UNIT- I PHP

TOPICS:

Introduction to PHP
- Declaring Variables
- Data Types
- Operators
-
- Control Structures
- Functions
  Reading data from WEB form controls
  like text boxes, radio buttons, lists etc..
- Handling File Uploads
- Handling Sessions and Cookies

PHP INTRODUCTION

PHP started out as a small open source project that evolved as more and more people found out how useful it was. **Rasmus Lerdorf** unleashed the first version of PHP way back in **1994**.

PHP is a recursive acronym for **"PHP: Hypertext Preprocessor".**

-
- PHP is a **server side scripting language** that is embedded in HTML. PHP scripts are executed on the server

- It is used to manage dynamic content, databases, session tracking, even build entire e-commerce sites.
- PHP supports many databases (MySQL, Informix, Oracle, Sybase, Solid, PostgreSQL, Generic ODBC, Microsoft SQL Server , etc.)
- PHP is an open source software.
- PHP is pleasingly zippy in its execution, especially when compiled as an Apache module on the Unix side. The MySQL server, once started, executes even very complex queries with huge result sets in record-setting time.
- PHP supports a large number of major protocols such as POP3, IMAP, and LDAP.
- PHP is forgiving: PHP language tries to be as forgiving as possible.
- PHP **Syntax is C-Like**.

Common uses of PHP:
PHP performs system functions, i.e. from files on a system it can create, open, read, write, and close them. **The other uses of PHP are:**
- PHP can handle forms, i.e. gather data from files, save data to a file, thru email you can send data, return data to the user.
- You add, delete, and modify elements within your database thru PHP.
- Access cookies variables and set cookies.
- Using PHP, you can restrict users to access some pages of your website.
- It can encrypt data.

Characteristics of PHP:

- Simplicity
- Efficiency
- Security
- Flexibility
- Familiarity

All PHP code must be included inside one of the three special markup tags are recognized by the PHP Parser.

```
<?php PHP code goes here ?>

<?    PHP code goes here ?>
```

Most common tag is the **<?php...?>**

SYNTAX OVERVIEW:

Canonical PHP tags *The most universally effective PHP tag style is:*
<?php...?>

Short-open (SGML-style) tags *Short or short-open tags look like this:*
<?...?>

HTML script tags *HTML script tags look like this:*
<script language="PHP">...</script>

PHP - VARIABLE TYPES

The main way to store information in the middle of a PHP program is by using a **variable**. Here are the most important things to know about variables in PHP.

- A variable is used to store information.
- All variables in PHP are denoted with a leading dollar sign (**$**).
- The value of a variable is the value of its most recent assignment.
- Variables are assigned with the **= operator**, with the variable on the left-hand side and the expression to be evaluated on the right.
- Variables can, but do not need, to be declared before assignment.
- Variables used before they are assigned have default values.
- PHP does a good job of **automatically converting types from one to another** when necessary.

- PHP variables are Perl- like.

Syntax: $var_name = value;

Eg: creating a variable containing a string, and a variable containing a number:
<?php
$txt="HelloWorld!";
$x=16;
?>

PHP is a Loosely Typed Language:
- ✓ In PHP, a variable does not need to be declared before adding a value to it.
- ✓ You 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.

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

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:

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

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

Example
```php
<?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.

PHP 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):

Example

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

PHP also stores all global variables in an array called **$GLOBALS[*index*]**. The *index* holds the name of the variable. This array is also accessible from within functions and can be used to update global variables directly. The example above can be rewritten likethis:

Example

```php
<?php
$x = 5;
$y = 10;
function myTest() {
   $GLOBALS['y'] = $GLOBALS['x'] + $GLOBALS['y'];
}
myTest();
echo $y; // outputs 15
?>
```

PHP The static Keyword

Normally, when a function is completed / executed, all of its variables are deleted. However, sometimes we want a local variable NOT to be deleted. We need it fo r a further job.
To do this, use the **static** keyword when you first declare the variable:

Example

