

# MODULE 3

# Module 3

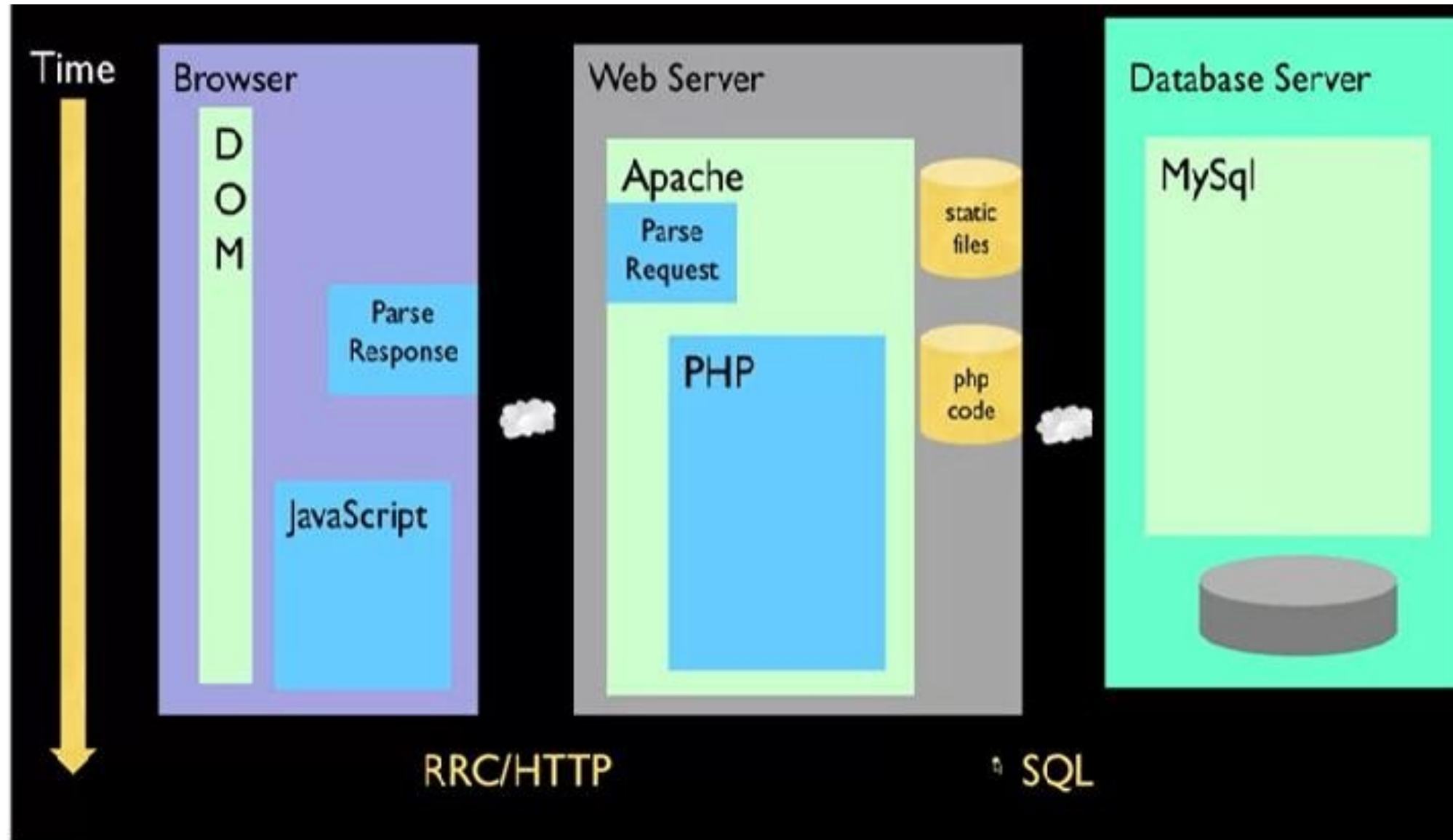
- PHP Language Structure: Introduction- Building blocks of PHP
- Variables, Data Types -Converting between Data Types-
- Operators and Expressions
- simple PHP program
- Flow Control functions
- Control statements
- Working with Functions
- Initializing and Manipulating Arrays
- Objects- String Comparisons-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

