2.1 Control Structures- if statement, Using the else clause with if statement, switch statement, The while statement, do while statement, for statement.

Conditional statements make it possible for your computer program to respond accordingly toa wide variety of inputs, using logic to discern between various conditions based on inputvalue. This functionality is so basic to the creation of computer software that it shouldn't comeas a surprise that a variety of conditional statements are a staple of all mainstreamprogramming languages, PHP included.

The if Statement

The if statement is one of the most commonplace constructs of any mainstream programminglanguage, offering a convenient means for conditional code execution. The following is the syntax:

```
if (expression) {
statement
}
```

As an example, suppose you want a congratulatory message displayed if the user guesses apredetermined secret number:

```
<?php
if ($bank_balance< 100)
{
$money = 1000;
$bank_balance += $money;
}
?>
```

One can forgo the use of brackets when the conditional body consists of only a singlestatement.

The else Statement

The problem with the previous example is that output is only offered for the user who correctly guessesthe secret number. All other users are left destitute, completely snubbed for reasons presumably linkedto their lack of psychic power. What if you want to provide a tailored response no matter the outcome?To do so you would need a way to handle those not meeting the if conditional requirements, a functionhandily offered by way of the else statement. Here's a revision of the previous example, this timeoffering a response in both cases:

```
<?php
if ($bank_balance< 100)
{
$money = 1000;
$bank_balance += $money;
}
else
{
$savings += 50;
$bank_balance -= 50;
}
?>
```

Like if, the else statement brackets can be skipped if only a single code statement is enclosed.

The elseif Statement

The if-else combination works nicely in an "either-or" situation—that is, a situation in which only twopossible outcomes are available. But what if several outcomes are possible? You would need a means forconsidering each possible outcome, which is accomplished with the elseif statement. Let's revise thesecret-number example again, this time offering a message if the user's guess is relatively close (withinten) of the secret number:

```
<?php
if ($bank_balance< 100)
{
$money = 1000;
$bank_balance += $money;
}
elseif ($bank_balance> 200)
{
$savings += 100;
$bank_balance -= 100;
}
else
{
$savings += 50;
$bank_balance -= 50;
}
?>
```

Like all conditionals, elseif supports the elimination of bracketing when only a single statement isenclosed.

The switch Statement

You can think of the switch statement as a variant of the if-else combination, often used when youneed to compare a variable against a large number of values:

```
<?php
if ($page == "Home") echo "You selected Home";
elseif ($page == "About") echo "You selected About";
elseif ($page == "News") echo "You selected News";
elseif ($page == "Login") echo "You selected Login";
elseif ($page == "Links") echo "You selected Links";
?>
```

If we use a switch statement, the code might look like

```
<?php
switch ($page)
{
case "Home":
echo "You selected Home";
break;
case "About":
echo "You selected About";
break;
case "News":
echo "You selected News";
break;
case "Login":
echo "You selected Login";
break;
case "Links":
echo "You selected Links";
break;}?>
```

Breaking out

If you wish to break out of the switch statement because a condition has been fulfilled,use the break command. This command tells PHP to break out of the switchand jump to the following statement.If you were to leave out the break commands in Example 4-23 and the case of Homeevaluated to be TRUE, all five cases would then be executed. Or if $page had the valueNews, all the case commands from then on would execute. This is deliberate andallows for some advanced programming, but generally you should always rememberto issue a break command every time a set of case conditionals has finished executing.In fact, leaving out the break statement is a common error.

Default action

A typical requirement in switch statements is to fall back on a default action if noneof the case conditions are met. For example, in the case of the menu code inExample 4-23, you could add the code in Example 4-24 immediately before the finalcurly brace.*Example 4-24. A default statement to add to Example 4-23*

```
default:
echo "Unrecognized selection";
break;
```

Although a break command is not required here because the default is the final substatement,and program flow will automatically continue to the closing curly brace,should you decide to place the default statement higher up, it would definitely needa break command to prevent program flow from dropping into the following statements. Generally, the safest practice is to always include the break command.

Looping Statements

Although varied approaches exist, looping statements are a fixture in every widespread programminglanguage. Looping mechanisms offer a simple means for accomplishing a commonplace task inprogramming: repeating a sequence of instructions until a specific condition is satisfied. PHP offersseveral such mechanisms, none of which should come as a surprise if you're familiar with otherprogramming languages.

The while Statement

The while statement specifies a condition that must be met before execution of its embedded code isterminated. Its syntax is the following:

```
while (expression) {
statements
}
```

In the following example, $count is initialized to the value 1. The value of $count is then squared andoutput. The $count variable is then incremented by 1, and the loop is repeated until the value of $countreaches 5.

```
<?php
$count = 1;
while ($count < 5) {
printf("%d squared = %d <br />", $count, pow($count, 2));
$count++;
}
?>
```

The output looks like this:

```
1 squared = 1
2 squared = 4
3 squared = 9
4 squared = 16
```

Like all other control structures, multiple conditional expressions may also be embedded into thewhile statement. For instance, the following while block evaluates either until it reaches the end-of-fileor until five lines have been read and output:

```
<?php
$linecount = 1;
$fh = fopen("sports.txt","r");
while (!feof($fh) && $linecount<=5) {
$line = fgets($fh, 4096);
echo $line. "<br />";$linecount++;}?>
```

Given these conditionals, a maximum of five lines will be output from the sports.txt file, regardlessof its size.

The do...while Statement

The do...while looping statement is a variant of while but it verifies the loop conditional at theconclusion of the block rather than at the beginning. The following is its syntax:

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

Both while and do...while are similar in function. The only real difference is that the codeembedded within a while statement possibly could never be executed, whereas the code embeddedwithin a do...while statement will always execute at least once. Consider the following example:

```
<?php
$count = 11;
do {
printf("%d squared = %d <br />", $count, pow($count, 2));
} while ($count < 10);
?>
```

The following is the outcome:11 squared = 121

The for Statement

The for statement offers a somewhat more complex looping mechanism than does while. The followingis its syntax: for (expression1; expression2; expression3) { statements }

There are a few rules to keep in mind when using PHP's for loops:

- The first expression, expression1, is evaluated by default at the first iteration of theloop.
- The second expression, expression2, is evaluated at the beginning of eachiteration. This expression determines whether looping will continue.
- The third expression, expression3, is evaluated at the conclusion of each loop.
- Any of the expressions can be empty, their purpose substituted by logic embeddedwithin the for block.