```php
<?php
function myTest() {
   static $x = 0;
   echo $x;
   $x++;
}
myTest();
myTest();
myTest();        ?>     Output: 0 1 2
```

Then, each time the function is called, that variable will still have the information it contained from the last time the function was called.
Note: The variable is still local to the function.

Variable Naming

Rules for naming a variable is-
- Variable names must begin with a letter or underscore character.
- A variable name can consist of numbers, letters, underscores but you cannot use characters like
    + , - , % , (,) . & , etc

There is no size limit for variables.

PHP - Data Types:

PHP has a total of **eight data types** which we use to construct our variables:

- **Integers:** are whole numbers, without a decimal point, like 4195.
- **Doubles:** are floating-point numbers, like 3.14159 or 49.1.   **Scalar types**
- **Booleans:** have only two possible values either true or false.
- **Strings:** are sequences of characters, like 'PHP supports string operations.'
- **Arrays:** are named and indexed collections of other values.
- **Objects:** are instances of programmer-defined classes.   **Compound types**
- **NULL:** is a special type that only has one value:NULL.
-
    Resources: are special variables that hold references to resources external   **Special types**

    to PHP (such as database connections).

The first four are simple types, and the next two (arrays and objects) are compound - the compound types can package up other arbitrary values of arbitrary type, whereas the simple types cannot.

PHP Integers

Integers are **primitive data types.** They are **whole numbers**, without a decimal point, like 4195. They are the simplest type. They correspond to simple whole numbers, both positive and negative {..., -2, -1, 0, 1, 2, ...}.
Integer can be in decimal (base 10), octal (base 8), and hexadecimal (base 16) format. Decimal format is the default, octal integers are specified with a leading 0, and hexadecimals have a leading 0x.
Ex: $v = 12345;

$var1 = -12345 + 12345;

notation.php
```
<?php
$var1 = 31; $var2 = 031; $var3 = 0x31;
 echo "$var1\n$var2\n$var3";     ?>
```
Output:

31
25
49

The default notation is the **decimal**. The script shows these three numbers in decimal. In Java and C, if an integer value is bigger than the maximum value allowed, integer overflow happens. PHP works differently. In PHP, the integer becomes a float number. Floating point numbers have greater boundaries. In 32bit system, an integer value size is four bytes. The maximum integer value is 2147483647.

boundary.php
```php
<?php
$var = PHP_INT_MAX;
 echo var_dump($var);
$var++;
echo var_dump($var);
 ?>
```
We assign a maximum integer value to the $var variable. We increase the variable by one. And we compare the contents.

Output:
```
int(2147483647)
float(2147483648)
```
As we have mentioned previously, internally, the number becomes a floating point value.

var_dump(): The PHP var_dump() function returns the data type and value.

PHP Doubles or Floating point numbers

Floating point numbers represent real numbers in computing. Real numbers measure continuous quantities like weight, height or speed. Floating point numbers in PHP can be larger than integers and they can have a decimal point. The size of a float is platform dependent.

We can use various syntaxes to create floating point values.

```php
<?php
 $a = 1.245;
$b = 1.2e3;
$c = 2E-10;
$d = 1264275425335735;
 var_dump($a);
var_dump($b);
var_dump($c);
var_dump($d);
 ?>
```

The **$d** variable is assigned a large number, so it is automatically converted to float type.

Output:
```
float(1.245)
float(1200)
float(2.0E-10)
float(1264275425340000)
```
This is the output of beside script

PHP Boolean

A Boolean represents two possible states: TRUE or FALSE.

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

Booleans are often used in conditional testing.

```php
<?php
 $male = False;
 $r = rand(0, 1);
 $male = $r ? True: False;
 if ($male) {
   echo "We will use name John\n";
} else {
   echo "We will use name Victoria\n";
}       ?>
```

The script uses a **random integer** generator to simulate our case. $r = **rand**(0, 1);

The **rand( )** function returns a random number from the given integer boundaries **0 or 1**.
$male = $r? True: False;

We use the ternary operator to set a $male variable. The variable is based on the random $r value. If $r equals to **1**, the $male variable is set to **True**. If $r equals to **0**, the $male variable is set to **False**.

PHP Strings

String is a data type representing textual data in computer programs. Probably the single most important data type in programming.

