
$a1=array_fill(3,4,"blue");
$b1=array_fill(0,1,"red");
print_r($a1);
echo "<br>";
print_r($b1);
?>
</body>
</htm
```

O/P

Array ([3] => blue [4] => blue [5] => blue [6] => blue)
Array ([0] => red)

array_splice(*array,start,length,array*)

Remove elements from an array and replace it with new elements

```
<?php
$a1=array(0=>"Dog",1=>"Cat",2=>"Horse",3=>"Bird");
$a2=array(0=>"Tiger",1=>"Lion");
array_splice($a1,0,2,$a2);
print_r($a1);
?>
```

The output of the code above will be:

Array ([0] => Tiger [1] => Lion [2] => Horse [3] => Bird)

array_slice(*array,start,length,preserve*)

Returns selected parts of an array

```
<?php
$a=array("red","green","blue","yellow","brown");
print_r(array_slice($a,1,2));
?>
```

Array ([0] => green [1] => blue)

PHP Array Constants

Constant	Description
CASE_LOWER	Used with array_change_key_case() to convert array keys to lower case
CASE_UPPER	Used with array_change_key_case() to convert array keys to upper case
SORT_ASC	Used with array_multisort() to sort in ascending order
SORT_DESC	Used with array_multisort() to sort in descending order
SORT_REGULAR	Used to compare items normally
SORT_NUMERIC	Used to compare items numerically
SORT_STRING	Used to compare items as strings
SORT_LOCALE_STRING	Used to compare items as strings, based on the current locale

The array() construct

$fruit_basket = array(‘apple’, ‘orange’, ‘banana’, ‘pear’);

$fruit_basket = array(0 => ‘apple’, 1 => ‘orange’,2 => ‘banana’, 3 => ‘pear’);

$fruit_basket = array(‘red’ => ‘apple’, ‘orange’ => ‘orange’,‘yellow’ => ‘banana’, ‘green’ => ‘pear’);

$fruit_basket[‘yellow’] // will be equal to ‘banana’

$my_empty_array = array();

creates an array with no elements.

Functions returning arrays

```
$my_array = range(1,5);
```

is equivalent to:

```
$my_array = array(1, 2, 3, 4, 5);
```

Functions returning arrays

Retrieving by index

$my_array[5]

The list() construct

$fruit_basket = array(‘apple’, ‘orange’, ‘banana’);

list($red_fruit, $orange_fruit) = $fruit_basket;

Multidimensional Arrays

- $multi_array[1][2][3][4][5] = “deeply buried treasure”;

```
$cornucopia = array(‘fruit’ =>
    array(‘red’ => ‘apple’,‘orange’ => ‘orange’,‘yellow’ => ‘banana’,
    ‘green’ => ‘pear’),
‘flower’ =>
    array(‘red’ => ‘rose’,‘yellow’ => ‘sunflower’,‘purple’ => ‘iris’));

$kind_wanted = ‘flower’;
$color_wanted = ‘purple’;
print(“The $color_wanted $kind_wanted is “ .
$cornucopia[$kind_wanted][$color_wanted]);
```

O/p The purple flower is iris

Simple Functions for Inspecting Arrays

Function	Behavior
`is_array()`	Takes a single argument of any type and returns a true value if the argument is an array, and false otherwise.
`count()`	Takes an array as argument and returns the number of nonempty elements in the array. (This will be 1 for strings and numbers.)
`sizeof()`	Identical to `count()`.
`in_array()`	Takes two arguments: an element (that might be a value in an array), and an array (that might contain the element). Returns `true` if the element is contained as a value in the array, `false` otherwise. (Note that this does not test for the presence of keys in the array.)
`isset($array[$key])`	Takes an *`array[key]`* form and returns `true` if the key portion is a valid key for the array. (This is a specific use of the more general function `isset()`, which tests whether a variable is bound.)

```
<?php
$people = array("Peter", "Joe", "Glenn", "Cleveland");

if (in_array("Glenn", $people))
  {
  echo "Match found";
  }
else
  {
  echo "Match not found";
  }
?>
```

Deleting from Arrays

```
$my_array[0] = ‘wanted’;
$my_array[1] = ‘unwanted’;
$my_array[2] = ‘wanted again’;
unset($my_array[1]);
```

```
$my_array[1] = ‘’;
```

Using iteration functions

```
$major_city_info = array();
$major_city_info[0] = 'Chicago';
$major_city_info['Chicago'] = 'United States';
$major_city_info[1] = 'Stockholm';
$major_city_info['Stockholm'] = 'Sweden';
$major_city_info[2] = 'Montreal';
$major_city_info['Montreal'] = 'Canada';
function city_by_number ($number_index, $city_array)
{
if (IsSet($city_array[$number_index]))
{
$the_city = $city_array[$number_index];
$the_country = $city_array[$the_city];
print("$the_city is in $the_country<BR>");
} }

city_by_number(0, $major_city_info);
```

Chicago is in United States

foreach

```
function print_all_foreach ($city_array)
{
foreach ($city_array as $name_value) {
print("$name_value<BR>");
}
}
print_all_foreach($major_city_info);
```

Chicago
United States
Stockholm
Sweden
Montreal
Canada

Iterating with current() and next()

```
function print_all_next($city_array)
{
$current_item = current($city_array);
if ($current_item)
print("$current_item<BR>");
else
print("There's nothing to print");
while($current_item = next($city_array))
print("$current_item<BR>");
}
print_all_next(&$major_city_info);
print_all_next(&$major_city_info);
```

Chicago
United States
Stockholm
Sweden
Montreal
Canada
There's nothing to print

Starting over with reset()

