UNIT-2

Programming with PHP: Conditional statements: if, if-else, switch, The ? Operator**,** Looping statements: while Loop, do-while Loop, for Loop

Arrays in PHP: Introduction- What is Array?, Creating Arrays, Accessing Array elements, Types of Arrays: Indexed v/s Associative arrays, Multidimensional arrays, Creating Array, Accessing Array, Manipulating Arrays, Displaying array, Using Array Functions, Including and Requiring Files- use of Include() and Require(), Implicit andn Explicit Casting in PHP.

DECISION MAKING

- ❖ PHP allows us to perform actions based on some type of conditionsthat may be logical or comparative.
- ❖ Based on the result of these conditions i.e., either TRUE or FALSE,an action would be performed.
- ❖ PHP provides us with four conditional statements:

1. if statement
2. if…else statement
3. if…elseif…else statement
4. switch statement

1. if Statement: The *if* statement is used to execute a block of code only if the specified condition evaluates to true. This is the simplest PHP's conditional statements and can be written like:

Syntax :

```
if (condition)
{
// if TRUE then execute this code
}
```

Example:

```
<?php              // if TRUE then execute this code
$x = 12; if ($x >
0)
{
echo "The number is positive";
}
?>
```

Output: The number is positive

2. **if…else Statement:** You can enhance the decision making process by providing an alternative choice through adding an *else* statement to the *if* statement. The *if...else* statement allows you to execute one block of code if the specified condition is evaluates to true and another block of code if it is evaluates to false. It can be written, like this:

Syntax:

```
if (condition)
{
                // if TRUE then execute this code
}
else
{

}               // if FALSE then execute this code
```

Example:

```
<?php
$x = -12;
if ($x > 0)
echo "The number is positive";
          else
                echo "The number is negative";
?>
```

Output: The number is negative

3. **if…elseif…else Statement:** he *if...elseif...else* a special statement that is used to combine multiple *if...else* statements.. We use this when there are multiple conditions of TRUE cases.

Syntax:

```
if (condition)
{
                // if TRUE then execute this code
}
elseif
{

}               // if TRUE then execute this code
elseif
{

}
else            // if TRUE then execute this code
{
```

```
// if FALSE then execute this code
}
```

Example:

```
<?php
$x = "August";
if ($x == "January")
{
echo "Happy Republic Day";
}
elseif ($x == "August")
{
            echo "Happy Independence Day!!!";
}
else
{

}           echo "Nothing to show";
?>
```

Output: Happy Independence Day!!!

4. **switch Statement:** The “switch” performs in various cases i.e., it has various cases to which it matches the condition and appropriately executes a particular case block. It first evaluates an expression and then compares with the values of each case. If a case matches then the same case is executed. To use switch, we need to get familiar with two different keywords namely, break and default.

The break statement is used to stop the automatic control flow intothe next cases and exit from the switch case.

The default statement contains the code that would execute if noneof the cases match.

Syntax:

```
switch(expression)
{
case value1:
code to be executed if n==statement1;break;
case value 2:
code to be executed if n==statement2;break;
case value 3:
code to be executed if n==statement3;break;
case value 4:
code to be executed if n==statement4;break;
……
default:
code to be executed if n != any case;
}
```

Example:

```
<?php
$n = "February";switch($n)
```

```
 {
   case "January": echo "Its January"; break;
case "February": echo "Its February";break;
case "March": echo "Its March";break;
case "April": echo "Its April";break;
case "May": echo "Its May";break;
default: echo "Doesn't exist";
}
 ?>
```

Output: Its February

LOOPS

- Loops are used to execute the same block of code again and again, until a certain condition is met. The basic idea behind a loop is to automate the repetitive tasks within a program to save the time and effort. PHP supports four different types of loops.

1. for loop
2. while loop
3. do-while loop
4. foreach loop

1. **for loop:** This type of loops is used when the user knows in advance, how many times the block needs to execute. These type of loops are also known as entry-controlled loops. There are three

main parameters to the code, namely the initialization, the testcondition and the counter.

Syntax:

```
for (initialization expression; test condition; update expression)
{
// code to be executed
}
```

In for loop, a loop variable is used to control the loop. Firstinitialize this loop variable to some value, then check whether this variable is less than or greater than counter value. If statement is true, then loop body is executed and loop variable gets updated. Steps are repeated till exit condition comes.

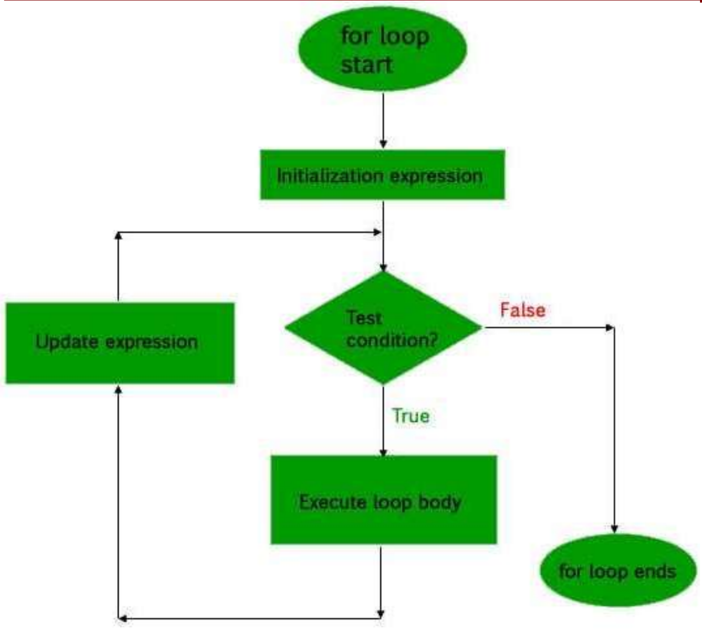
Example:

```
<?php
for ($num = 1; $num <= 10; $num += 2)
{
echo "$num \n";
}
?>
```

Output:

```
1    3    5    7    9
```

Flow Diagram:



1. **while loop:** The while loop is also an entry control loop like for loops i.e., it first checks the condition at the start of the loop and if its true then it enters the loop and executes the block of statements, and goes on executing it as long as the condition holds true.