# PHP Language Structure

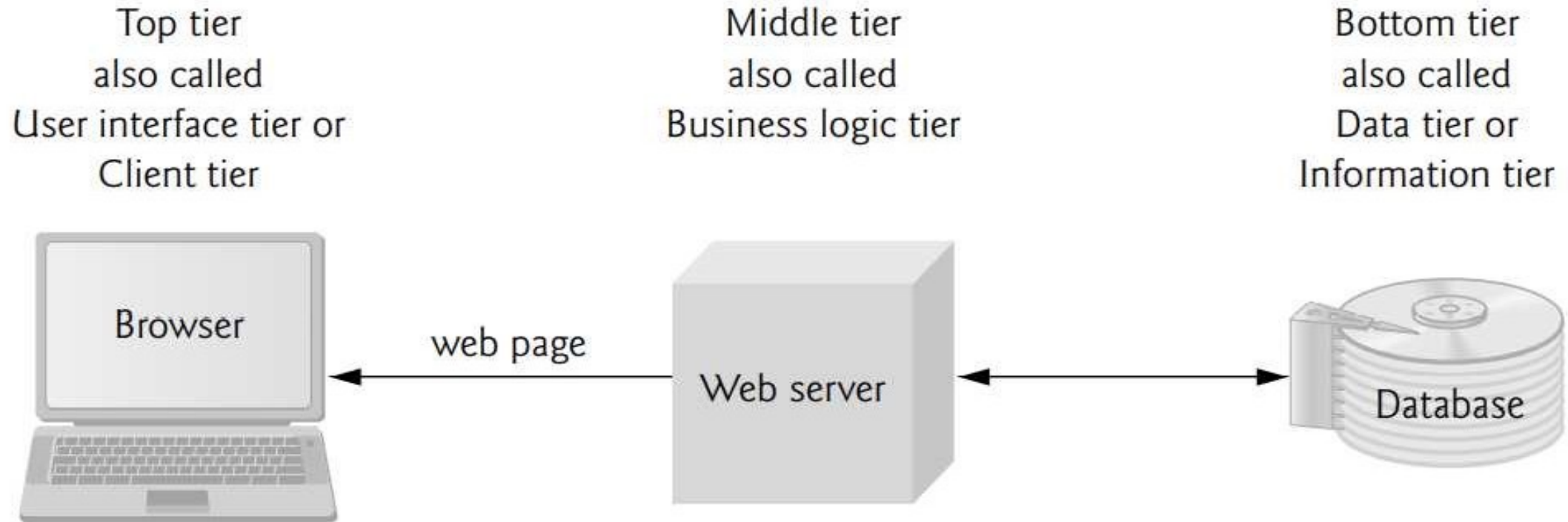
- [Rasmus Lerdorf](#) developed PHP in 1994.
  - A PHP script is executed on the server, and the plain HTML result is sent back to the browser.
  - A PHP script can be placed anywhere in the document.
  - A PHP script starts with `<?php` and ends with `?>`
  - The default file extension for PHP files is `".php"`.

- A PHP file normally contains HTML tags, and some PHP scripting code.
- PHP code is executed on the server, generating HTML which is then sent to the client.
- The client would receive the results of running that script, but would not know what the underlying code was.
- PHP is compatible with almost all servers used today (Apache, IIS, etc.). PHP is easy to learn and runs efficiently on the server side
- PHP is FREE to download from : [www.php.net](http://www.php.net)

# Introduction to PHP



# 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

ON SERVER

```
<html>
<head> <title>Welcome</title> </head>
<body>
<?
    echo "Hello";
    print "<br />";
    echo "<b>I'm here..</b>";
??
</body>
</html>
```



```
<html>
<head> <title>Welcome</title> </head>
<body>
Hello<br /><b>I'm here..</b></body>
</html>
```



Welcome - Mozilla

Hello  
I'm here..

Done

2006© justMy illustrations

- 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>

- PHP: “Hypertext Preprocessor”
- Originally, PHP was an acronym for Personal Home Page
- open source general-purpose scripting language
- web development tool (not case sensitive) and can be embedded into HTML
- File ends in .php—
- Separated in files with the `<?php     ?>` tag
- php commands can make up an entire file, or can be contained in html.

- Program lines end in ";" or you get an error
- PHP scripts are executed on the server and is a case sensitive language . but PHP function names are case in-sensitive
- PHP supports many databases (MySQL, Informix, Oracle, Sybase, Solid, PostgreSQL, Generic ODBC, etc.)
- PHP runs on different platforms (Windows, Linux, Unix, etc.)

# Building blocks of PHP

- ▶ Variables
- ▶ Data Types
- ▶ Operators and Expressions
- ▶ Constants



# Variable

- 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*

- 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**

## Global Variables

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

```
<?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
$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.

# Super Global Variables

- GLOBALS
- \$\_SERVER
- \$\_REQUEST
- \$\_POST
- \$\_GET
- \$\_ENV
- \$\_COOKIE
- \$\_SESSION



- 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
```

```
$a = 1;
```

```
$b = 2;
```

```
function Sum()
```

```
{
```

```
    $GLOBALS['b'] = $GLOBALS['a'] + $GLOBALS['b'];
```

```
}
```

```
Sum();
```

```
echo $b;
```

```
?>
```

- `$GLOBALS`- PHP super global variable which is used to access global variables from anywhere in the PHP script
- `$_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.

# RESERVED WORDS

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

# 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. [ <i>Note:</i> 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.

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



# 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

# 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;`

- <?php
- \$height=100;
- \$width=50;
- if (\$width == 0)
- {
- echo "The width needs to be a non-zero number";
- }
- ?>

- 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.