```
function print_all_array_reset($city_array)
{
$current_item = reset($city_array);
if ($current_item)
print("$current_item<BR>");
else
print("There's nothing to print");
while($current_item = next($city_array))
print("$current_item<BR>");
}
```

Reverse order with end() and prev()

```
function print_all_array_backwards($city_array)
{
$current_item = end($city_array);
if ($current_item)
print("$current_item<BR>");
else
print("There's nothing to print");
while($current_item = prev($city_array))
print("$current_item<BR>");
}
print_all_array_backwards($major_city_info);
```

Extracting keys with key()

```
function print_keys_and_values($city_array)
{
reset($city_array);
$current_value = current($city_array);
$current_key = key($city_array);
if ($current_value)
print("Key: $current_key; Value: $current_value<BR>");
else
print("There's nothing to print");
while($current_value = next($city_array))
{
$current_key = key($city_array);
print("Key: $current_key; Value: $current_value<BR>");
}
}
print_keys_and_values($major_city_info);
```

Key: 0; Value: Chicago
Key: Chicago; Value: United States
Key: 1; Value: Stockholm
Key: Stockholm; Value: Sweden
Key: 2; Value: Montreal
Key: Montreal; Value: Canada

Empty values and the each() function

```
function print_keys_and_values_each($city_array)
{
reset($city_array);
while ($array_cell = each($city_array))
{
$current_value = $array_cell[‘value’];
$current_key = $array_cell[‘key’];
print(“Key: $current_key; Value: $current_value<BR>”);
}
}
print_keys_and_values_each($major_city_info);
```

Key: 0; Value: Chicago
Key: Chicago; Value: United States
Key: 1; Value: Stockholm
Key: Stockholm; Value: Sweden
Key: 2; Value: Montreal
Key: Montreal; Value: Canada

Walking with array_walk()

```
function  print_value_length($array_value, $array_key_ignored)
{
$the_length = strlen($array_value);
print("The length of $array_value is $the_length<BR>");
}


array_walk($major_city_info, 'print_value_length');
```

The length of Chicago is 7
The length of United States is 13
The length of Stockholm is 9
The length of Sweden is 6
The length of Montreal is 8
The length of Canada is 6

```
<?php
function myfunction($value,$key)
{
echo "The key $key has the value
$value<br>";
}
$a=array("a"=>"red","b"=>"green","c"=>"blue");
array_walk($a,"myfunction");
?>
```

Functions for Iterating over arrays

Function	Arguments	Side Effect	Return Value
`current()`	One array argument	None.	The value from the key/value pair currently pointed to by the internal "current" pointer (or false if no such value).
`next()`	One array argument	Advances the pointer by one. If already at the last element, it will move the pointer "past the end," and subsequent calls to `current()` will return false.	The value pointed to after the pointer has been advanced (or false if no such value).
`prev()`	One array argument	Moves the pointer back by one. If already at the first element, will move the pointer "before the beginning."	The value pointed to after the pointer has been moved back (or false if no such value).
`reset()`	One array argument	Moves the pointer back to point to the first key/value pair, or "before the beginning" if the array is empty.	The first value stored in the array, or false for an empty array.
`end()`	One array argument	Moves the pointer ahead to the last key/value pair.	The last value that is currently in the list of key/value pairs.

Function	Arguments	Side Effect	Return Value
`array_walk()`	1) An array argument, 2) the name of a two- (or three-) argument function to call on each key/value, and 3) an optional third argument.	This function invokes the function named by its second argument on each key/value pair. Side effects depend on the side effects of the passed function.	(Returns 1.)
`pos()`	One array argument	None. (This function is an alias for `current()`.)	The value of the key/value pair that is currently pointed to.
`each()`	One array argument	Moves the pointer ahead to the next key/value pair.	An array that packages the keys and values of the key/value pair that was current before the pointer was moved (or false if no such pair). The returned array stores the key and value under its own keys 0 and 1, respectively, and also under its own keys `'key'` and `'value'`.

Arithmetic operators

Arithmetic Operators		
Operator	**Behavior**	**Examples**
+	Sum of its two arguments.	4 + 9.5 evaluates to 13.5
-	If there are two arguments, the right-hand argument is subtracted from the left-hand argument. If there is just a right-hand argument, then the negative of that argument is returned.	50 - 75 evaluates to -25 - 3.9 evaluates to -3.9
*	Product of its two arguments.	3.14 * 2 evaluates to 6.28
/	Floating-point division of the left-hand argument by the right-hand argument.	5 / 2 evaluates to 2.5
%	Integer remainder from division of left-hand argument by the absolute value of the right-hand argument. (See discussion in the following section.)	101 % 50 evaluates to 1 999 % 3 evaluates to 0 43 % 94 evaluates to 43 -12 % 10 evaluates to -2 -12 % -10 evaluates to -2

Incrementing operators

$result = $count++; is exactly equivalent to:

```
$result = $count;
$count = $count + 1;
```

while $result = ++$count; is equivalent to:

```
$count = $count + 1;
$result = $count;
```

Assignment operators

+=, -=, *=, /=, and %=

Comparison operators

<, >,<=,>=, ==, !=, ===, !==

Precedence and parentheses

Arithmetic operators have higher precedence than comparison operators.

Comparison operators have higher precedence than assignment operators.

The *, /, and % arithmetic operators have the same precedence.

The + and – arithmetic operators have the same precedence.

The *, /, and % operators have higher precedence than + and –.

When arithmetic operators are of the same precedence, associativity is from left to right

