MODULE 3

PHP Language Structure: Introduction- Building blocks of PHP-Variables, Data Types -simple PHP program-Converting between Data Types- Operators and Expressions -Flow Control functions - Control statements- Working with Functions- Initialising and Manipulating Arrays-- Objects- String Comparisons-String processing with Regular Expression

Introduction to PHP

- PHP stands for Hypertext Preprocessor.
- PHP is an interpreted language, i.e., there is no need for compilation.
- It is faster than other scripting languages, for example, ASP and JSP.
- PHP is a server-side scripting language, which is used to manage the dynamic content of the website.
- PHP can be embedded into HTML.
- PHP is an object-oriented language , open source scripting language.
- PHP is simple and easy to learn language.

- **Platform Independent:**
- PHP is available for WINDOWS, MAC, LINUX & UNIX operating system. A PHP application developed in one OS can be easily executed in other OS also.
- **Database Support:**
- PHP supports all the leading databases such as MySQL, SQLite, ODBC, etc.
- **Error Reporting -**
- PHP has predefined error reporting constants to generate an error notice or warning at runtime. E.g., E_ERROR, E_WARNING, E_STRICT, E_PARSE.

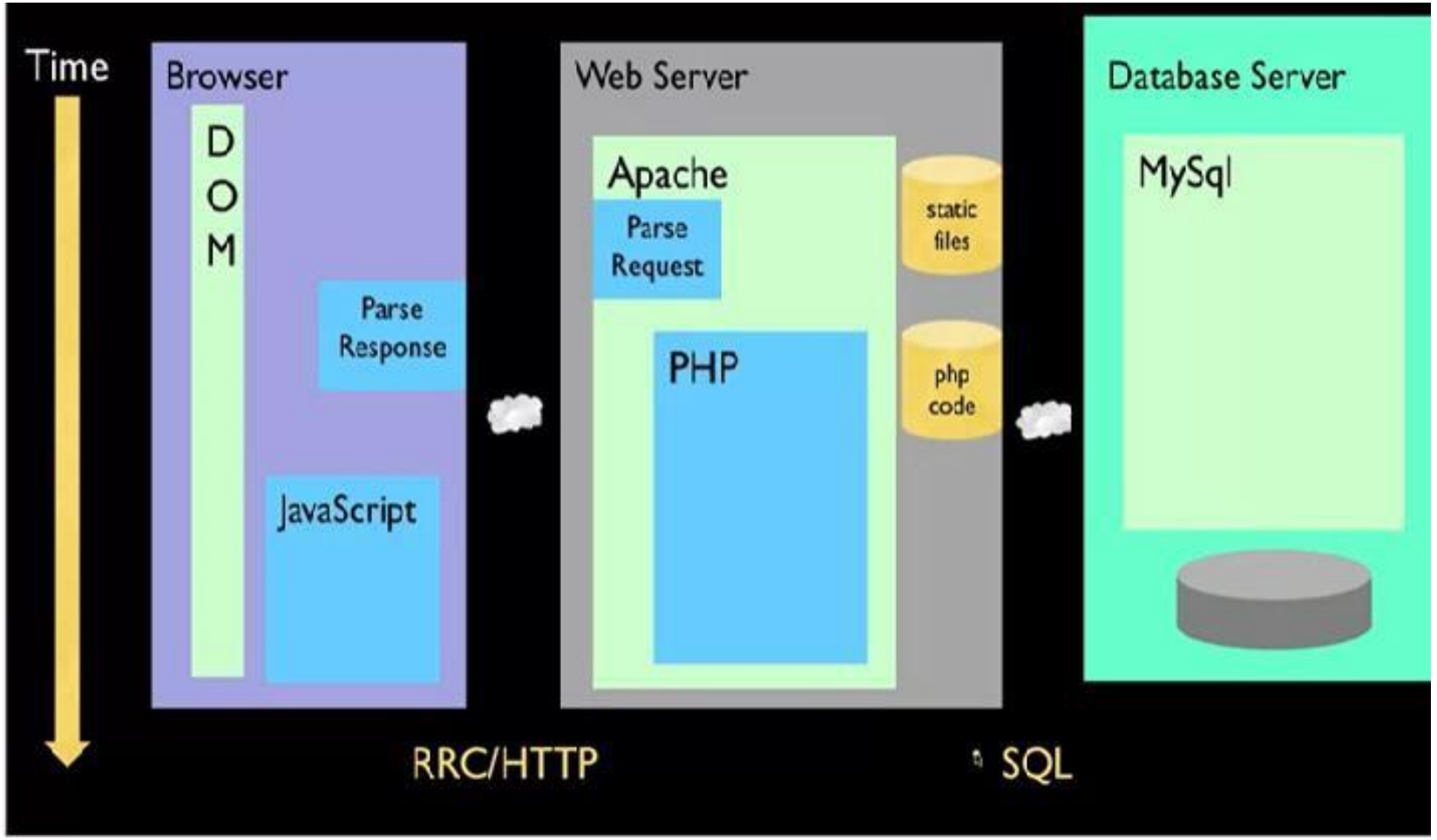
- **Loosely Typed Language:**

PHP allows us to use a variable without declaring its datatype. It will be taken automatically at the time of execution based on the type of data it contains on its value

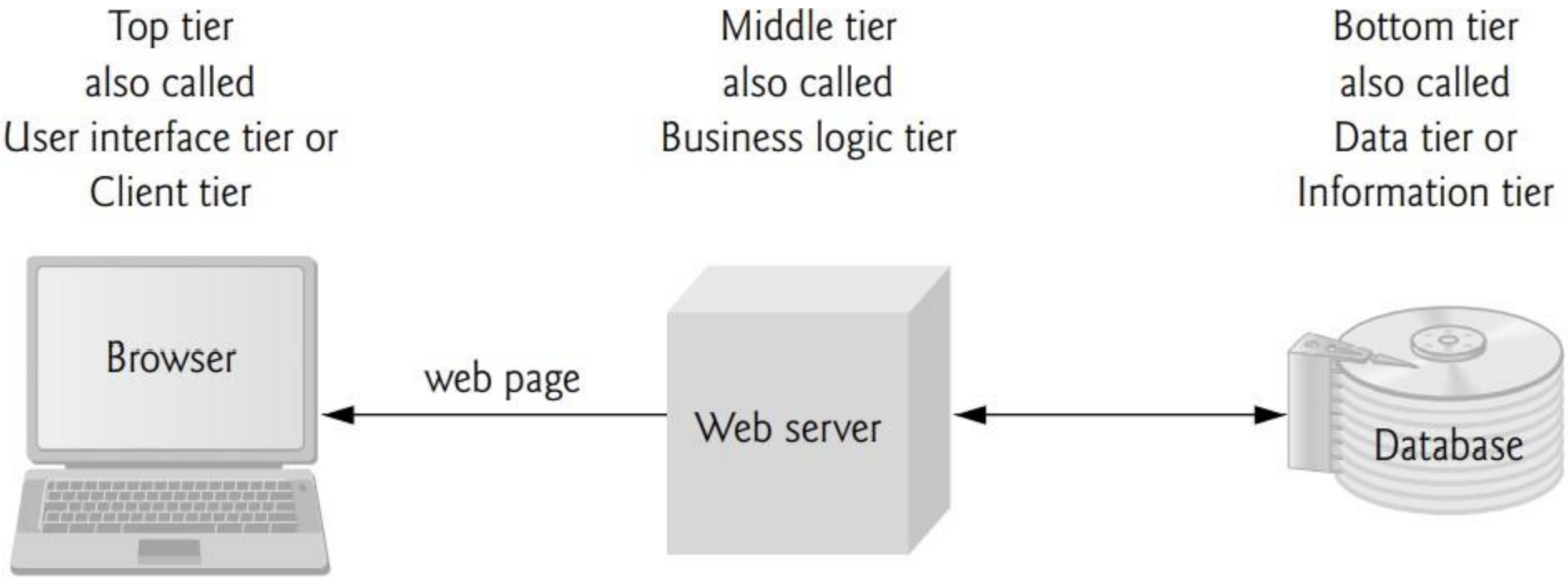
- **Dynamically typed**

PHP checks the type of variables at runtime

- PHP—a language written specifically for handling client requests—a valuable tool.
- PHP code is embedded directly into text-based documents, such as HTML, though these script segments are interpreted by the server before being delivered to the client.
- PHP script file names end with .php.
- PHP code is inserted between the delimiters and can be placed anywhere in HTML markup.