```php
<?php
 $a = "PHP ";
$b = 'PERL';
 echo $a . $b;  ?>
```

Output: PHP PERL

We can use single quotes and double quotes to create string literals.

The script outputs two strings to the console. The \n is a special sequence, a new line.

The escape-sequence replacements are −
- \n is replaced by the newline character
- \r is replaced by the carriage-return character
- \t is replaced by the tab character
- \$ is replaced by the dollar sign itself ($)
- \" is replaced by a single double-quote (")
- \\ is replaced by a single backslash (\)

The Concatenation Operator
There is only one string operator in PHP.
The concatenation operator (**.**) is used to put two string values together. To concatenate two string variables together, use the concatenation operator:

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

O/P: Hello Kalpana! What a nice day!

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**. The example below searches for the text "world" in the string "Hello world!":

Example

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

output: 6
Tip: The first character position in a string is 0 (not 1).

Replace Text within a String
The PHP **str_replace**() function replaces some characters with some other characters in a string. The example below replaces the text "world" with "Dolly"

Example

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

Output: Hello Kalpana!

The strlen() function:

The **strlen**() function is used to return the length of a string. Let's find the length of a string:

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

The output of the code above will be: 12

PHP Array

Array is a complex data type which handles a collection of elements. Each of the elements can be accessed by an index. An array stores multiple values in one single variable. In the following example $cars is an array. The PHP var_dump() function returns the data type and value:

Example

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

The **array keyword** is used to create a collection of elements. In our case we have names.
The print_r function prints human readable information about a variable to the console.

O/P: Array ([0] => Volvo [1] => BMW [2] => Toyota)
  array(3) { [0]=> string(5) "Volvo" [1]=> string(3) "BMW" [2]=> string(6) "Toyota" }

PHP 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:

Example

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

Output: VW

PHP NULL

NULL is a special data type that only has **one value: NULL**. To give a variable the NULL value, simply assign it like this −

Ex: $my_var = NULL;

The special constant NULL is capitalized by convention, but actually it is case insensitive; you could just as well have typed −

$my_var = null;

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:

Example1

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

PHP Resource

The special resource type is not an actual data type. It is the storing of a reference to functions and resources external to PHP. A common **example** of using the resource data type is a **database call**. Resources are handlers to opened files, database connections or image canvas areas. We will not talk about the resource type here, since it is an advanced topic.

constant() function

As indicated by the name, this function will return the value of the constant. This is useful when you want to retrieve value of a constant, but you do not know its name, i.e. It is stored in a variable or returned by a function.constant()example

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

Output: 50 50

Only scalar data (boolean, integer, float and string) can be contained in constants.

PHP - Operators:

What is Operator?

Simple answer can be given using expression 4 + 5 is equal to 9. **Here 4 and 5 are called operands and + is called operator.** PHP language supports following type of operators.

Arithmetic Operators	Assignment Operators
Increment/Decrement operators	Conditional (or ternary) Operators
Comparison Operators	String Operators
Logical (or Relational) Operators	Array Operators

Arithmetic Operators:

There are following arithmetic operators supported by PHP language:
Assume variable **A holds 10** and variable **B holds 20** then:

Operator	Description	Example
+	Adds two operands	$A + $B will give 30
-	Subtracts second operand from the first	$A - $B will give -10
*	Multiply both operands	$A *$B will give 200
/	Divide numerator by denumerator	$B / $A will give 2

%	Modulus Operator and remainder of after division	an integer	$B % $A will give 0
**	Exponentiation ($x to the $y'th power)		$A ** $B

Increment/Decrement operators

Operator	Description	Example
++	Increment operator, increases integer value by one	$A++ - 11 / ++$A
--	Decrement operator, decreases integer value by one	$A-- will give 9 / --$A

Comparison Operators:

There are following comparison operators supported by PHP language Assume variable A holds 10 and variable B holds 20 then:

Operator	Description	Example
==	Checks if the value of two operands are equal or not	($A==$B) is not true.
===	Identical(Returns true if $A is equal to $B, and they are of the same type)	$A === $B
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	($A != $B) is true.
<>	Returns true if $x is not equal to $y	$A <> $B
!==	Not identical (Returns true if $A is not equal to $B, or they are not of the same type)	$A !== $B
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	($A > $B) is not true.
<	Checks if the value of left operand is less (A < B) is true. Than the value of right operand, if yes then condition becomes true.	
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then returns true.	($A >= $B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	($A <= $B) is true.

Logical Operators:

There are following logical operators supported by PHP language
Assume variable A holds 10 and variable B holds 20 then:

Operator	Description	Example
and (or) &&	Called Logical AND operator. If both the operands are true then then condition becomes true.	($A and $B) is true. ($A && $B) is true.
or (or) ||	Called Logical OR Operator. If any of the two operands are non zero then then condition becomes true.	($A or $B) is true. ($A || $B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will makefalse.	!($A && $B) is false.

Assignment Operators:
There are following assignment operators supported by PHP language:

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	$C = $A + $B

+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	$C += $A is equivalent to $C = $C + $A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	$C -= $A is equivalent to $C = $C - $A
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	$C *= $A is equivalent to $C = $C * $A
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	$C /= $A is equivalent to $C = $C / $A
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	$C %= $A is equivalent to $C = $C % $A

Conditional Operator

There is one more operator called conditional operator. This first evaluates an expression for a true or false value and then execute one of the two given statements depending upon the result of the evaluation.

The conditional operator has this syntax:

Operator	Description	Example
? :	Conditional Expression	If Condition is true ? Then value X : Otherwise value Y

PHP String Operators

PHP has two operators that are specially designed for strings.

Operator	Description	Example
.	Concatenation	$txt1 . $txt2 (Concatenation of $txt1 and $txt2)
.=	Concatenation assignment	$txt1 .= $txt2 (Appends $txt2 to $txt1)

PHP Array Operators

The PHP array operators are used to compare arrays.

Operator	Description	Example
+	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)
!= or <>	Inequality	$x != $y or $x <> $y Returns true if $x is not equal to $y
!==	Non-identity	$x !== $y (Returns true if $x is not identical to $y)

Precedence of PHP Operators

Operator precedence determines the grouping of terms in an expression. This affects how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has higher precedence than the addition operator −

For example x = 7 + 3 * 2; Here x is assigned 13, not 20 because operator * has higher precedence than + so it first get multiplied with 3*2 and then adds into 7. **Ans:13**

Here operators with the highest precedence appear at the top of the table; those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

Category	Operator	Associativity
Unary	! ++ --	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=	Right to left

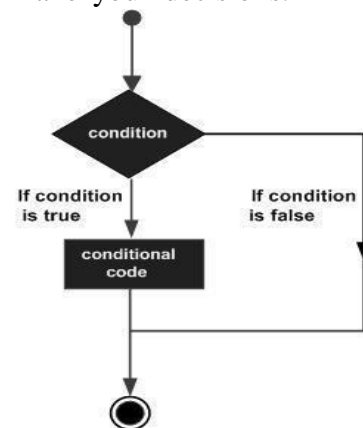|		
Ex:**<!DOCTYPE html>** **<html>** **<body>** **<?php** **$x = 10;** **$y = 6;** **echo $x + $y;** **?>** **</body>** **</html>** O/P: 16	**<!DOCTYPE html>** **<html>** **<body>** **<?php** **$x = 100;** **$y = 50;** **var_dump($x > $y); //** **returns true because $x is** **greater than $ y** **?>** O/P: **bool(true)**	**<!DOCTYPE html>** **<html>** **<body>** **<?php** **$x = 100;** **$y = 50;** **if ($x == 100 xor $y == 80) {** **echo "Hello world!";** **}** **?>** O/P: **Hello world!**

PHP -Decision Making

The if, elseif ...else and switch statements are used to take decision based on the different condition. You can use conditional statements in your code to make your decisions. PHP supports following three decision making statements –

• **if...else statement** − use this statement if you want to execute a set of code when a condition is true and another if the condition is not true

• **elseif statement** − is used with the if...else statement to execute a set of code if **one** of the several condition is true

• **switch statement** − is used if you want to select one of many blocks of code to be executed, use the Switch statement. The switch statement is used to avoid long blocks of if..elseif..else code.

Syntax	EX: <html>

if (*condition*)
  code to be executed if condition is true;
else
  code to be executed if condition is false;
	<body>
	<?php
	$d=date("D");
	    if ($d=="Fri")
	  echo "Have a nice weekend!";
	    else
	  echo "Have a nice day!";
	?>
	</body> </html>

Example The following example will output "Have a nice weekend!" if the current day is Friday, Otherwise, it will output "Have a nice day!":	

The If...Else Statement

If you want to execute some code if a condition is true and another code if a condition is false, use the if... else statement.

The elseif Statement

If you want to execute some code if one of the several conditions is true use the elseif statement

Syntax	EX: <html>

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

Example The following example will output "Have a nice weekend!" if the current day is Friday, and "Have a nice Sunday!" if the current day is Sunday. Otherwise, it will output "Have a nice day!"	

The Switch Statement

If you want to select one of many blocks of code to be executed, use the Switch statement. The switch statement is used to avoid long blocks of if..elseif..else code.

Syntax	Example

switch (*expression*)
{
  case *label1:*
    code to be executed if expression = label1;
    break;
  case *label2:*
    code to be executed if expression = label2;
    break;
  default:
  code to be executed
  if expression is different
  from both label1 and label2;
}
	The *switch* statement works in an unusual way. First it evaluates given expression then seeks a lable to match the resulting value. If a matching value is found then the code associated with the matching label will be executed or if none of the lable matches then statement will execute any specified default code.

PHP -Loop Types

Loops in PHP are used to execute the same block of code a specified number of times. PHP supports following four loop types.

- **for** − loops through a block of code a specified number of times.

- **while** − loops through a block of code if and as long as a specified condition is true.

- **do.. while** − loops through a block of code once, and then repeats the loop as long as a special condition is true.

- **foreach** − loops through a block of code for each element in an array.

We will discuss about **continue** and **break** keywords used to control the loops execution.

The for loop statement

The for statement is used when you know how many times you want to execute a statement or a block of statements.

Syntax