```
<html><body><?php
$x = 10;
echo $x++;    // Returns $x, then increments $x by one
echo"<br>";
$x = 10;
echo ++$x;   // Increments $x by one, then returns $x
echo"<br>";
$count=10;
$result = $count++;
echo  $result."    ".$count;
echo"<br>";
$count=10;
$result = ++$count;
echo  $result."    ".$count;
?>
</body></html>
```

Simple Math Functions

Function	Behavior
`floor()`	Takes a single argument (typically a double) and returns the largest integer that is less than or equal to that argument.
`ceil()`	Short for ceiling — takes a single argument (typically a double) and returns the smallest integer that is greater than or equal to that argument.
`round()`	Takes a single argument (typically a double) and returns the nearest integer. If the fractional part is exactly 0.5, it returns the nearest even number.
`abs()`	Short for absolute value — if the single numerical argument is negative, the corresponding positive number is returned; if the argument is positive, the argument itself is returned.
`min()`	Takes any number of numerical arguments (but at least one) and returns the smallest of the arguments.
`max()`	Takes any number of numerical arguments (but at least one) and returns the largest of the arguments.

min(3, abs(-3), max(round(2.7), ceil(2.3), floor(3.9)))

Randomness

Random Number Functions

Function	Behavior
`srand()`	Takes a single positive integer argument and seeds the random number generator with it.
`rand()`	If called with no arguments, returns a "random" number between 0 and `RAND_MAX` (which can be retrieved with the function `getrandmax()`). The function can also be called with two integer arguments to restrict the range of the number returned — the first argument is the minimum and the second is the maximum (inclusive).
`getrandmax()`	Returns the largest number that may be returned by `rand()`. This number is limited to 32768 on Windows platforms.
`mt_srand()`	Like `srand()`, except that it seeds the "better" random number generator.
`mt_rand()`	Like `rand()`, except that it uses the "better" random number generator.
`mt_getrandmax()`	Returns the largest number that may be returned by `mt_rand()`.

Randomness

rand();
 	
or
 	
rand(*min,max*);
	echo(rand() . "
	");
	echo(rand(10,100));
	echo(mt_rand() . "
	");
	echo(mt_rand(10,100));
	echo(getrandmax()); 32767
	echo(mk_getrandmax()); 2147483647
	
	srand(mktime());
	echo(rand());

Randomness

```
function random_char($string)
{
$length = strlen($string);
$position = mt_rand(0, $length - 1);
return($string[$position]);
}
function random_string ($charset_string, $length)
{
$return_string = “”; // the empty string
for ($x = 0; $x < $length; $x++)
$return_string .= random_char($charset_string);
return($return_string);
}

$charset = “abcdefghijklmnopqrstuvwxyz”;
$random_string = random_string($charset, 8);
print(“random_string: $random_string<BR>”);
```

with the result:
random_string: eisexkio

Randomness

```
<html>
<body>
<?php
$charset = 'abcdefghijklmnopqrstuvwxyz';
$result = ''; // the empty string
$length = strlen($charset);
for ($x = 0; $x < 8; $x++)
{
$position = mt_rand(0, $length - 1);
$result .= $charset[$position];
}
echo($result);
?>
</body>
</html>
```

The heredoc syntax

```
echo<<<ENDOFFORM
<FORM METHOD=POST ACTION=”{$_ENV[‘PHP_SELF’]}“>
<INPUT TYPE=TEXT NAME=FIRSTNAME VALUE=$firstname>
<INPUT TYPE=SUBMIT NAME=SUBMIT VALUE=SUBMIT>
</FORM>
ENDOFFORM;
```

```html
<html>
<body>
<form method="post" action="test.php">
Enter number<input type="text"
name="num"/>
<br/>
<input type="submit" value="submit">
</form>
</body>
</html
```

```php
<?php
$n=$_POST['num'];
$sum=0;
for($i=1;$i<=$n;$i++)
$sum=$sum+$i;
echo "Sum of first $n natural
numbers is $sum";
?>
```

```
<?php
if (isset($_POST['SUBMIT']))
{
$n=$_POST['num'];
$sum=0;
for($i=1;$i<=$n;$i++)
$sum=$sum+$i;
echo "Sum of first $n natural numbers is $sum";
}
$message= <<<xyz
<FORM METHOD=POST ACTION=heredoc2.php>
<INPUT TYPE=TEXT NAME=num>
<INPUT TYPE=SUBMIT NAME=SUBMIT VALUE=SUBMIT>
</FORM>
xyz;
?>
<HTML><BODY>
<?php echo $message; ?>
</BODY></HTML>
```

String Functions

- <?php
echo strlen("Hello world!");
?>
- $twister = “Peter Piper picked a peck of pickled peppers”;

print(“Location of ‘p’ is “ . **strpos**($twister, ‘p’) .’
’);

print(“Location of ‘q’ is “ . strpos($twister, ‘q’) .’
’);

O/p Location of ‘p’ is 8 Location of ‘q’ is

print(“Location of ‘p’ is “ . **strrpos**($twister, ‘p’) .’
’);

O/p Location of ‘p’ is 40

strcmp() returns a negative number if the first string is less than the second and a positive number if the second string is less. It returns 0 if they are identical.

- **strcasecmp**(“hey!”, “HEY!”) return 0 //case insensitive comparison

String Functions