Syntax:

```
        while (if the condition is true)
        {
                // code is executed
        }
```

Example:

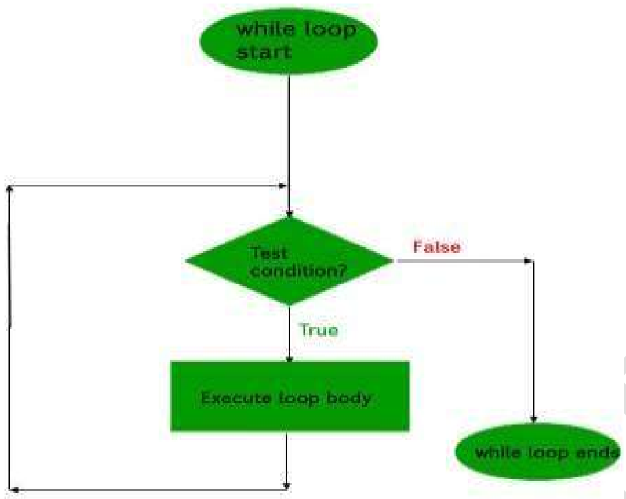
```
<?php
        $num = 2;
         while ($num < 12)
        {
                $num += 2;
                echo $num, "\n";
        }



?>
```

Output: 4 6 8 10 12

Flowchart



2. **do-while loop:** This is an exit control loop which means that it first enters the loop, executes the statements, and then checks the condition. Therefore, a statement is executed at least once on using the do…while loop. After executing once, the program is executedas long as the condition holds true.

Syntax:

```
        do
        {
                        //code is executed
              } while (if condition is true);
```

Example:

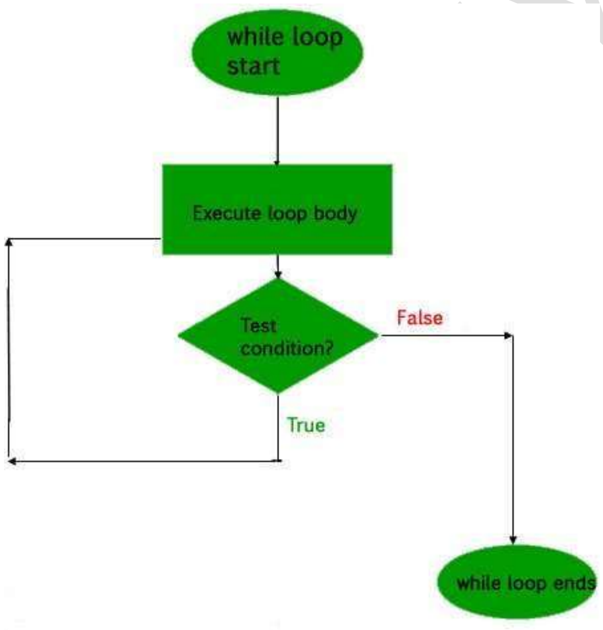
```
<?php
```

```
$num = 2;
do {
        {
            $num += 2;
            echo $num, "\n";
        } while ($num < 12);

?>
```

Output: 4 6 8 10 12

Flowchart:



1. **foreach loop:** 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_element as value)
   {
   ```

```
                //code to be executed
        }
```

Example:

```
<?php
        $arr = array (10, 20, 30, 40, 50, 60);
        foreach ($arr as $value)
        {
                echo "$val \n";
        }
                $arr = array ("Ram", "Laxman", "Sita");
        foreach ($arr as $value)
        {
                echo "$val \n";
        }
 ?>
```

Output:

```
10    20    30    40    50    60
Ram  Laxman     Sita
```

Break:

The PHP break keyword is used to terminate the execution of a loop prematurely. The break statement is placed inside the statement block. It 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
```

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

Output: Loop stopped at i = 3

Continue:

The PHP continue keyword is used to halt the current iteration of aloop but it does not terminate the loop.

Just like the break statement the continue statement is placed 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.

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

Output:

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

swapping Using Third Variable

```
<?php
$a = 45;
$b = 78;
// Swapping Logic
$third = $a;
$a = $b;
$b = $third;
echo "After swapping:<br><br>";echo
"a =".$a."  b=".$b;
?>
```

Swapping Without using Third Variable(+ and -):

```
<?php
$a=234;
$b=345;
//using arithmetic operation
$a=$a+$b;
$b=$a-$b;
$a=$a-$b;
echo "Value of a: $a</br>";
echo "Value of b: $b</br>";
?>
```

Example for (* and /):

```
<?php
$a=234;
$b=345;
// using arithmetic operation
$a=$a*$b;
$b=$a/$b;
$a=$a/$b;
echo "Value of a: $a</br>";echo
"Value of b: $b</br>";
?>
```

PHP script for generating a list of prime numbers below 100.

```
<?php
$number = 2 ;
while ($number < 100 )
{
        $div_count=0;
        for ( $i=1;$i<=$number;$i++)
        {
                if (($number%$i)==0)
                {
                        $div_count++;
                }
        }
        if ($div_count<3)
        {
                echo $number." , ";
        }
                $number=$number+1;
}
?>
```

output: 2 , 3 , 5 , 7 , 11 , 13 , 17 , 19 , 23 , 29 , 31 , 37 , 41 , 43 , 47 , 53 , 59 , 61 , 67 , 71 , 73 , 79 , 83 , 89 , 97 ,

Factorial of a number

```
<?php
$num = 4;
$factorial = 1;
for ($x=$num; $x>=1; $x--)
{
        $factorial = $factorial * $x;
}
echo "Factorial of $num is $factorial";
?>
```

Palindrome Number Program Without of Using PHP PredefinedFunction :

```
<?php
$number = 53235;
$p = $number;
$revnum =0;
while($number != 0)
{
      $revnum = $revnum*10 + $number % 10 ;
```

```
    $number = (int)($number/10);
}
if($revnum==$p)
        echo $p.' is palindrome number';
else
        echo 'number is not palindrome';
?>
```

PHP | Ternary Operator

ternary operator: The ternary operator (?:) is a conditional operator used to perform a simple comparison or check on a condition having simple statements. It decreases the length of the code performing conditional operations. The order of operation of this operator is from left to right. It is called a ternary operator because it takes three operands-a condition, a result statement for true, and a result statement for false. The syntax for the ternary operator is as follows.