```
for (initialization; condition; increment)

{
  code to be executed;
```

The initializer is used to set the start value for the counter of the number of loop iterations. A variable may be declared here for this purpose and it is traditional to name it $i.

Example

The following example makes five iterations and changes the assigned value of two variables on eachpass
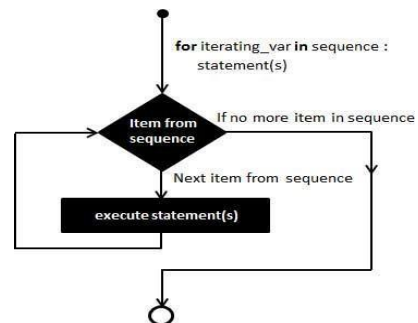` of the loop −

```
<html> <body>

    <?php
  $a = 0;
  $b = 0;
   for( $i=0; $i<5; $i++ )
  {   $a += 10;    $b += 5;
  }
      echo ("At the end of the loop a=$a and b=$b" );
```



```
for iterating_var in sequence :
          statement(s)
```

This will produce the following result −
At the end of the loop a=50 and b=25

The while loop statement

The while statement will execute a block of code if and as long as a test expression is true. If the test expression is true then the code block will be executed. After the code has executed the test expression will again be evaluated and the loop will continue until the test expression is found to befalse.

Syntax

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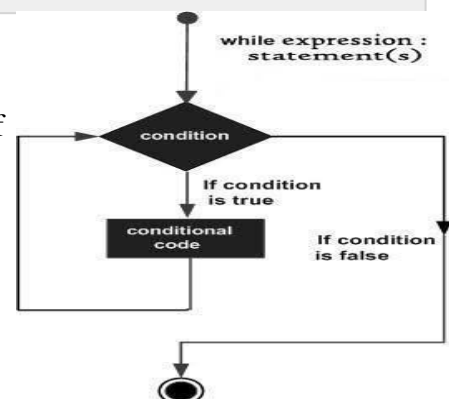
Example

This example decrements a variable value on each iteration of the loop and the counter increments until it reaches 10 when the evaluation is false and the loop ends.

```
<html>
  <body>
    <?php
    $i = 0;
```



```
while expression :
     statement(s)
```

```
    $num = 50;
     while( $i < 10)
    {
      $num--;
      $i++;
    }
    echo ("Loop stopped at i = $i and num = $num" );
  ?>   </body> </html>
```

This will produce the following result –

Loop stopped at i = 10 and num = 40

The do...while loopstatement

The do...while statement will execute a block of code at least once. It then will repeat the loop as long as a condition is true.

Syntax

```
do
{
   code to be executed;

}while (condition);
```

Example

The following example will increment the value of i at least once, and it will continue incrementing the variable i as long as it has a value of less than 10 −

```
<html>                                      }while( $i < 10 );
  <body>                                       echo ("Loop stopped at i = $i" );
    <?php                                    ?>
     $i = 0; $num = 0;                     </body>
    do{                                  </html>
       $i++;
```

O/P: Loop stopped at i = 10

| **The foreach loop statement**
 The foreach statement is used to **loop through arrays**. For each pass the value of the current array element is assigned to $value and the array pointer is moved by one and in the next pass next element will be processed.
 Syntax

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

 Example
 Try out beside example to list out the values of an array. | ```<html>```
 ``` <body>```
 ```   <?php```
 ``` $array = array(1, 2, 3, 4, 5);```
 ```   foreach($array as $value)```
 ```   {```
 ```   echo "Value is $value
";```
 ```   }```
 ```   ?>```
 ``` </body> </html>```
 This will produce the following result −

 Value is 1
 Value is 2
 Value is 3
 Value is 4
 Value is 5 |

The break statement

The PHP **break** keyword is used to terminate the execution of a loop prematurely. The **break** statement is situated inside the statement block. If gives you full control and whenever you want to exit from the loop you can come out. After coming out of a loop immediate statement to the loop will be executed.