With these rules in mind, consider the following examples, all of which display a partialkilometer/mile equivalency chart:// Example One

```
for ($kilometers = 1; $kilometers <= 5; $kilometers++) {
printf("%d kilometers = %f miles <br />", $kilometers, $kilometers*0.62140);
}

// Example Two
for ($kilometers = 1; ; $kilometers++) {
if ($kilometers > 5) break;
printf("%d kilometers = %f miles <br />", $kilometers, $kilometers*0.62140);
}
// Example Three
$kilometers = 1;
for (;;) {
// if $kilometers > 5 break out of the for loop.
if ($kilometers > 5) break;
printf("%d kilometers = %f miles <br />", $kilometers, $kilometers*0.62140);
$kilometers++;          }
```

The results for all three examples follow:

```
1 kilometers = 0.6214 miles
2 kilometers = 1.2428 miles
3 kilometers = 1.8642 miles
4 kilometers = 2.4856 miles
5 kilometers = 3.107 miles
```

The foreach Statement

The foreach looping construct syntax is adept at looping through arrays, pulling each key/value pairfrom the array until all items have been retrieved or some other internal conditional has been met. Twosyntax variations are available, each of which is introduced with an example.The first syntax variant strips each value from the array, moving the pointer closer to the end witheach iteration. The following is its syntax:

foreach (array_expr as $value) {   statement   }

Suppose you want to output an array of links, like so: Example 1

```
<?php
$x = array("a","b","c");
echo "<b>Online Resources</b>:<br />";
foreach($x as $x1) {
echo "$x1 <br/>";
}?>
```

This would result in the following:

```
a
b
c
```

Example 2:

```
<?php
$links = array("www.apress.com","www.php.net","www.apache.org");
echo "<b>Online Resources</b>:<br />";
foreach($links as $link) {
echo "<a href=\"http://$link\">$link</a><br />";
}        ?>
```

This would result in the following:

```
Online Resources:<br />
<a href="http://www.apress.com">http://www.apress.com</a><br />
<a href="http://www.php.net">http://www.php.net</a><br />
<a href="http://www.apache.org">http://www.apache.org</a><br />
```

The break and goto Statements

Encountering a break statement will immediately end execution of a do...while, for, foreach, switch,or while block. For example, the following for loop will terminate if a prime number is pseudo-randomlyhappened upon:

```
<?php
$primes = array(2,3,5,7,11,13,17,19,23,29,31,37,41,43,47);
for($count = 1; $count++; $count < 1000) {
$randomNumber = rand(1,50);
if (in_array($randomNumber,$primes)) {
break;
} else {
printf("Non-prime number found: %d <br />", $randomNumber);   }        }        ?>
```

Sample output follows:

```
Non-prime number found: 48
Non-prime number found: 42 Prime number found: 17
```

Through the addition of **the goto statement** in PHP 5.3, the break feature was extended to supportlabels. This means you can suddenly jump to a specific location outside of a looping or conditionalconstruct. An example follows:

```
<?php
for ($count = 0; $count < 10; $count++)       {
$randomNumber = rand(1,50);
if ($randomNumber< 10)
goto less;
else
echo "Number greater than 10: $randomNumber<br />";     }
less:
echo "Number less than 10: $randomNumber<br />";          ?>
```

It produces the following (your output will vary):

```
Number greater than 10: 22
Number greater than 10: 21
Number greater than 10: 35
Number less than 10: 8
```

The continue Statement

The continue statement causes execution of the current loop iteration to end and commence at thebeginning of the next iteration. For example, execution of the following while body will recommence if$usernames[$x] is found to have the value missing:

```
<?php
$usernames = array("Grace","Doris","Gary","Nate","missing","Tom");
for ($x=0; $x < count($usernames); $x++) {
if ($usernames[$x] == "missing") continue;
printf("Staff member: %s <br />", $usernames[$x]); }          ?>
```

This results in the following output:

```
Staff member: Grace
Staff member: Doris
Staff member: Gary
Staff member: Nate
Staff member: Tom
```

2.2 Functions - Defining a function, User Defined function, argument function, variable function, Return function, default argument, variable length argument.
Variable Function: gettype, settype, isset, strval, floatval, intval, print_r.MATH functions: Abs, ceil, floor, round, fmod, min, max, pow, sqrt, randDatefunction:Date, getdate, setdate, checkdate, time, mktime.

The basic requirements of any programming language include somewhere to store data, a means of directing program flow, and a few bits and pieces such as expression evaluation, file management, and text output. PHP has all these, plus tools like else and elseif to make life easier. But even with all these in our toolkit, programming can be clumsy and tedious; especially if you have to rewrite portions of very similar code each time you need them.
That's where functions and objects come in. As you might guess, a *function* is a set of statements that performs a particular function and—optionally—returns a value. You can pull out a section of code that you have used more than once, place it into a function, and call the function by name when you want the code. Functions have many advantages over contiguous, inline code. For example, they:
• Involve less typing
• Reduce syntax and other programming errors
• Decrease the loading time of program files
• Decrease execution time, because each function is compiled only once, no matter how often you call it
• Accept arguments and can therefore be used for general as well as specific cases
Objects take this concept a step further. An *object* incorporates one or more functions, and the data they use, into a single structure called a *class*.

PHP comes with hundreds of ready-made, built-in functions, making it a very rich language. To use a function, call it by name. For example, you can see the print function in action here:
print("print is a pseudo-function"); The parentheses tell PHP that you're referring to a function.

Defining a Function
The general syntax for a function is as follows:

```
Function function_name ([parameter [, ...]]) {  // Statements }
```

The first line of the syntax indicates the following:
• A definition starts with the word function.
• A name follows, which must start with a letter or underscore, followed by any number of letters, numbers, or underscores.
• The parentheses are required.
• One or more parameters, separated by commas, are optional.

Function names are case-insensitive, so all of the following strings can refer to the print function: PRINT, Print, and PrInT.

