Introduction to PHP -Online Lesion 2

❖ Bitwise Operators.

Bitwise operators allow evaluation and manipulation of specific bits within an integer.

Bitwise Operators

Example	Name	Result
$a & $b	And	Bits that are set in both $a and $b are set.
$a | $b	Or (inclusive or)	Bits that are set in either $a or $b are set.
$a ^ $b	Xor (exclusive or)	Bits that are set in $a or $b but not both are set.
~ $a	Not	Bits that are set in $a are not set, and vice versa.
$a << $b	Shift left	Shift the bits of $a $b steps to the left (each step means "multiply by two")
$a >> $b	Shift right	Shift the bits of $a $b steps to the right (each step means "divide by two")

Example of PHP Bit Shifting (left shift)

```php
<?php
$x=8;
$y=3;
echo $x << $y;
?>
```

In the below example, the value of $x that is 8 is taken and a BIT SHIFT LEFT operation is performed.

So, 8 is multiplied by 2 thrice. Thus we get **8 x 2 x 2 x 2 = 64**.

Explanation

1 Byte (8 bits)										
Place Value	**128**	**64**	**32**	**16**	**8**	**4**	**2**	**1**		
$x	0	0	0	0	1	0	0	0	=	8
Output	0	1	0	0	0	0	0	0	=	64

2011-14

Consider the following PHP code.

```php
<?php
    $a =11;
    $b =15;
    $c =2;
    $d=30;

    $e=$b+$a*$c;
    $f=$e%$c;
    $g=$e<<$f;

    print $f.$g;
    echo $e;
?>
```

The output is

(a) 174437
(b) 17437
(c) 12757
(d) 11873
(e) 1118377

2016 – 5

5) Which of the following PHP scripts produce(s) the output "**Nice Day**"?

(a)

```php
<?php
    $a = "Nice";
    $b = "Day";
    echo $a . $b;

?>
```

(b)

```php
<?php
    /*
        $a = "Nice";
        $b = "Day";
        echo $a . $b;
    */
?>
```

(c)

```php
<?php
    $a = "Nice";
    $b = " Day";
    echo $a , $b;
?>
```

(d)

```php
<?php
    $a = "Nice";
    $b = "Day";
    echo $a ," ", $b;
 ?>
```

(e)

```php
<?php
    $a = "Nice";
    $b = " Day";
    echo $a + $b;
?>
```

2016 – 6

Which of the following PHP scripts print(s) the value 3 when executed?

(a)

```
<?php
    echo 7/4;
?>
```

(b)

```
<?php
    echo 7%4;
?>
```

(c)

```
<?php
    echo -1*3+12/2;
?>
```

(d)

```
<?php
    echo b11;
?>
```

(e)

```
<?php
    echo 0b11;
?>
```

2018-10

Which of the following PHP scripts is/are syntactically valid?

(a)
```
<?php
    $a = 4;
    echo $a;
?>
```

(b)
```
<php
    $a = $b = 4;
    echo $a,$b;
?>
```

(c)
```
<?php
    $a = 4;
    if ($a == 4){
    echo $a;
    }
?
```

(d)
```
<?php
    $a = 4;
    if ($a){
        echo $a;
    }
```

(e)
```
<?php
    $a = 2;
    while $a==4 {
        $a--;
    }
?>
```

2019-09

Consider the following PHP code segment:

```
if ($a <= $b and $b <= $c){
    echo "True";
}
```

Which of the following values for variable $a, $b and $c, respectively, would produce an output 'True'?

(a)	1,2,3	(b)	3,2,1
(c)	1,1,1	(d)	1,3,2
(e)	2,3,1		

2016 – 3

3) Which of the following PHP scripts is/are syntactically valid?

(a)

```
<?php
    if(true){
       echo "True";
    }
?>
```

(b)

```
<?php

   if(2 != 3)
     echo "In if part";
   else
     echo "In else part";
?>
```

(c)

```
<?php
    if 2 != 3
        echo "In if part";
    else
        echo "In else part";
?>
```

(d)

```
<?php
    if(2 != 3){
        echo "In if part";
    } else {
        echo "In else part";
    }
?>
```

(e)

```
<?php
    If 2 != 3{
        echo "In if part";
    } else {
        echo "In else part";
    }
?>
```

2017-8

Which of the following PHP scripts display(s) the clause **Inside if** when executed?