Syntax:

(Condition) ? (Statement1) : (Statement2);

- **Condition:** It is the expression to be evaluated and returns a boolean value.
- **Statement 1:** It is the statement to be executed if the condition results in a true state.
- **Statement 2:** It is the statement to be executed if the condition results in a

false state.The result of this comparison can also be assigned to a variable using the assignment operator. The syntax is as follows:

Variable = (Condition) ? (Statement1) : (Statement2);

If the statement executed depending on the condition returns any value, it will be assignedto the variable.

Advantages of Ternary Operator: Following are some advantages of ternary operator:

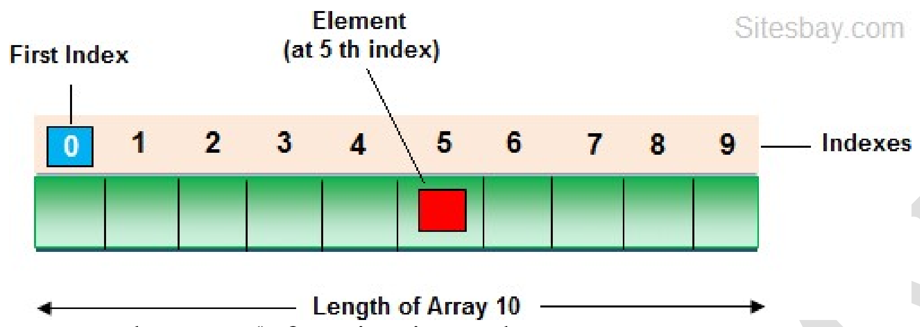
- The use of the ternary operator will make the code shorter in comparison to the IFELSE statement.
- The code can be quick in length in comparison to the IF ELSE statement.
- The readability of the code will increase with the usage of conditional statements.
- The use of the ternary operator makes the code simpler.

Example 1: In this example, if the value of $a is greater than 15, then 20 will be returned and will be assigned to $b, else 5 will bereturned and assigned to $b.

ARRAYS

Array is used to store multiple values of same type in single variable. An array is a special variable, which can hold more than one value ata time.

CREATE AN ARRAY IN PHP



In PHP, the array() function is used to create an array.

Syntax: **array( );**

TYPES OF ARRAY IN PHP

There are three types of array in PHP, which are given below.

1. Indexed arrays - Arrays with a numeric index
2. Associative arrays - Arrays with named keys
3. Multidimensional arrays - Arrays containing one or more arrays

1. Indexed Arrays

The index can be assigned automatically (index always starts at 0).

Example

```
<?php
```

```
        $student = array("Harry", "Varsha", "Gaurav");
        echo "Class 10th Students " . $student[0] . ", " . $student[1] . "and " .
$student[2] . ".";
?>
```

Output: Class 10th Students Harry, Varsha and Gaurav

Find Length of an Array in PHP

Using count() function you can find length of an array in php.

Example

```
<?php
        $student = array("Harry", "Varsha", "Gaurav");echo
        "Length of Array: ";
        echo count($student);
?>
```

Output: Length of Array: 3

Arrays using for Loop Example

```
<?php
        $student = array("Harry", "Varsha", "Gaurav");
        $arrlength = count($student); for($i =
        0; $i < $arrlength; $i++)
        {
                echo $student[$i];echo
                "<br>";
        }
?>
```

Output:

```
Harry
Varsha
```

Gaurav

2. Associative Arrays in PHP

In this type of array; arrays use named keys that you assign to them.

Syntax

```
$age = array("Harry"=>"10", "Varsha"=>"20", "Gaurav"=>"30");or
$age['Harry'] = "10";
$age['Varsha'] = "20";
$age['Gaurav'] = "30";
```

3. Multidimensional Arrays in PHP

A multidimensional array is an array containing one or more arrays. For a two-dimensional array you need two indices to select an element **Example**

```
<?php
        $student = array( array("Harry",300,11),
        array("Varsha",400,10),
                    array("Gaurav",200,8), array("Hitesh",220,8));


        echo $student[0][0].": Marks: ".$student[0][1].", Class:
        ".$student[0][2].".<br>";
        echo $student[1][0].": Marks: ".$student[1][1].", Class:
        ".$student[1][2].".<br>";
        echo $student[2][0].": Marks: ".$student[2][1].", Class:
        ".$student[2][2].".<br>";
        echo $student[3][0].": Marks: ".$student[3][1].", Class:
        ".$student[3][2].".<br>";
?>
```

Output:
```
Harry: Marks: 300 Class: 11
Varsha: Marks: 400 Class: 10
Gaurav: Marks: 200 Class: 8
Hitesh: Marks: 220 Class: 8
```

SORT FUNCTIONS FOR ARRAYS

1. sort() - sort arrays in ascending order

Example

```
<?php
        $cars = array("Volvo", "BMW", "Toyota");
        sort($cars);
        $clength = count($cars);
        for($x = 0; $x < $clength; $x++)
        {
                echo $cars[$x];
                echo "<br>";
        }
?>
```

Output:
```
BMW
Toyota
Volvo
```

2. rsort() - sort arrays in descending order

Example

```
<?php
        $cars = array("Volvo", "BMW", "Toyota");
        rsort($cars);
        $clength = count($cars);
        for($x = 0; $x < $clength; $x++)
```

```
            {
                echo $cars[$x];
                echo "<br>";
            }
    ?>
```

Output:

```
Volvo
ToyotaBMW
```

3. asort() - sort associative arrays in ascending order, according to thevalue

Example:

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

Output:

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

4. ksort() - sort associative arrays in ascending order, according to thekey

Example:

```
<?php
```

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

Output:

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

5. arsort() - sort associative arrays in descending order, according tothe value

Example:

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

Output:

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

6. krsort() - sort associative arrays in descending order, according tothe key

Example:

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

Output:

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

array_change_key_case()   Changes all keys in an array to lowercaseor uppercase

Syntax:

array_change_key_case(array $array[, int $case = CASE_LOWER])

Example

```
<?php
$salary=array("Sonoo"=>"550000","Vimal"=>"250000","Ratan"=>"2000 00");
print_r(array_change_key_case($salary,CASE_UPPER));
?>
```

Output:

```
Array ( [SONOO] => 550000 [VIMAL] => 250000 [RATAN] => 200000 )
```

array_chunk(): Splits an array into chunks of arrays**Syntax:**

array_chunk (array $array , int $size) **Example**

```
<?php
$salary=array("Sonoo"=>"550000","Vimal"=>"250000","Ratan"=>"2000 00");
print_r(array_chunk($salary,2));
?>
```

Output:

```
Array (
[0] => Array ( [0] => 550000 [1] => 250000 )
[1] => Array ( [0] => 200000 )
)
```

ARRAY FUNCTIONS

Function	Description
array()	Creates an array
array_combine()	Creates an array by using the elements from one "keys" array and one "values" array
array_count_values()	Counts all the values of an array
array_diff()	Compare arrays,and returns the differences(compare values only)
array_diff_assoc()	Compare arrays, and returns the differences (compare keys and values)
array_diff_key()	Compare arrays, and returns the differences (compare keys only)
array_fill()	Fills an array with values

array_fill_keys() Fills an array with values, specifying keys array_filter() Filters the values of an array using a callback function

array_flip() Flips/Exchanges all keys with their associatedvalues in an array

array_intersect() Compare arrays, and returns the matches (compare values only)

Example

```
<?php
$name1=array("sonoo","john","vivek","smith");
$name2=array("umesh","sonoo","kartik","smith");
$name3=array_intersect($name1,$name2); foreach(
$name3 as $n )
{
  echo "$n<br />";
}
?>
```

Output:

```
sonoo
smith
```

array_intersect_assoc() Compare arrays and returns the matches(compare keys and values)

array_intersect_key() Compare arrays, and returns the matches(compare keys only)

array_intersect_uassoc() Compare arrays, and returns the matches (compare keys and values, using a user-definedkey comparison function)

array_intersect_ukey()	Compare arrays, and returns the matches (compare keys only, using a user-defined keycomparison function)
array_key_exists()	Checks if the specified key exists in thearray
array_keys()	Returns all the keys of an array array_map() Sends each value of an array to a user-made function, which returns new values
array_merge()	Merges one or more arrays into one array
array_merge_recursive() recursively	Merges one or more arrays into one array
array_multisort()	Sorts multiple or multi-dimensional arrays
array_pad()	Inserts a specified number of items, with a specified value, to an array
array_pop()	Deletes the last element of an array array_product() Calculates the product of the values in an array array_push() Inserts one or more elements to the end ofan array
array_rand()	Returns one or more random keys from anarray
array_reduce() function	Returns an array as a string, using a user-defined
array_replace()	Replaces the values of the first array with thevalues from following arrays
array_replace_recursive()	Replaces the values of the first array with the values from following arrays recursively
array_reverse()	Returns an array in the reverse order

Example

```
<?php
$season=array("summer","winter","spring","autumn");
$reverseseason=array_reverse($season);foreach(
$reverseseason as $s )
{
  echo "$s<br />";
}
?>
```

Output:

```
autumn
spring winter
summer
```

array_search(): Searches an array for a given value and returns thekey

Example

```
<?php
$season=array("summer","winter","spring","autumn");
$key=array_search("spring",$season); echo
$key;
?>
```

Output:

```
2
```

array_shift()	Removes the first element from an array, and returns the value of the removed element
array_slice()	Returns selected parts of an array array_splice() Removes and replaces specified elements ofan array

array_sum() Returns the sum of the values in an array

Example:

```
<?php
$a = array(2, 4, 6, 8);
echo "sum(a) = " . array_sum($a) . "\n";
$b = array("a" => 1.2, "b" => 2.3, "c" => 3.4);
echo "sum(b) = " . array_sum($b) . "\n";
?>
```

Output:

```
sum(a) = 20
sum(b) = 6.9
```

array_unique()	Removes duplicate values from an array
array_unshift()	Adds one or more elements to the beginning of an array
array_values()	Returns all the values of an array
count()	Returns the number of elements in an array
current()	Returns the current element in an array
each()	Returns the current key and value pairfrom an array
end()	Sets the internal pointer of an array to its lastelement
extract()	Imports variables into the current symbol tablefrom an array
in_array()	Checks if a specified value exists in an arraykey() Fetches a key from an array
list()	Assigns variables as if they were an array
range()	Creates an array containing a range of elements

reset()	Sets the internal pointer of an array to its first element
shuffle()	Shuffles an array
sizeof()	Alias of count()
uksort()	Sorts an array by keys using a user-defined comparison function
usort()	Sorts an array using a user-defined comparison function

natsort(): The natsort() function is used to sorts an array using a "natural order" algorithm. The function implements a sort algorithm but maintains original keys/values.

This function implements a sort algorithm that orders alphanumeric strings in the way a human being would while maintaining key/value associations.