The opening curly brace starts the statements that will execute when you call the function; a matching curly brace must close it. These statements may include one or more return statements, which force the function to cease execution and return to the calling code. If a value is attached to the return statement, the calling code can retrieve it.

Passing Arguments by Value
You'll often find it useful to pass data into a function. As an example, let's create a function that calculates an item's total cost by determining its sales tax and then adding that amount to the price:

```
Function calcSalesTax()
{ $price=10;
  $tax=10;
 $total = $price + $tax;
 echo "Total cost: $total"; }
calcSalesTax();
```

Of course, you're not bound to passing static values into the function. You can also pass variables like this:

```
Function calcSalesTax1($price, $tax) {
 $total = $price + $tax;
     echo "Total cost: $total"; }
calcSalesTax1(15,20);
```

When you pass an argument in this manner, it's called passing by value. This means that any changes made to those values within the scope of the function are ignored outside of the function. If you want these changes to be reflected outside of the function's scope, you can pass the argument by reference, introduced next.

Passing Arguments by Reference

On occasion, you may want any changes made to an argument within a function to be reflected outside of the function's scope. Passing the argument by reference accomplishes this. Passing an argument by reference is done by appending an ampersand to the front of the argument. Here's an example:

Example:

```
$cost = 20;     $tax = 5;
Function calculateCost2 (&$cost, $tax)     {
// Modify the $cost variable
 $cost = $cost + $tax;
 // Perform some random change to the $tax variable.
$tax += 4;    }
calculateCost2($cost,$tax);
printf("Tax is %01.2f%% ", $tax*100);    printf("Cost is: $%01.2f", $cost);

// calculateCost2(30,$tax); error
calculateCost2($cost,10);
printf("Tax is %01.2f%% ", $tax*100);    printf("Cost is: $%01.2f", $cost); ?>
```

Here's the result: Tax is 500.00% Cost is: $25.00   Tax is 500.00% Cost is: $35.00

Default Argument Values

Default values can be assigned to input arguments, which will be automatically assigned to the argument if no other value is provided. To revise the sales tax example, suppose that the majority of your sales take place in Franklin County, Ohio. You could then assign $tax the default value of 6.75 percent, like this:

```
Function calcSalesTax3($tax=100,$price )
{    $total = $price + $tax;  echo "Total cost: $total"; }
$x=10;$y=20;
calcSalesTax3($x,$y);//Total cost: 30
calcSalesTax3($x);//Total cost: 110
```

Default argument values must appear at the end of the parameter list and must be constant expressions; you cannot assign nonconstant values such as function calls or variables.

You can designate certain arguments as optional by placing them at the end of the list and assigning them a default value of nothing, like so:

```
Function calcSalesTax($price, $tax="")  {     $total = $price + ($price * $tax);     echo "Total cost: $total"; }
```

This allows you to call calcSalesTax() without the second parameter if there is no sales tax:

```
calcSalesTax(42.00);
```

This returns the following output: Total cost: $42

If multiple optional arguments are specified, you can selectively choose which ones are passed along. Consider this example:

function calculate($price, $price2="", $price3="") {   echo $price + $price2 + $price3; }

You can then call calculate(), passing along just $price and $price3, like so:

calculate(10, "", 3);   This returns the following value: 13

Returning a Value
The return Statement The return() statement returns any ensuing value back to the function caller, returning program control back to the caller's scope in the process. If return() is called from within the global scope, the script execution is terminated. Revising the calcSalestax() function again, suppose you don't want to immediately echo the sales total back to the user upon calculation, but rather want to return the value to the calling block:

Function calcSalesTax($price, $tax=.0675) {
  $total = $price + ($price * $tax);   return $total; }

Alternatively, you could return the calculation directly without even assigning it to $total, like this:

Function calcSalesTax($price, $tax=.0675) {   return $price + ($price * $tax); }

Here's an example of how you would call this function:

<?php $price = 6.99; $total = calcSalesTax($price); ?>

Returning Multiple Values
It's often convenient to return multiple values from a function. For example, suppose that you'd like to create a function that retrieves user data from a database (say the user's name, e-mail address, and phone number) and returns it to the caller. Accomplishing this is much easier than you might think, with the help of a very useful language construct, list(). The list() construct offers a convenient means for retrieving values from an array, like so:

<?php $colors = array("red","blue","green");   list($red, $blue, $green) = $colors; ?>

Once the list() construct executes, $red, $blue, and $green will be assigned red, blue, and green, respectively. Building on the concept demonstrated in the previous example, you can imagine how the three prerequisite values might be returned from a function using list():

<?php function retrieveUserProfile()   { $user[] = "Jason Gilmore"; $user[] = "jason@example.com"; $user[] = "English";   return $user;   }

list($name, $email, $language) = retrieveUserProfile();   echo "Name: $name, email: $email, language: $language"; ?>
Executing this script returns the following: Name: Jason Gilmore, email: jason@example.com, language: English

Let's take a look at a simple function to convert a person's full name to lowercase and then capitalize the first letter of each name.
Some of the built-in functions that use one or more arguments appear in
<?php
echo **strrev**(" .dlrow olleH"); // Reverse string
echo **str_repeat**("Hip ", 2); // Repeat string
echo **strtoupper**("hooray!"); // String to uppercase
?>
This example uses three string functions to output the following text:
Hello world. Hip Hip HOORAY!
$lowered = **strtolower**("aNY # of Letters and Punctuation you WANT");
echo $lowered;   The output of this experiment is as follows: any # of letters and punctuation you want

PHP also provides **aucfirst** function that sets the first character of a string to uppercase:
$ucfixed = **ucfirst**("any # of letters and punctuation you want");
Suppose you want to **raise** five to the third power. You could invoke PHP's pow() function like this:

```
<?php     $value = pow(5,3); // returns 125     echo $value; ?>
```

If you want to output the function results, you can bypass assigning the value to a variable, like this:

```
<?php     echo pow(5,3); ?>
```

If you want to output the function outcome within a larger string, you need to concatenate it like this:

```
echo "Five raised to the third power equals ".pow(5,3).".";
```

Or perhaps more eloquently, you could use printf(): **printf("Five raised to the third power equals %d.", pow(5,3));**

Variable Function: gettype, settype, isset, strval, floatval, intval, print_r.
MATH functions: Abs, ceil, floor, round, fmod, min, max, pow, sqrt, rand
Datefunction:Date, getdate, setdate, checkdate, time, mktime.