Time
Browser
D
O
M
Parse
Response
JavaScript
Web Server
Apache
Parse
Request
PHP
static
files
php
code
Database Server
MySql
RRC/HTTP
SQL

Three-tier web-based application



Three-tier architecture.

Three-tier web-based application

- The bottom tier (also called the data tier or the information tier) maintains the application's data. This tier typically stores data in a relational database management system (RDBMS)
- The middle tier implements business logic, controller logic and presentation logic to control interactions between the application's clients and its data. The middle tier acts as an intermediary between data in the information tier and the application's clients. The middle-tier controller logic processes client requests (such as requests to view a product catalog) and retrieves data from the database
- The top tier, or client tier, is the application's user interface, which gathers input and displays output. Users interact directly with the application through the user interface, which is typically a web browser or a mobile device

Client-Side Scripting versus Server-Side Scripting

- Client-side scripting with JavaScript can be used to validate user input, to interact with the browser, to enhance web pages, and to add client/server communication between a browser and a web server.
- Client-side scripting does have limitations, such as browser dependency; the browser or scripting host must support the scripting language and capabilities.
- client-side scripts can be viewed by the client using the browser's source-viewing capability
- server-side scripts, often generate custom responses for clients
- Server-side scripting languages have a wider range of programmatic capabilities than their client-side equivalents.
- Server-side scripts also have access to server-side software that extends server functionality—Microsoft web servers use ISAPI (Internet Server Application Program Interface) extensions and Apache HTTP Servers use modules.

Accessing Web Server

- To request documents from web servers, users must know the hostnames on which the web server software resides.
- Users can request documents from local web servers (i.e., ones residing on users' machines) or remote web servers (i.e., ones residing on different machines).
- Local web servers can be accessed through your computer's name or through the name **localhost**
- **localhost**—a hostname that references the local machine and normally translates to the IP address 127.0.0.1 (known as the loopback address).

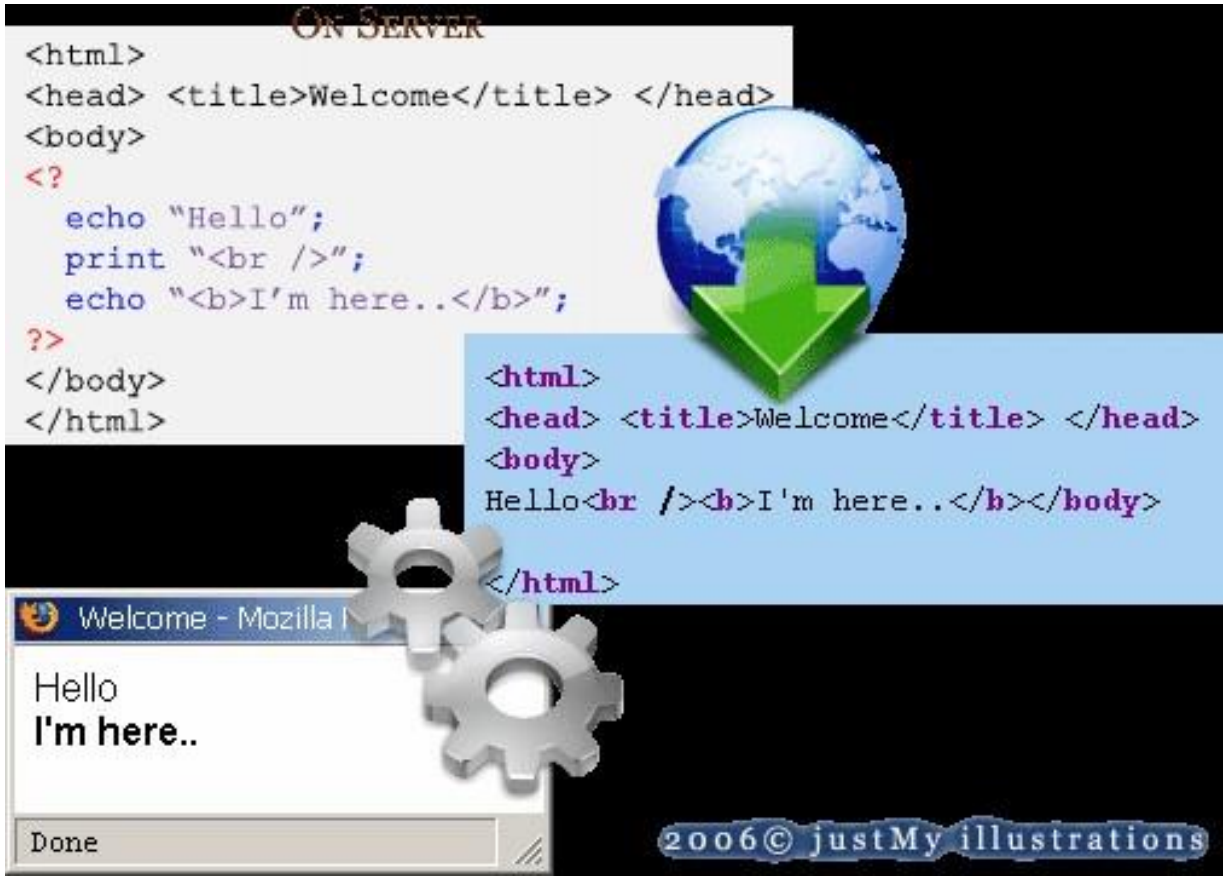
Installing to PHP

- XAMPP integrated installer for Apache, MySQL and PHP is available for Windows, Mac OS X and Linux
- On windows, you can download and install WAMP. With one installation and you get an Apache webserver, database server and php.

http://www.wampserver.com

- On mac, you can download and install MAMP.

http://www.mamp.info/en/index.html



Building blocks of PHP

- Variables
- Data Types
- Operators and Expressions
- Constants

- Variables
- All variable names in PHP begin with dollar signs ($)
- a letter or an underscore followed by any number (including zero) of letters, digits, or underscores.
- PHP variable names are case sensitive.
- Assigned by value
  - `$foo = "Bob"; $bar = $foo;`
- Some are preassigned, server and env vars
  - For example, there are PHP vars, eg. *PHP_SELF, HTTP_GET_VARS*

Building blocks of PHP

- Variables
- Line 5 outputs the value of variable $name by calling function print. The value of $name is printed, not the string "$name".
- When a variable is encountered inside a doublequoted ("") string, PHP interpolates the variable
- . All operations of this type execute on the server before the HTML5 document is sent to the client
- PHP variables are loosely typed—they can contain different types of data (e.g., integers, doubles or strings) at different time

C: > xampp > htdocs > dashboard > smith > index.php

```
1  <?php
2  $name="smith";
3  $txt = "Web programming-using PHP";
4  echo "<center><b>I love $txt!</b></center>";
5  print( "<h1><center>Welcome to PHP, $name!</center></h1>" );
6  ?>
```


I love Web programming-using PHP!

Welcome to PHP, smith!

Building blocks of PHP

- Global Variables
- In PHP global variables must be declared global inside a function if they are going to be used in that function.

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

function test()
{
    echo $a; /* reference to local scope variable */
}

test();
?>
```

- 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. There is no limit to the number of global variables that can be manipulated by a function.

```php
<?php
$a = 1;
$b = 2;

function Sum()
{
    global $a, $b;

    $b = $a + $b;
}

Sum();
echo $b;
?>
```

Building blocks of PHP

- Super Global Variables
- Super global variables are built-in variables that are always available in all scopes.
- Some predefined variables in PHP are "superglobals", which means that they are always accessible, regardless of scope - and you can access them from any function, class or file without having to do anything special.

- The PHP superglobal variables are:
- $GLOBALS
- $_SERVER
- $_REQUEST
- $_POST
- $_GET
- $_FILES
- $_ENV
- $_COOKIE
- $_SESSION

Building blocks of PHP

- second way to access variables from the global scope is to use the special PHP-defined *$GLOBALS* array
- The *$GLOBALS* array is an associative array with the name of the global variable being the key and the contents of that variable being the value of the array element.
- $b is a variable present within the $GLOBALS array, it is also accessible from outside the function!