Syntax: natsort(array_name)

Example:

```
<?php
$php_files = array("code12.php", "code22.php","code2.php",
"Code3.php", "code1.php"); natsort($php_files);
echo "List of file using natural order: ";print_r($php_files);
?>
```

Output: List of file using natural order:

```
Array (
    [3] => Code3.php
    [4]  =>  code1.php
    [2]  =>  code2.php
    [0]  =>  code12.php
    [1]  =>  code22.php
)
```

natcasesort(): The natcasesort() function is used to sort an array using a case insensitive "natural array" algorithm. The function implements a sort algorithm but maintains original keys/values.

natcasesort() is a case insensitive version of natsort()

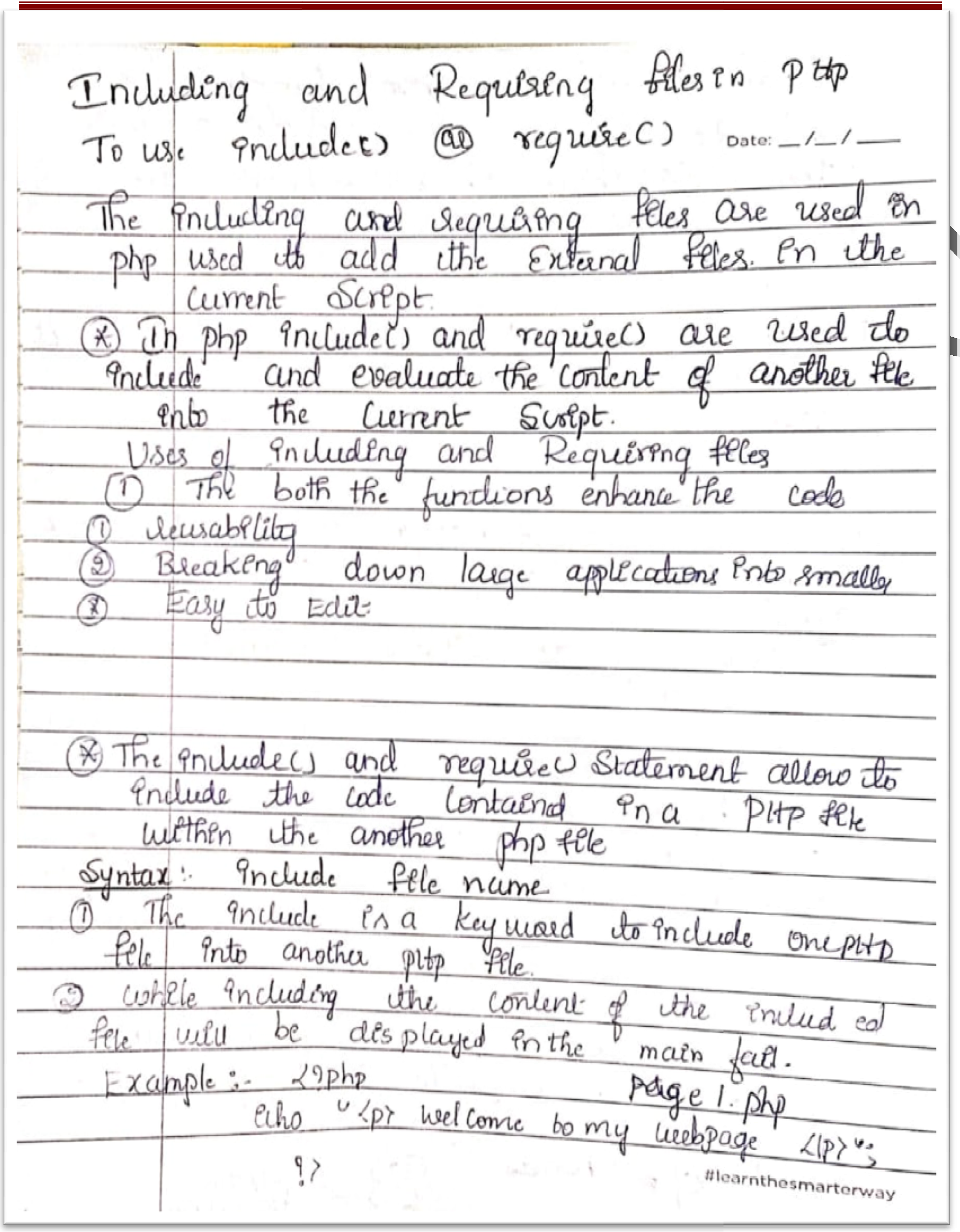
Syntax: natcasesort(array_name)

Example:

```
<?php
$php_files = array("code12.php", "code22.php","code2.php",
"Code3.php", "code1.php"); natcasesort($php_files);
echo "List of file using natural order: ";print_r($php_files);
?>
```

Output: List of file using natural order:

```
 Array
        (
            [4]  =>  code1.php
            [2]  =>  code2.php
            [3] => Code3.php
            [0]  =>  code12.php
            [1]  =>  code22.php
        )
```



Including and Requiring files in php

To use include() (or) require()

Date: __/__/__

The including and requiring files are used in php used to add the External files in the current Script.

(*) In php include() and require() are used to include and evaluate the content of another file into the current Script.

Uses of Including and Requiring files

(1) The both the functions enhance the code

(1) reusability

(2) Breaking down large applications into smaller

(3) Easy to Edit.

(*) The include() and require() Statement allow to include the code contained in a PHP file within the another php file

Syntax :- include file name.

(1) The include is a keyword to include one php file into another php file.