Variable Function: gettype, settype, isset, strval, floatval, intval, print_r.

PHP provides over thirty functions that can perform various functions related to variables.
Before we learn about the functions, let's take a look at the different types of data that can be stored in variables. Although PHP automatically assigns the data type of each variable based on the data that it contains, it will be good to know a little about each one.

Type	Type	Description
String	Scalar	Sequences of Characters, Such As This Statement
Integer	Scalar	Whole Number, Positive or Negative, Without a Decimal Point
Float (Also Floating-Point or Double)	Scalar	Floating Numbers, With a Decimal Point
Boolean	Scalar	Has Only Two Possible Values: "TRUE" or "FALSE"
Array	Compound	Named/Indexed Collection of Other Values
Object	Compound	Programmer-Defined Classes
Resource	Special	Hold References to External Resources (Database Connections, Etc.)
NULL	Special	Has Only One Possible Value: NULL

isset : Determines If a Variable Is Set and Is Not NULL
Probably the most commonly used variable function is the isset() function, and, as the name implies, it can tell you whether or not a variable has been set.

```
<?php
  $question = "What's the unluckiest kind of cat to have?<br>";
  if (isset($question)) {
    echo $question;
    echo "A catastrophe!";   } else {
    echo "If you don't have a question you don't get an answer.";   } ?>
```

empty : Determines Whether a Variable Is Empty
The empty() function will determine whether or not a variable contains actual data. An empty string "", 0, NULL & FALSE are all considered empty variables.

```
<?php
  $question = "What do you get when two giraffes collide?<br>";
  if (!empty($question)) {
    echo $question;
    echo "A giraffic jam.";
    unset($question);
  } else {     echo "If you don't have a question you don't get an answer.";   } ?>
```

gettype Gets the Type of a Variable. string **gettype** (mixed $var)
Return Values : Possible values for the returned string are:
"boolean"   "integer" "string" "NULL" array" "object" "resource" "unknown type"
"double" (for historical reasons "double" is returned in case of a float, and not simply "float")

```
 <?php
$data = array(1, 1., NULL, new stdClass, 'foo');
foreach ($data as $value) {    echo gettype($value), "\n";  } ?>
```

settype Sets the Type of a Variable. bool **settype** (mixed &$var , string $type)
type:
Possibles values of type are:"boolean"or"bool""integer" or "int" "float" or "double" "string" "array" "object" "null"

```
<?php
$foo = "5bar"; // string
$bar = true;   // boolean
settype($foo, "integer"); // $foo is now 5   (integer)
settype($bar, "string");  // $bar is now "1" (string) ?>
```

Strval
string **strval** (mixed $var) Get the string value of a variable.
This function performs no formatting on the returned value. If you are looking for a way to format a numeric value as a string, please see sprintf() or number_format().

```
<?php
class StrValTest{
public function __toString()    {        return __CLASS__;    } }
// Prints 'StrValTest'
echo strval(new StrValTest); ?>
```

Floatval float **floatval** (mixed $var) - - Gets the float value of var.

```
<?php
$var = '122.34343The';
$float_value_of_var = floatval($var);
echo $float_value_of_var; // 122.34343
?>
```

```
<?php
$var = 'The122.34343';
$float_value_of_var = floatval($var);
echo $float_value_of_var; // 0
?>
```

Intval int **intval** (mixed $var [, int $base = 10])
Returns the integer value of var, using the specified base for the conversion (the default is base 10). **intval()** should not be used on objects, as doing so will emit an **E_NOTICE** level error and return 1.

Return Values ¶
The integer value of var on success, or 0 on failure. Empty arrays return 0, non-empty arrays return 1.
The maximum value depends on the system. 32 bit systems have a maximum signed integer range of -2147483648 to 2147483647. So for example on such a system, *intval('1000000000000')* will return 2147483647. The maximum signed integer value for 64 bit systems is 9223372036854775807.
Strings will most likely return 0 although this depends on the leftmost characters of the string. The common rules of integer casting apply.

```
<?php
echo intval(42);                      // 42
echo intval(4.2);                     // 4
echo intval('42');                   // 42
echo intval('+42');                   // 42
echo intval('-42');                  // -42
echo intval(042);                     // 34
echo intval('042');                   // 42
echo intval(1e10);                    // 1410065408
echo intval('1e10');                  // 1
echo intval(0x1A);                    // 26
echo intval(42000000);                // 42000000
echo intval(420000000000000000000);   // 0
echo intval('420000000000000000000'); // 2147483647
echo intval(42, 8);                   // 42
echo intval('42', 8);                 // 34
echo intval(array());                 // 0
echo intval(array('foo', 'bar'));     // 1
?>
```

print_r
The print_r() function is a built-in function in PHP and is used to print or display information stored in a variable.
print_r($variable, $isStore)

Parameters: This function accepts two parameters as shown in above syntax and described below.
$variable: This parameter specifies the variable to be printed and is a mandatory parameter.
$isStore: This an option parameter. This parameter is of boolean type whose default value is FALSE and is used to store the output of the print_r() function in a variable rather than printing it. If this parameter is set to TRUE then the print_r() function will return the output which it is supposed to print.
Return Value: If the $variable is an integer or a float or a string the function prints the value of the variable. If the variable is an array the function prints the array in a format which displays the keys as well as values, a similar notation is used for objects. If the parameter $isStore is set to TRUE then the print_r() function will return a string containing the information which it is supposed to print.

```
<?php
$var1 = "Welcome to GeeksforGeeks";  // string variable
$var2 = 101; // integer variable
$arr = array('0' => "Welcome", '1' => "to", '2' => "GeeksforGeeks"); // array variable
  // printing the variables
print_r($var1); echo"\n";
print_r($var2); echo"\n";
print_r($arr); ?>
```

```
Welcome to GeeksforGeeks
101
Array
(
    [0] => Welcome
    [1] => to
    [2] => GeeksforGeeks
)
```

unset : Unsets a Given Variable
The **unset**() function, confusingly stuck in the example above, will unset the variable so that it is not longer set. It can be reset later in the script, but until then it no longer exists.