```php
<?php
$a = 1;
$b = 2;

function Sum()
{
    $GLOBALS['b'] = $GLOBALS['a'] + $GLOBALS['b'];
}

Sum();
echo $b;
?>
```

Superglobal variables

- $GLOBALS- PHP super global variable which is used to access global variables from anywhere in the PHP script
- $_SERVER is a PHP super global variable which holds information about headers, paths, and script locations.
- $_REQUEST is a PHP super global variable which is used to collect data after submitting an HTML form.
- $_POST is a PHP super global variable which is used to collect form data after submitting an HTML form with method="post". $_POST is also widely used to pass variables.
- $_GET is a PHP super global variable which is used to collect form data after submitting an HTML form with method="get".$_GET can also collect data sent in the URL.

Building blocks of PHP

- Variables
- Data Types
- Operators and Expressions
- Constants

Reserved Words

and	else	global	require	virtual
break	elseif	if	return	xor
case	extends	include	static	while
class	false	list	switch	
continue	for	new	this	
default	foreach	not	true	
do	function	or	var	

Building blocks of PHP

Data Types

Type	Description
`int, integer`	Whole numbers (i.e., numbers without a decimal point).
`float, double, real`	Real numbers (i.e., numbers containing a decimal point).
`string`	Text enclosed in either single (`''`) or double (`""`) quotes. [*Note:* Using double quotes allows PHP to recognize more escape sequences.]
`bool, boolean`	`true` or `false`.
`array`	Group of elements.
`object`	Group of associated data and methods.
`resource`	An external source—usually information from a database.
`NULL`	No value.

Building blocks of PHP

Data Types

- PHP has four scalar types: Boolean, integer, double, and string; two compound types, array and object; and two special types, resource and NULL

1 Integer Type

- PHP has a single integer type, named integer. (corresponds to the long type of C)
- In most cases, this is 32 bits, or a bit less (not fewer) than ten decimal digits.

2 Double Type

- PHP's double type corresponds to the double type of C and its successors.
- Double literals can include a decimal point, an exponent, or both. The exponent has an E or an e
- Floating point numbers (also known as "floats", "doubles", or "real numbers") can be specified using any of the following syntaxes:

Order of Precedence of Style Sheet

3.String Type

- Characters in PHP are single bytes. no character type.
- A single character data value is represented as a string of length 1.
- String literals with either single (') or double quotes (")
- In single-quoted string literals, escape sequences, such as \n, are not recognized and the values of embedded variables are not substituted
- 'The sum is :$sum' exactly as it is typed.
- In double-quoted string literals, escape sequences are recognized, and embedded variables are replaced by their current values.

```
$sum=10;
print "The sum : $sum";
o/p :The sum : 10
```

4.Boolean Type

- The only two possible values for the Boolean type are TRUE and FALSE, both of which are case insensitive.
- It is used in control structure like the testing portion of an if statement.
- If an integer expression is used in Boolean context, it evaluates to FALSE if it is zero; otherwise, it is TRUE.
- If a string expression is used in Boolean context, it evaluates to FALSE if it is either the empty string or the string "0"; otherwise, it is TRUE.

Ex: $var = true;

Building blocks of PHP

Variables

Data Types

Operators & Expressions

- Constants

```
//Result of the equal operator is a Boolean
<?php
$height=100;
$width=50;
if ($width == 0)
{
echo "The width needs to be a non-zero number";
}
?>
```

- //Result of the condition is a Boolean
- <?php
- $height=100;
- $width= 50; if ($width)
- {
- echo "The area of the rectangle is".$height*$width;
- }
- else
- {
- echo "The width needs to be a non-zero number";

Determining data types

- PHP has several useful functions for determining the type of a variable.
- is_array() True if variable is an array.
- is_bool() True if variable is a bool.
- is_float(), is_double(), is_real() True if variable is a float.
- is_int(), is_integer(), is_long() True if variable is an integer.
- is_null() True if variable is set to null.
- is_numeric() True if variable is a number or numeric string.
- is_scalar() True if variable is an int, float, string, or bool.
- is_object() True if variable is an object.
- is_resource() True if variable is a resource.
- is_string() True if variable is a string.

changing data types

- PHP includes both implicit and explicit type conversions.
- Implicit type conversions are called coercions
- The context can cause a coercion of the type of the value of the expression.
- Whenever a numeric value appears in string context, the numeric value is coerced to a string.
- whenever a string value appears in numeric context, the string value is coerced to a numeric value. If the string contains a period, an e, or an E, it is converted to double; otherwise, it is converted to an integer.
- When a double is converted to an integer, the fractional part is dropped; rounding is not done

changing data types

- Explicit type conversions can be specified in three ways.
- Using the syntax of C, an expression can be cast to a different type. The cast is a type name in parentheses preceding the expression. For example, if the value of $sum is 4 . 777, the following produces 4: (int)$sum
- Another way to specify explicit type conversion is to use one of the functions intval(), doubleval(), or strval(). For example, if $sum is still 4.77 7, the following call returns 4:

```
intval($sum)
```

- The third way to specify an explicit type conversion is the settype() function, which takes two parameters: a variable and a string that specifies a type name. For example, if $sum is still 4 . 777, the following statement converts the value of $sum to 4 and its type to integer: settype($sum, "integer");

changing data types with settype()

- The settype() function converts a variable to a specific type.
- Syntax: settype(*variable*, *type*);

```
<?php
$a = "32"; // string
settype($a, "integer"); // $a is now integer
$b = 32; // integer
settype($b, "string"); // $b is now string
$c = true; // boolean
settype($c, "integer"); // $c is now integer (1)
?>
```

Parameter	Description
variable	Required. Specifies the variable to convert
type	Required. Specifies the type to convert *variable* to. The possible types are: boolean, bool, integer, int, float, double, string, array, object, null

changing data types by casting

- We can cast following data type variable in PHP
- (int), (integer) - cast to integer
- (bool), (boolean) - cast to boolean
- (float), (double), (real) - cast to float
- (string) - cast to string
- (array) - cast to array
- (object) - cast to object
- (unset) - cast to NULL (PHP 5)

Building blocks of PHP

Variables
Data Types
Operators & Expressions
Constants

Operators &expressions

- PHP has the usual (for C-based programming languages) collection of arithmetic operators (+, -, *, /, %, ++, and --)
- If either operand is a double, the operation is double and a double result is produced
- Operators are used to perform operations on variables and values.
- PHP divides the operators in the following groups:
- Arithmetic operators
- Assignment operators
- Comparison operators
- Increment/Decrement operators
- Logical operators
- String operators
- Array operators
- Conditional assignment operators

Building blocks of PHP

- Operators &expressions

Variables
Data Types
Operators & Expressions
Constants

Arithmetic Operators

Operator	Description	Example	Result
+	Addition	x=2 x+2	4
-	Subtraction	x=2 5-x	3
*	Multiplication	x=4 x*5	20
/	Division	15/5 5/2	3 2.5
%	Modulus (division remainder)	5%2 10%8 10%2	1 2 0
++	Increment	x=5 x++	x=6
--	Decrement	x=5 x--	x=4

Building blocks of PHP

- Operators &expressions

Variables
Data Types
Operators & Expressions
Constants

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
%=	x%=y	x=x%y

Building blocks of PHP

- Operators &expressions

Variables
Data Types
Operators & Expressions
Constants

Comparison Operators

Operator	Description	Example
==	is equal to	5==8 returns false
!=	is not equal	5!=8 returns true
<>	is not equal	5<>8 returns true
>	is greater than	5>8 returns false
<	is less than	5<8 returns true
>=	is greater than or equal to	5>=8 returns false
<=	is less than or equal to	5<=8 returns true

Building blocks of PHP

- Operators &expressions

Variables
Data Types
Operators & Expressions
Constants

Logical Operators

Operator	Description	Example
&&	and	x=6 y=3 (x < 10 && y > 1) returns true
||	or	x=6 y=3 (x==5 || y==5) returns false
!	not	x=6 y=3 !(x==y) returns true

Building blocks of PHP

- Operators &expressions-predefined Functions

Function	Parameter Type	Returns
floor	Double	Largest integer less than or equal to the parameter
ceil	Double	Smallest integer greater than or equal to the parameter
round	Double	Nearest integer
srand	Integer	Initializes a random number generator with the parameter
rand	Two numbers	A pseudorandom number greater than the first parameter and smaller than the second
abs	Number	Absolute value of the parameter
min	One or more numbers	Smallest
max	One or more numbers	Largest

Operator	Type	Associativity
=	assignment	right to left
+=	addition assignment	
-=	subtraction assignment	
*=	multiplication assignment	
/=	division assignment	
%=	modulus assignment	
&=	bitwise AND assignment	
|=	bitwise OR assignment	
^=	bitwise exclusive OR assignment	
.=	concatenation assignment	
<<=	bitwise shift left assignment	
>>=	bitwise shift right assignment	
=>	assign value to a named key	
and	logical AND	left to right
xor	exclusive OR	left to right
or	logical OR	left to right
,	list	left to right

floor(),ceil()

- The floor() function rounds a number DOWN to the nearest integer, if necessary.

