What is PHP?

What is PHP?

- PHP is an acronym for "PHP: Hypertext Preprocessor"
- PHP is a widely-used, open source scripting language
- PHP scripts are executed on the server
- PHP is free to download and use

What is a PHP File?

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

What Can PHP Do?

- PHP can generate dynamic page content
- PHP can create, open, read, write, delete, and close files on the server
- PHP can collect form data
- PHP can send and receive cookies
- PHP can add, delete, modify data in your database
- PHP can be used to control user-access
- PHP can encrypt data
- With PHP you are not limited to output HTML. You can output images, PDF files, and even Flash movies. You can also output any text, such as XHTML and XML.

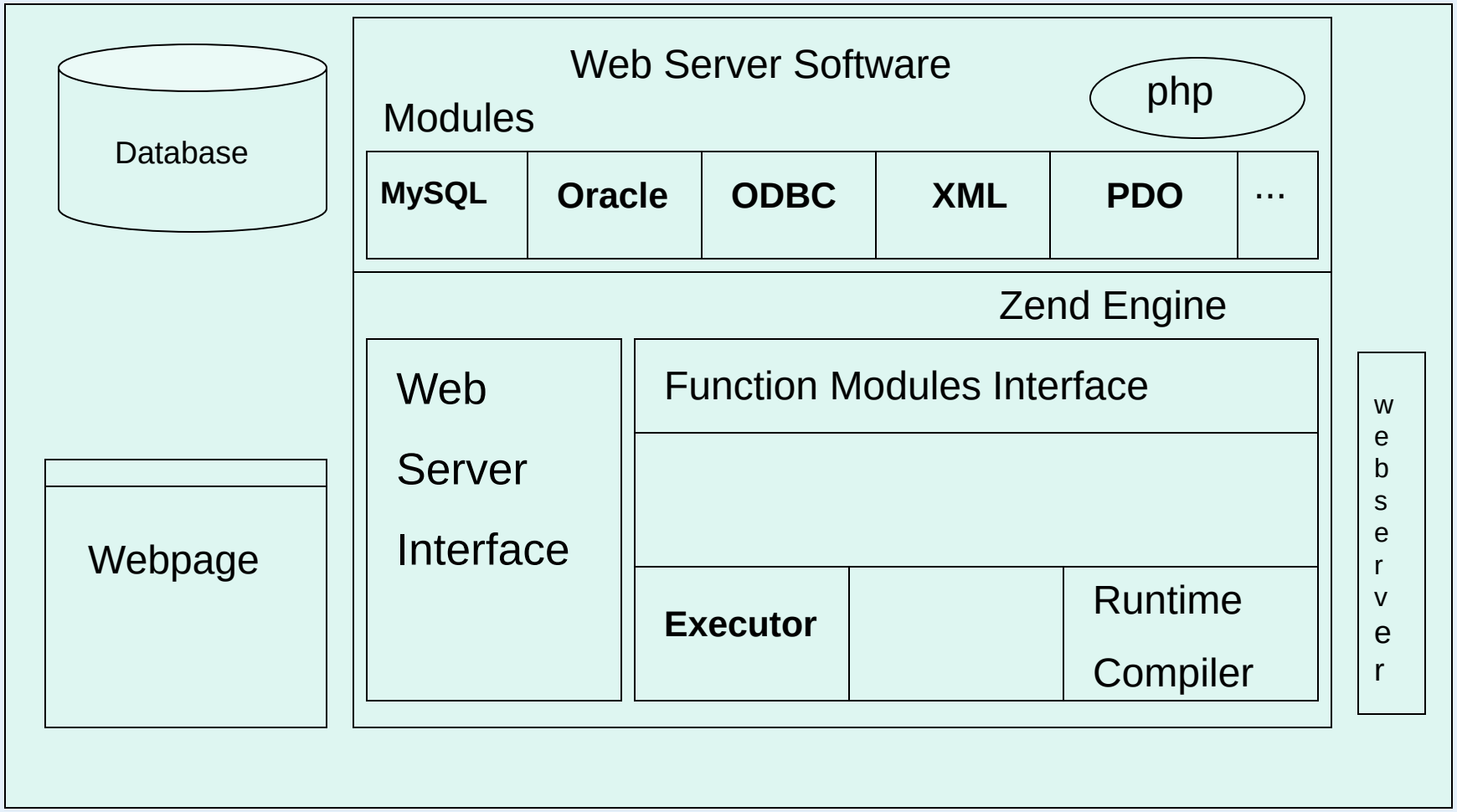
Why PHP?

- PHP runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)
- PHP is compatible with almost all servers used today (Apache, IIS, etc.)
- PHP supports a wide range of databases
- PHP is free. Download it from the official PHP resource: www.php.net
- PHP is easy to learn and runs efficiently on the server side
- PHP Help is available via excellent manuals

Zend & PHP

- Zend is PHP's central core.
- **Zend** Framework (ZF) is an open source, object-oriented web application framework implemented in **PHP** 5.
- The **Zend Engine** is the open source scripting **engine** that interprets the PHP programming language.
  - It was originally developed by Zeev Suraski and Andi Gutmans while they were students at the Technion - Israel Institute of Technology.
  - They later founded a company called **Zend** Technologies in Ramat Gan, Israel.

Internal Structure of PHP



PHP Installation

- To run PHP Script at any web server, three components are required :-
  1. An interpreter that analyzes PHP code snippets, translates them to something CPU understands, checks the translation for any kind of error & finally passes it to CPU for execution.
  2. A functionality section of the interpreter which implements functionality of PHP.
  3. An interface section that talks to the Web Server.

- Install a web server
- Install PHP
- Install a database, such as MySQL

PHP Installation

- PHP can be configured on IIS, WAMP and XAMPP servers.
- IIS is a web-server application just like Apache is, except it's made by Microsoft and is Windows only (Apache runs on both Windows and Linux).
- **XAMPP** is an acronym for **X** (any Operating System), **A**pache (Web server), **M**ySQL Database, **P**HP Language and **P**ERL.
- **WAMP** is an acronym for **W**indows (OS), **A**pache (web-server), **M**ySQL (database), **P**HP (language).
- **IIS**(Internet Information Server) is also more geared towards using ASP.NET (vs. PHP) and "SQL Server" (vs. MySQL), though it can use PHP and MySQL too.

XAMPP or WAMP?