```
<html>
<body>
<?php
$x="Hello World";
echo "Position of World in $x is". strpos($x,"World")  ."<br>";
echo strpos($x,"o")  ."<br>";
echo strrpos($x,"o")  ."<br>";
echo strcmp("Hello world!","HELLO WORLD!")  ."<br>";
echo strcasecmp("Hello world!","HELLO WORLD!")  ."<br>";
echo strcasecmp("Hello world!","HELLO1 WORLD!") ."<br>";
?>
</body>
</html>
```

Position of World in Hello World is6
4
7
8192
0
-17

String Functions

- The **strstr()** function takes a string to search in and a string to look for (in that order).
- If it succeeds,it returns the portion of the string that starts with (and includes) the first instance of the string it is looking for
- If the string is not found, a false value is returned

```
$string_to_search = “showsuponceshowsuptwice”;
$string_to_find = “up”;
print(“Result of looking for $string_to_find” .
strstr($string_to_search, $string_to_find) . “<br>”);
$string_to_find = “down”;
print(“Result of looking for $string_to_find” .
strstr($string_to_search, $string_to_find));
```

Result of looking for up: uponceshowsuptwice
Result of looking for down:

String Functions

echo(**substr**(“Take what you need, and leave the rest behind”,23));

o/p leave the rest behind

echo(**substr**(“Take what you need, and leave the rest behind”,5, 13));

o/p what you need

Simple Inspection, Comparison, and Searching Functions

Function	Behavior
`strlen()`	Takes a single string argument and returns its length as an integer.
`strpos()`	Takes two string arguments: a string to search, and the string being searched for. Returns the (0-based) position of the beginning of the first instance of the string if found and a false value otherwise. It also takes a third optional integer argument, specifying the position at which the search should begin.
`strrpos()`	Like `strpos()`, except that it searches backward from the end of the string, rather than forward from the beginning. The search string must only be one character long, and there is no optional position argument.
`strcmp()`	Takes two strings as arguments and returns 0 if the strings are exactly equivalent. If `strcmp()` encounters a difference, it returns a negative number if the first different byte is a smaller ASCII value in the first string, and a positive number if the smaller byte is found in the second string.
`strcasecmp()`	Identical to `strcmp()`, except that lowercase and uppercase versions of the same letter compare as equal.
`strstr()`	Searches its first string argument to see if its second string argument is contained in it. Returns the substring of the first string that starts with the first instance of the second argument, if any is found — otherwise, it returns false.
`strchr()`	Identical to `strstr()`.
`stristr()`	Identical to `strstr()` except that the comparison is case independent.

String Functions

Substr() function takes a string (that the substring will be selected from), an integer (the position at which the desired substring starts), and an optional third integer argument that is the length of the desired substring

If the start position is negative, it means that the starting character is determined by counting backward from the end of the string

Negative-length argument means that the final character is determined by counting backward from the end rather than forward from the start position

```
$alphabet_test = “abcdefghijklmnop”;
print(“3: “ . substr($alphabet_test, 3) . “<BR>”);
print(“-3: “ . substr($alphabet_test, -3) . “<BR>”);
print(“3, 5: “ . substr($alphabet_test, 3, 5) . “<BR>”);
print(“3, -5: “ . substr($alphabet_test, 3, -5) . “<BR>”);
print(“-3, -5: “ . substr($alphabet_test, -3, -5) . “<BR>”);
print(“-3, 5: “ . substr($alphabet_test, -3, 5) . “<BR>”);
```

```
3: defghijklmnop
-3: nop
3, 5: defgh
3, -5: defghijk
-3, -5:
-3, 5: nop
```

String Functions

chop(), ltrim(), and trim() trim whitespace off the end, the beginning, and the beginning and end

```
$original = “ More than meets the eye “;
$chopped = chop($original);
$ltrimmed = ltrim($original);
$trimmed = trim($original);
print(“The original is ‘$original’<BR>”);
print(“Its length is “ . strlen($original) . “<BR>”);
print(“The chopped version is ‘$chopped’<BR>”);
print(“Its length is “ . strlen($chopped) . “<BR>”);
print(“The ltrimmed version is ‘$ltrimmed’<BR>”);
print(“Its length is “ . strlen($ltrimmed) . “<BR>”);
print(“The trimmed version is ‘$ltrimmed’<BR>”);
print(“Its length is “ . strlen($trimmed) . “<BR>”);
```

```
The original is ‘   More than meets the eye   ‘
Its length is 28
The chopped version is ‘   More than meets the eye’
Its length is 25
The ltrimmed version is ‘More than meets the eye   ‘
Its length is 26
The trimmed version is ‘More than meets the eye’
Its length is 23
```

String Functions

```
$first_edition =“Burma is similar to Rhodesia in at least one way.”;
$second_edition = str_replace(“Rhodesia”, “Zimbabwe”,$first_edition);
$third_edition = str_replace(“Burma”, “Myanmar”,$second_edition);
print($third_edition);
```

Myanmar is similar to Zimbabwe in at least one way.

```
print(substr_replace(“ABCDEFG”, “-“, 2, 3));
```

AB-FG

```
print(str_repeat(“cheers “, 3));
```

cheerscheerscheers

String Functions

The strtolower() function returns an all-lowercase string

The strtoupper() function returns an all-uppercase string

The ucfirst() function capitalizes only the first letter of a string:

The ucwords() function capitalizes the first letter of each word in a string