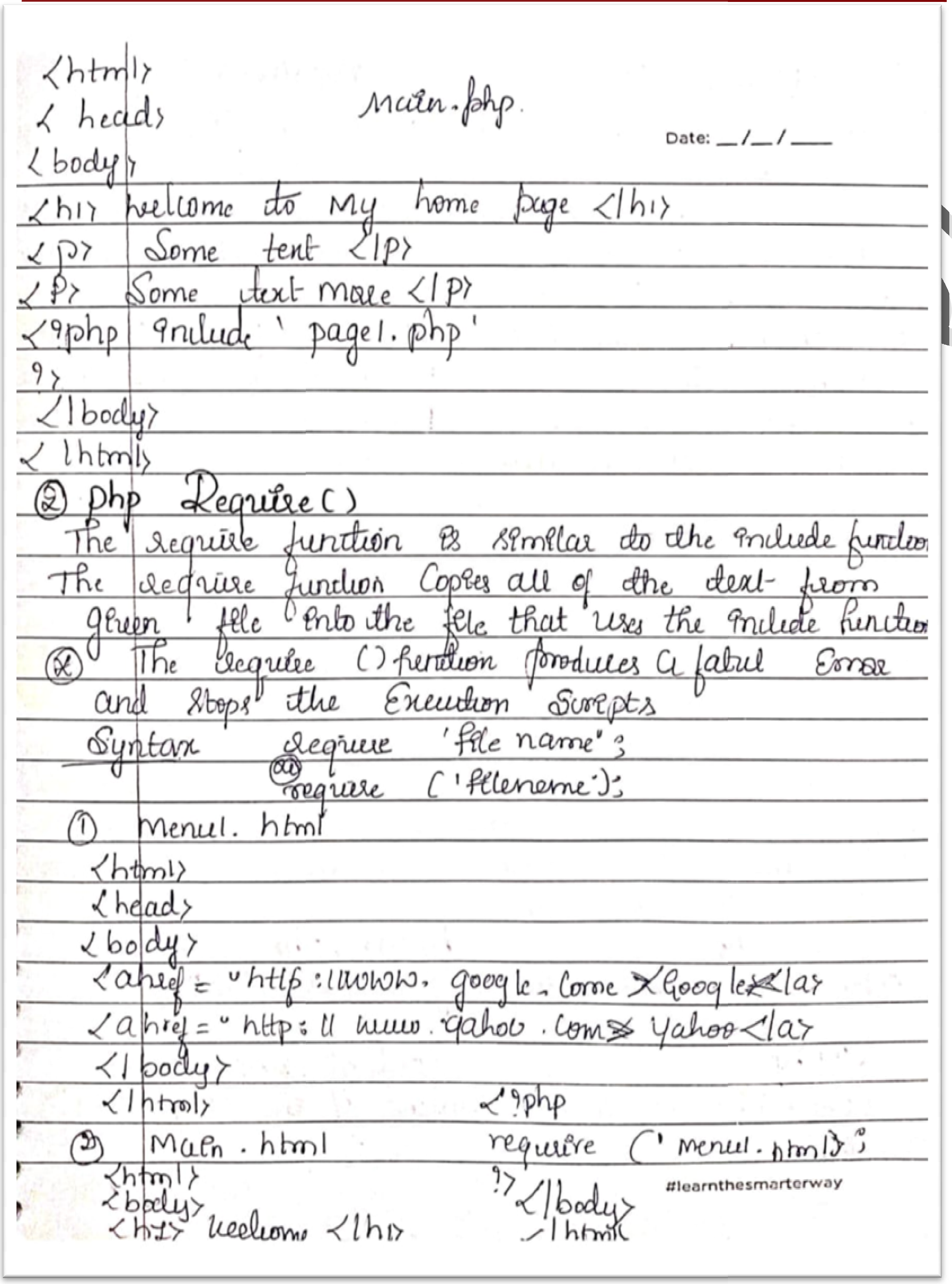
(2) while including the content of the included file will be displayed in the main file.

Example :- <?php   page1.php

echo "<p> welcome to my webpage </p>";

?>

#learnthesmarterway



main.php

Date: __/__/__

```
<html>
< head>
<body>
<h1> welcome to My home page </h1>
<p> Some text </p>
<p> Some text more </p>
<?php include ' page1.php '
?>
</body>
< /html>
```

② php Require ( )

The require function is similar to the include function. The require function copies all of the text from given file into the file that uses the include function.

ⓧ The require ( ) function produces a fatal Error and stops the Execution Scripts

Syntax   require 'file name';
(or)
require ('filename');

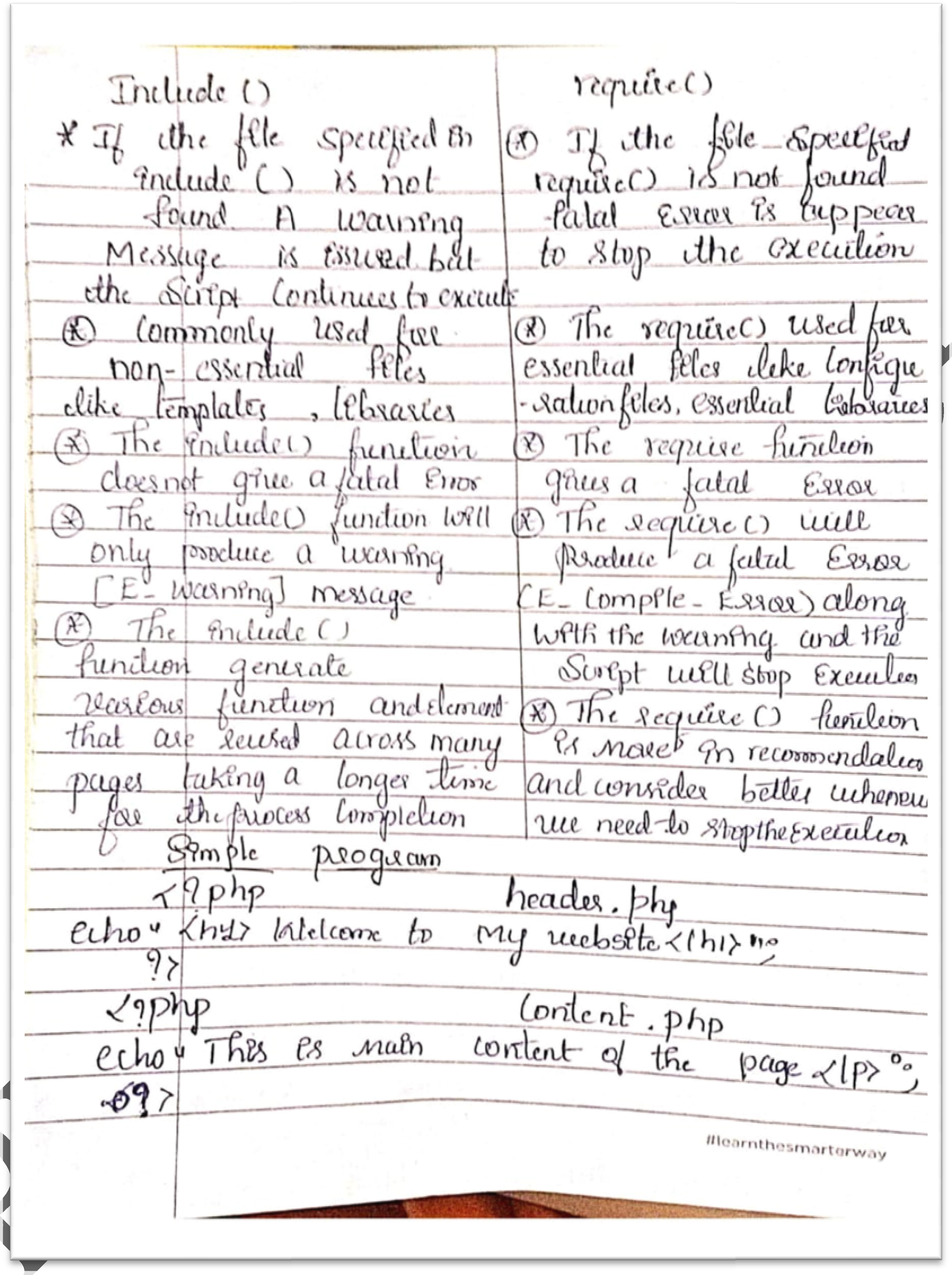
① menu1. html

```
<html>
<head>
<body>
<a href = "http://www. google. com">Google</a>
<a href = " http:// www. yahoo . com"> Yahoo</a>
</body>
</html>
```