- XAMPP is easy to use than WAMP. XAMPP is more powerful.
- XAMPP has a control panel from that you can start and stop individual components (such as MySQL, Apache etc.).
- XAMPP is more resource consuming than WAMP because of heavy amount of internal component software like Tomcat , FileZilla FTP server, Webalizer, Mercury Mail etc.
- So if you do not need high features better to go with WAMP. XAMPP also has SSL feature which WAMP doesn't.
- (Secure Sockets Layer (SSL) is a networking protocol that manages server authentication, client authentication and encrypted communication between servers and clients.)
- **Conclusion:** If your applications need to deal with native web apps only, Go for WAMP. If you need advanced features as stated above, go for XAMPP.

Syntax

- 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 statements end with a semicolon (;).
- Next we have an example of a simple PHP file, with a PHP script that uses a built-in PHP function "echo" to output the text "Hello World!" on a web page.

Example

```
<html>
<body>
<h1>My first PHP page</h1>

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

Comments in PHP

- A comment in PHP code is a line that is not read/executed as part of the program. Its only purpose is to be read by someone who is looking at the code.
- Comments can be used to:
  - Let others understand what you are doing
  - Remind yourself of what you did - Most programmers have experienced coming back to their own work a year or two later and having to re-figure out what they did.
  - Comments can remind you of what you were thinking when you wrote the code

Several ways of commenting

```
<html>
<body>

    <?php
      // This is a single-line comment

      # This is also a single-line comment

    /*
    This is a multiple-lines comment block
    that spans over multiple lines
    */

    // You can also use comments to leave out parts of a code line
    $x = 5 /* + 15 */ + 5;
    echo $x;
    ?>
</body>
</html>
```

Case Sensitivity

- In PHP, all keywords (e.g. if, else, while, echo, etc.), classes, functions, and user-defined functions are NOT case-sensitive.
- E.g.

```
<html>
<body>

<?php
ECHO "Hello World!<br>";
echo "Hello World!<br>";
EcHo "Hello World!<br>";
?>

</body>
</html>
```

Case Sensitivity

- However; all variable names are case-sensitive. E.g.

```
<html>
<body>

<?php
$color = "red";
echo "My car is " . $color . "<br>";
echo "My house is " . $COLOR . "<br>";
echo "My boat is " . $coLOR . "<br>";
?>

</body>
</html>
```

- Here only the first statement will display the value of the $color variable (this is because $color, $COLOR, and $coLOR are treated as three different variables)

Variables

- In PHP, a variable starts with the $ sign, followed by the name of the variable

```
<?php
    $txt = "Hello world!";
    $x = 5;
    $y = 10.5;
?>
```

- When you assign a text value to a variable, put quotes around the value.
- Unlike other programming languages, PHP has no command for declaring a variable. It is created the moment you first assign a value to it.

Variables

- A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume).
- Rules for PHP variables:
  - A variable starts with the $ sign, followed by the name of the variable
  - A variable name must start with a letter or the underscore character
  - A variable name cannot start with a number
  - A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
  - Variable names are case-sensitive ($age and $AGE are two different variables)

Output Variables

- The PHP echo statement is often used to output data to the screen.
- Example 1:

```
<?php
    $txt = “PHP learning";
    echo "I love $txt!";
?>
```

- Example 2:

```
<?php
    $txt = " Welcome to PHP learning ";
    echo "I love " . $txt . "!";
?>
```

One more example

- Sum of two variables

```
<?php
    $x = 5;
    $y = 4;
    echo $x + $y;
?>
```

Is a Loosely Typed Language

- We do not have to tell PHP which data type the variable is.
- PHP automatically converts the variable to the correct data type, depending on its value.
- In other languages such as C, C++, and Java, the programmer must declare the name and type of the variable before using it.

Testing type of variable

- To test the type of variable use gettype().
- string **gettype** (mixed $var)
- Mixed - $var can be of any type.
- Example :-

```
<?php
$testvar;
echo gettype($testvar);  // Null
echo "<br>";

$testvar = 15;
echo gettype($testvar); //integer
echo "<br>";

$testvar = "ABC";
echo gettype($testvar); //string
?>
```

Testing variable for specific type

Function	Description
is_int($var)	Returns true if $var is an integer
is_float($var)	Returns true if $var is a float
is_string($var)	Returns true if $var is a string
is_bool($var)	Returns true if $var is a boolean
is_array($var)	Returns true if $var is an array
is_object($var)	Returns true if $var is an object
is_resource($var)	Returns true if $var is a resource
is_null($var)	Returns true if $var is a null

Changing variable's datatype

- Using settype() :-
  - To use settype pass the name of variable which you want to change the type and new datatype
  - bool **settype** (mixed &$var , string $type)
- Using Type casting :-
  - PHP provides type conversion using cast operator

Operator	Changes To
int or integer	Integer
float, real, double	Floating point
String	String
bool or boolean	Boolean
array	Array
Object	Object

Changing variable's datatype

- PHP function's to cast a value

Function	Description
intval($var or value)	Returns value cast to integer
floatval($var or value)	Returns value cast to float
strval($var or value)	Returns value cast to string

Datatypes

- PHP has several different types of variables.
- All holds specific class/type of information.
- PHP has 8 basic data types which are categorized into 3 categories :-
  1. Scalar Datatype
  2. Compound Datatype
  3. Special Datatype

Scalar Data Type

- Scalar data means data that contains only a single value.

Data type	Description
Integer	-Stores whole numbers either +ve or – ve. -If value assigned is out of the range it is converted to float.
Float	- Store floating point numbers as well as higher integers.
String	- Stores sequence of characters terminated by NULL.
Boolean	-It holds true or false. -Internally it holds integer value - 0 for false - rest values for true.

Compound Data types

- Compound data is data that can contain more than one variable.

Data type	Description
Array	-Ordered map -It holds multiple values.
Object	-They have multiple values within it. -They have their own functions for accessing and/ or manipulating its data.

Special Datatype

- PHP supports two special data types which have a specific meaning.

Data types	Description
Resource	Contains a reference to an external resource, such as file or database. It can be used in same manner as we use another variables but only difference is it should be freed up when not required.
Null	Contains null as a value. Explicitly do not contain any value.

PHP Variables Scope

- In PHP, variables can be declared anywhere in the script.
- The scope of a variable is the part of the script where the variable can be referenced/used.
- PHP has three different variable scopes:
  - local
  - global
  - static

Global and Local Scope

- A variable declared **outside** a function has a GLOBAL SCOPE and can only be accessed outside a function
- E.g.

```
<?php
    $x = 5; // global scope

    function myTest() {
        // using x inside this function will generate an error
        echo "<p>Variable x inside function is: $x</p>";
    }
    myTest();

    echo "<p>Variable x outside function is: $x</p>";
?>
```

Cont...

- A variable declared **within** a function has a LOCAL SCOPE and can only be accessed within that function
- E.g.