```
<?php
$original = “They DON’T KnoW they’re SHOUTING”;
$lower = strtolower($original);
echo $lower;
?>
```

Printing and output

The % character signals the beginning of a *conversion specification, which indicates how to* print one of the arguments that follow the format string

After the %, there are six elements that make up the conversion specification, some of which are optional: padding, alignment, minimum width, precision, and type

■■An optional *sign character used for numbers to indicate whether the number will be* negative (-).

■■The single (optional) *padding character is either a 0 or a space ( ). This character is used to* fill any space that would otherwise be unused but that you have insisted (with the minimum width argument) be filled with something. If this padding character is not given, the default is to pad with spaces.

■■The optional *alignment character (-) indicates whether the printed value should be left- or* right-justified. If present, the value will be left-justified; if absent, it will be right-justified.

Printing and output

■■An optional *minimum width number that indicates how many spaces this value should take* up, at a minimum. (If more spaces are needed to print the value, it will overflow beyond its bounds.)

■■An optional precision specifier is written as a dot (.) followed by a number. It indicates how many decimal points of *precision a double should print with. (This has no effect on* printing things other than doubles.)

■■A single character indicating how the *type of the value should be interpreted. The f character* indicates printing as a double, the s character indicates printing as a string, and then the rest of the possible characters (b, c, d, o, x, X) mean that the value should be interpreted as an integer and printed in various formats. Those formats are b for binary, c for printing the character with the corresponding ASCII values, o for octal, x for hexadecimal (with lowercase letters) and X for hexadecimal with uppercase letters.

Printing and output

```
<?php
$value = 3.14159;
printf("%f,%10f,%-010f,%2.2f\n",$value, $value, $value, $value);
?>
```

3.141590, 3.141590,3.141590000000000, 3.14

Examining Regular Expressions

Tokenizing and Parsing Functions

- The process of breaking up a long string into *words is called tokenizing*
  - The strtok() function takes two arguments: the string to be broken up into tokens and a string containing all the *delimiters (characters that count* as boundaries between tokens)
  - On the first call, both arguments are used,and the string value returned is the first token
  - To retrieve subsequent tokens, make the same call, but omit the source string argument
  - $token = strtok("open-source HTML-embedded server-side Web scripting"," ");

- explode() function stores the tokens all at once in an array
  - $explode_result = explode("AND", "one AND a two AND a three");

Tokenizing and Parsing Functions

```
$token = strtok(
“open-source HTML-embedded server-side
Web scripting”,“ “);
while($token){
print($token . “<BR>”);
$token = strtok(“ “);
}
```

```
$token = strtok(
“open-source HTML-embedded server-side
Web-scripting”,“ -“);
while($token){
print($token . “<BR>”);
$token = strtok(“ -“);
}
```

open-source
HTML-embedded
server-side
Web
scripting

open
source
HTML
embedded
server
side
Web
scripting

Regex in PHP

- Regular expressions are patterns for string matching, with special wildcards that can match entire portions of the target string
- *POSIX* (extended) regex
  - Regex pattern-matching machinery used in Unix command-line shells
- Perl-compatible regex
  - Regular expressions in Perl

Regex in PHP

- Characters that are not *special are matched literally. The letter a in a pattern, for example,* matches the same letter in a target string
- The special character ^ matches the beginning of a string only, and the special character $ matches the end of a string only
- The special character . matches any character
- The special character * matches zero or more instances of the previous regular expression,and + matches one or more instances of the previous expression
- A set of characters enclosed in square brackets matches any of those characters — the pattern [ab] matches either a or b. You can also specify a range of characters in brackets by using a hyphen — the pattern [a-c] matches a, b, or c
- Special characters that are escaped with a backslash (\) lose their special meaning and are matched literally

Regex in PHP

```
function simple_dot_com ($url)
{
return(ereg(‘^www\\.[a-z]+\\.com$’, $url));
}
$urls_to_test =array(‘www.ibm.com’, ‘www.java.sun.com’,‘www.zend.com’,
‘java.sun.com’,‘www.java.sun.com’, ‘www.php.net’,‘www.IBM.com’);
while($test = array_pop($urls_to_test)){
if (simple_dot_com($test))
print(“\“$test\“ is a simple dot-com<BR>”);
else
print(“\“$test\“ is NOT a simple dot-com<BR>”);
}
```

“www.IBM.com” is NOT a simple dot-com
“www.php.net” is NOT a simple dot-com
“www.java.sun.com” is NOT a simple dot-com
“java.sun.com” is NOT a simple dot-com
“www.zend.com” is a simple dot-com
“www.java.sun.com” is NOT a simple dot-com
“www.ibm.com” is a simple dot-com

POSIX Regular Expression Functions

Function	Behavior
`ereg()`	Takes two string arguments and an optional third-array argument. The first string is the POSIX-style regular expression pattern, and the second string is the target string that is being matched. The function returns `TRUE` if the match was successful and `FALSE` otherwise. In addition, if an array argument is supplied and portions of the pattern are enclosed in parentheses, the parts of the target string that match successive parenthesized portions will be copied into successive elements of the array.
`ereg_replace()`	Takes three arguments: a POSIX regular expression pattern, a string to do replacement with, and a string to replace into. The function scans the third argument for portions that match the pattern and replaces them with the second argument. The modified string is returned. If there are parenthesized portions of the pattern (as with `ereg()`), the replacement string may contain special substrings of the form `\\digit` (that is, two backslashes followed by a single-digit number), which will themselves be replaced with the corresponding piece of the target string.
`eregi()`	Identical to `ereg()`, except that letters in regular expressions are matched in a case-independent way.
`eregi_replace()`	Identical to `ereg_replace()`, except that letters in regular expressions are matched in a case-independent way.
`split()`	Takes a pattern, a target string, and an optional limit on the number of portions to split the string into. Returns an array of strings created by splitting the target string into chunks delimited by substrings that match the regular expression. (Note that this is analogous to the `explode()` function, except that it splits on regular expressions rather than literal strings.)
`spliti()`	Case-independent version of `split()`.

Regex in PHP