```
<?php
echo(floor(0.60) . "<br>");  //0
echo(floor(0.40) . "<br>");//0
echo(floor(5) . "<br>");//5
echo(floor(5.1) . "<br>");//5
echo(floor(-5.1) . "<br>");//-6
echo(floor(-5.9));//-6
echo (ceil(0.70)); //1
echo(ceil(-4.1));//-4
echo(ceil(6.2));//7
echo(round(0.70878, 2));  //0.71?>
```

- ceil():To round a number UP to the nearest integer use the ceil() function.
- round(): To round a floating-point number, use the round() function

rand()

- The rand() function generates a random integer.
- If you want a random integer between 10 and 100 (inclusive), use rand (10,100).

```
<!DOCTYPE html>
<html>
<body>

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

</body>
</html>
```

```
397779448
1364525460
51
```

Building blocks of PHP

- Constants
- A constant is a name or an identifier for a simple value. A constant value cannot change during the execution of the script. By default, a constant is case-sensitive. By convention, constant identifiers are always uppercase.
- A constant name starts with a letter or underscore, followed by any number of letters, numbers, or underscores.
- If you have defined a constant, it can never be changed or undefined.
- To define a constant you have to use define() function and to retrieve the value of a constant, you have to simply specifying its name

```
<?php
  define("MINSIZE", 50);
   echo MINSIZE;
  echo constant("MINSIZE"); // same thing as the previous line?>
```

Building blocks of PHP

- Constants
- Only scalar data (boolean, integer, float and string) can be contained in constants.
- Differences between constants and variables are
- There is no need to write a dollar sign ($) before a constant, where as in Variable one has to write a dollar sign.
- Constants cannot be defined by simple assignment, they may only be defined using the define() function.
- Constants may be defined and accessed anywhere without regard to variable scoping rules.
- Once the Constants have been set, may not be redefined or undefined.

Valid and invalid constant names

```
// Valid constant names
define("ONE",     "first thing");
define("TWO2",    "second thing");
define("THREE_3", "third thing");
define("__THREE__", "third value");

// Invalid constant names
define("2TWO",    "second thing");
```

Echo and Print

- PHP echo and print Statements
- echo and print are more or less the same.
- They are both used to output data to the screen.
- echo has no return value while print has a return value of 1 so it can be used in expressions.
- echo can take multiple parameters (although such usage is rare) while print can take one argument.
- echo is marginally faster than print.

```
<!DOCTYPE html >
<html><head>
<title> today.php </title> </head>
<body> <p>
<?php
print "<b>Welcome to my home page <br /> <br />";

 print "Today is:</b>date("Y/m/d"); print "<br />";
?>
</p>
</body>
</html>
```

o/p

Welcome to my Home Page

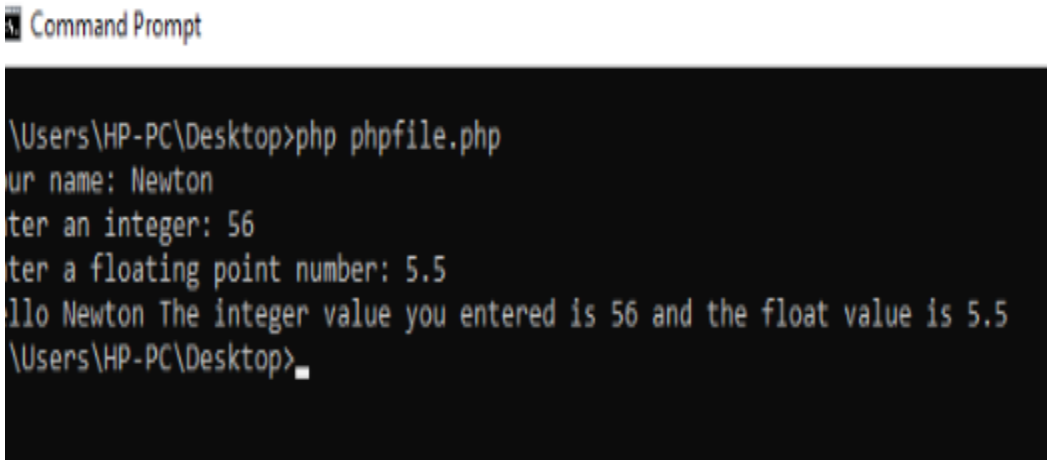
Today is 2014/09/24

l-day of the week, F-month, j-date value and S –suffix for the day

date(l,F,jS)=>Saturday June 1st

Reading single input from keyboard

```php
<?php

// Input section
// get users name
$name = (string)readline("Your name: ")

$int = (int)readline('Enter an integer: ');

$float = (float)readline('Enter a floating'
                . ' point number: ');

// Entered integer is 10 and
// entered float is 9.78
echo "Hello ".$name." The integer value you entered is "
. $int
    . " and the float value is " . $float;
?>

```



```php
<?php

// For input
// 1 2 3 4 5 6
$arr = explode(' ', readline());

// For output
print_r($arr)

?>
```

```
C:\Users\HP-PC\Desktop>php phpmulti.php
Enter multiple space separated values: 1 2 3 4 5 27 38 49 50
Array
(
    [0] => 1
    [1] => 2
    [2] => 3
    [3] => 4
    [4] => 5
    [5] => 27
    [6] => 38
    [7] => 49
    [8] => 50
)
```

Flow Control functions and control statements

- if (expression)

{ // code to execute if the expression evaluates to true }

LISTING 13.23 An `if` Statement

```
1: <?php
2: $mood = "happy";
3: if ($mood == "happy") {
4:     echo "Hooray! I'm in a good mood!";
5: }
6: ?>
```

if else Statement

```
if (expression) {
   // code to execute if the expression evaluates to true
} else {
   // code to execute in all other cases
}
```

```
1: <?php
2: $mood = "sad";
3: if ($mood == "happy") {
4:     echo "Hooray! I'm in a good mood!";
5: } else {


       echo "I'm in a $mood mood.";
   }
   ?>
```

```
if (expression) {
   // code to execute if the expression evaluates to true
} elseif (another expression) {
   // code to execute if the previous expression failed
   // and this one evaluates to true
} else {
   // code to execute in all other cases
}
```

LISTING 13.25 An `if` Statement That Uses `else` and `elseif`

```
1:  <?php
2:  $mood = "sad";
3:  if ($mood == "happy") {
4:      echo "Hooray! I'm in a good mood!";
5:  } elseif ($mood == "sad") {
6:      echo "Awww. Don't be down!";
7:  } else {
8:      echo "I'm neither happy nor sad, but $mood.";
9:  }
10: ?>
```

```
switch (expression) {
      case result1:
           // execute this if expression results in result1
           break;


      case result2:
           // execute this if expression results in result2
           break;
      default:
           // execute this if no break statement
           // has been encountered hitherto
}
```

LISTING 13.26 A `switch` Statement

```
1: <?php
2: $mood = "sad";
3: switch ($mood) {
4:     case "happy":
5:        echo "Hooray! I'm in a good mood!";
6:        break;
7:     case "sad":
8:        echo "Awww. Don't be down!";
9:        break;
10:    default:
11:       echo "I'm neither happy nor sad, but $mood.";
12:       break;
13: }
14: ?>
```

For loop

- for (initialization expression; test expression; modification expression) { // code to be executed }

LISTING 13.30 Using the for Statement

```
1: <?php
2: for ($counter = 1; $counter <= 12; $counter++) {
3:     echo $counter . " times 2 is " . ($counter * 2) . "<br>";
4: }
5: ?>
```

```
1 times 2 is 2
2 times 2 is 4
3 times 2 is 6
4 times 2 is 8
5 times 2 is 10
6 times 2 is 12
7 times 2 is 14
8 times 2 is 16
9 times 2 is 18
10 times 2 is 20
11 times 2 is 22
12 times 2 is 24
```

while Statement

- while (expression)
- { // do something }

LISTING 13.28 A `while` Statement

```
1: <?php
2: $counter = 1;
3: while ($counter <= 12) {
4:     echo $counter . " times 2 is " . ($counter * 2) . "<br>";
5:     $counter++;
6: }
7: ?>
```

do…while Statement

- do
- { // code to be executed } while (expression);

LISTING 13.29 The do…while Statement

```
1: <?php
2: $num = 1;
3: do {
4:     echo "The number is: " . $num . "<br>";
5:     $num++;
6: } while (($num > 200) && ($num < 400));
7: ?>
```

Basic Programs-Palindrome number