```
<?php
    function myTest() {
        $x = 5; // local scope
        echo "<p>Variable x inside function is: $x</p>";
    }
    myTest();

    // using x outside the function will generate an error
    echo "<p>Variable x outside function is: $x</p>";
?>
```

- You can have local variables with the same name in different functions, because local variables are only recognized by the function in which they are declared.

The global Keyword

- The global keyword is used to access a global variable from within a function.
- To do this, use the global keyword before the variables (inside the function)
- E.g.

```
<?php
    $x = 5;
    $y = 10;

    function myTest() {
        global $x, $y;
        $y = $x + $y;
    }

    myTest();
    echo $y; // outputs 15
?>
```

The static Keyword

- When a function is completed/executed, all of its variables are deleted.
- To do this, use the **static** keyword when you first declare the variable.

```
<html> <body>
<?php
function myTest() {
    static $x = 0;
    echo $x;
    $x++;
}
myTest();    echo "<br>";
myTest();    echo "<br>";
myTest();    echo "<br>";
myTest();    echo "<br>";
myTest();
?>
</body> </html>
```

echo and print Statements

- In PHP there are two basic ways to get output: echo and print.
- We use echo (and print) in almost every example.
- echo and print are more or less the same. They are both used to output data to the screen.
- The differences are small:
  - 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.

The echo Statement

- The echo statement can be used with or without parentheses: echo or echo().
- Example 1

```
<?php
    echo "<h2>PHP is Fun!</h2>";
    echo "Hello world!<br>";
    echo "I'm about to learn PHP!<br>";
    echo "This ", "string ", "was ", "made ", "with multiple parameters.";
?>
```

The echo Statement

- Example 2

```
<?php
    $txt1 = "Learn PHP";
    $txt2 = “Hello World";
    $x = 5;
    $y = 4;

    echo "<h2>$txt1</h2>";
    echo "Study PHP at $txt2<br>";
    echo $x + $y;
?>
```

The print Statement

- The print statement can be used with or without parentheses: print or print().
- Example 1

```
<?php
    print "<h2>PHP is Fun!</h2>";
    print "Hello world!<br>";
    print "I'm about to learn PHP!";
?>
```

The print Statement

- Example 2

```
<?php
    $txt1 = "Learn PHP";
    $txt2 = " Welcome to PHP learning";
    $x = 5;
    $y = 4;

    print "<h2>$txt1</h2>";
    print "Study PHP at $txt2<br>";
    print $x + $y;
?>
```

Data Types

- PHP supports the following data types:
  - String
  - Integer
  - Float (floating point numbers - also called double)
  - Boolean
  - Array
  - Object
  - NULL
  - Resource

String

- A string is a sequence of characters, like "Hello world!".
- A string can be any text inside quotes. You can use single or double quotes:
- **Example**

```
<?php
    $x = "Hello world!";
    $y = 'Hello world!';

    echo $x;
    echo "<br>";
    echo $y;
?>
```

Integer

- An integer is a whole number (without decimals). It is a number between -2,147,483,648 and +2,147,483,647.
- Rules for integers:
  - An integer must have at least one digit (0-9)
  - An integer cannot contain comma or blanks
  - An integer must not have a decimal point
  - An integer can be either positive or negative
  - Integers can be specified in three formats: decimal (10-based), hexadecimal (16-based - prefixed with 0x) or octal (8-based - prefixed with 0)
- **Example**

```
<?php
$x = 5985;
var_dump($x);
?>
```

Float

- A float (floating point number) is a number with a decimal point or a number in exponential form.
- In the following example $x is a float. The PHP var_dump() function returns the data type and value:
- **Example**

```
<?php
$x = 10.365;
var_dump($x);
?>
```

- N.B. The **var_dump()** function is used to display structured information (type and value) about one or more variables.

Boolean

- A Boolean represents two possible states: TRUE or FALSE.

```
$x = true;
$y = false;
```

- Booleans are often used in conditional testing. You will learn more about conditional testing in a later chapter of this tutorial.

Array

- An array stores multiple values in one single variable.
- E.g.

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

- Here $cars is an array. The PHP var_dump() function returns the data type and value

Object

- An object is a data type which stores data and information on how to process that data.
- In PHP, an object must be explicitly declared.
- First we must declare a class of object. For this, we use the class keyword. A class is a structure that can contain properties and methods. E.g.

```
<?php
class Car {

    function Car() {

            $this->model = "VW";
    }
}
// create an object
$herbie = new Car();
// show object properties
echo $herbie->model;
?>
```

NULL Value

- Null is a special data type which can have only one value: NULL.
- A variable of data type NULL is a variable that has no value assigned to it.
- **Tip:** If a variable is created without a value, it is automatically assigned a value of NULL.
- Variables can also be emptied by setting the value to NULL

```
<?php
$x = "Hello world!";
$x = null;
var_dump($x);
?>
```

Strings

- A string is a sequence of characters, like "Hello world!".
- **Get The Length of a String**
  - The PHP strlen() function returns the length of a string (number of characters).
  - The example below returns the length of the string "Hello world!":

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

Cont...

- **Count The Number of Words in a String**
- The PHP str_word_count() function counts the number of words in a string:
- **Example**

```
<?php

echo str_word_count("Hello world!"); // outputs 2

?>
```

Cont...

- **Reverse a String**
- The PHP strrev() function reverses a string. E.g.

```
<?php
echo strrev("Hello world!"); // outputs !dlrow olleH
?>
```

- **Search For a Specific Text Within a String**
- The PHP strpos() function searches for a specific text within a string.
- If a match is found, the function returns the character position of the first match. If no match is found, it will return FALSE.
- E.g.

```
<?php
echo strpos("Hello world!", "world"); // outputs 6
?>
```

Cont...

- **Replace Text Within a String**
- The PHP str_replace() function replaces some characters with some other characters in a string.
- E.g.