```
<?php
$copy_date = "Copyright 1999";
$copy_date = ereg_replace("([0-9]+)", "2000", $copy_date);
print $copy_date;
?>
```

```
Copyright
2000
```

```
<?php
$ip = "123.456.789.000"; // some IP address
$iparr = split ("\.", $ip);
print "$iparr[0] <br />";
print "$iparr[1] <br />" ;
print "$iparr[2] <br />" ;
print "$iparr[3] <br />" ;
?>
```

```
123
456
789
000
```

Regex in PHP

```
<?php
$str = 'one,two,three,four';
// zero limit
print_r(explode(',',$str,0));
print "<br>";
// positive limit
print_r(explode(',',$str,2));
print "<br>";
// negative limit
print_r(explode(',',$str,-1));
?>
```

```
Array ( [0] => one,two,three,four )
Array ( [0] => one [1] => two,three,four )
Array ( [0] => one [1] => two [2] => three )
```

Common Perl-Compatible Pattern Constructs

Construct	Interpretation
Simple literal character matches	If the character involved is not special, Perl will match characters in sequence. The example pattern `/abc/` matches any string that has the substring `'abc'` in it.
Character class matches: [<*list of characters*>]	Will match a single instance of any of the characters between the brackets. For example, `/[xyz]/` matches a single character, as long as that character is either x, y, or z. A sequence of characters (in ASCII order) is indicated by a hyphen, so that a class matching all digits is `[0-9]`.
Predefined character class abbreviations	The patterns `\d` will match a single digit (from the character class `[0-9]`), and the pattern `\s` matches any whitespace character.
Multiplier patterns	Any pattern followed by * means: "Match this pattern 0 or more times." Any pattern followed by ? means: "Match this pattern exactly once." Any pattern followed by + means: "Match this pattern 1 or more times."

Construct	Interpretation
Anchoring characters	The caret character ^ at the beginning of a pattern means that the pattern must start at the beginning of the string; the $ character at the end of a pattern means that the pattern must end at the end of the string. The caret character at the beginning of a character class `[^abc]` means that the set is the complement of the characters listed (that is, any character that is not in the list).
Escape character '\'	Any character that has a special meaning to regex can be treated as a simple matching character by preceding it with a backslash. The special characters that might need this treatment are: `. \ + * ? [ ] ^ $ ( ) { } = ! < > | :`
Parentheses	A parenthesis grouping around a portion of any pattern means: "Add the substring that matches this pattern to the list of substring matches."

```
/phone number\s+(\d\d\d\d\d\d\d)/
```

Perl-Compatible Regular Expression Functions

Function	Behavior
`preg_match()`	Takes a regex pattern as first argument, a string to match against as second argument, and an optional array variable for returned matches. Returns `0` if no matches are found, and `1` if a match is found. If a match is successful, the array variable contains the entire matching substring as its first element, and subsequent elements contain portions matching parenthesized portions of the pattern. As of PHP 4.3.0, an optional flag of `PREG_OFFSET_CAPTURE` is also available. This flag causes preg match to return into the specified array a two-element array for each match, consisting of the match itself and the offset where the match occurs.
`preg_match_all()`	Like `preg_match()`, except that it makes all possible successive matches of the pattern in the string, rather than just the first. The return value is the number of matches successfully made. The array of matches is not optional (If you want a true/false answer, use `preg_match()`). The structure of the array returned depends on the optional fourth argument (either the constant `PREG_PATTERN_ORDER`, or `PREG_SET_ORDER`, defaulting to the former). (See further discussion following the table.) `PREG_OFFSET_CAPTURE` is also available with this function.
`preg_split()`	Takes a pattern as first argument and a string to match as second argument. Returns an array containing the string divided into substrings, split along boundary strings matching the pattern. (Analogous to the POSIX-style function `split()`.) An optional third argument (`limit`) controls how many elements to split before returning the list; `-1` means no limit. An optional flag in the fourth position can be `PREG_SPLIT_NO_EMPTY` causing the function to return only nonempty pieces, `PREG_SPLIT_DELIM_CAPTURE` causing any parenthesized expression in the delimiter pattern to be returned, or `PREG_SPLIT_OFFSET_CAPTURE`, which does the same as `PREG_OFFSET_CAPTURE`.
`preg_replace()`	Takes a pattern, a replacement string, and a string to modify. Returns the result of replacing every matching portion of the modifiable string with the replacement string. An optional limit argument determines how many replacements will occur (as in `preg_split()`).

`preg_replace_callback()`	Like `preg_replace()`, except that the second argument is the name of a callback function, rather than a replacement string. This function should return the string that is to be used as a replacement.
`preg_grep()`	Takes a pattern and an array and returns an array of the elements of the input array that matched the pattern. Surviving values of the new array have the same keys as in the input array.
`preg_quote()`	A special-purpose function for inserting escape characters into strings that are intended for use as regex patterns. The only required argument is a string to escape; the return value is that string with every special regex character preceded by a backslash.

Regex in PHP

```
<?php
$line = "Vi is the greatest word processor ever created!";
// perform a case-Insensitive search for the word "Vi“
 if (preg_match("/\bVi\b/i", $line, $match))
print "Match found!";
endif;
?>
```

Match found!