- 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");`

- 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)

# 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**

## 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



## Assignment Operators

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

## 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

## 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

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

- 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>`

# 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?>

## 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");
```

- 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.

# Echo and Print

- **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**

# Flow Control statements in PHP

- If statement can include any number of elseif clauses.
- example of an if construct:
- `if ($day == "Saturday" || $day == "Sunday")`
- `{ $today = "weekend"`
- `};`
- `else`
- `{ $today = "weekday";`
- `}`

- `if($num>0)`
- `$pcount++;`
- `elseif($num<0)`
- `$ncount++;`
- `else`
- `print "Zero";`

- switch (*n*)
  - { case *label1*:
- *code to be executed if n=label1;*
- break;
- case *label2*:
- *code to be executed if n=label2;*
- break;

- *case label3:*
- *code to be executed if n=label3;*
- *break;*
- *...*
- *default:*
- *code to be executed if n is different from all labels;*
- *}*

- The while, for, and do-while statements of PHP are exactly like those of Java.
- PHP also has a foreach statement

```
<html>
<body>

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

</body>
</html>
```



```
<?php
$count = 1;
$sum = 0;
do
| {
$sum += $count;
$count++;
} while ($count <= 100);
?>
```

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

The number is 1  
The number is 2  
The number is 3  
The number is 4  
The number is 5

```
<html>
<body>

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

</body>
</html>
```

- The break statement can be used to terminate the execution of a for, foreach, while, or do-while construct.
- The continue statement is used in loop constructs to skip the remainder of the current iteration but continue execution at the beginning of the next.

```

<html>
  <head> <title> powers.php </title> </head>
  <body>
    <table border = "border">
      <caption> Powers table </caption>
      <tr>
        <th> Number </th>
        <th> Square Root </th>
        <th> Square </th>
        <th> Cube </th>
        <th> Quad
      </th>
    </tr>
    <?php
    for ($number = 1; $number <=10; $number++)
    {
      $root = sqrt($number);
      $square = pow($number, 2);
      $cube = pow($number, 3);
      $quad = pow($number, 4);
      print("<tr align = 'center'> <td> $number </td>");
      print("<td> $root </td> <td> $square </td>");
      print("<td> $cube </td> <td> $quad </td> </tr>");
    ?>
  </table>
</body>
</html>

```

Powers table

Number	Square Root	Square	Cube	Quad
1	1	1	1	1
2	1.4142135623731	4	8	16
3	1.7320508075689	9	27	81
4	2	16	64	256
5	2.2360679774998	25	125	625
6	2.4494897427832	36	216	1296
7	2.6457513110646	49	343	2401
8	2.8284271247462	64	512	4096
9	3	81	729	6561
10	3.1622776601684	100	1000	10000

```
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>
```



one

two

three

```
<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>
```

# Arrays in PHP

- PHP provides the capability to store data in arrays.
- Arrays are divided into elements that behave as individual variables.
- Array names, begin with the \$ symbol
- Individual array elements are accessed by following the array's variable name with an index enclosed in square brackets ([]).
- If a value is assigned to an array element of an array that does not exist, then the array is created.

- assigning a value to an element where the index is omitted appends a new element to the end of the 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

# Creating numeric Arrays in PHP

- A numeric array stores each array element with a numeric index.
- There are two methods to create a numeric array.
  - 2)second way to create an array is with the array construct.
  - In the following example the index is automatically assigned (the index starts at 0):

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

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

# Creating Associative Arrays in PHP

- With an associative array, each ID key is associated with a value.
- When storing data about specific named values, a numerical array is not always the best way to do it.
- With associative arrays we can use the values as keys and assign values to them



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

The code above will output:

```
Peter is 32 years old.
```

```
<?php
```

```
$mon = array(
```

```
    "January" => "first", "February" => "second",
```

```
    "March" => "third", "April" => "fourth",
```

```
    "May" => "fifth", "June" => "sixth",
```

```
    "July" => "seventh", "August" => "eighth",
```

```
    "September" => "ninth", "October" => "tenth",
```

```
    "November" => "eleventh", "December" => "twelfth" );
```

```
foreach ( $mon as $monname => $monvalue )
```

```
    print( "<p>$monname is the $monvalue month</p>" );
```

```
?>
```

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

test1.php

```
1  <?php
2  print( "<p class = 'head'>Creating the first array</p>" );
3  $first[ 0 ] = "zero";
4  $first[ 1 ] = "one";
5  $first[ 2 ] = "two";
6  $first[] = "three";
7  for ( $i = 0; $i < count( $first ); ++$i )
8  | print( "Element $i is $first[$i]</p>" );
9  print( "<p class = 'head'>Creating the second array</p>" );
10 $second = array( "aa", "bb", "cc", "dd" );
11 for ( $i = 0; $i < count( $second ); ++$i )
12 | print( "Element $i is $second[$i]</p>" );
13 | print( "<p class = 'head'>Creating the third array</p>" );
14 $third[ "Amy" ] = 21;
15 $third[ "Bob" ] = 18;
16 $third[ "Carol" ] = 23;
17 | foreach ( $third as $name => $age )
18 | | print( "AGE of $name is $age <br />" );
19 print( "<p class = 'head'>Creating the fourth array</p>" );
20 $fourth = array( "Mon" => 23, "Tue" => 18, "Wed" => 27 );
21 | | foreach ( $fourth as $day => $temp )
22 | | | print( "The temperature on $day was $temp <br />" );
23 ?>
```

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 aa

Element 1 is bb

Element 2 is cc

Element 3 is dd

Creating the third array

AGE of Amy is 21

AGE of Bob is 18

AGE of Carol is 23

Creating the fourth array

The temperature on Mon was 23

The temperature on Tue was 18

The temperature on Wed was 27


# Functions dealing with Array

- **Unset**

- A whole array can be deleted with `unset()`, as with a scalar variable.
- Individual elements of an array also can be removed with `unset`, as in
- the following:
- **`$list = array(2, 4, 6, 8);`**
- **`unset($list[2]);`**
- Now `$list` has three remaining elements with keys 0,1, and 3 and elements 2, 4, and 8.



- The collection of keys and the collection of values of an array can be extracted with built-in functions.
- The **array\_keys** function takes an array as its parameter and returns an array of the keys of the given array.
- The returned array uses 0, 1, and so forth as its keys.

 test2.php