```
<?php
echo str_replace("world", "Dolly", "Hello world!"); // outputs Hello Dolly!
?>
```

Constants

- A constant is an identifier (name) for a simple value whose value cannot be changed during the script.
- A valid constant name starts with a letter or underscore (no $ sign before the constant name).
- Unlike variables, constants are automatically global across the entire script. The syntax is as follows:
    - define(*name*, *value*, *case-insensitive*) where,
        - *name*: Specifies the name of the constant
        - *value*: Specifies the value of the constant
        - *case-insensitive*: Specifies whether the constant name should be case-insensitive. Default is false

Cont...

- Example 1

```
<?php
define("GREETING", "Welcome to PHP World!");
echo GREETING;
?>
```

- Example 2

```
<?php
   define("GREETING", "Welcome to PHP World!", true);
   echo greeting;
?>
```

Constants are Global

- Constants are automatically global and can be used across the entire script.
- **Example**

```
<?php
   function myTest() {
   define("GREETING", "Welcome to world of PHP!");
            echo GREETING;
   }
   myTest();
   echo GREETING;
?>
```

Differences between constants and variables

- 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.

Operators

- PHP divides the operators in the following groups:
    - Arithmetic operators
    - Assignment operators
    - Comparison operators
    - Increment/Decrement operators
    - Logical operators
    - String operators
    - Array operators

Arithmetic Operators

- The PHP arithmetic operators are used with numeric values to perform common arithmetical operations, such as addition, subtraction, multiplication etc.

Operator	Name	Example	Result
+	Addition	$x + $y	Sum of $x and $y
-	Subtraction	$x - $y	Difference of $x and $y
*	Multiplication	$x * $y	Product of $x and $y
/	Division	$x / $y	Quotient of $x and $y
%	Modulus	$x % $y	Remainder of $x divided by $y
**	Exponentiation	$x ** $y	Result of raising $x to the $y'th power (Introduced in PHP 5.6)

Assignment Operators

- The PHP assignment operators are used with numeric values to write a value to a variable.
- The basic assignment operator in PHP is "=". It means that the left operand gets set to the value of the assignment expression on the right.

Assignment	Same as...	Description
x = y	x = y	The left operand gets set to the value of the expression on the right
x += y	x = x + y	Addition
x -= y	x = x - y	Subtraction
x *= y	x = x * y	Multiplication
x /= y	x = x / y	Division
x %= y	x = x % y	Modulus

Comparison Operators

- The PHP comparison operators are used to compare two values (number or string)

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

Increment / Decrement Operators

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

Operator	Name	Description
++$x	Pre-increment	Increments $x by one, then returns $x
$x++	Post-increment	Returns $x, then increments $x by one
--$x	Pre-decrement	Decrements $x by one, then returns $x
$x--	Post-decrement	Returns $x, then decrements $x by one

Logical Operators

- The PHP logical operators are used to combine conditional statements.

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

String Operators

- PHP has two operators that are specially designed for strings.

Operator	Name	Example	Result
.	Concatenation	$txt1 . $txt2	Concatenation of $txt1 and $txt2
.=	Concatenation assignment	$txt1 .= $txt2	Appends $txt2 to $txt1

Array Operators

- The PHP array operators are used to compare arrays.

Operator	Name	Example	Result
+	Union	$x + $y	Union of $x and $y
==	Equality	$x == $y	Returns true if $x and $y have the same key/value pairs
===	Identity	$x === $y	Returns true if $x and $y have the same key/value pairs in the same order and of the same types
!=	Inequality	$x != $y	Returns true if $x is not equal to $y
<>	Inequality	$x <> $y	Returns true if $x is not equal to $y
!==	Non-identity	$x !== $y	Returns true if $x is not identical to $y

Conditional Statements

- if statement
- switch statement
- goto statement

If statement

- **Simple if**

```
if(test-expr)
{
True part
}
statement
```

If...else statement

```
if(test-expr)
{
True part
}
else
{
False part
}
```

Nested if statement

```
if(test-exp 1)
{
        if(test-exp 2)
        {
            true 1 & 2
        }
        else
        {
            false 2
        }
}
else
{
        false part all
}
```

else if ladder :-

```
if(test-exp 1)
        statement -1;
 else if(test-exp 2)
        statement – 2;
  else if(test-exp 3)
        statement – 3;
    .
     .
      .
     else
        false statement
```

switch statement :-

```
switch(ch)
{
        case value1:
          statement 1;break;
       case value2:
         statement 2;break;
       case value3:
          statement 3; break;
       .
       .
       .
        default :
           statement-false; break;
}
```

Case value can be of any data type or even any condition.

We can use break and continue interchangeable.

Default can be placed anywhere in switch.

The if Statement

- **Syntax**
  - if (*condition*) {
    code to be executed if condition is true;
    }
- The example below will output "Have a good day!" if the current time (HOUR) is less than 20:
- **Example**

```
<?php
  $t = date("H");

  if ($t < "20") {
    echo "Have a good day!";
  }
?>
```

The if...else Statement

- Use the if....else statement to execute some code **if a condition is true and another code if the condition is false.**
- **Syntax**

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

- **Example**

```
<?php
$t = date("H");
   if ($t < "20")
      echo "Have a good day!";
   else
      echo "Have a good night!";

?>
```

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

- Use the if....elseif...else statement to specify a new condition to .
- **Syntax**

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

E.g

```
<?php
$t = date("H");
if ($t < "10")
        echo "Have a good morning!";
elseif ($t < "20")
        echo "Have a good day!";
else
        echo "Have a good night!";
?>
```

The switch Statement

- Use the switch statement to select one of many blocks of code to be executed. Syntax:
- 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;
  }

Example

```
<?php
$favcolor = "red";
switch ($favcolor) {
    case "red":
                    echo "Your favorite color is red!";
                    break;
    case "blue":
                    echo "Your favorite color is blue!";
                    break;
    case "green":
                    echo "Your favorite color is green!";
                    break;
    default:
                    echo "Your favorite color is neither red, blue, or green!";
    }
?>
```

goto statement

- The *goto* operator can be used to jump to another section in the program. The target point is specified by a label followed by a colon, and the instruction is given as *goto* followed by the desired target label.
- The target label must be within the same file and context, meaning that you cannot jump out of a function or method, nor can you jump into one.
- You also cannot jump into any sort of loop or switch structure.

goto statement

Forward Jump :-

```
<?php
    goto label;
     statement -1
label:
      statement -2
    ?>
```

Example :

```
<?php
goto a;
echo "here";
a:
    echo "there";

?>
```

Backward Jump :-

```
<?php
label:
   statement -2
goto label;
 statement -1
?>
```

Example :

```
<?php
a:
            echo "there";
            exit;
goto a;
            echo "here";
?>
```

Loops

- In PHP, we have the following looping statements:
  - **while** - loops through a block of code as long as the specified condition is true
  - **do...while** - loops through a block of code once, and then repeats the loop as long as the specified condition is true
  - **for** - loops through a block of code a specified number of times
  - **foreach** - loops through a block of code for each element in an array

Loops

while loop

```
while (expr) statement;


while (expr): statement
   ...
endwhile;
```

do..while loop :-

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

for loop :-

```
for(intialization; condition; incr/decr)
{
        statement;
}
```

do..while loop is executed at least once whereas while and for loop executes only when condition is satisfied.

The while Loop

- The while loop executes a block of code as long as the specified condition is true.
- **Syntax**

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

- **Example**

```
<?php
   $x = 1;

   while($x <= 5) {
      echo "The number is: $x <br>";
      $x++;
   }
?>
```

The do...while Loop

- The do...while loop will always execute the block of code once, it will then check the condition, and repeat the loop while the specified condition is true.
- **Syntax**

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

- **Example**

```
<?php
  $x = 1;

  do {
     echo "The number is: $x <br>";
     $x++;
  } while ($x <= 5);
?>
```

Cont...

- **Example**

```
<?php
   $x = 6;

   do {
      echo "The number is: $x <br>";
      $x++;
   } while ($x<=5);
?>
```

The for Loop

- The for loop is used when you know in advance how many times the script should run.
- **Syntax**
    - for (*init counter; test counter; increment counter*) {
      code to be executed;
      }
- Parameters:
    - *init counter*: Initialize the loop counter value
    - *test counter*: Evaluated for each loop iteration. If it evaluates to TRUE, the loop continues. If it evaluates to FALSE, the loop ends.
    - *increment counter*: Increases the loop counter value

Cont...