```
<html><body>
<?php
$num = 123454321;
$x = 0;
$n =$num;
while(floor($num))
{
$mod = $num%10;
$x = $x * 10 + $mod;
$num = $num/10;
}
if($n==$x)
{
echo "$n is a Palindrome number.";
}
else
{
echo "$n is not a Palindrome number.";
}
?>
</body></html>
```

Array-Creation & Initialization

- $rainbow = array("red", "orange", "yellow", "green", "blue", "indigo", "violet");
- $rainbow[] = "red";
  $rainbow[] = "orange";
  $rainbow[] = "yellow";
  $rainbow[] = "green";
  $rainbow[] = "blue";
  $rainbow[] = "indigo";
  $rainbow[] = "violet";

Both snippets create a seven-element array called $rainbow, with values starting at index position 0 and ending at index position 6.

- $rainbow[0] = "red";

$rainbow[1] = "orange";

$rainbow[2] = "yellow";

$rainbow[5] = "green";

$rainbow[6] = "blue";

$rainbow[7] = "indigo";

$rainbow[8] = "violet";

Regardless of whether you initially create your array using the array() function or the array operator, you can still add to it using the array operator. In the first line of the following snippet, six elements are added to the array, and one more element is added to the end of the array in the second line:

$rainbow = array("red", "orange", "yellow", "green", "blue", "indigo");

$rainbow[] = "violet";

Array Manipulatior

```
<!DOCTYPE html>

<!-- Fig. 19.7: arrays.php -->
<!-- Array manipulation. -->
<html>
   <head>
      <meta charset = "utf-8">
      <title>Array manipulation</title>
      <style type = "text/css">
         p    { margin: 0; }
         .head { margin-top: 10px; font-weight: bold; }
      </style>
   </head>
   <body>
      <?php
         // create array first
         print( "<p class = 'head'>Creating the first array</p>" );
         $first[ 0 ] = "zero";
         $first[ 1 ] = "one";
         $first[ 2 ] = "two";
         $first[] = "three";

         // print each element's index and value
         for ( $i = 0; $i < count( $first ); ++$i )
            print( "Element $i is $first[$i]</p>" );

         print( "<p class = 'head'>Creating the second array</p>" );

         // call function array to create array second
         $second = array( "zero", "one", "two", "three" );

         for ( $i = 0; $i < count( $second ); ++$i )
            print( "Element $i is $second[$i]</p>" );

         print( "<p class = 'head'>Creating the third array</p>" );

         // assign values to entries using nonnumeric indices
         $third[ "Amy" ] = 21;
         $third[ "Bob" ] = 18;
         $third[ "Carol" ] = 23;

         // iterate through the array elements and print each
         // element's name and value
         for ( reset( $third ); $element = key( $third ); next( $third
            print( "<p>$element is $third[$element]</p>" );
```

```
47          print( "<p class = 'head'>Creating the fourth array</p>" );
48
49          // call function array to create array fourth using
50          // string indices
51          $fourth = array(
52             "January"   => "first",   "February" => "second",
53             "March"     => "third",   "April"    => "fourth",
54             "May"       => "fifth",   "June"     => "sixth",
55             "July"      => "seventh", "August"   => "eighth",
56             "September" => "ninth",   "October"  => "tenth",
57             "November"  => "eleventh","December" => "twelfth" );
58
59          // print each element's name and value
60          foreach ( $fourth as $element => $value )
61             print( "<p>$element is the $value month</p>" );
62       ?><!-- end PHP script -->
63    </body>
64 </html>
```

Creating the first array
Element 0 is zero
Element 1 is one
Element 2 is two
Element 3 is three

Creating the second array
Element 0 is zero
Element 1 is one
Element 2 is two
Element 3 is three

Creating the third array
Amy is 21
Bob is 18
Carol is 23

Creating the fourth array
January is the first month
February is the second month
March is the third month
April is the fourth month
May is the fifth month
June is the sixth month
July is the seventh month
August is the eighth month
September is the ninth month
October is the tenth month
November is the eleventh month
December is the twelfth month

Creating Associative Arrays

- Associative arrays use actual named keys
- $character = array("name" => "Bob", "occupation" => "superhero", "age" => 30, "special power" => "x-ray vision");
- You can reference specific elements of an associative array using the specific key, such as in this example:

echo $character['occupation’];

- you can use the array operator to add to an associative array: $character['supername'] = "Mega X-Ray Guy";
- This example adds a key called supername with a value of Mega X-Ray Guy

Creating Multidimensional Arrays - an array that contains another array

- ```
  $cars = array (
    array("Volvo",22,18),
    array("BMW",15,13),
    array("Saab",5,2),
    array("Land Rover",17,15)
  );
  ```
  ```

COMPUTER SCIENCE AND E | (4) WhatsApp | PHP, MySQL & JavaScript All | PHP Multidimensional Arrays | W3Schools Tryit Editor

w3schools.com/php/phptryit.asp?filename=tryphp_array_multi

Gmail YouTube

Run ›

Result Size: 668 x 462

Get your own website

```
<!DOCTYPE html>
<html>
<body>

<?php
$cars = array (
  array("Volvo",22,18),
  array("BMW",15,13),
  array("Saab",5,2),
  array("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>
```

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.


337


338

LISTING 12.5 Defining a Multidimensional Array

```
1:  <?php
2:  $characters = array(
3:                    array(
4:                      "name" => "Bob",
5:                      "occupation" => "superhero",
6:                      "age" => 30,
7:                      "special power" => "x-ray vision"
8:                     ),
9:                    array(
10:                     "name" => "Sally",
11:                     "occupation" => "superhero",
12:                     "age" => 24,
13:                     "special power" => "superhuman strength"
14:                    ),
15:                    array(
16:                     "name" => "Jane",
17:                     "occupation" => "arch villain",
18:                     "age" => 45,
19:                     "special power" => "nanotechnology"
20:                    )
21:                );
22: ?>
```

- Each element consists of an associative array, itself containing four elements: name, occupation, age, and special_power
- echo $characters[1];

Array

echo $characters[1]['occupation’];

superhero

- count() and sizeof()—Each of these functions counts the number of elements in an array; sizeof() is an alias of count(). Given the array $colors = array("blue", "black", "red", "green");
- foreach()—This control structure (which looks like a function) is used to step through an array, assigning the value of an element to a given variable, as you saw in the previous section

- **reset()**—This function rewinds the pointer to the beginning of an array, as in this example:

```
reset($character);
```

  This function proves useful when you are performing multiple manipulations on an array, such as sorting, extracting values, and so forth.

- **array_push()**—This function adds one or more elements to the end of an existing array, as in this example:

```
array_push($existingArray, "element 1", "element 2", "element 3");
```

- **array_pop()**—This function removes (and returns) the last element of an existing array, as in this example:

```
$last_element = array_pop($existingArray);
```

- **array_unshift()**—This function adds one or more elements to the beginning of an existing array, as in this example:

```
array_unshift($existingArray, "element 1", "element 2", "element 3");
```

- **array_shift()**—This function removes (and returns) the first element of an existing array, as in this example, where the value of the element in the first position of `$existingArray` is assigned to the variable `$first_element`:

```
$first_element = array_shift($existingArray);
```

- **`array_merge()`**—This function combines two or more existing arrays, as in this example:

```
$newArray = array_merge($array1, $array2);
```

- **`array_keys()`**—This function returns an array containing all the key names within a given array, as in this example:

```
$keysArray = array_keys($existingArray);
```

- **`array_values()`**—This function returns an array containing all the values within a given array, as in this example:

```
$valuesArray = array_values($existingArray);
```

- **`shuffle()`**—This function randomizes the elements of a given array. The syntax of this function is simply as follows:

```
shuffle($existingArray);
```

Sorting 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

```
<?php
$fruits = array("Orange", "Grapes", "Apple","Banana");
sort($fruits);
foreach($fruits as $x )
 {
    echo $x;
    echo "<br>";
}
?>
```

o/p:

Apple

Banana

Grapes

Orange

Sorting Arrays

- sort() - sort arrays in ascending order
- rsort() - sort arrays in descending order

test2.php

```php
<?php
$fruits = array("Orange", "Grapes", "Apple","Banana");
rsort($fruits);
foreach($fruits as $x ) {
    echo $x;
    echo "<br>";
}
?>
```

Orange
Grapes
Banana
Apple

Sorting Arrays

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

test2.php

```php
<?php
$age = array("Peter"=>"45", "Ben"=>"47", "Joe"=>"43");
asort($age);
foreach($age as $x => $x_value)
{
    echo "Key=" . $x . ", Value=" . $x_value;
    echo "<br>";
}
?>
```