```
1  <?php
2  $temp = array("Mon" => 74, "Tue" => 70, "Wed" => 67, "Thu" => 62, "Fri" => 65);
3  $days = array_keys($temp);
4  $temps = array_values($temp);
5  echo "Days<br>";
6
7  for ( $i = 0; $i < count( $days ); ++$i )
8  print( "Day $i is $days[$i]</p>" );
9  |
10 echo "Temperatures<br>";
11 for ( $i = 0; $i < count( $temps ); ++$i )
12 print( "Day $i temp is $temps[$i]</p>" );
13 ?>
```



Days	Temperatures
Day 0 is Mon	Day 0 temp is 74
Day 1 is Tue	Day 1 temp is 70
Day 2 is Wed	Day 2 temp is 67
Day 3 is Thu	Day 3 temp is 62
Day 4 is Fri	Day 4 temp is 65

- The existence of an element of a specific key can be determined with the `array_key_exists` function, which returns a Boolean value.
- For example, consider the following:

 test2.php

```
1  <?php
2  $temp = array("Mon" => 74, "Tue" => 70, "Wed" => 67, "Thu" => 62, "Fri" => 65);
3  if (array_key_exists("Tue", $temp))
4  | { $tues_high = $temp["Tue"];
5    print "The high on Tuesday was $tues_high <br />";
6  }
7  ?>
```

The `is_array` function is similar to the `is_int` function: It takes a variable as its parameter and returns `TRUE` if the variable is an array, `FALSE` otherwise.

10 of 30

- The `in_array` function takes two parameters an expression and an array and returns `TRUE` if the value of the expression is a value in the array;
- otherwise, it returns `FALSE`.

- **sizeof()**
- The **number of elements in an array can be determined with the sizeof function.**
- For example, consider the following code:
- `$list = array("Bob", "Fred", "Alan", "Bozo");`
- `$len = sizeof($list);`
- After executing this code, \$len will be 4.

- `count()` -counts the number of elements in an array;
- `sizeof()` is an alias of `count()`.
- Given the array
- `$colors = array("blue", "black", "red", "green");`
- both `count($colors);` and `sizeof($colors);` return a value of 4

- **The explode function explodes a string into substrings and returns them in an array.**
- The delimiters of the substrings are defined by the first parameter to explode, which is a string; the second parameter is the string to be converted.
- `$str = "May god bless you all";`
- `$words = explode(" ", $str);`



- \$words creates an array which contains ("May", "god", "bless", "you", "all").
- **The implode function does the inverse of explode.**
- Given a separator character (or string) and an array, it concatenates the elements of the array together, using the given separator string between the elements, and returns the result as a string.

```
$words = array("Are", "you", "coming", "to", "India?");  
$str = implode(" ", $words);  
$str has "Are you coming to India?"
```

- The `array_push` and `array_pop` functions provide a simple way to implement a stack in an array.
- The `array_push` function takes as its first parameter an array.
- After this first parameter, there can be any number of additional parameters.
- The values of all subsequent parameters are placed at the end of the array.

- The `array_push` **function returns the new number of elements in the array.**

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

- array\_pop function takes a single parameter, the name of an array.
- **It removes the last element from the array** and returns it.
- The value NULL is returned if the array is empty.

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

- `array_unshift()`—This function adds one or more elements to the beginning of an existing array

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

`array_shift()`—This function removes (and returns) the first element of an existing array,

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

- `array_merge()`—This function combines two or more existing arrays.

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



# To process Array use foreach statement

- The foreach statement is designed to build loops that process all of the elements of an array.
- This statement has two forms:
- `foreach (array as scalar_variable) loop body`
- `foreach (array as key => value) loop body`

- In the first form, one of the array's values is set to the scalar variable for each iteration of the loop body.
- Ex: `$list=array(2,4,6,8);`
- `foreach ($list as $temp)`
- `print("$temp <br />");`
- This code will produce the values of all of the elements of `$list`

- The second form of foreach provides both the key and the value of each element of the array.
- **foreach (array as key => value) loop body**
- For example:
- **\$lows = array("Mon" => 23, "Tue" => 18, "Wed" => 27);**
- **foreach (\$lows as \$day => \$temp)**
- **print("The low temperature on \$day was \$temp <br />");**

- `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

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

 test2.php

```
1  <?php
2  $fruits = array("Orange", "Grapes", "Apple","Banana");
3  rsort($fruits);
4  foreach($fruits as $x ) {
5      echo $x;
6      echo "<br>";
7  }
8  ?>
```



Orange  
Grapes  
Banana  
Apple

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

```
test2.php
1  <?php
2  $age = array("Peter"=>"45", "Ben"=>"47", "Joe"=>"43");
3  asort($age);
4  foreach($age as $x => $x_value)
5  {
6      echo "Key=" . $x . ", Value=" . $x_value;
7      echo "<br>";
8  }
9  ?>
```

Key=Joe, Value=43

Key=Peter, Value=45

Key=Ben, Value=47

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