Example
In the following example condition test becomes true when the counter value reaches 3 and loop terminates.

```php
<?php
$i = 0;
 while( $i < 10)        {
   $i++;
   if( $i == 3 )break;       }
 echo ("Loop stopped at i = $i" );
?> O/P: Loop stopped at i=3
```

The continue statement

The PHP **continue** keyword is used to halt the current iteration of a loop but it does not terminate the loop. Just like the **break** statement the **continue** statement is situated inside the statement block containing the code that the loop executes, preceded by a conditional test. For the pass encountering **continue** statement, rest of the loop code is skipped and next pass starts.
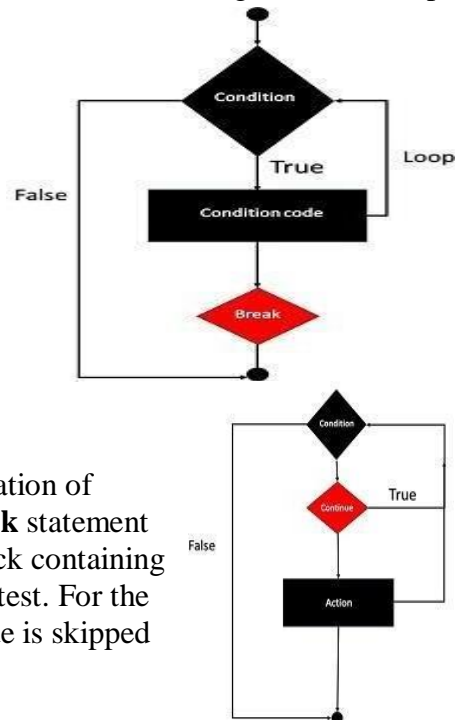
Example
In the following example loop prints the value of array but for which condition becomes true it just skip the code and next value is printed.

```html
<html>
  <body>
     <?php
     $nos = array( 1, 2, 3, 4, 5);
       foreach( $nos as $value )
     {
       if( $value == 3 )
         continue;
       echo "Value is $value <br />";
     }
      ?>
  </body>
</html>
```

This will produce the following result −

Value is 1
Value is 2
Value is 4
Value is 5

PHP -Functions

PHP functions are similar to other programming languages. A function is a piece of code which takes one more input in the form of parameter and does some processing and returns a value. You already have seen many functions like **fopen()** and **fread()** etc. They are built- in functions but PHP gives you option to create your own functions as well.
There are two parts which should be clear to you –
   - **Creating a PHP Function**
   - **Calling a PHP Function**

In fact you hardly need to create your own PHP function because there are already more than 1000 of built- in library functions created for different area and you just need to call them according to your requirement.

Creating PHP Function

It's very easy to create your own PHP function. Suppose you want to create a PHP function which will simply write a simple message on your browser when you will call it. Following example creates a function called writeMessage() and then calls it just after creating it.

```
<html> <head>
    <title>Writing PHP Function</title>
  </head>
  <body>
   <?php
     /* Defining a PHP Function */
     function writeMessage()
     {
```

```
     echo "Have a nice time Kalpana!";

     }    /* Calling a PHP Function */
     writeMessage();

   ?>
```

Output: Have a nice time Kalpana!

PHP Functions with Parameters

PHP gives you option to pass your parameters inside a function. You can pass as many as parameters you're like. These parameters work like variables inside your function. Following example takes two integer parameters and add them together and then print them.

```
<html>

  <head>      <title>Writing PHP Function with Parameters</title> </head>

  <body>

  <?php

     function addFunction($num1, $num2)

       {

         $sum = $num1 + $num2;
```

Output: Sum of the two numbers is : 30

Passing Arguments by Reference

It is possible to pass arguments to functions by reference. This means that a reference to the variable is manipulated by the function rather than a copy of the variable's value. Any changes made to an argument in these cases will change the value of the original variable. You can pass an argument by reference by adding an ampersand to the variable name in either the function call or the function definition.

```
<html>
    <head>
    <title>Passing Argument by Reference</title>
  </head>
  <body>
  <?php
     function addFive($num)
     {
```

```
        $num += 5;

    }

    function addSix(&$num)

    {

        $num += 6;

    }

    $orignum = 10;
    addFive( $orignum );

    echo "O riginal Value is $orignum<br/>";
```
Output: Original Value is 10

Original Value is 16

PHP Functions returning value

A function can return a value using the **return** statement in conjunction with a value or object. return stops the execution of the function and sends the value back to the calling code. You can return more than one value from a function using **returnarray(1,2,3,4)**.

```
<html>   <head>      <title>Writing PHP Function which returns value</title> </head>

    <body>

      <?php

        function addFunction($num1, $num2)

        {

            $sum = $num1 + $num2;
            return $sum;
```
Output: Returned value from the function : 30