② Main . html

```
<html>
<body>
<h1> welcome </h1>
<?php
require (' menu1. html');
?>
</body>
</html>
```

#learnthesmarterway

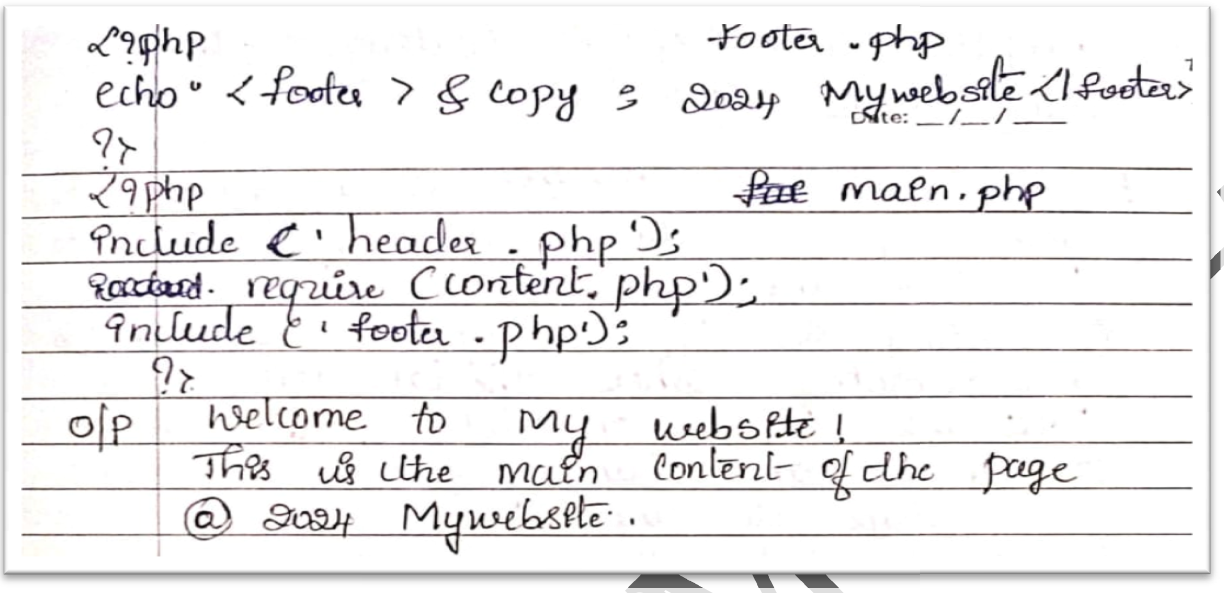


Include ()	require()
* If the file specified in include () is not found A warning Message is issued but the Script Continues to execute	(*) If the file specified require() is not found fatal Error is appear to stop the execution
(*) Commonly used for non-essential files like templates, libraries	(*) The require() used for essential files like configuration files, essential libraries
(*) The include() function does not give a fatal Error	(*) The require function gives a fatal Error
(*) The include() function will only produce a warning [E_warning] message	(*) The require() will Produce a fatal Error (E_Compile_Error) along with the warning and the Script will stop Execution
(*) The include () function generate various function and element that are reused across many pages taking a longer time for the process completion	(*) The require () function is more in recommendation and consider better whenever we need to stop the Execution