```
test2.php
1  <?php
2  $age = array("Peter"=>"45", "Ben"=>"47", "Joe"=>"43");
3  ksort($age);
4  foreach($age as $x => $x_value)
5  {
6      echo "Key=" . $x . ", Value=" . $x_value;
7      echo "<br>";
8  }
9  ?>
```

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

### **Original Array**

[Fred] => 31  
[Al] => 27  
[Gandalf] => wizzard  
[Betty] => 42  
[Frodo] => hobbit

### **Array sorted with sort**

[0] = hobbit  
[1] = wizzard  
[2] = 27  
[3] = 31  
[4] = 42

### **Array sorted with asort**

[Frodo] = hobbit  
[Gandalf] = wizzard  
[Al] = 27  
[Fred] = 31  
[Betty] = 42

### **Array sorted with ksort**

[Al] = 27  
[Betty] = 42  
[Fred] = 31  
[Frodo] = hobbit  
[Gandalf] = wizzard

# Arrays-Using array\_search()


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





```
<?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);
?>
```


← → ↻ 🏠 ⓘ localhost/dashboard/smith/test3.php

 Paatshala: Log in to...

 Advance your skills...

 CSE-new LMS SJCE...

 Log in | MongoDB

 Portfolio of Smitha...

```
array(4) { [0]=> string(3) "jan" [1]=> string(3) "feb" [3]=> string(5) "april" [4]=> string(3) "may" }
```

# Arrays-Using array\_diff()

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

---

```
<?php
$delete_item = 'april';
// take a list of months in an array
$months= array('jan', 'feb', 'march', 'april', 'may');
$final_months= array_diff($months, array($delete_item));

// var_dump - Dumps information about a variable
var_dump($final_months);
?>
```

← → ↻ 🏠 ⓘ localhost/dashboard/smith/test3.php

 Paatshala: Log in to...  Advance your skills...  CSE-new LMS SJCE...  Log in | MongoDB  Portfolio

```
array(4) { [0]=> string(3) "jan" [1]=> string(3) "feb" [2]=> string(5) "march" [4]=> string(3) "may" }
```


- You need to find whether an array is subset of another array.
- Let us suppose that there are two arrays.
- First array is large which have 6 values.
- Second array is small which have 2 values.
- Find if second array is subset of first which means that all values of second array should exists in first array.


```
<?php
// Define two array
$array1 = array('a','1','2','3','4');
$array2 = array('a','3');


// intersect/matched values should be equal to first array
if (array_intersect($array2, $array1) === $array2) {
    echo "<br/>It is a subset<br/>";
} else {
    echo "Not a subset";
}
?>
```



localhost/dashboard/smith/test3.php

 Paatshala: Log in to...

 Advance your skills...

 CSE-new LMS

It is a subset

# PHP Program to check whether an array is empty or not

- An empty array can sometimes cause software crash or unexpected outputs.
- To avoid this, it is better to check whether an array is empty or not .
- There are various methods and functions available in PHP to check whether the defined or given array is an empty or not.



```
<?php
// Declare an array and initialize it
$non_empty_array = array('URL' => 'https://www.abc.com');
// Declare an empty array
$empty_array = array();
// Condition to check array is empty or not
if(!empty($non_empty_array))
    echo "Given Array is not empty <br>";
if(empty($empty_array))
    echo "Given Array is empty";
?>
```

localhost/dashboard/smith/test3.php



Advance your skills...



CSE-new LMS S.

Given Array is not empty

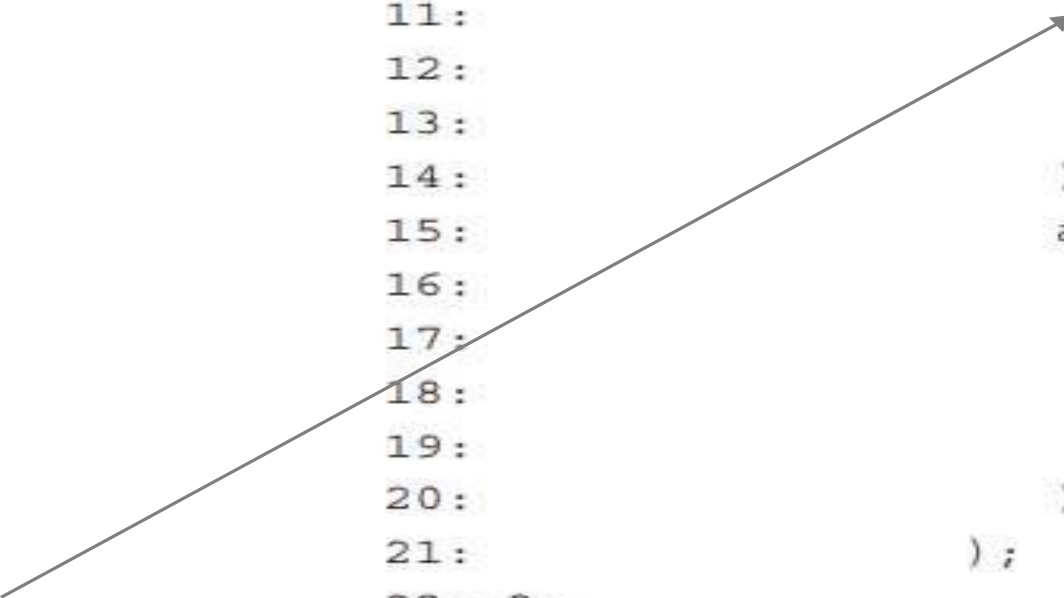
Given Array is empty

# MultiDimensional Arrays

- The first two types of arrays hold strings and integers, whereas this third type holds other arrays.
- If each set of key/value pairs constitutes a dimension, a multidimensional array holds more than one series of these key/value pairs.
- The \$characters array is initialized using the array() function.  
\$characters[0], \$characters[1]...

- Each element consists of an associative array, itself containing four elements: name, occupation, age, and special\_power
- `echo $characters[1]['occupation'];` will display “superhero”