Setting Default Values for Function Parameters

You can set a parameter to have a default value if the function's caller doesn't pass it. Following function prints NULL in case use does not pass any value to this function.

```
<html>   <head>      <title>Writing PHP Function which returns value</title></head>

    <body>

    <?php

        function printMe($param = NULL)

        {

            print $param;
```
Output: This is test

Dynamic Function Calls

It is possible to assign function names as strings to variables and then treat these variables exactly as you would the function name itself.

```html
<html>
 <head>
<title>Dynamic Function Calls</title>
</head>
  <body>
   <?php
     function sayHello()
     {
       echo "Hello<br />";
     }
     $function_holder = "sayHello";
     $function_holder();
   ?>   </body> </html>
```
Output: Hello

```html
<html>
 <head>
<title>Dynamic Function Calls</title>
</head>
  <body>
   <?php
     function add($x,$y)
     {
       echo "addition=" . ($x+$y);
     }
     $function_holder = "add";
     $function_holder(20,30);
   ?>   </body> </html>
```
Output: addition=50

PHP Default Argument Value

The following example shows how to use a default parameter. If we call the function setHeight() without arguments it takes the default value as argument:

Example

```php
<?php
function setHeight($minheight = 50) {
   echo "The height is : $minheight \t";
}
setHeight(350);
setHeight(); // will use the default value of 50
setHeight(135);
setHeight(80);
?>
```
O/P: 350   50   135   80

PHP -Web Concepts and Reading data from WEB

Identifying Browser & Platform

PHP creates some useful **environment variables** that can be seen in the **phpinfo.php** page that was used to setup the PHP environment. One of the environment variables set by PHP is **HTTP_USER_AGENT** which identifies the user's browser and operating system.

Using HTML Formwith name validation in PHP: test.php

```php
<?php
 if( $_POST["name"] || $_POST["age"] )
 {
   if (preg_match("/[^A- Za-z'-]/",$_POST['name'] ))
   {
     die ("invalid name and name  should be alpha");      }
   echo "Welcome ". $_POST['name']. "<br/>";
```

```
    echo "You are ". $_POST['age']. " years old.";
    exit();
  }   ?>
<html>
  <body>
  <form action="<?php $_PHP_SELF ?>" method="POST">
    Name: <input type="text" name="name" />
    Age: <input type="text" name="age" />
    <input type="submit" />
  </form> </body> </html>
```

- The PHP default variable **$_PHP_SELF** is used for the PHP script name and when you click "submit" button then same PHP script will be called and will produce followingresult

- The method = "POST" is used to post user data to the server script.

PHP Forms and User Input:

The PHP $_GET and $_POST variables are used to retrieve information from forms, like user input.

PHP - GET & POST Methods

There are two ways the browser client can send information to the web server.
  - The GET Method
  - The POST Method

Before the browser sends the information, it encodes it using a scheme called **URL encoding or URL Parameters**. In this scheme, name/value pairs are joined with equal signs and different pairs are separated by the ampersand.

name1=value1&name2=value2&name3=value3

The GET Method

The GET method sends the encoded user information appended to the page request. The page and the encoded information are separated by the **?**character.

http://www.sampletest.com/index.htm?name1=value1&name2=value2

- The GET method produces a long string that appears in your server logs, in the **browser's Location: box.**

- The GET method is restricted to send upto **1024 characters only.**

- Never use GET method if you have **password or other sensitive information** to be sent to the server.

- GET **can't be used to send binary data**, like images or word documents, to the server. The PHP
- provides **$_GET** associative array to access all the sent information using GET method.

```
<?php
  if( $_GET["name"] || $_GET["age"] )
  {
    echo "Welcome ". $_GET['name']. "<br />";
    echo "You are ". $_GET['age']. " years old.";
    exit();
  }
?> <html>  <body>
```

```
<form action="<?php $_PHP_SELF ?>" method="GET">
    Name: <input type="text" name="name" />
    Age: <input type="text" name="age" />
    <input type="submit" />
</form>  </body>   </html>
```

The POST Method

The POST method transfers information **via HTTP headers or HTTP Parameters**. The information is encoded as described in case of GET method and put into a header called QUERY_STRING.

- The POST method does not have any restriction on data size to be sent.
- The POST method can be used to send ASCII as well as binary data.
- The data sent by POST method goes through HTTP header so security depends on HTTP protocol. By using Secure HTTP you can make sure that your information is secure.
- The PHP provides **$_POST** associative array to access all the sent information using POST method.

PHP Form Handling

Example : The example below contains an HTML form with two input fields and a submit button:

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

When a user fills out the form above and click on the submit button, the form data is sent to a PHP file, called "**welcome.php**": its looks like this:

```
<html> <body>
Welcome <?php echo $_POST["fname"]; ?>!
You are <?php echo $_POST["age"]; ?> years old.
</body>
</html> Output: Welcome Kalpana! You are 29 years old.
```

$_GET is an array of variables passed to the current script via the **URL parameters.**
$_POST is an array of variables passed to the current script via the **HTTP POST method.**

PHP -File Uploading

A PHP script can be used with a HTML form to allow users to upload files to the server. Initially files are uploaded into a temporary directory and then relocated to a target destination by a PHPscript.