```
<?php
   for ($x = 0; $x <= 10; $x++) {
       echo "The number is: $x <br>";
   }
?>
```

The Foreach Loop

- The foreach loop works only on arrays, and is used to loop through each key/value pair in an array.
- **Syntax**
  - foreach ($*array* as $*value*) {
    code to be executed;
    }
- **Example**

```
<?php
  $colors = array("red", "green", "blue", "yellow");

  foreach ($colors as $value) {
    echo "$value <br>";
  }
?>
```

foreach loop

Indexed Array :

```
<?php
$array = array(1,2,3,4);
foreach($array as $val) {
    print $val; }
?>
```

Associative Array :

```
<?php
$arr = array(1=>'ABC',2=>'PQR');
foreach ($arr as $key => $val) {
    print "$key = $val\n";
}
?>
```

Super Globals

- Super Globals are variables that are automatically available throughout all program code in all scopes.
- These variables do not require declaration and then too they can be accessed.
- Super global variables provide :-
  - Useful information about the environment.
  - Allow access to HTML form variables or parameters.
  - Access to cookies stored on a client.
  - Keeping track of sessions and file uploads.

PHP Super Globals

Name	Functionality
$GLOBALS	Contains all global variables in your script, including other super globals. This is not generally recommended for use, unless you are, for some reason, not sure where a variable will be stored.
$_GET	Contains all variables sent via a HTTP GET request. That is, sent by way of the URL.
$_POST	Contains all variables sent via a HTTP POST request.
$_FILES	Contains all variables sent via a HTTP POST file upload.
$_COOKIE	Contains all variables sent via HTTP cookies.
$_SESSION	Contains all variables stored in a user's session.
$_SERVER	Contains all variables set by the web server you are using, or other sources that directly relate to the execution of your script.

PHP Super Globals

Name	Functionality
$_REQUEST	Contains all variables sent via HTTP GET, HTTP POST, and HTTP cookies. This is basically the equivalent of combining $_GET, $_POST, and $_COOKIE, and is less dangerous than using $GLOBALS. However, as it does contain all variables from untrusted sources (that is, your visitors), you should still try to steer clear unless you have very good reason to use it.
$_ENV	Contains all environment variables set by your system or shell for the script.

- globals.php, ServerGlobal.php, RequestGlobal.php (includes the example for POST request also, GetGlobal.html/GetGlobal.php
- More details will be covered in User Requests and Form Handling.

Here Documents

- Here documents use special form of I/O redirection to feed a command list to an interactive program or command.
- To allow people to easily write large amounts of text from within PHP, but without the need to constantly escape things, heredoc syntax was developed.

Here Documents

```
<?php
    $mystring = <<<EOT
        This is some PHP text.
        It is completely free
        I can use "double quotes"
        and 'single quotes',
        plus $variables too, which will
        be properly converted to their values,
        you can even type EOT, as long as it
        is not alone on a line, like this:
    EOT;
  echo $mystring;
?>
```

Here Documents

- There are several key things to note about heredoc, and the example above:
  - You can use anything you like; "EOT" is just an example
  - You need to use <<< before the delimiter to tell PHP you want to enter heredoc mode
  - Variable substitution is used in PHP, which means you do need to escape dollar symbols - if you do not, PHP will attempt variable replacement.
  - You can use your delimiter anywhere in the text, but not in the first column of a new line
  - At the end of the string, just type the delimiter with no spaces around it, followed by a semi-colon to end the statement

Functions

- A function is a block of code that has a name and it has a property that it is reusable.
- Function groups a number of script statements into a unit and gives it a name.
- To keep the script from being executed when the page loads, you can put it into a function.
- A function will be executed by a call to the function.
- You may call a function from anywhere within a page.

Functions

- The real power of PHP comes from its functions; it has more than 1000 built-in functions.
- **PHP User Defined Functions**
    - Besides the built-in PHP functions, we can create our own functions.
    - A function is a block of statements that can be used repeatedly in a program.
    - A function will not execute immediately when a page loads.
    - A function will be executed by a call to the function.

Syntax for function

- function *functionName*($param1, $param2,...)
  {
  code to be executed;
  return $value;
  }
- Example :-

```
<?php
    function writeName()
    {
    echo “ABC";
    }
    echo "My name is ";
    writeName();
?>
```

Example : with parameter

```
<?php
function writeName($fname){
        echo $fname .
"Paul.<br />";
}
echo "My name is ";
writeName("Jim");
echo "My sister's name is ";
writeName("Agnes");
echo "My brother's name is ";
writeName("Steve");
?>
```

Create a User Defined Function in PHP

- A user defined function declaration starts with the word "function":
- **Syntax**

```
function functionName() {
    code to be executed;
}
```

- A function name can start with a letter or underscore (not a number). Function names are NOT case-sensitive.
- Give the function a name that reflects what the function does!
- **Example**

```
<?php
    function writeMsg() {
        echo "Hello world!";
    }
    writeMsg(); // call the function
?>
```

Function Arguments

- Information can be passed to functions through arguments. An argument is just like a variable.
- Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.
- **Example**

```
<?php
    function familyName($fname){
        echo "$fname Sharma.<br>";
    }
    familyName("Raj");
    familyName("Kavita");
    familyName("Pooja");
    familyName("Suyash");
    familyName("Vivaan");
?>
```

Cont...

- **Example**

```
<?php
   function familyName($fname, $year){
      echo "$fname Sharma. Born in $year <br>";
   }

   familyName("Pooja", "1975");
   familyName("Raj", "1978");
   familyName("Vivaan", "1983");
?>
```

Default Argument Value

- **Example**

```
<?php
    function setHeight($minheight = 50) {
        echo "The height is : $minheight <br>";
    }

    setHeight(350);
    setHeight(); // will use the default value of 50
    setHeight(135);
    setHeight(80);
?>
```

Functions - Returning values

- **Example**