```
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: 2;
```



# Multidimensional arrays

```
<?php
$characters = array (
    array("Volvo",22,18),
    array("BMW",15,13),
    array("Toyota",5,2),
    array("Maruthi",17,15)
);
for ($row = 0; $row < 4; $row++)
{
    for ($col = 0; $col < ; $col++)
    {
        echo "\t".$characters[$row][$col]."\t\t";
    }
    echo "<br/><br/>";
}
?>
```

---

Volvo 22 18

BMW 15 13

Toyota 5 2

Maruthi 17 15

# Functions

- Besides the built-in PHP functions, it is possible to create your own functions.
- A function is a **block of statements** that can be used repeatedly in a program.
- A **function will not execute automatically when a page loads.**
- **A function** will be executed by a **call** to the function.
- A user-defined function declaration starts with the word function.



- A function name must start with a **letter or an underscore**.
- Function names are **NOT case-sensitive**.
- They are **case-insensitive** for the ASCII characters A to Z
- All functions and classes in PHP have the **global scope** - they can be called outside a function even if they were defined inside and vice versa.
- PHP does not support **function overloading**, nor is it possible to undefine or redefine previously-declared functions.

# Cntd...

- 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.
- Functions come in two flavors: those built in to the language and those you define yourself.
- PHP has **hundreds of built-in functions**.
- **strtoupper()** returns a string value, so its usage requires the presence of a variable to accept the new string, such as the following


```
$new_string = strtoupper("Hello World!");  
echo $new_string;
```

HELLO WORLD!

- The number of **actual parameters** in a call to a function **does not need to match the number of formal parameters defined in that function**.
- If there are **too few actual parameters** in a call, the corresponding **formal parameters will be unbound variables**.
- If there are too many actual parameters, the excess parameters will be ignored.

- The default **scope** of a variable defined in a function is **local**.
- If a variable defined in a function has the same name as a variable used outside the function, there is no **interference** between the two.
- A local variable is visible only in the function in which it is used
- PHP has the **global declaration**.
- When a variable is listed in a global declaration in a function, that variable is expected to be defined outside the function.
- So, such a variable has the same meaning inside the function as outside.

- The default (that is, when storage for it is allocated) **lifetime of local variables in a PHP function is from the time the variable is first used** until the function's execution terminates.
- **The lifetime of a static variable in a function begins when the variable is first used in the first execution of the function.**
- Its lifetime ends when the script execution ends.

 test2.php

```
1    <?php
2    function writeMsg()
3    {
4        echo "Hello world!";
5    }
6    writeMsg();
7    // call the function
8
9    ?>
```





test2.php


```
1  <?php
2  function family($fname, $year)
3  {
4      echo "$fname . Born in $year <br>";
5  }
6  family("Smith", "1975");
7  family("Stalin", "1978");
8  family("Kai Jim", "1983");
9  ?>
```

localhost/dashboard/smith/test2.php

...  Advance your skills...  CSE-new LMS SJCE..

Smith . Born in 1975  
Stalin . Born in 1978  
Kai Jim . Born in 1983

- PHP is a Loosely Typed Language
- PHP automatically associates a data type to the variable, depending on its value.
- Since the data types are not set in a strict sense, you can do things like adding a string to an integer without causing an error.

 test2.php

```
1  <?php
2  function addNumbers(int $a, int $b) {
3      return $a + $b;
4  }
5  echo addNumbers(5, "5 days");
6  // since strict is NOT enabled "5 days"
7  // is changed to int(5), and it will return 10
8  ?>
```

- In PHP 7, type declarations were added.
- This gives us an option to specify the expected data type when declaring a function, and by adding the strict declaration, it will throw a "Fatal Error" if the data type mismatches.

- To specify strict we need to set `declare(strict_types=1);`.
- This must be on the very first line of the PHP file.
  - In the following example we try to send both a number and a string to the function, but here we have added the strict declaration: it will throw a "Fatal Error" if the data type mismatches

```
1 <?php declare(strict_types=1);
2 // strict requirement
3 function addNum(int $a, int $b) {
4     return $a + $b;
5 }
6 echo addNum(5, "5 days");
7 // since strict is enabled and "5 days"
8 // is not an integer, an error will be thrown
9 ?>
```

← → ↺ 🏠 ⓘ localhos... 🔗 ☆ 🧩 🖱️ Paused ⋮

 Paatshala: Log in to...  Advance your skills... »



**Fatal error:** Uncaught TypeError: addNumbers(): Argument #2 (\$b) must be of type int, string given, called in C:\xampp\htdocs\dashboard\smith\test2.php on line 6 and defined in C:\xampp\htdocs\dashboard\smith\test2.php:10 Stack trace: #0 C:\xampp\htdocs\dashboard\smith\test2.php(6): addNumbers(5, '5 days') #1 {main} thrown in **C:\xampp\htdocs\dashboard\smith\test2.php** on line 10