Information in the **phpinfo.php** page describes the temporary directory that is used for file uploads as **upload_tmp_dir** and the maximum permitted size of files that can be uploaded is stated as **upload_max_filesize**. These parameters are set into PHP configuration file **php.ini**

The process of uploading a file follows these steps −

- The user opens the page containing a HTML form featuring a text files, a browse button and a submit button.
- The user clicks the browse button and selects a file to upload from the local PC.
- The full path to the selected file appears in the text filed then the user clicks the submit button.
- The selected file is sent to the temporary directory on the server.
- The PHP script that was specified as the form handler in the form's action attr ibute checks that the file has arrived and then copies the file into an intended directory.
- The PHP script confirms the success to the user.

As usual when writing files it is necessary for both temporary and final locations to have permissions set that enable file writing. If either is set to be read-only then process will fail. An uploaded file could be a text file or image file or any document.

Creating an upload form

The following HTML code below creates an up loader form. This form is having method attribute set to **post** and **enctype attribute** is set to **multipart/form-data**

There is one global PHP variable called **$_FILES**. This variable is an associate double dimension array and keeps all the information related to uploaded file. So if the value assigned to the input's name attribute in uploading form was **file**, then PHP would create following five variables –

- **$_FILES['file']['tmp_name']** the uploaded file in the temporary dir on the web server.

- **$_FILES['file']['name']** – the actual name of the uploaded file.

- **$_FILES['file']['size']** – the size in bytes of the uploaded file.

- **$_FILES['file']['type']** – the MIME type of the uploaded file.

- **$_FILES['file']['error']** – the error code associated with this file upload.

Example: Below example should allow upload images and gives back result as uploaded file information.

```php
<?php
  if(isset($_FILES['image']))
  {
    $file_name =
$_FILES['image']['name'];
    $file_size =$_FILES['image']['size'];
    $file_type=$_FILES['image']['type'];
  }
?>
<html> <body>
    <form action="" method="POST"
enctype="multipart/form-data">
       <input type="file" name="image" />
       <input type="submit"/>
```

```php
    <ul>
       <li>Sent file: <?php echo
$_FILES['image']['name']; ?>
       <li>File size: <?php echo
$_FILES['image']['size'];  ?>
       <li>File type: <?php echo
$_FILES['image']['type'] ?>
    </ul>
    <img src=" <?php echo
$_FILES['image']['name']; ?>"
height="200" width="200" />
  </form>
 </body>
</html>
```

PHP -Cookies

Cookies are text files stored on the client computer and they are kept of use tracking purpose. PHP transparently supports HTTP cookies.

There are three steps involved in identifying returning users −

- Server script sends a set of cookies to the browser. For example name, age, or identification number etc.
- Browser stores this information on local machine for future use.
- When next time browser sends any request to web server then it sends those cookies information to the server and server uses that information to identify the user.

Setting Cookies with PHP

PHP provided **setcookie()** function to set a cookie. This function requires upto six arguments and should be called before <html> tag. For each cookie this function has to be called separately.

<div style="background-color:#d98880; padding:10px; text-align:center;">

setcookie(name, value, expire, path, domain, security);

</div>

Here is the detail of all the arguments −

- **Name** − This sets the name of the cookie and is stored in an environment variable called HTTP_COOKIE_VARS. This variable is used while accessing cookies.
- **Value** − This sets the value of the named variable and is the content that you actually want to store.
- **Expiry** − This specify a future time in seconds since 00:00:00 GMT on 1st Jan 1970. After this time cookie will become inaccessible. If this parameter is not set then cookie will automatically expire when the Web Browser is closed.
- **Path** − This specifies the directories for which the cookie is valid. A single forward slash character permits the cookie to be valid for all directories.
- **Domain** − This can be used to specify the domain name in very large domains and must contain at least two periods to be valid. All cookies are only valid for the host and domain which created them.
- **Security** − This can be set to 1 to specify that the cookie should only be sent by secure transmission using HTTPS otherwise set to 0 which mean cookie can be sent by regular HTTP.

Following example will create two cookies **name** and **age** these cookies will be expired after one hour.

```
<?php
  setcookie("name", "KALPANA", time()+3600, "/","", 0);
  setcookie("age", "36", time()+3600, "/", "", 0);
?>
<html>
  <head>      <title>Setting Cookies with PHP</title>  </head>
  <body>
    <?php echo "Set Cookies"?>
  </body>
  </html>
```

Accessing Cookies with PHP

PHP provides many ways to access cookies. Simplest way is to use either $_COOKIE or $HTTP_COOKIE_VARS variables. Following example will access all the cookies set in above example.

```html
<html>
    <head>      <title>Accessing Cookies with PHP</title> </head>
    <body>
     <?php
     echo $_COOKIE["name"]. "<br />";
       /* is equivalent to */
     echo $HTTP_COOKIE_VARS["name"]. "<br />";
     echo $_COOKIE["age"] . "<br />";
       /* is equivalent to */
     echo $HTTP_COOKIE_VARS["name"] . "<br />";
    ?>
    </body> </html>
```

You can use **isset()** function to check if a cookie is set or not.

```html
<html>   <head>      <title>Accessing Cookies with PHP</title> </head>
    <body>
    <?php
      if( isset($_COOKIE["name"]))
        echo "Welcome " . $_COOKIE["name"] . "<br />";
      else
        echo "Sorry... Not recognized" . "<br />";
    ?>
    </body> </html>
```