```
?php
$foods = array("pasta", "steak", "fish", "potatoes");
// find elements beginning with "p", followed by one or more
letters.
$p_foods = preg_grep("/p(\w+)/", $foods);
print "Found food is " . $p_foods[0];
print "Found food is " . $p_foods[1];
print "Found food is " . $p_foods[3];

?>
```

Found food is pasta
Found food is
Found food is potatoes

Regex in PHP

<A HREF=”http://mysite.com/mypage.php”>My cool page on my cool site</A>

/<A\sHREF=”[^”]+”>/	**left angle bracket, followed by A, followed by a space, followed by the string HREF=, followed by a double-quotation mark, followed by any number of characters that are not quotation marks, followed by a closing quotation mark, followed by a right angle bracket**
/<A\s+HREF=”[^”]+”\s*>/	Allowing more space + -one or more *-zero or more
/<A\s+HREF=”[^\“]+”\s*>[^>]*<\/A>/	with text and close tag
/<A\s+HREF=”[^\“]+”\s*>[^>]*<\/A>/i	case-independent

PHP Gotchas

Installation-Related Problems

- Symptom: Text of file displayed in browser window
  - PHP engine is clearly not being invoked. Check that you are accessing the site through the web server and not via the filesystem.
  - Use this: http://localhost/mysite/mypage.php

- Symptom: PHP blocks showing up as text under HTTP or browser prompts you to save file
  - Haven't specified all the fileextensions you want to be served by the web server and parsed with the PHP interpreter
  - php.ini file is in the wrong place or has a bad configuration directive
- Symptom: Server or host not found/Page cannot be displayed
  - DNS (Domain Name Service) or Web-server configuration issue

PHP Gotchas

Rendering Problems-PHP does not report an error per se, but what you see is not what you thought you would get

Symptom: Totally blank page

- Caused by a fatal error in the PHP code from which the PHP interpreter cannot recover
- <?php
  die(print “hello”);
- Error reporting should probably be turned off. Check your php.ini file’s display_errors setting

Symptom: PHP code showing up in Web browser

- omitted a PHP start tag

```
<HTML><HEAD></HEAD><BODY>
<?php include(“secret.php”);
secret_function(); ?>
</BODY></HTML>
```

```
function secret_function ()
{
echo “Open sesame!”;
}
```

PHP Gotchas

Failures to Load Page

- Symptom: Page cannot be found
  - Misspelling the filename or path
- Symptom: Failed opening [file] for inclusion
  - Included-file version of Page cannot be found

Parse Errors

- Symptom: Parse error message
  - The missing semicolon
  - No dollar signs

What we have here is	What we have here is
<?php Problem = “a big ball”; echo $Problem; ?>.	<?php $Problem = “a big ball ”; print(“Problem”); ?>.

PHP Gotchas

Parse Errors

- Mode issues
  - A parse error will result if you fail to close off a PHP block properly
  - Eg: Missed PHP double-quote and the HTML closing bracket

```
<FORM>
<INPUT TYPE=”TEXT” SIZE=15 NAME=”FirstName”
VALUE=”<?php print(“$FirstName”); ?>”>
<INPUT TYPE=”TEXT” SIZE=10 NAME=”PhoneNumber”
VALUE=”<?php print($PhoneNumber”); ?>”
<INPUT TYPE=”SUBMIT” NAME=”Submit”>
</FORM>
```

- Unescaped quotation marks

```
<?php
print(“She said, /“What we have here is “);
$Problem = “a difference of opinion\“”;
print(“$Problem”); ?>
```

PHP Gotchas

Parse Errors

- Unterminated strings
  - print(“I am a guilty print statement!);
- Other parse error causes
  - unclosed parentheses,unclosed brackets, operators without arguments, control structure tests without parentheses

Missing Includes

- Symptom: Include warning
  - If a file is included and PHP can’t locate the file, execution of the script will continue with a PHP warning
  - If a file is required and PHP can’t locate that file, execution will stop with an error

```php
<?php
echo '<a href="/default.asp">Home</a> -
<a href="/html/default.asp">HTML
Tutorial</a> -
<a href="/css/default.asp">CSS Tutorial</a> -
<a href="/js/default.asp">JavaScript
Tutorial</a> -
<a href="default.asp">PHP Tutorial</a>';
?>
```

```html
<html>
<body>

<div class="menu">
<?php include 'menu.php';?>
</div>

<h1>Welcome to my home page!</h1>
<p>Some text.</p>
<p>Some more text.</p>

</body>
</html>
```

PHP Gotchas

Unbound Variables

- Symptom: Variable not showing up in print string
  - print("like $this")
- Symptom: Numerical variable unexpectedly zero
- Causes of unbound variables
  - Unbound variables are interpreted as 0 in a numerical context, "" in a string context, FALSE in a Boolean context, and as an empty array in an array context

```
<?php
$one_string = "one";
$three_string = "three";
$one = 1;
$two = 2;
print("This math is as easy as $one_string, $two_string,$three_string!<BR>");
print("$one_string is equal to $one<BR>");
print("$two_string is equal to $two<BR>");
print("$three_string is equal to $three<BR>");
print("$one_string divided by $two_string is " .($one / $two) . "<BR>");
print("$one_string divided by $three_string is " .($one / $three) . "<BR>");
?>
```

PHP Gotchas

Unbound Variables

- Case problems
  - Variables in PHP are case sensitive, so the same name with different capitalization results in a different variable
- Scoping problems

```
$name = “Steve Suehring”;
$rank = “Intarweb Programmer”;
$serial_no = “4”;
function Answer($name)
{
global $rank;
print(“Name: $name; Rank: $rank;
serial no: $serial_no<BR>”);
}
Answer($name);
```

PHP Gotchas

Function Problems

Symptom: Call to undefined function my_function()

- Misspelled the name of a function (built-in or user-defined) or omitted the function definition
- If you use include/require files to load user-defined functions,make sure that you are loading the appropriate files

Symptom: Call to undefined function ()

- Accidentally put a $ in front of a sensible call to my_function()

Symptom: Call to undefined function array()

- $my_amendments = array();

  $my_amendments(5) = “the fifth”;

Symptom: Cannot redeclare my_function()

Symptom: Wrong parameter count

PHP Gotchas

Math Problems

Symptom: Division-by-zero warning