```
1 <?php declare(strict_types=1);  
2 // strict requirement  
3 function addNum(int $a, int $b) {  
4     return $a + $b;  
5 }  
6 echo "5+15=".addNum(5, 15);  
7 // since strict is enabled and "5 days"  
8 // is not an integer, an error will be thrown  
9 ?>
```

```
<?php declare(strict_types=1); // strict requirement  
function sum(int $x, int $y) {  
    $z = $x + $y;  
    return $z;  
}  
  
echo "5 + 10 = " . sum(5, 10) . "<br>";  
echo "7 + 13 = " . sum(7, 13) . "<br>";  
echo "2 + 4 = " . sum(2, 4);  
?>
```

localhost/dashboard/smith/test2.php

..  Advance your skills...  CSE-new LMS

$5+15=20$

localhost/dashboard/smith/test2.php

 Advance your skills...  CSE-new LMS

$5 + 10 = 15$

$7 + 13 = 20$

$2 + 4 = 6$



- PHP 7 also supports Type Declarations for the return statement.
- Like with the type declaration for function arguments, by enabling the strict requirement, it will throw a "Fatal Error" on a type mismatch.
- To declare a type for the function return, add a colon ( : ) and the type right before the opening curly ( { ) bracket when declaring the function.

```
1  <?php declare(strict_types=1); // strict requirement
2  function add(float $a, float $b) : float {
3      |   return $a + $b;
4      |   }
5  echo "sum of 1.2 +5.2 is".add(1.2, 5.2);
6  ?>
```

localhost/dashboard/smith/test2.php



Advance your skills...



CSE-new LMS

sum of  $1.2 + 5.2$  is  $6.4$

- Default parameter-passing mechanism of PHP is pass by value.
- The values of actual parameters are copied into the memory locations associated with the corresponding formal parameters in the called function.
- The values of the formal parameters are never copied back to the caller, so passing by value a one-way communication to the function.

```
<?php declare(strict_types=1);  
function max_abs($first, $second):int  
{  
    $first = abs($first);  
    $second = abs($second);  
    if ($first >= $second)  
        return $first;  
    else  
        return $second;  
}  
echo "Max of 100,87 is ".max_abs(100,87);  
?>
```

localhost/dashboard/smith/test2.php



Advance your skills...



CSE-new LMS

Max of 100,87 is 100

- Pass-by-reference parameters can be done in PHP in two ways.
  - One way is to add an ampersand (&) to the beginning of **the name of the formal parameter that you want to be passed by reference.**
  - The other way is to add an ampersand **to the actual parameter in the function call.**

- In PHP, arguments are usually passed by value, which means that a copy of the value is used in the function and the variable that was passed into the function cannot be changed



```
<?php
function add_five(&$value)
{
    $value += 98;
}
$num = 2;
add_five($num);
echo "Num=$num";
?>
```

localhost/dashboard/smith/test2.php



Advance your skills...



CSE-new LMS

Num=102

# 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.

- Creating object is so simple

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

# Create an instance of class as:

- Create an instance of class as:

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

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



a: blue Jeep Renegade

```
<?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;
?>
```


## 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: ?>
```

---

I drive a: red Porsche Boxster



# 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();
?>
```

## 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
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();
?>
```

# Object Inheritance

- One class inherits functionality from its parent class

```
2  <?php
3  class myClass {
4  | public $name;
5  function __construct($n)
6  | {
7  $this->name = $n;
8  | }
9  public function sayHello()
10 | {
11 |     echo "HELLO! My name is ".$this->name;
12 | }
13 | }
14 class childClass extends myClass {
15     public function sayHello1()
16     {
17         echo "<br/>i won't tell my name ".$this->name;
18         echo "<br/><br/>";
19     }
20 }
21 $object2 = new myClass("Benson");
22 $object2->sayHello();
23 $object1 = new childClass("Baby Benson");
24 $object1->sayHello1();
25 ?>
```

localhost/dashboard/smith/test3.php

..  Advance your skills...  CSE-new LMS SJCE...

HELLO! My name is Benson  
i won't tell my name Baby Benson

# String Comparisons

- Many string-processing tasks can be accomplished by using the equality and comparison operators,
- function **strcmp** to 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

```
2  <?php
3  // create array fruit
4  $fruits = array( "apple", "orange", "banana" );
5  // iterate through each array element
6  for ( $i = 0; $i < count( $fruits ); ++$i )
7  { // call function strcmp to compare the array element
8    // to string "banana"
9    if(strcmp($fruits[$i],"banana"<0))
10   print( "<p>" . $fruits[ $i ] . " is less than banana " );
11   elseif ( strcmp( $fruits[ $i ], "banana" ) > 0 )
12     print( "<p>" . $fruits[ $i ] . " is greater than banana " );
13     else
14       print( "<p>" . $fruits[ $i ] . " is equal to banana " );
15     // use relational operators to compare each element
16     // to string "apple"
17     if ( $fruits[ $i ] < "apple" )
18       print( "and less than apple!</p>" );
19     elseif ( $fruits[ $i ] > "apple" )
20       print( "and greater than apple!</p>" );
21     elseif ( $fruits[ $i ] == "apple" )
22       print( "and equal to apple!</p>" );
23   }
24  ?>
```



localhost/dashboard/smith/test3.php



Advance your skills...