MATH functions:
Abs
This function takes negative value as input and returns the absolute (positive) value of a integer or float number.
Syntax :
abs(number);   In this function ‘number’ can be float or integer.

Ceil
This function takes numeric value as argument and returns the next highest integer value by rounding up value if necessary. **Syntax :** ceil($number);

Floor
This function takes numeric value as argument and returns the next lowest integer value (as float) by rounding down value if necessary. **Syntax :** floor($number);

Round
This function takes numeric value as argument and returns the next highest integer value by rounding up value if necessary. **Syntax :** round(number, precision, mode);

In this, **number** specifies the value to round, **precision** specifies the number of decimal digits to round to (Default is 0) and, **mode(optional)** specifies one of the following constants to specify the mode in which rounding occurs :
PHP_ROUND_HALF_UP : (set by Default) Rounds number up to precision decimal, when it is half way there. Rounds 1.5 to 2 and -1.5 to -2
PHP_ROUND_HALF_DOWN : Round number down to precision decimal places, when it is half way there. Rounds 1.5 to 1 and -1.5 to -1
PHP_ROUND_HALF_EVEN : Round number to precision decimal places towards the next even value.
PHP_ROUND_HALF_ODD : Round number to precision decimal places towards the next odd value.

Fmod
This function takes two arguments as input returns the floating point remainder (modulo) of division of arguments.
Syntax : fmod(x, y);
Here, **x** is dividend and **y** is divisor. The remainder (r) is defined as: x = i * y + r, for some integer i. If y is non-zero, r has the same sign as x and a magnitude less than the magnitude of y.

Min
In this function, if the first and only parameter is an array, min() returns the lowest value in that array. If at least two parameters are provided, min() returns the smallest of these values.
Syntax : min(array_values); or min(value1,value2,...);

Max
In this function, if the first and only parameter is an array, max() returns the highest value in that array. If at least two parameters are provided, max() returns the biggest of these values.
max(array_values); or max(value1,value2,...);

Pow
pow() : This function takes base and exponent as arguments and returns base raised to the power of exponent.
Syntax : pow(base,exponent);

Sqrt
This function takes numeric value as arguments and returns the square root of value.
Syntax : sqrt(number);

rand
If this function is called without the optional min, max arguments rand() returns a pseudo-random integer between 0 and getrandmax(). If you want a random number between 12 and 56 (inclusive). Example, use rand(12, 56).
Syntax : rand(); or rand(min,max);

Datefunction:

Date The date() function formats a local date and time, and returns the formatted date string.

date(*format,timestamp);*

```
echo "Today is " . date("Y/m/d") . "<br>";// Today is 2019/01/07
echo "Today is " . date("Y.m.d") . "<br>";// Today is 2019.01.07
echo "Today is " . date("Y-m-d") . "<br>";// Today is 2019-01-07
echo "Today is " . date("l");// Today is Monday
```

Getdate

Return date/time information of the current local date/time:

The getdate() function returns date/time information of a timestamp or the current local date/time.

getdate(*timestamp);*

timestamp Optional. Specifies an integer Unix timestamp. Default is the current local time (time())

```
print_r(getdate());
```

```
o/p: Array ( [seconds] => 47 [minutes] => 59 [hours] => 5 [mday] => 7 [wday] => 1 [mon] => 1 [year] => 2019
[yday] => 6 [weekday] => Monday [month] => January [0] => 1546858787 )
```

```
$mydate=getdate(date("U"));
echo "$mydate[weekday], $mydate[month] $mydate[mday], $mydate[year]"; // Monday, January 7, 2019
```

Setdate

We can assign new date to any date object by using setDate function. Using this we can change the date object.

setDate(Year, month, day)

```
date_default_timezone_set('America/Chicago');
$today = new DateTime;
echo $today->format('Y-m-d ');//19-01-07
echo "<br>";
$today->setDate(13,11,20);
echo $today->format('Y-m-d'); //0013-11-20
```

Checkdate

The checkdate() function is used to validate a Gregorian date.

checkdate(*month,day,year);*

Return Value: TRUE if the date is valid. FALSE otherwise

```
var_dump(checkdate(12,31,-400));// bool(false)
echo "<br>";
var_dump(checkdate(2,29,2003));// bool(false)
echo "<br>";
var_dump(checkdate(2,29,2004));// bool(true)
```

Time

Return the current time as a Unix timestamp, then format it to a date:

The time() function returns the current time in the number of seconds since the Unix Epoch (January 1 1970 00:00:00 GMT).

time();

Return Value: Returns an integer containing the current time as a Unix timestamp

```
$t=time();
echo($t . "<br>");
echo(date("Y-m-d",$t));
```

Mktime

The optional *timestamp* parameter in the date() function specifies a timestamp. If you do not specify a timestamp, the current date and time will be used. The mktime() function returns the Unix timestamp for a date. The Unix timestamp contains the number of seconds between the Unix Epoch (January 1 1970 00:00:00 GMT) and the time specified.

```
$d=mktime(11, 14, 54, 8, 12, 2014);
echo "Created date is " . date("Y-m-d h:i:sa", $d); // Created date is 2014-08-12 11:14:54am
```

2.3 Arrays: Single-Dimensional Arrays Multidimensional Arrays, Casting Arrays

An array is traditionally defined as a group of items that share certain characteristics, such as similarity (car models, baseball teams, types of fruit, etc.) and type (e.g., all strings or integers). Each item is distinguished by a special identifier known as a key. PHP takes this definition a step further, forgoing the requirement that the items share the same data type. For example, an array could quite possibly contain items such as state names, ZIP codes, exam scores, or playing card suits. Each item consists of two components: the aforementioned key and a value. The key serves as the lookup facility for retrieving its counterpart, the value. Keys can be numerical or associative. Numerical keys bear no real relation to the value other than the value's position in the array. As an example, the array could consist of an alphabetically sorted list of state names, with key 0 representing Alabama and key 49 representing Wyoming. Using PHP syntax, this might look like the following:

$states = array(0 => "Alabama", 1 => "Alaska"...49 => "Wyoming");
Using numerical indexing, you could reference the first state in the array (Alabama) like so: $states[0]

Like many programming languages, PHP's numerically indexed arrays begin with position 0, not 1.