```
$numerator = 5;
$ratio = $numerator / $denominator;
```

where $denominator is unbound

```
$numerator = 5;
if (isset($denominator) && $denominator != 0)
$ratio = $numerator / $denominator;
else
print("I'm sorry, Dave, I cannot do that<BR>");
```

Symptom: Unexpected arithmetic result

Symptom: NaN (or NAN)

Arccosine is defined only when applied to numbers between –1.0 and 1.0

```
$value = acos(45);
print("acos result is $value<BR>");
```

PHP Gotchas

Timeouts

- Reason for a timeout is an infinite loop

```
//compute the factorial of 10
$Fact = 1;
for ($Index = 1; $Index <= 10; $index++)
$Fact *= $Index;
```

- The lowercase $index that is incremented has nothing to do with the $Index that is being tested

Revision

```
<?php
session_start();
?>
<HTML><HEAD><TITLE>Greetings</TITLE></HEAD><BODY>
<?php
if (!IsSet($_SESSION['visit_count'])) {
echo "Hello, you must have just arrived.
Welcome!<BR>";
$_SESSION['visit_count'] = 1;
}
else {
$visit_count = $_SESSION['visit_count'] + 1;
echo "Back again are ya? That makes $visit_count times now "."<BR>";
$_SESSION['visit_count'] = $visit_count;
}
$self_url = $_SERVER['PHP_SELF'];
$session_id = SID;
if (IsSet($session_id) &&
$session_id) {
$href = "$self_url?$session_id";
}
else {
$href = $self_url;
}
echo "<BR><A HREF=\"$href\">Visit us again</A> sometime";
?>
</BODY></HTML>
```

Sending HTTP Headers

Comparing Things That Are Not Integers

```
$string_1 = “00008”;
$string_2 = “007”;
$string_3 = “00008-OK”;
if ($string_2 < $string_1)
print(“$string_2 is less than $string_1<BR>”);
if ($string_3 < $string_2)
print(“$string_3 is less than $string_2<BR>”);
if ($string_1 < $string_3)
print(“$string_1 is less than $string_3<BR>”);
```

gives this output (with comments added):

007 is less than 00008 // numerical comparison

00008-OK is less than 007 // string comparison

00008 is less than 00008-OK // string comp. - contradiction!

Basic Use of Exceptions

```
<?php
//create function with an exception
function checkNum($number) {
  if($number>1) {
    throw new Exception("Value must be 1 or below");
  }
  return true;
}
//trigger exception
checkNum(2);
?>
```

Fatal error: Uncaught exception 'Exception'
with message 'Value must be 1 or below' in C:\webfolder\test.php:6
Stack trace: #0 C:\webfolder\test.php(12):
checkNum(28) #1 {main} thrown in C:\webfolder\test.php on line 6

Try, throw and catch

Try - A function using an exception should be in a "try" block. If the exception does not trigger, the code will continue as normal. However if the exception triggers, an exception is "thrown"

Throw - This is how you trigger an exception. Each "throw" must have at least one "catch"

Catch - A "catch" block retrieves an exception and creates an object containing the exception information

Try, throw and catch

```
<?php
//create function with an exception
function checkNum($number) {
  if($number>1) {
    throw new Exception("Value must be 1 or below");
  }
  return true;
}
//trigger exception in a "try" block
try {
  checkNum(2);
  //If the exception is thrown, this text will not be shown
  echo 'If you see this, the number is 1 or below';
}

//catch exception
catch(Exception $e) {
  echo 'Message: ' .$e->getMessage();
}
?>
```

```
<HTML>
<HEAD>
<TITLE>A POST example: retirement savings worksheet</TITLE>
<STYLE TYPE=”text/css”>
<!--
BODY {font-size: 14pt}
.heading {font-size: 18pt; color: red}
-->
</STYLE>
</HEAD>
<?php
// This test, along with the Submit button value in the form// below, will check to see if the form is being rendered for//
the first time (in which case it will display with only the
// default annual gain filled in).
if (!IsSet($_POST[‘Submit’]) || $_POST[‘Submit’] != ‘Calculate’)
{
$_POST[‘CurrentAge’] = “”;
$_POST[‘RetireAge’] = “”;
$_POST[‘Contrib’] = “”;
$Total = 0;
$AnnGain = 7;
} else {
$AnnGain = $_POST[‘AnnGain’];
$Years = $_POST[‘RetireAge’] - $_POST[‘CurrentAge’];
$YearCount = 0;
$Total = $_POST[‘Contrib’];
while ($YearCount<= $Years) {
$Total = round($Total * (1.0 + $AnnGain/100) +
$_POST[‘Contrib’]);
$YearCount = $YearCount + 1;
}
}
?>
<BODY>
<DIV ID=”Div1” class=”heading”>
A retirement-savings calculator</DIV>
<P class=blurb>Fill in all the values (except “Nest Egg”)and see how much money you’ll have for your retirementunder
different scenarios. You can change the values andresubmit the form as many times as you like. You must fillin the two
```

PHP Superglobal Arrays

GET, POST, COOKIE, ENVIRONMENT, and SERVER variables were made global by the register_globals directive in php.ini and were directly accessible by their names by default