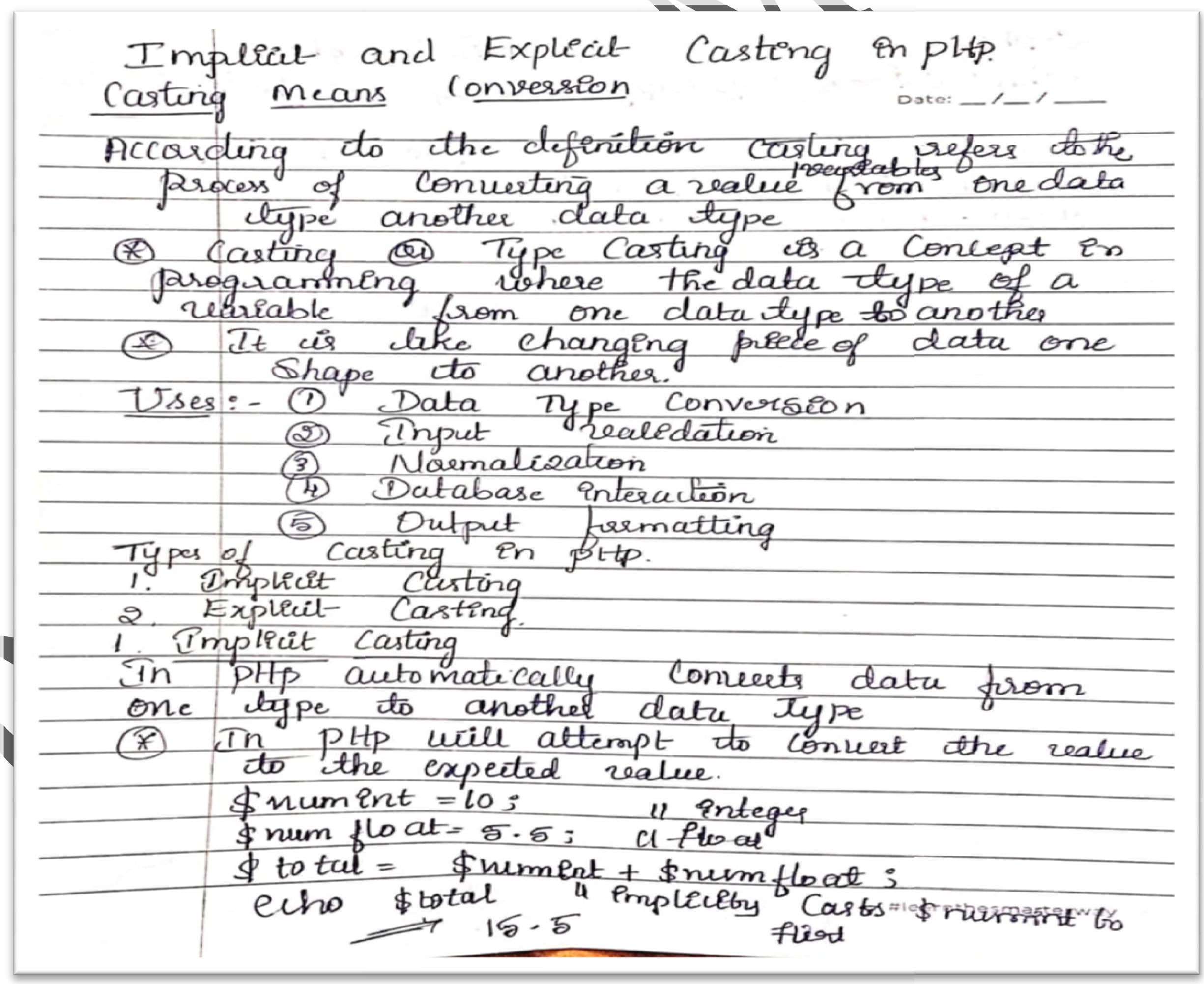
Simple program

```
<?php                                   header.php
echo "<h1> Welcome to My website</h1>";
?>
<?php                                   content.php
echo "This is main content of the page </p>";
?>
```

#learnthesmarterway



```
<?php                                   Footer.php
echo "<footer>&copy; 2024 Mywebsite</footer>
?>                                       Date: __/__/___
<?php                                   ~~Fot~~ main.php
include ('header.php');
~~Rcontent~~. require ('content.php');
include ('footer.php');
?>
o/p   Welcome to My website!
      This is the main content of the page
      © 2024 Mywebsite.
```



Implicit and Explicit Casting in php.

Casting Means Conversion   Date: __/__/___

According to the definition casting refers to the process of Converting a value (variables) from one data type another data type

⊛ Casting or Type Casting is a Concept in programming where the data type of a variable from one data type to another

⊛ It is like changing piece of data one shape to another.

Uses:-
① Data Type Conversion
② Input validation
③ Normalization
④ Database interaction
⑤ Output formatting

Types of Casting in php.
1. Implicit Casting
2. Explicit Casting.

1. Implicit Casting

In pHp automatically Converts data from one type to another data type

⊛ In pHp will attempt to Convert the value to the expected value.

```
$numint = 10;              // Integer
$numfloat = 5.5;           // float
$total = $numint + $numfloat;
echo $total                // implicitly Casts $numint to float
      → 15.5
```

```
$string = "10";
$integer = 5;
$result = $string + $integer;
echo $result ;    o/p:-15
```

In implicit conversion in php typically refers to the process of converting data types implicitly during operations (or) assignments.

② Explicit Casting :-

Explicitly or Explicit casting involves manually converting a value from one data type to another data type using casting operators.

(*) In PHP provides casting operators for various data types (int), (float), (string) (array) (object)

Example

```
$numfloat = 5.5;
$num_int = (int) $numfloat ;        to integer
echo $num_int; -> Explicitly casts $numfloat
```

② Casting to float (float) :

```
$intnum = 10;
$floatnum = (float) $intnum;   o/p :- 10.0
```

③

```
$num = 123;          Casting string
$strnum = (string) $num;      o/p :- "123"
```

④ Casting to Boolean

```
$num = 0;
$boolvalue = (bool) $num;      o/p = false.
```

⑤ Casting to Array

```
$num = 123;
$arrayvalue = (array) $num     o/p array(123)
```

In PHP, explicit casting is the process of manually converting a value from one data type to another. This can be particularly useful when dealing with operations that require a specific type or when working with mixed data types. PHP provides a straightforward syntax for explicit casting, and the language supports casting between various types such as integers, floats, strings, arrays, objects, and more.

Basic Syntax for Explicit Casting

The general syntax for casting in PHP is to specify the desired type in parentheses before the variable or value you want to convert.

The syntax:

$newValue = (type) $originalValue;

Example:

```
<?php
// Original values
$intVal = 10;
$floatVal = 10.5;
$stringVal = "100";
$arrayVal = [1, 2, 3];
$objVal = (object) ['name' => 'Alice'];
// Casting
$castToInt = (int) $floatVal;         // 10
$castToFloat = (float) $stringVal;    // 100.0
$castToString = (string) $intVal;     // "10"
$castToArray = (array) $objVal;       // Array with 'name' => 'Alice'
$castToObject = (object) $arrayVal;   // Object with properties 0, 1, 2
// Output results
echo "Int cast: $castToInt\n";        // 10
echo "Float cast: $castToFloat\n";    // 100.0
echo "String cast: $castToString\n";  // "10"
print_r($castToArray);                // Array ( [name] => Alice )
print_r($castToObject);               // stdClass Object ( [0] => 1 [1] => 2 [2] => 3 )
?>
```