Key=Joe, Value=43
Key=Peter, Value=45
Key=Ben, Value=47

Sorting Arrays

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

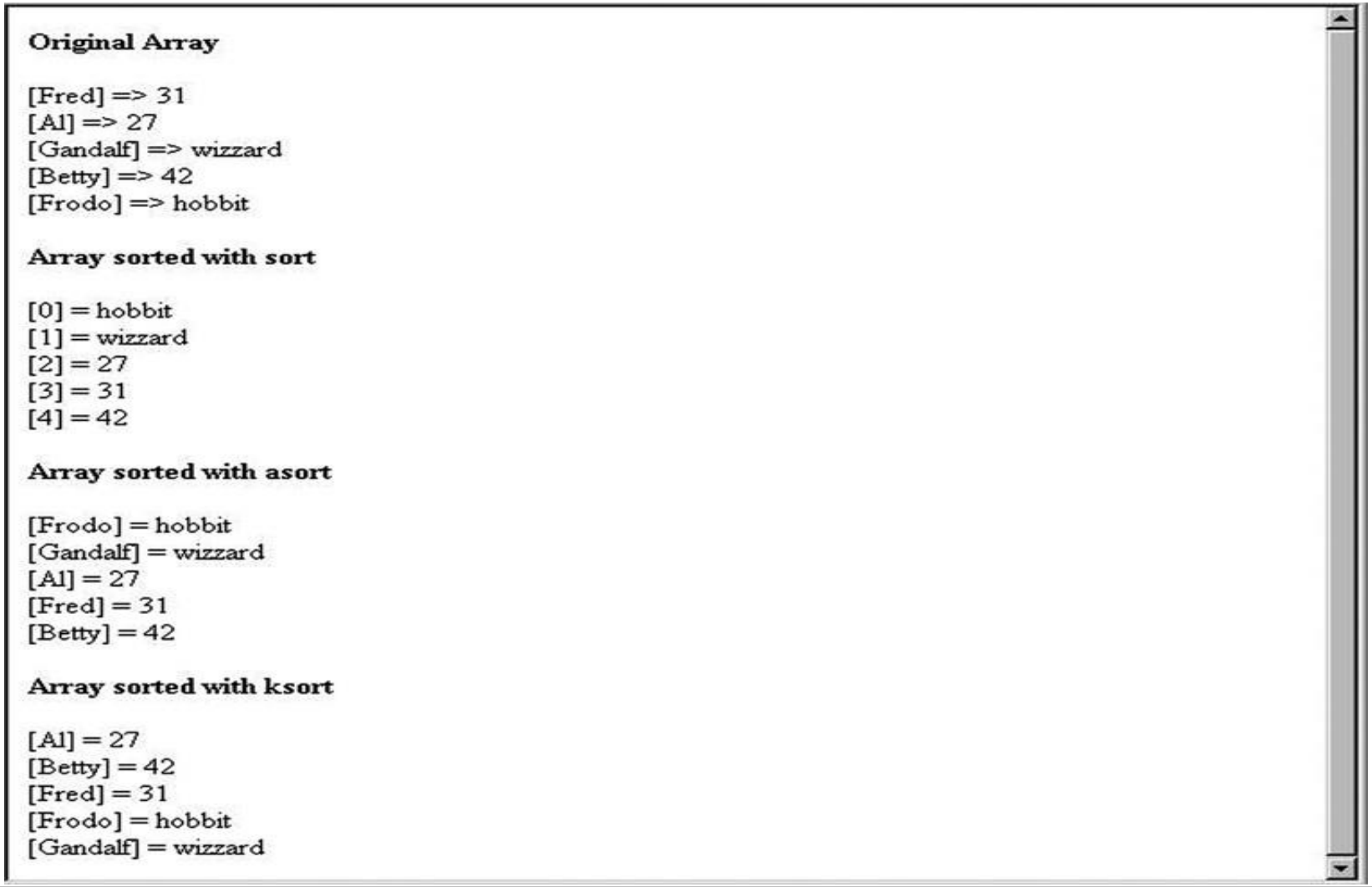
test2.php

```php
<?php
$age = array("Peter"=>"45", "Ben"=>"47", "Joe"=>"43");
ksort($age);
foreach($age as $x => $x_value)
{
    echo "Key=" . $x . ", Value=" . $x_value;
    echo "<br>";
}
?>
```

Key=Ben, Value=47
Key=Joe, Value=43
Key=Peter, Value=45

- The sort function, which takes an array as a parameter, sorts the values in the array, replacing the keys with the numeric keys, 0, 1, 2, ...
- The array can have both string and numeric values.
- The string values migrate to the beginning of the array in alphabetical order. The numeric values follow in ascending order

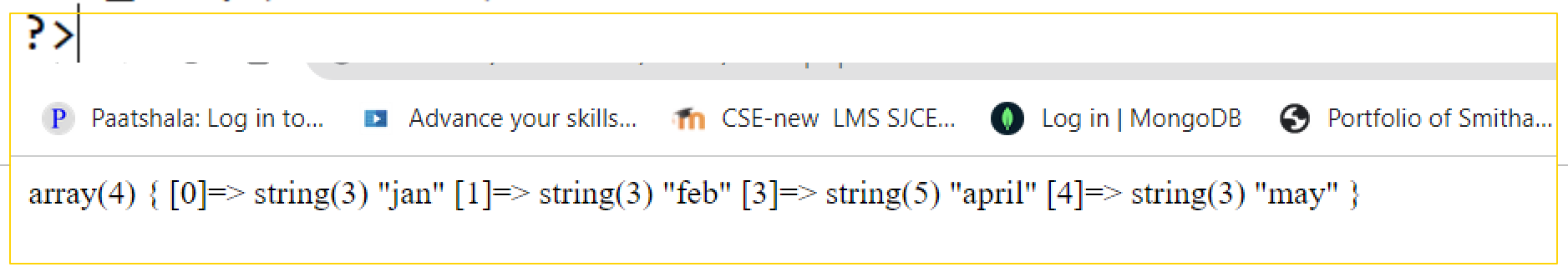
Sorting Arrays



Arrays-Using array_search()

- With the help of array_search() function, we can remove specific elements from an array.

```php
<?php
$delete_item = 'march';
// take a list of months in an array
$months = array('jan', 'feb', 'march', 'april', 'may');
if (($key = array_search($delete_item, $months)) !== false) {
    unset($months[$key]);
}

// print array to see latest values
var_dump($months);
?>
```

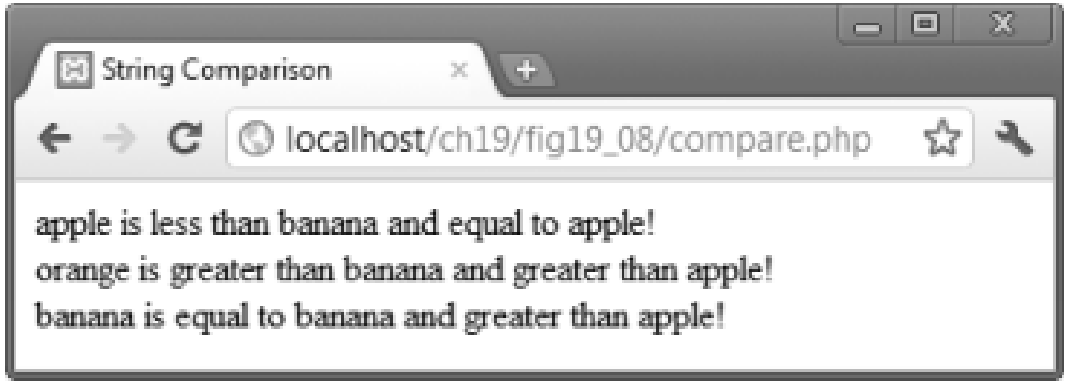


String Comparisons

- strcmp -compare two strings.
- The function returns -1 if the first string alphabetically precedes the second string. 0 if the strings are equal, and 1 if the first string alphabetically follows the second.
- Relational operators (==, !=, <=, > and >=) can also be used to compare strings.

```
<!DOCTYPE html>

<!-- Fig. 19.8: compare.php -->
<!-- Using the string-comparison operators. -->
<html>
   <head>
      <meta charset = "utf-8">
      <title>String Comparison</title>
      <style type = "text/css">
         p { margin: 0; }
      </style>
   </head>
   <body>
      <?php
         // create array fruits
         $fruits = array( "apple", "orange", "banana" );

         // iterate through each array element
         for ( $i = 0; $i < count( $fruits ); ++$i )
         {
            // call function strcmp to compare the array element
            // to string "banana"
            if ( strcmp( $fruits[ $i ], "banana" ) < 0 )
               print( "<p>" . $fruits[ $i ] . " is less than banana " );
```

```
            elseif ( strcmp( $fruits[ $i ], "banana" ) > 0 )
               print( "<p>" . $fruits[ $i ] . " is greater than banana ");
            else
               print( "<p>" . $fruits[ $i ] . " is equal to banana " );

            // use relational operators to compare each element
            // to string "apple"
            if ( $fruits[ $i ] < "apple" )
               print( "and less than apple!</p>" );
            elseif ( $fruits[ $i ] > "apple" )
               print( "and greater than apple!</p>" );
            elseif ( $fruits[ $i ] == "apple" )
               print( "and equal to apple!</p>" );
         } // end for
      ?><!-- end PHP script -->
   </body>
</html>
```