An associative key logically bears a direct relation to its corresponding value. Mapping arrays associatively is particularly convenient when using numerical index values just doesn't make sense. For instance, you might want to create an array that maps state abbreviations to their names. Using PHP syntax, this might look like the following:

$states = array("OH" => "Ohio", "PA" => "Pennsylvania", "NY" => "New York")
You could then reference Ohio like this: $states["OH"]

Two-dimensional arrays

Example 3-5. Defining a two-dimensional array

```
<?php
$oxo = array(array('x', ' ', 'o'),  array('o', 'o', 'x'),        array('x', 'o', ' '));        ?>
```

To then return the third element in the second row of this array, you would use the following PHP command, which will display an x: echo $oxo[1][2];

It's also possible to create arrays of arrays, known as multidimensional arrays. For example, you could use a multidimensional array to store U.S. state information. Using PHP syntax, it might look like this:

$states = array ("Ohio" => array("population" => "11,353,140", "capital" => "Columbus"),   "Nebraska" => array("population" => "1,711,263", "capital" => "Omaha"));

You could then reference Ohio's population: $states["Ohio"]["population"]
This would return the following: 11,353,140

Unlike other languages, PHP doesn't require that you assign a size to an array at creation time. In fact, because it's a loosely typed language, PHP doesn't even require that you declare the array before using it, although you're free to do so. Each approach is introduced in this section, beginning with the informal variety. Individual elements of a

PHP array are referenced by denoting the element between a pair of square brackets. Because there is no size limitation on the array, you can create the array simply by making reference to it, like this:
$state[0] = "Delaware";   You can then display the first element of the array $state, like this: echo $state[0];

Additional values can be added by mapping each new value to an array index, like this:

$state[1] = "Pennsylvania"; $state[2] = "New Jersey"; ... $state[49] = "Hawaii";

Interestingly, if you intend for the index value to be numerical and ascending, you can omit the index value at creation time:

$state[] = "Pennsylvania"; $state[] = "New Jersey"; ... $state[] = "Hawaii";

The array building code consists of the following construct: array(); with five strings inside. Each string is enclosed in apostrophes. $team = array('Bill', 'Mary', 'Mike', 'Chris', 'Anne');
If we then wanted to know who player 4 is, we could use this command: echo $team[3]; // Displays the name Chris

Testing for an Array
The **is_array() function** determines whether variable is an array, returning TRUE if it is and FALSE otherwise. Note that even an array consisting of a single value will still be considered an array. An example follows:

$states = array("Florida"); $state = "Ohio"; printf("\$states is an array: %s <br />", (is_array($states) ? "TRUE" : "FALSE")); printf("\$state is an array: %s <br />", (is_array($state) ? "TRUE" : "FALSE"));

Executing this example produces the following:
$states is an array: TRUE $state is an array: FALSE

Outputting an Array
The most common way to output an array’s contents is by iterating over each key and echoing the corresponding value. For instance, a foreach statement does the trick nicely:

$states = array("Ohio", "Florida", "Texas"); foreach ($states AS $state) {   echo "{$state}<br />"; }

Adding and Removing Array Elements

Adding a Value to the Front of an Array
The **array_unshift()** function adds elements to the front of the array. All preexisting numerical keys are modified to reflect their new position in the array, but associative keys aren’t affected. Its prototype follows:

Int **array_unshift**(array array, mixed variable [, mixed variable...])

The following example adds two states to the front of the $states array:
$states = array("Ohio", "New York"); array_unshift($states, "California", "Texas"); // $states = array("California", "Texas", "Ohio", "New York");

Adding a Value to the End of an Array
The **array_push()** function adds a value to the end of an array, returning the total count of elements in the array after the new value has been added. You can push multiple variables onto the array simultaneously by passing these variables into the function as input parameters. Its prototype follows:

intarray_push(array array, mixed variable [, mixed variable...])

The following example adds two more states onto the $states array:

$states = array("Ohio", "New York"); array_push($states, "California", "Texas"); // $states = array("Ohio", "New York", "California", "Texas");

Removing a Value from the Front of an Array
The **array_shift()** function removes and returns the first item found in an array. If numerical keys are used, all corresponding values will be shifted down, whereas arrays using associative keys will not be affected. Its prototype follows:
mixedarray_shift(array array)

The following example removes the first state from the $states array:

$states = array("Ohio", "New York", "California", "Texas"); $state = array_shift($states); // $states = array("New York", "California", "Texas") // $state = "Ohio"

Removing a Value from the End of an Array
The **array_pop()** function removes and returns the last element from an array. Its prototype follows:
mixedarray_pop(array array)
The following example removes the last state from the $states array:

$states = array("Ohio", "New York", "California", "Texas"); $state = array_pop($states); // $states = array("Ohio", "New York", "California" // $state = "Texas"

Searching an Array
The **in_array()** function searches an array for a specific value, returning TRUE if the value is found and FALSE otherwise. Its prototype follows:
booleanin_array(mixed needle, array haystack [, boolean strict])

In the following example, a message is output if a specified state (Ohio) is found in an array consisting of states having statewide smoking bans:
$state = "Ohio"; $states = array("California", "Hawaii", "Ohio", "New York"); if(in_array($state, $states)) echo "Not to worry, $state is smoke-free!";
The optional third parameter, strict, forces in_array() to also consider type.