CSE-new LMS SJCE...



apple is less than banana and equal to apple!

orange is less than banana and greater than apple!

banana is less than banana and greater than apple!

- PHP can process text easily and efficiently, enabling straightforward searching, substitution, extraction and concatenation of strings.
- **Text manipulation** is usually done with **regular expressions**—a series of characters that serve as pattern-matching templates (or search criteria) in strings, text files and databases.
- **A regular expression is a sequence of characters that forms a search pattern.**
- When you search for data in a text, you can use this search pattern to describe what you are searching for.

- **A *pattern* is a sequence of characters to be searched for in a character string.**
- In PHP, patterns are normally enclosed in slash characters.
- `/def/` represents the pattern `def`.
- If the pattern is found, a match occurs.
- For example, if you search the string `redefine` for the pattern `/def/`, the pattern matches the third, fourth, and fifth characters

# String Processing with Regular Expressions

- PHP includes two different kinds of string pattern matching using regular expressions:
- one that is based on **POSIX regular expressions** and one that is based on **Perl regular expressions**
- Function **preg\_match** uses regular expressions to search a string for a specified pattern using Perl-compatible regular expressions (PCRE).
- POSIX regular expressions are compiled into PHP
- `preg_match(regex, str)`- Returns a Boolean value
- Perl-Compatible Regular Expression (PCRE) library must be compiled before Perl regular expressions can be used
- `preg_split(regex, str)` - Returns an array of the substrings

# String Processing with Regular Expressions

- **The ^ and \$ Pattern Anchors**
- The pattern anchors ^ and \$ ensure that the pattern is matched only at the beginning or the end of a string.
- For example, the pattern /^def/ matches 'def' only if these are the first three characters in the string.
- Similarly, the pattern /def\$/ matches 'def' only if these are the last three characters in the string.
- You can combine ^ and \$ to force matching of the entire string, as follows:
- /^def\$/ This matches only if the string is def

- **preg\_match function**

- The preg\_match()function takes two parameters, the first of which is the Perl-style regular expression as a string.
- The second parameter is the string to be searched

```
1  <?php
2      $str="PHP is Hypertext Preprocessor";
3      if (preg_match("/^PHP/", $str))
4          print "\n$str begins with PHP <br />";
5      else
6          print "\n$str does not begin with PHP <br />";
7      ?>
8
9  <?php
```

localhost/dashboard/smith/test3.php



Advance your skills...



CSE-new LMS SJCE...



PHP is Hypertext Preprocessor begins with PHP



- The `preg_split` function operates on strings but returns an array and uses patterns
- `preg_split` takes two parameters, the first of which is a Perl-style pattern as a string. The second parameter is the string to be split.
- For example, consider the following sample code:
- **`$fruit_string = "apple : orange : banana";`**
- **`$fruits = preg_split("/ : /", $fruit_string);`**
- The array `$fruits` now has ( "apple", "orange", "banana").

Function	Description
<code>preg_match()</code>	Returns 1 if the pattern was found in the string and 0 if not
<code>preg_match_all()</code>	Returns the number of times the pattern was found in the string, which may also be 0
<code>preg_replace()</code>	Returns a new string where matched patterns have been replaced with another string

```

1  <?php
2      $str="today is friday.The time is Now";
3      if (preg_match("/Now/", $str))
4          print "\n$str contains Now <br/> <br />";
5      else
6          print "\n$str does not begin with Now <br/><br />";
7      if (preg_match("/^Now/", $str))
8          print "\n$str begins with Now <br/><br />";
9      else
10         print "\n$str does not begin with Now <br/><br />";
11     if (preg_match("/Now$/", $str))
12         print "\n$str ends with Now <br/> <br/>";
13     else
14         print "\n$str does not ends with Now <br />";
15     ?>

```

localhost/dashboard/smith/test3.php

Advance your skills... CSE-new LMS SJCE...

today is friday.The time is Now contains Now

today is friday.The time is Now does not begin with N

today is friday.The time is Now ends with Now

# Regular Expression Modifiers

Modifier	Description
i	Performs a case-insensitive search
m	Performs a multiline search
u	Enables correct matching of UTF-8 encoded patterns

# Regular Expression Patterns

---

Expression	Description
[abc]	Find one character from the options between the brackets
[^abc]	Find any character NOT between the brackets
[0-9]	Find one character from the range 0 to 9

# Quantifiers

Quantifiers define quantities:

Quantifier	Description
$n^+$	Matches any string that contains at least one $n$
$n^*$	Matches any string that contains zero or more occurrences of $n$
$n?$	Matches any string that contains zero or one occurrences of $n$
$n\{x\}$	Matches any string that contains a sequence of $X$ $n$ 's
$n\{x,y\}$	Matches any string that contains a sequence of $X$ to $Y$ $n$ 's
$n\{x, \}$	Matches any string that contains a sequence of at least $X$ $n$ 's

```
1  <?php
2  $str = "Apples and bananas.";
3  $pattern = "/ba(na){2}/i";
4  echo preg_match($pattern, $str);
5  // Outputs 1
6  $str = "Visit Microsoft!";
7  $pattern = "/\b([a-zA-Z]*ft)\b/i";
8  echo preg_match($pattern, $str);
9  //output 1 end with ft
10 ?>
```