Deleting Cookie with PHP

Officially, to delete a cookie you should call setcookie() with the name argument only but this does not always work well, however, and should not be relied on.
It is safest to set the cookie with a date that has already expired –

```php
<?php
  setcookie( "name", "", time()- 60, "/","", 0);
  setcookie( "age", "", time()- 60, "/","", 0);
?>
<html>
 <head>      <title>Deleting Cookies with PHP</title> </head>
  <body>
    <?php echo "Deleted Cookies" ?>
  </body>   </html>
```

PHP - Session

- When you work with an application, you open it, do some changes, and then you close it. This is much like a Session.
- Session ID is stored as a cookie on the client box or passed along through URL's.
- The values are actually stored at the server and are accessed via the session id from your cookie. On the client side the session ID expires when connection is broken.
- Session variables solve this problem by storing user information to be used across multiple pages (e.g. username, favorite color, etc). By default, session variables last until the user closes the browser.
- Session variable values are stored in the 'superglobal' associative array '$_SESSION'

Start a PHP Session

A session is started with the **session_start()** function.

Session variables are set with the PHP global variable: **$_SESSION**.

demo_session1.php	Modify a PHP Session Variable

``php
<?php
// Start the session
session_start();
?>
<html>
<body>
<?php
// Set session variables
$_SESSION["favcolor"] = "green";
$_SESSION["favanimal"] = "cat";
echo "Session variables are set.";
?>
</body> </html>```
	To change a session variable, just overwrite it:
	
	``php
	<?php
	session_start();
	?>
	<html>
	<?php
	$_SESSION["favcolor"] = "yellow";
	print_r($_SESSION); ?> </html>```

Get PHP Session Variable Values

Next, we create another page called **"demo_session2.php".** From this page, we will access the session information we set on the first page **("demo_session1.php").**

Notice that session variables are not passed individually to each new page, instead they are retrieved from the session we open at the beginning of each page (session_start()).

```php
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>
<?php
// Echo session variables that were set on previous page
echo "Favorite color is " . $_SESSION["favcolor"] . ".<br>";
echo "Favorite animal is " . $_SESSION["favanimal"] . ".";
?>
</body>  </html>
```

Destroy a PHP Session

To remove all global session variables and destroy the session, use session_unset() and session_destroy():

Example
```php
<?php
session_start();
?>
<html>
<body>
<?php
session_unset();   // remove all session variables
session_destroy(); // destroy the session
?> </body> </html>
```

Difference between a session and a cookie

● The main **difference between** a **session** and a **cookie** is that **session** data is stored on the server, whereas **cookies** store data in the visitor's browser.

● **Sessions** are more secure than **cookies** as it is stored in server. **Cookie** can be turn off from browser.

	Sessions	Cookies
1	Sessions are **server-side** files that contain user information	Cookies are **client-side** files that contain user information
2	Session Max life time is 1440 Seconds(24 Minutes) as defined in php.ini file in **php.ini** on line 1604 you can find ; http://php.net/session.gc- maxlifetime session.gc_maxlifetime = 1440 You can edit this value if you need custom session life.	We have to set cookie max life time manually with php code with setcookie function. setcookie("email", 'test@example.com', time()+3600); /* expire in 1 hour */ Expire time : I hour after current time
3	In php **$_SESSION** super global variable is used to manage session.	In php **$_COOKIE** super global variable is used to manage cookie.
4	Before using $_SESSION, you have to write session_start(); In that way session will start	You don't need to start Cookie as It is stored in your local machine.
5	You can store as much data as you like within in sessions.(default is 128MB.) memory_limit= 128M php.ini line 479 ;http://php.net/memory- limit	Official MAX Cookie size is 4KB
6	Session is dependent on COOKIE. Because when you start session with session_start() then SESSIONID named key will be set in COOKIE with Unique Identifier Value for your system.	
7	**session_destroy();** is used to **"Destroys all data registered to a session"**, and if you want to unset some key's of SESSION then use unset() function. unset($_SESSION["key1"], $_SESSION["key2"])	There is no function named unsetcookie() time()-3600);//expire before 1hour In that way you unset cookie(Set cookie in previous time)
8	Session ends when user closes his browser.	Cookie ends depends on the life time you set for it.
9	A **session** is a group of information on the server that is associated with the cookie information.	**Cookies** are used to identify sessions.

Write a Program to create simple Login and Logout example using sessions.

login.php
```
<html>
<head>
<title>Login Form</title>
</head>
<body>
<h2>Login Form</h2>
<form method="post" action="checklogin.php">
User Id: <input type="text" name="uid"><br>
```

```
Password: <input type="password" name="pw"><br>
<input type="submit" value="Login">
</form>
</body>
</html>
```

checklogin.php

```php
<?php
$uid = $_POST['uid'];
$pw =$_POST['pw'];
if($uid == 'arun' and $pw == 'arun123')
{
        session_start();
        $_SESSION['sid
        ']=session_id();
        header("location
        :securepage.php
        ");
}
?>
```

securepage.php

```php
<?php
        session_start();
        if($_SESSION['s
        id']==session_id(
        ))
        {
echo "Welcome to you<br>";
echo "<a href='logout.php'>Logout</a>";
        }
        else
        {
                header("location:login.php");
        }
?>
```

logout.php

```php
<?php
echo "Logged
out
scuccessfully"
;
session_start()
;
session_destro
y();
setcookie(PHPSESSID,session_id(),time()-1); ?
```