Searching Associative Array Keys
The function **array_key_exists()** returns TRUE if a specified key is found in an array and FALSE otherwise. Its prototype follows: booleanarray_key_exists(mixed key, array array)
The following example will search an array's keys for Ohio, and if found, will output information about its entrance into the Union:

$state["Delaware"] = "December 7, 1787"; $state["Pennsylvania"] = "December 12, 1787"; $state["Ohio"] = "March 1, 1803"; if (array_key_exists("Ohio", $state)) printf("Ohio joined the Union on %s", $state["Ohio"]);
The following is the result: Ohio joined the Union on March 1, 1803

Searching Associative Array Values
The **array_search()** function searches an array for a specified value, returning its key if located and FALSE otherwise. Its prototype follows: mixedarray_search(mixed needle, array haystack [, boolean strict])
The following example searches $state for a particular date (December 7), returning information about the corresponding state if located:

$state["Ohio"] = "March 1"; $state["Delaware"] = "December 7"; $state["Pennsylvania"] = "December 12"; $founded = array_search("December 7", $state); if ($founded) printf("%s was founded on %s.", $founded, $state[$founded]);
The output follows: Delaware was founded on December 7.

2.4 Constructors

Declaring a Class

Before you can use an object, you must define a class with the class keyword. Class definitions contain the class name (which is case-sensitive), its properties, and its methods. Example 5-10 defines the class User with two properties, which are: $name and $password. It also creates a new instance (called $object) of this class.

Example 5-10. Declaring a class and examining an object

```
<?php
$object = new User;
print_r($object);

class User  {
public $name, $password;
functionsave_user()
{
echo "Save User code goes here";    }  } ?>
```

Here I have also used an invaluable function called print_r. It asks PHP to display information about a variable in human-readable form. The _r stands for in humanreadable format. In the case of the new object $object, it prints the following:

```
User Object (  [name]     =>  [password] => )
```

However, a browser compresses all the whitespace, so the output in a browser is slightly harder to read:

```
User Object ( [name] => [password] => )
```

In any case, the output says that $object is a user-defined object that has the properties name and password.

Creating an Object

To create an object with a specified class, use the new keyword, like this:

object = new Class.

Here are a couple of ways in which we could do this:

```
$object = new User;
$temp   = new User('name', 'password');
```

On the first line, we simply assign an object to the User class. In the second, we pass parameters to the call. A class may require or prohibit arguments; it may also allow arguments but not require them.

Accessing Objects

Let’s add a few lines to Example 5-10 and check the results. Example 5-11 extends the previous code by setting object properties and calling a method.

Example 5-11. Creating and interacting with an object

```
<?php
$object = new User;
print_r($object);
echo "<br>";
  $object->name     = "Joe";
$object->password = "mypass";
print_r($object); echo "<br>";
 $object->save_user();
class User  {    public $name, $password;
function save_user()    {      echo "Save User code goes here";    }  } ?>
```

Constructors

When creating a new object, you can pass a list of arguments to the class being called. These are passed to a special method within the class, called the *constructor*, which initializes various properties.

Example 5-14. Creating a constructor method

```
<?php
class User
{ function User($param1, $param2) {
// Constructor statements go here
public $username = "Guest"; } } ?>
```

However, PHP 5 provides a more logical approach to naming the constructor, which is to use the function name __construct (that is, construct preceded by two underscore characters), as in Example 5-15.

Example 5-15. Creating a constructor method in PHP 5

```
<?php
class User {
function __construct($param1, $param2)
{ // Constructor statements go here
public $username = "Guest"; } } ?>
```

```
<?php
class A{
 function testA()
 {
 echo "This is test method of class A"; }
 function A() {
 echo "This is user defined constructor of class A"."<br/>";  } }
/* predefined constructor
function  __construct() {
echo "This is user defined constructor of class A"."<br/>"; }
*/
 $obj= new A();
 $obj->testA(); ?>
```

Predefine Constructor

PHP introduce a new functionality to define a constructor i.e __construct().
By using this function we can define a constructor.
It is known as predefined constructor. Its better than user defined constructor because if we change class name then user defined constructor treated as normal method.
If predefined constructor and user defined constructor, both define in the same class, then predefined constructor treat like a Constructor while user defined constructor treated as normal method.

```
<?php
class A {
function A() {
 echo "user defined constructor"; }
 function __construct() {
echo "This is predefined constructor"; } }
$obj= new A(); // This is predefined constructor
$obj->A(); // user defined constructor ?>
```

Parametrized Constructor

```
<?php
class employee{
Public $name;
Public  $profile;
 function __construct($n,$p) {
 $this->name=$n;
 $this->profile=$p;
 echo "Welcome  "; }
function show()
 {
 echo $this->name."... ";
 echo "Your profile is ".$this->profile."<br/>";}}
$obj= new employee("shashi","developer");
 $obj->show();
$obj1= new employee("pankaj","Tester");
 $obj1->show(); ?>
```

Destructor in PHP

The Destructor method will be called as soon as there are no other references to a particular object, or in any order during the shutdown sequence.

Destructor automatically call at last.

Note : _ _destruct() is used to define destructor.

```
<?php
class demo {
 function  __construct(){
 echo  "object is initializing their propertie"."<br/>";
 }
function work(){
 echo  "Now works is going "."<br/>"; }
 function __destruct() {
 echo  "after completion the work,  object  destroyed automatically";}}
$obj= new demo();
 $obj->work();
//to check object is destroyed or not
echo is_object($obj);?>
```

```
o/p: object is initializing their propertie
Now works is going
1after completion the work, object destroyed automatically
```

2.5 Error types: Parse error, Run time errors, Logic errors, Debugging Methodology, Error levels

Types of Errors in PHP

An error is a unwanted result of program. Error is a type of mistake.

An error can also be defined as generate wrong output of program caused by a fault.

An error message display with error message, filename and line number.

An error can be categorized in :

Syntax Error

Logical Error

Run-time Error

The syntax error is a any mistake in symbol or missing symbol in a syntax.

Logical error means you expressed something that is logically incorrect and it will produce unwanted result.

The logical error occur at run-time.

The **run-time error** produced by a logical error. Both logical error and run-time error are same thing.

In PHP there are four types of errors.

- Notice Error
- Fatal Error
- Waring Error
- Parse Error

Parse error or Syntax Error: It is the type of error done by the programmer in the source code of the program. The syntax error is caught by the compiler. After fixing the syntax error the compiler compile the code and execute it. Parse errors can be caused dues to unclosed quotes, missing or Extra parentheses, Unclosed braces, Missing semicolon etc

Example:

```
<?php
$x = "geeks";
y = "Computer science";
echo $x;
echo $y;  ?>
```

o/p: PHP Parse error: syntax error, unexpected '=' in /home/18cb2875ac563160a6120819bab084c8.php on line 3

Fatal Error: It is the type of error where PHP compiler understand the PHP code but it recognizes an undeclared function. This means that function is called without the definition of function.