```
<?php
    function sum($x, $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);
?>
```

The Date() Function

- The PHP date() function formats a timestamp to a more readable date and time.
- **Syntax**
    - date(*format*, *timestamp*)

Parameter	Description
format	Required. Specifies the format of the timestamp
timestamp	Optional. Specifies a timestamp. Default is the current date and time

Cont...

- Get a Simple Date
  - The required *format* parameter of the date() function specifies how to format the date (or time).
  - Here are some characters that are commonly used for dates:
    - » d - Represents the day of the month (01 to 31)
    - » m - Represents a month (01 to 12)
    - » Y - Represents a year (in four digits)
    - » l (lowercase 'L') - Represents the day of the week
  - Other characters, like"/", ".", or "-" can also be inserted between the characters to add additional formatting.

Cont...

- **Example**

```
<?php
    echo "Today is " . date("Y/m/d") . "<br>“;
    echo "Today is " . date("Y.m.d") . "<br>“;
    echo "Today is " . date("Y-m-d") . "<br>";
    echo "Today is " . date("l");
?>
```

- Date1.php

Get a Simple Time

- Here are some characters that are commonly used for times:
    - h - 12-hour format of an hour with leading zeros (01 to 12)
    - i - Minutes with leading zeros (00 to 59)
    - s - Seconds with leading zeros (00 to 59)
    - a - Lowercase Ante meridiem and Post meridiem (am or pm)
- The example below outputs the current time in the specified format:
- **Example**

```
<?php
   echo "The time is " . date("h:i:sa");
?>
```

- Date2.php

Get Your Time Zone

- If the time you got back from the code is not the right time, it's probably because your server is in another country or set up for a different time-zone.
- So, if you need the time to be correct according to a specific location, you can set a time-zone to use.
- The example below sets the time-zone to "Asia/Calcutta", then outputs the current time in the specified format:
- **Example**

```
<?php
  date_default_timezone_set("Asia/Calcutta");
  echo "The time is " . date("h:i:sa");
?>
```

- Date3.php

String Functions

- **substr() :-**This function returns the part of the string as an output.
- Syntax : -
- string substr(<string s>,<int start>,[<int length>]);
  - s: mandatory parameter. The string from which the part is to be extracted is mentioned here.
  - Start : The start in the string from which the characters are to be extracted
    - Positive number – Start at a specified position in the string
    - Negative number – Start at a specified position from the end of the string
    - 0 – Start at the first character in string
  - Length : It is an optional parameter. It specifies the length of the string which is to be extracted.
    - Positive number – The length to be returned from the start parameter
    - Negative number – The length to be returned from the end of the string

String Functions

- Examples:-

  <?php echo substr("Hello world",6); ?> //Returns world

  <?php echo substr("Hello world",6,4); ?> // Returns worl

  <?php echo substr("Hello world", -1); ?> // Returns d

  <?php echo substr("Hello world", -3, -1); ?> // Returns rl

- strlen() :-This function returns the length of the string.
- Syntax :- int strlen(<string s>);
  - string s: It is mandatory field. The string whose length is to be found out is mentioned here.

- Example :-

  <?php echo strlen("Hello world"); ?> // Returns 11

String Functions

- **trim() :-**This function removes the whitespaces from both start and the end of the string.
- Syntax :- string trim(<string s>);
  - string s: It is mandatory field. The string of which the whitespaces are to be removed is passed as parameter.
- Example :-

```
<?php echo trim( "        Hello World    "); ?>//Hello World
```

- **ltrim() :-** This function removes the whitespaces from the left part of the string.
- Syntax:- string ltrim(<string s>);
- Example :-

```
<?php echo ltrim( "        Hello World    "); ?>
            // returns Hello World
```

String functions

- **rtrim():-** This function removes the whitespaces from the right part of the string.
- Syntax :- string rtrim(<string s>);
  - string s: It is mandatory field. The string of which the whitespaces are to be removed from right side is passed as parameter.
- Example :-

```
<?php echo rtrim( "        Hello World    "); ?>//        Hello World
```

- **strtolower():-** This function converts the string to lower case
- Syntax : string strtolower(<string s>);
  - String s : It is mandatory field. The string which is to be converted to lower case is passed here.
- Example :-

```
<?php echo strtolower("HELLO WORLD"); ?>
// Returns hello world
```

String functions

- **strtoupper():-** This function converts the string to upper case
- Syntax :strtoupper(<string s>);
  - string s: It is mandatory field. The string which is to be converted to upper case is passed here.
- Example :-

```
<?php echo strtoupper("hello world"); ?>
// Returns HELLO WORLD
```

- **strcmp():-** The strcmp() function compares two strings.
- This function returns:
  - 0 – if the two strings are equal
  - <0 – if string1 is less than string2
  - >0 – if string1 is greater than string2
- Syntax : strcmp(<string1>,<string2>);
  - String1 and String 2 are mandatory.
- Example :-

```
<?php echo strcmp("Hello world!","Hello world!"); ?> // 0
```

String Functions

- **str_replace() :-**
- The str_replace() function replaces some characters with some other characters in a string.
- If the string to be searched is an array, it returns an array
- If the string to be searched is an array, find and replace is performed with every array element
- If both find and replace are arrays, and replace has fewer elements than find, an empty string will be used as replace
- If find is an array and replace is a string, the replace string will be used for every find value

String Functions

- Syntax

  str_replace(<search>,<replace>,<string/array>,[<count>]);

- Explanation :
  - Search : It is mandatory . The string or value to be searched comes here.
  - Replace : It is mandatory. The string or value to be replaced comes here.
  - String/Array : It is mandatory. The string or array in which the value is to be found out comes here.
  - Count : It is optional. It counts the number of replacements to be done.

Mathematical Functions

- PHP provides you wide set of mathematical functions.
- float **ceil** (float *value*) – rounds to nearest int above current value.
- float **floor** (float *value*) - rounds to the nearest int below current value.
- float **round** (float *value* [, int *precision*]) – rounds up the specified value as first parameter to the specified precision as second parameter.
- Trigonometric functions as:
  - float **sin** (float *value*)
  - float **cos** (float *value*)
  - float **tan** (float *value*)
  - float **asin** (float *value*)
  - float **acos** (float *value*)
  - float **atan** (float *value*)
  - float **deg2rad** (float *value*)
  - float **rad2deg** (float *value*)

Other Mathematical Functions

- number **abs** (number *value*)
- float **sqrt** (float *value*)
- number **pow** (number *base*, number *exponent*)
- int **bindec** (string *binary_string*)
- string **decbin** (int *number*)
- string **dechex** (int *number*)
- string **decoct** (int *number*)
- int **hexdec** (string *hex_string*)
- int **octdec** (string *octal_string*)
- string **base_convert** (string *number*, int *from_base*, int *to_base*)

Mathematical Constants

Constant	Meaning
M_PI	PI (Value : 3.14159265358979323846)
M_SQRTPI	Sqrt(M_PI) (Value :1.77245385090551602729)
M_SQRT2	Sqrt(2) Value : 1.414213562373095
M_SQRT3	Sqrt(3) Value : 1.732050807568877

Arrays

- An array stores multiple values in one single variable.
- **Example**