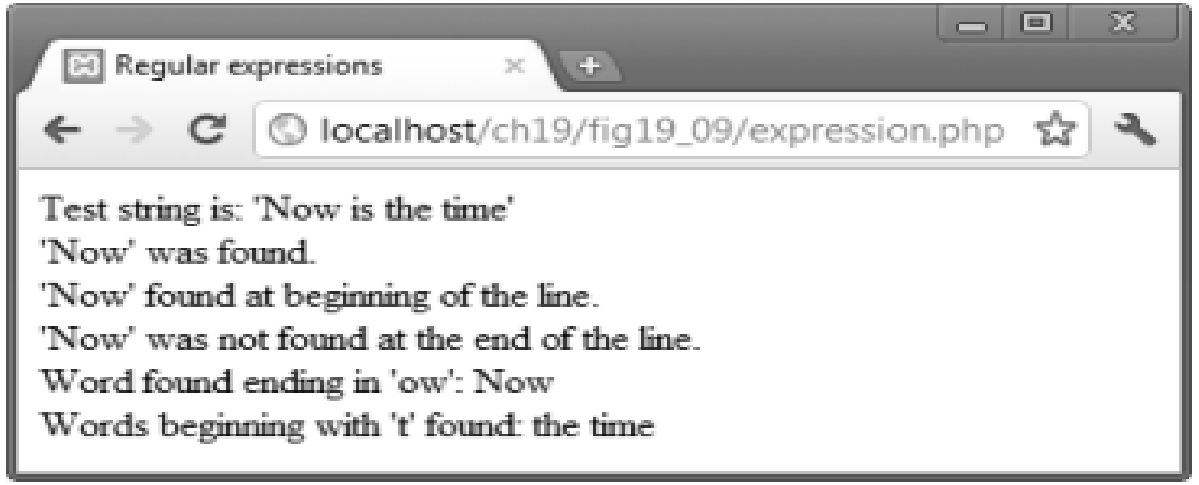
String Processing with Regular Expressions

- Function **preg_match** uses regular expressions to search a string for a specified pattern using Perl-compatible regular expressions

```
<!DOCTYPE html>

<!-- Fig. 19.9: expression.php -->
<!-- Regular expressions. -->
<html>
   <head>
      <meta charset = "utf-8">
      <title>Regular expressions</title>
      <style type = "text/css">
         p { margin: 0; }
      </style>
   </head>
```

```
<body>
   <?php
      $search = "Now is the time";
      print( "<p>Test string is: '$search'</p>" );

      // call preg_match to search for pattern 'Now' in variable search
      if ( preg_match( "/Now/", $search ) )
         print( "<p>'Now' was found.</p>" );

      // search for pattern 'Now' in the beginning of the string
      if ( preg_match( "/^Now/", $search ) )
         print( "<p>'Now' found at beginning of the line.</p>" );

      // search for pattern 'Now' at the end of the string
      if ( !preg_match( "/Now$/", $search ) )
         print( "<p>'Now' was not found at the end of the line.</p>" );

      // search for any word ending in 'ow'
      if ( preg_match( "/\b([a-zA-Z]*ow)\b/i", $search, $match ) )
         print( "<p>Word found ending in 'ow': " .
            $match[ 1 ] . "</p>" );
```

```
        // search for any words beginning with 't'
        print( "<p>Words beginning with 't' found: " );

        while ( preg_match( "/\b(t[[:alpha:]]+)\b/", $search, $match ) )
        {
            print( $match[ 1 ] . " " );

            // remove the first occurrence of a word beginning
            // with 't' to find other instances in the string
            $search = preg_replace("/" . $match[ 1 ] . "/", "", $search);
        } // end while

        print( "</p>" );
    ?><!-- end PHP script -->
  </body>
</html>
```



String functions

- strlen() - Return the Length of a String

```php
<?php
echo strlen("Hello world!"); // outputs 12
?>
```

str_word_count() - Count Words in a String

```php
<?php
echo str_word_count("Hello world!"); // outputs 2
?>
```

- strrev() - Reverse a String
- ```
  <?php
  echo strrev("Hello world!"); // outputs !dlrow
  olleH
  ?>
  ```
- str_replace() - Replace Text Within a String
- ```
  <?php
  echo str_replace("world", "Dolly", "Hello
  world!"); // outputs Hello Dolly!
  ?>
  ```

Implode-The implode() function returns a string from the elements of an array.

- <!DOCTYPE html>
- <html>
- <body>

- <?php
- $arr = array('Hello','World!','Beautiful','Day!');
- echo implode(" ",$arr);
- ?>

- </body>
- </html>

Hello World! Beautiful Day!

Explode-The explode() function breaks a string into an array.

- <!DOCTYPE html>
- <html>
- <body>

Array ([0] => Hello [1] => world. [2] => It's [3] => a [4] => beautiful [5] => day.)

- <?php
- $str = "Hello world. It's a beautiful day.";
- print_r (explode(" ",$str));
- ?>

- </body>
- </html>The "separator" parameter cannot be an empty string.

FUNCTIONS

- A function is a self-contained block of code that can be called by your scripts.
- When called, the function's code is executed and performs a particular task.
- You can pass values to a function, which then uses the values appropriately—storing them, transforming them, displaying them, or whatever the function is told to do.
- When finished, a function can also pass a value back to the original code that called it into action

- Two types
- Built in functions

$text = strtoupper("Hello World!")

LISTING 13.1 Calling the Built-In `abs()` Function

```
<?php
$num = -321;
$newnum = abs($num);
echo $newnum;
//prints "321"
?>
```

User defined functions

- function some_function($argument1, $argument2)

{

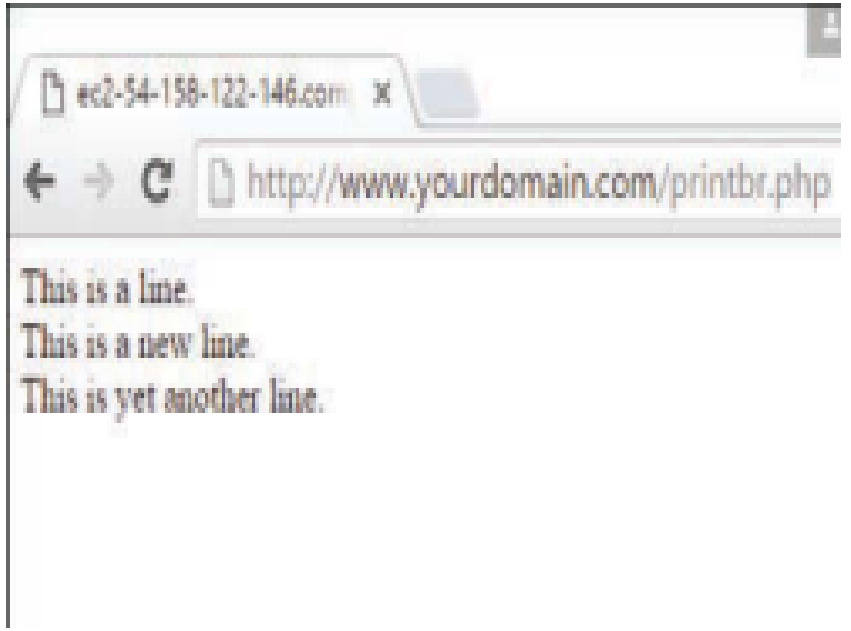
//function code here

}

```
<?php
function bighello()
{
      echo "<h1>HELLO!</h1>";
}
bighello();
?>
```

LISTING 13.3 Declaring a Function That Requires an Argument

```
1: <?php
2: function printBR($txt)
3: {
4:       echo $txt."<br>";
5: }
6: printBR("This is a line.");
7: printBR("This is a new line.");
8: printBR("This is yet another line.");
9: ?>
```



Return

LISTING 13.4 A Function That Returns a Value

```
1: <?php
2: function addNums($firstnum, $secondnum)
3: {
4:       $result = $firstnum + $secondnum;
5:       return $result;
6: }
7: echo addNums(3,5);
8: //will print "8"
9: ?>
```

The return statement stops the execution of the function and sends the value back to the calling code.