Example: <?php

```
function add($x, $y) {
    $sum = $x + $y;
    echo "sum = " . $sum; }
$x = 0;
$y = 20;
add($x, $y);
  diff($x, $y);
?>
```

Explanation : In line 12, function is called but the definition of function is not available. So it gives error.

Warning Errors : The main reason of warning errors are including a missing file. This means that the PHP function call the missing file.

Example:

```
<?php
$x = "GeeksforGeeks";
  include ("gfg.php");
echo $x . "Computer science portal"; ?>
```

This program call an undefined file gfg.php which are not available. So it produces error.

Notice Error: It is similar to warning error. It means that the program contains something wrong but it allows the execution of script.

Example:

```
<?php
 $x = "GeeksforGeeks";
echo $x;   echo $geeks; ?>
```

This program use **undeclared variable** $geeks so it gives error message.

Error Report levels

These error report levels are the different types of error the user-defined error handler can be used for:

Value	Constant	Description
2	E_WARNING	Non-fatal run-time errors. Execution of the script is not halted
8	E_NOTICE	Run-time notices. The script found something that might be an error, but could also happen when running a script normally
256	E_USER_ERROR	Fatal user-generated error. This is like an E_ERROR set by the programmer using the PHP function trigger_error()
512	E_USER_WARNING	Non-fatal user-generated warning. This is like an E_WARNING set by the programmer using the PHP function trigger_error()
1024	E_USER_NOTICE	User-generated notice. This is like an E_NOTICE set by the programmer using the PHP function trigger_error()
4096	E_RECOVERABLE_ERROR	Catchable fatal error. This is like an E_ERROR but can be caught by a user defined handle (see also set_error_handler())
8191	E_ALL	All errors and warnings (E_STRICT became a part of E_ALL in PHP 5.4)

Exceptions
Exceptions are used to change the normal flow of a script if a specified error occurs.
What is an Exception
With PHP 5 came a new object oriented way of dealing with errors.
Exception handling is used to change the normal flow of the code execution if a specified error (exceptional) condition occurs. This condition is called an exception.
This is what normally happens when an exception is triggered:

- The current code state is saved
- The code execution will switch to a predefined (custom) exception handler function
- Depending on the situation, the handler may then resume the execution from the saved code state, terminate the script execution or continue the script from a different location in the code

We will show different error handling methods:

- Basic use of Exceptions
- Creating a custom exception handler
- Multiple exceptions
- Re-throwing an exception
- Setting a top level exception handler

Note: Exceptions should only be used with error conditions, and should not be used to jump to another place in the code at a specified point.

Basic Use of Exceptions
When an exception is thrown, the code following it will not be executed, and PHP will try to find the matching "catch" block.
If an exception is not caught, a fatal error will be issued with an "Uncaught Exception" message.

```
<?php
//create function with an exception
function checkNum($number) {
  if($number>1) {
    throw new Exception("Value must be 1 or below");  }
  return true; } //trigger exception
checkNum(2); ?>
```

Try, throw and catch

To avoid the error from the example above, we need to create the proper code to handle an exception.
Proper exception code should include:

- try - A function using an exception should be in a "try" block. If the exception does not trigger, the code will continue as normal. However if the exception triggers, an exception is "thrown"
- throw - This is how you trigger an exception. Each "throw" must have at least one "catch"
- catch - A "catch" block retrieves an exception and creates an object containing the exception information

```
<?php
//create function with an exception
function checkNum($number) {
  if($number>1) {
    throw new Exception("Value must be 1 or below");  }
  return true; }
//trigger exception in a "try" block
try {
  checkNum(2);
  //If the exception is thrown, this text will not be shown
  echo 'If you see this, the number is 1 or below'; } //catch exception
catch(Exception $e) {
  echo 'Message: ' .$e->getMessage(); } ?>
```

Creating a Custom Exception Class

To create a custom exception handler you must create a special class with functions that can be called when an exception occurs in PHP. The class must be an extension of the exception class.

The custom exception class inherits the properties from PHP's exception class and you can add custom functions to it.

Lets create an exception class:

```
<?php
class customException extends Exception {
  public function errorMessage() {
    //error message
    $errorMsg = 'Error on line '.$this->getLine().' in '.$this->getFile()
    .': <b>'.$this->getMessage().'</b> is not a valid E-Mail address';
    return $errorMsg;   } }
$email = "someone@example...com";
try {
  //check if
  if(filter_var($email, FILTER_VALIDATE_EMAIL) === FALSE) {
    //throw exception if email is not valid
    throw new customException($email);   } }
catch (customException $e) {
  //display custom message
  echo $e->errorMessage(); } ?>
```

Multiple Exceptions

It is possible for a script to use multiple exceptions to check for multiple conditions.

It is possible to use several if..else blocks, a switch, or nest multiple exceptions. These exceptions can use different exception classes and return different error messages:

```
<?php
class customException extends Exception {
  public function errorMessage() {
    //error message
    $errorMsg = 'Error on line '.$this->getLine().' in '.$this->getFile()
    .': <b>'.$this->getMessage().'</b> is not a valid E-Mail address';
    return $errorMsg;   } }
$email = "someone@example.com";
try {
  //check if
  if(filter_var($email, FILTER_VALIDATE_EMAIL) === FALSE) {
    //throw exception if email is not valid
    throw new customException($email);   }
  //check for "example" in mail address
  if(strpos($email, "example") !== FALSE) {
    throw new Exception("$email is an example e-mail");   } }
catch (customException $e) {
  echo $e->errorMessage(); }
catch(Exception $e) {
  echo $e->getMessage(); } ?>
```

Rules for exceptions

- Code may be surrounded in a try block, to help catch potential exceptions
- Each try block or "throw" must have at least one corresponding catch block
- Multiple catch blocks can be used to catch different classes of exceptions
- Exceptions can be thrown (or re-thrown) in a catch block within a try block