```
<?php
   $cars = array("Volvo", "BMW", "Toyota");
   echo "I like " . $cars[0] . ", " . $cars[1] . " and " . $cars[2] .
   ".";
?>
```

- **What is an Array?**
- An array is a special variable, which can hold more than one value at a time.
- If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:
- $cars1 = "Volvo";
  $cars2 = "BMW";
  $cars3 = "Toyota";

Create an Array

- In PHP, the array() function is used to create an array:
- array();
- In PHP, there are three types of arrays:
    - **Indexed arrays** - Arrays with a numeric index
    - **Associative arrays** - Arrays with named keys
    - **Multidimensional arrays** - Arrays containing one or more arrays

Indexed Arrays

- There are two ways to create indexed arrays:
- The index can be assigned automatically (index always starts at 0), like this:
- $cars = array("Volvo", "BMW", "Toyota");
- or the index can be assigned manually:

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

- E.g.

```
<?php
    $cars = array("Volvo", "BMW", "Toyota");
    echo "I like " . $cars[0] . ", " . $cars[1] . " and " . $cars[2] . ".";
?>
```

Get The Length of an Array - The count() Function

- The count() function is used to return the length (the number of elements) of an array:
- **Example:**

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

- **Loop Through an Indexed Array**

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

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

Associative Arrays

- Associative arrays are arrays that use named keys that you assign to them.
- There are two ways to create an associative array:
- $age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43"); **or:**
- $age['Peter'] = "35";
  $age['Ben'] = "37";
  $age['Joe'] = "43";
- The named keys can then be used in a script:
- **Example**

```
<?php
   $age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
   echo "Peter is " . $age['Peter'] . " years old.";
?>
```

Loop Through an Associative Array

- To loop through and print all the values of an associative array, you could use a foreach loop, like this:
- **Example**

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

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

Multidimensional Arrays

- A multidimensional array is an array containing one or more arrays.
- PHP understands multidimensional arrays that are two, three, four, five, or more levels deep.
- However, arrays more than three levels deep are hard to manage for most people.

Two-dimensional Arrays

- A two-dimensional array is an array of arrays (a three-dimensional array is an array of arrays of arrays).

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

- We can store the data from the table above in a two-dimensional array, like this:
- $cars = array (array("Volvo",22,18), array("BMW",15,13), array("Saab",5,2), array("Land Rover",17,15));

Cont...

- Now the two-dimensional $cars array contains four arrays, and it has two indices: row and column.
- To get access to the elements of the $cars array we must point to the two indices (row and column):
- **Example**

```
<?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>";
?>
```

Cont...

- We can also put a For loop inside another For loop to get the elements of the $cars array (we still have to point to the two indices):
- **Example**

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

Adding elements with [ ]

- The empty brackets add an element to the array.
- The element has a numeric key that's one more than the biggest numeric key already in the array.
- If the array doesn't exist yet, the empty brackets add an element with a key of 0.

Adding elements with [ ]

- Example

```
// Create $lunch array with two elements
// This sets $lunch[0]
$lunch[] = 'Chana masala';

// This sets $lunch[1]
$lunch[] = 'Chole bhature';

// Create $dinner with three elements
$dinner=array('Dum Aloo‘,'Gajar ka Halwa','Moong dal ka halwa');

// Add an element to the end of $dinner
// This sets $dinner[3]
$dinner[] = 'Palak Paneer';
```

Arrays

- Finding the size of an array
- echo count($cars);

Assigning a Range of Values

- range( ) function creates an array of consecutive integer or character values between the two values you pass to it as arguments.

  array range ($start , $limit [, number $step = 1])
- Parameters
  - Start - First value of the sequence.
  - Limit - The sequence is ended upon reaching the limit value.
- Step - If a step value is given, it will be used as the increment between elements in the sequence. step should be given as a positive number. If not specified, step will default to 1.
- rangeArray.php

Padding an Array

- To create an array initialized to the same value, use array_pad( ).

  array array_pad (array $input, int $pad_size mixed $pad_value)

  input - Initial array of values to pad.

  pad_size - New size of the array.

  pad_value-Value to pad if *input* is less than pad_size.

  Example

```
$scores = array(3, 10);
$padded = array_pad($scores, 5, 0);
// $padded is now array(3, 10, 0, 0, 0)
```

- If you want the new values added to the start of the array, use a negative second argument:

```
$padded = array_pad($scores, -5, 0);
```

- paddingArray.php

Multidimensional Array

- The values in an array can themselves be arrays. This lets you easily create multidimensional arrays:

```
$row_0 = array(1, 2, 3);
$row_1 = array(4, 5, 6);
$row_2 = array(7, 8, 9);
$multi = array($row_0, $row_1, $row_2);
```

- You can refer to elements of multidimensional arrays by appending more []s:

```
$value = $multi[2][0];
// row 2, column 0. $value = 7
```

Extracting Multiple Values

- To copy all of an array's values into variables, use the list( ) construct:

  list(*$variable, ...*) = *$array*;

- array's values are copied into the listed variables, in the array's internal order.

  Example :-

```
$person = array('name' => 'Fred', 'age' => 35, 'wife' => 'Betty');
list($n, $a, $w) = $person; // $n is 'Fred', $a is 35, $w is 'Betty'
```

- If you have more values in the list( ) than in the array, the extra values are set to NULL:

```
$values = array('hello', 'world');
list($a, $b, $c) = $values;
// $a is 'hello', $b is 'world', $c is NULL
```

Extracting Multiple Values

- If you have more values in the array than in the list( ), the extra values are ignored:

```
$person = array('name' => 'Fred', 'age' => 35, 'wife' => 'Betty');
list($n, $a) = $person; // $n is 'Fred', $a is 35
```

- Two or more consecutive commas in the list( ) skip values in the array:

```
$values = range('a', 'e');
list($m,,$n,,$o) = $values; // $m is 'a', $n is 'c', $o is 'e'
```

arrtoList.php

Iterator Functions

- Every PHP array keeps track of the current element you're working with; the pointer to the current element is known as the iterator.
- PHP has functions to set, move, and reset this iterator. The iterator functions are:

  current() : Returns the element currently pointed at by the iterator

  reset( ) : Moves the iterator to the first element in the array and returns it

  next( ) : Moves the iterator to the next element in the array and returns it

  prev(): Moves the iterator to the previous element in the array and returns it

  end() : Moves the iterator to the last element in the array and returns it

  each(): Returns the key and value of the current element as an array and moves the iterator to the next element in the array key( ) returns the key of the current element.

  The each( ) function is used to loop over the elements of an array. It processes elements according to their internal order.

  iteratorArray.php

Slicing an array

- To extract only a subset of the array, use the array_slice( ) function:

  $subset = array_slice(*array*, *offset*, *length*);
- The array_slice( ) function returns a new array consisting of a consecutive series of values from the original array.
- The *offset* parameter identifies the initial element to copy (0 represents the first element in the array), and the *length* parameter identifies the number of values to copy.
- The new array has consecutive numeric keys starting at 0.

Keys

- The array_keys( ) function returns an array consisting of only the keys in the array, in internal order:
- $array_of_keys = array_keys(*array*);
- Example:

```
$person = array('name' => 'Fred', 'age' => 35, 'wife' => 'Wilma');
$keys = array_keys($person);
// $keys is array('name', 'age', 'wife')
```

Key_values.php

Values

- PHP also provides a function to retrieve an array of just the values in an array, array_values( ):
- $array_of_values = array_values(*array*);
- Example:

```
$person = array('name' => 'Fred', 'age' => 35, 'wife' => 'Wilma');
$values = array_values($person); // $values is array('Fred', 35, 'Wilma');
```

Creating variables from array

- The extract( ) function automatically creates local variables from an array. The indexes of the array elements are the variable names

```
$person = array('name' => 'Fred', 'age' => 35, 'wife' => 'Betty');
extract($person); // $name, $age, and $wife are now set
print $name;
```

- If a variable created by the extraction has the same name as an existing one, the extracted variable overwrites the existing variable.

```
$shape = "round";
$array = array("cover" => "bird", "shape" => "rectangular");
extract($array, EXTR_PREFIX_SAME, "book");
echo "Cover: $cover, Book Shape: $book_shape, Shape: $shape";
```

- exctractArray.php

Creating variables from array

- Syntax
- extract(*array, extract_rules, prefix*)

- *array* Required. Specifies the array to use *extract_rules* Optional. The extract() function checks for invalid variable names and collisions with existing variable names. This parameter specifies how invalid and colliding names are treated. Possible values are given in the next slide.

- *prefix* Optional. If EXTR_PREFIX_SAME, EXTR_PREFIX_ALL, EXTR_PREFIX_INVALID or EXTR_PREFIX_IF_EXISTS are used in the extract_rules parameter, a specified prefix is required.
  This parameter specifies the prefix. The prefix is automatically separated from the array key by an underscore character.

Creating variables from array

- EXTR_OVERWRITE - Default. On collision, the existing variable is overwritten
- EXTR_SKIP - On collision, the existing variable is not overwritten
- EXTR_PREFIX_SAME - On collision, the variable name will be given a prefix
- EXTR_PREFIX_ALL - All variable names will be given a prefix
- EXTR_PREFIX_INVALID - Only invalid or numeric variable names will be given a prefix
- EXTR_IF_EXISTS - Only overwrite existing variables in the current symbol table, otherwise do nothing
- EXTR_PREFIX_IF_EXISTS - Only add prefix to variables if the same variable exists in the current symbol table
- EXTR_REFS - Extracts variables as references. The imported variables are still referencing the values of the array parameter

Creating array from variables

- The compact( ) function is the complement of extract( ).
- Pass it the variable names to compact either as separate parameters or in an array.
- The compact( ) function creates an associative array whose keys are the variable names and whose values are the variable's values.
- Any names in the array that do not correspond to actual variables are skipped.
- $color = 'indigo'; $shape = 'curvy'; $floppy = 'none'; $a = compact('color', 'shape', 'floppy'); // or $names = array('color', 'shape', 'floppy'); $a = compact($names);

Searching for elements in array

- The in_array( ) function returns true or false, depending on whether the first argument is an element in the array given as the second argument:

  in_array(*to_find*, *array*)

- findArray.php

Sorting one array

Effect	Ascending	Descending	User-defined order
Sort array by values, then reassign indexes starting with 0	sort( ) (sort.php)	rsort( )	usort( ) (usort.php)
Sort array by values	asort( ) (asort.php)	arsort( )	uasort( )
Sort array by keys	ksort( ) (ksort.php)	krsort( )	uksort( ) (uksort.php)

Merging arrays

- array_merge( ) function intelligently merges two or more arrays:
  - $merged = array_merge(*array1*, *array2* [, *array ...*])
  - Indexed Array merge :-

```
$first = array('hello', 'world'); // 0 => 'hello', 1 => 'world'
$second = array('exit', 'here'); // 0 => 'exit', 1 => 'here'
$merged = array_merge($first, $second);
// $merged = array('hello', 'world', 'exit', 'here')
```

  - Associative Array merge :-

```
$first = array('bill' => 'clinton', 'tony' => 'danza');
$second = array('bill' => 'gates', 'adam' => 'west');
$merged = array_merge($first, $second);
// $merged = array('bill' => 'gates', 'tony' => 'danza', 'adam' => 'west')
```

  merge.php

Printing arrays

- print_r()
  - **print_r()** displays information about a variable in a way that's readable by humans.

  var_dump()
  - This function displays structured information about one or more expressions that includes its type and value.
  - Example :-

```
<?php
$a = array ('a' => 'apple', 'b' => 'banana', 'c' => array ('x', 'y', 'z'));
print_r ($a);
?>
Array ( [a] => apple [b] => banana [c] => Array ( [0] => x [1] => y [2] => z ) )
```

Cont...

- Example :-

```
<?php
    $a = array(1, 2, array("a", "b", "c"));
    var_dump($a);
?>
```

- array(3)
  { [0]=> int(1)
  [1]=> int(2)
  [2]=> array(3)
  { [0]=> string(1) "a"
  [1]=> string(1) "b"
  [2]=> string(1) "c" }
  }

explode()

- explode() takes a string and splits it into separate elements of an array using the argument provided as a delimiter.
  - explode(separator, string,limit)

Parameter	Description
separator	Required.Delimitor using which you can break string.
string	Required. The string to split
limit	Optional. Specifies the maximum number of array elements to return.

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

Implode()

- implode() traverses through the elements of an array and re-creates a single string.
- implode(separator, array)

Parameter	Description
separator	Optional. Specifies what to put between the array elements. Default is "" (an empty string)
Array	Required. The array to join to a string

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

- Output :- Hello World! Beautiful Day!