LISTING 13.5 Variable Scope: A Variable Declared Within a Function Is Unavailable Outside the Function

```
<?php
function test()
{
     $testvariable = "this is a test variable";
}
echo "test variable: ".$testvariable."<br>";
?>
```

LISTING 13.6 Variables Defined Outside Functions Are Inaccessible from Within a Function by Default

```
1: <?php
2: $life = 42;
3: function meaningOfLife()
4: {
5:       echo "The meaning of life is ".$life";
6: }
7: meaningOfLife();
8: ?>
```

LISTING 13.7 Accessing Global Variables with the `global` Statement

```
1: <?php
2: $life=42;
3: function meaningOfLife()
4: {
5:       global $life;
6:       echo "The meaning of life is ".$life";
7: }
8: meaningOfLife();
9: ?>
```

You can declare more than one variable at a time with the global statement by simply separating each of the variables you want to access with commas: global $var1, $var2, $var3;

LISTING 13.10 A Function Requiring Two Arguments

```
1:  <?php
2:  function fontWrap($txt, $fontsize)
3:  {
4:     echo "<span style=\"font-size:".$fontsize."\">".$txt."</span>";
5:  }
6:  fontWrap("really big text<br/>","24pt");
7:  fontWrap("some body text<br/>","16pt");
8:  fontWrap("smaller body text<br/>","12pt");
9:  fontWrap("even smaller body text<br/>","10pt");
10: ?>
```

LISTING 13.12 Passing an Argument to a Function by Value

```
1: <?php
2: function addFive($num)
3: {
4:       $num += 5;
5: }
6: $orignum = 10;
7: addFive($orignum);
8: echo $orignum;
9: ?>
```

LISTING 13.13 Using a Function Definition to Pass an Argument to a Function by Reference

```
1: <?php
2: function addFive(&$num)
3: {
4:       $num += 5;
5: }
6: $orignum = 10;
7: addFive($orignum);
8: echo $orignum;
9: ?>
```

Objects

- An object is a sort of theoretical box of things—variables, functions, and so forth—that exists in a templated structure called a class
- Think of an object as a little box with inputs and outputs on either side of it. The input mechanisms are methods, and methods have properties.
- an object exists as a data structure, and a definition of that structure called a class. In each class, you define a set of characteristics.
- For example, suppose you have created an automobile class. In the automobile class, you might have color, make, and model characteristics
- Each automobile object uses all the characteristics, but each object initializes the characteristics to different values, such as blue, Jeep, and Renegade, or red, Porsche, and Boxster.

Objects

- Creating object is so simple

```
class myClass {
      //code will go here
}
```

- Create an instance of class as:

```
$object1 = new myClass();
```

```
<?php
class myClass {
    //code will qo here  }
$object1 = new myClass();
echo "\$object1 is an ".gettype($object1).".<br>";
if (is_object($object1)) {
    echo "Really! I swear \$object1 is an object!";
 }
 ?>
```

```
$object1 is an object.
Really! I swear $object1 is an object!
```

Properties of Objects

- The variables declared inside an object are called properties.
- These properties can be values, arrays, or even other objects.
- If you use the keyword public, protected, or private before the variable name, you can indicate if the class member (the variable) can be accessed everywhere (public), within the class itself or a parent class or an inherited class (protected), or only by the class itself (private) :

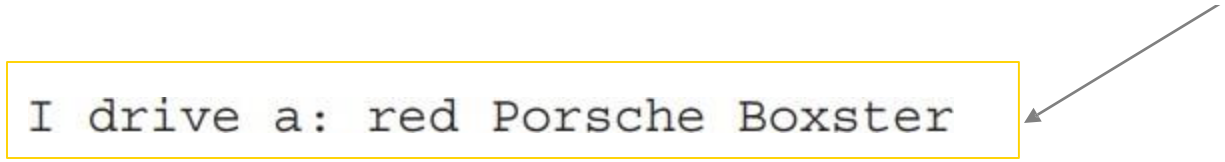
```
class myCar {
      public $color = "silver";
      public $make = "Mazda";
      public $model = "Protege5";
}
```

```
<?php
class myCar {
    public $color = "blue";
    public $make = "Jeep";
    public $model = "Renegade"; }
    $car = new myCar();
    echo "I drive a: " . $car->color
    . " ".$car->make . " " . $car->model;
    ?>
```

```
I drive a: blue Jeep Renegade
```

Changing Object Properties

```
1:  <?php
2:  class myCar {
3:     public $color = "blue";
4:     public $make = "Jeep";
5:     public $model = "Renegade";
6:  }
7:  $car = new myCar();
8:  $car->color = "red";
9:  $car->make = "Porsche";
10: $car->model = "Boxster";
11: echo "I drive a: " . $car->color . " " . $car->make . " ". $car->model;
12: ?>
```



Object Methods

- Methods add functionality to your objects
- The -> operator is used to call the object method in the context of your script. Had there been any variables stored in the object, the method would have been capable of accessing them for its own purposes

```
<?php
class myClass {
    public function sayHello() {
        echo "HELLO!";
    }
}
$object1 = new myClass();
$object1->sayHello();
?>
```

Object Methods

Accessing Class Properties Within a Method

```
1:  <?php
2:  class myClass {
3:      public $name = "Jimbo";
4:      public function sayHello() {
5:          echo "HELLO! My name is " . $this->name;
6:      }
7:  }
8:  $object1 = new myClass();
9:  $object1->sayHello();
10: ?>
```

```
HELLO! My name is Jimbo
```

- The special variable $this is used to refer to the currently instantiated object
- Any time an object refers to itself, you must use the $this variable.
- Using the $this variable in conjunction with the -> operator enables you to access any property or method in a class, within the class itself

Changing the value of a property from within a method

```
<?php
class myClass {
 public $name = "Jimbo";
public function setName($n)
 {
$this->name = $n;
 }
 public function sayHello()
 {
     echo "HELLO! My name is ".$this->name;
     }
 }
     $object1 = new myClass();
     $object1->setName("Julie");
     $object1->sayHello();
 ?>
);
```

- when the sayHello() function is called and it looks for $this->name, it uses Julie, which is the new value that was just set by the setName() function.

Constructors

- A constructor is a function that lives within a class and, given the same name as the class, is automatically called when a new instance of the class is created using new classname.
- Using constructors enables you to provide arguments to your class, which will then be processed immediately when the class is called.
- A constructor allows you to initialize an object's properties upon creation of the object.If you create a__construct() function, PHP will automatically call this function when you create an object from a class.

```php
<?php
class myClass {
 public $name;
function __construct($n)
 {
$this->name = $n;
 }
 public function sayHello()
 {
     echo "HELLO! My name is ".$this->name;
     echo "<br/><br/>";
    }
 }
     $object2 = new myClass("Benson");
     $object2->sayHello();
?>
```

Inheritance

LISTING 13.21 A Class Inheriting from Its Parent

```
1:  <?php
2:  class myClass {
3:     public $name = "Benson";
4:     public function myClass($n) {
5:          $this->name = $n;
6:     }
7:     public function sayHello() {
8:          echo "HELLO! My name is ".$this->name;
9:     }
10: }
11: class childClass extends myClass {
12: //code goes here
13: }
14: $object1 = new childClass("Baby Benson");
15: $object1->sayHello();
16: ?>
```

HELLO! My name is Baby Benson

LISTING 13.22 The Method of a Child Class Overriding That of Its Parent

```
1:  <?php
2:  class myClass {
3:     public $name = "Benson";
4:     public function myClass($n) {
5:          $this->name = $n;
6:     }
7:     public function sayHello() {
8:          echo "HELLO! My name is ".$this->name;
9:     }
10: }
11: class childClass extends myClass {
12:     public function sayHello() {
13:          echo "I will not tell you my name.";
14:     }
15: }
16: $object1 = new childClass("Baby Benson");
17: $object1->sayHello();
18: ?>
```

I will not tell you my name