(a)

```php
<?php

if(true){
    echo "Inside if";
}

?>
```

(b)

```php
<?php

    if(True and 0){
        echo "Inside if";
    }

?>
```

(c)

```php
<?php
if( "some text" or 0){
    echo "Inside if";
}
?>
```

(d)

```php
<?php
    if( "some text" and ""){
        echo "Inside if";
    }
?>
```

(e)

```php
<?php
    $a = [];
    if(!$a){
        echo "Inside if";
    }
?>
```

2017-09

Which of the following PHP scripts is/are syntactically valid?

(a)

```php
<?php
    $a = 4
    echo $a
?>
```

(b)

```php
<?php
    $a = $b = 4;
    echo $a,$b;
?>
```

(c)

```php
<?php
    if $a = 4{
        echo $a;
    }
?>
```

(d)

```php
<?php
    if (($a = 4) > 0){
        echo $a;
    }
?>
```

(e)

```php
<?php
    $a = 2;
    while ($a) {
        echo "Inside while";
        $a--;
    }
?>
```

❖ PHP Literals.

Data literals are ways to provide values for different data types in PHP source code. PHP data literals rules are summarized below:

1. Boolean data literals: Data literals for Boolean data type are two reserved key words: **true** and **false**. Note that true and false are **case in-sensitive** - **TRUE** and **FALSE** are **also valid Boolean literals**.

2. integer data literals: Data literals for integer data type have 4 forms:

```php
<?php

    $a = 1234; // decimal number
    $a = 0123; // octal number (equivalent to 83 decimal)
    $a = 0x1A; // hexadecimal number (equivalent to 26 decimal)
    $a = 0b11111111; // binary number (equivalent to 255 decimal)
    $a = 1_234_567; // decimal number (as of PHP 7.4.0)

?>
```

- To use **octal** notation**, precede the number with a 0 (zero).**
- To use **hexadecimal** notation, **precede the number with 0x.**
- To use **binary** notation, **precede the number with 0b.**
- As of **PHP 7.4.0**, integer literals may contain **underscores (_) between digits**, for **better readability of literals**. These **underscores are removed by PHP's scanner**.

3. Floating point numbers (**also known as** "**floats**", "**doubles**", or "**real numbers**") can be specified using any of the following syntaxes:

```php
<?php
        $a = 1.234;
        $b = 1.2e3;
        $c = 7E-10;
        $d = 1_234.567; // as of PHP 7.4.0
?>
```

The size of a float is platform-dependent, although a maximum of approximately 1.8e308 with a precision of roughly 14 decimal digits is a common value (the 64 bit IEEE format).

We can use the "**gettype()**" function to check type of the variables.

```php
$a = 3;
echo gettype($a) . "<br>";

$b = 3.2;
echo gettype($b) . "<br>";

$c = "Hello";
echo gettype($c) . "<br>";

$d = array();
echo gettype($d) . "<br>";

$f = NULL;
echo gettype($f) . "<br>";

$g = false;
echo gettype($g) . "<br>";
```

4. A string literal can be specified in **four** different ways:

a. **single quoted.**
b. **double quoted.**
c. **heredoc syntax.**
d. **nowdoc syntax (since PHP 5.3.0).**

a. **single quoted.**
b. **double quoted.**

If the string is enclosed in double-quotes ("), PHP will interpret the following escape sequences for special characters:

Escaped characters

Sequence	Meaning
\n	linefeed (LF or 0x0A (10) in ASCII)
\r	carriage return (CR or 0x0D (13) in ASCII)
\t	horizontal tab (HT or 0x09 (9) in ASCII)
\v	vertical tab (VT or 0x0B (11) in ASCII) (since PHP 5.2.5)
\e	escape (ESC or 0x1B (27) in ASCII) (since PHP 5.4.4)
\f	form feed (FF or 0x0C (12) in ASCII) (since PHP 5.2.5)
\\	backslash
\$	dollar sign
\"	double-quote
\[0-7]{1,3}	the sequence of characters matching the regular expression is a character in octal notation, which silently overflows to fit in a byte (e.g. "\400" === "\000")
\x[0-9A-Fa-f]{1,2}	the sequence of characters matching the regular expression is a character in hexadecimal notation
\u{[0-9A-Fa-f]+}	the sequence of characters matching the regular expression is a Unicode codepoint, which will be output to the string as that codepoint's UTF-8 representation (added in PHP 7.0.0)

c. **heredoc syntax**

A third way to delimit strings is the **heredoc** syntax: **<<<.** After this operator, an identifier is provided, then a newline

https://www.php.net/manual/en/language.types.string.php

```php
<?php
    $str = <<<EOD
    Example of string
    spanning multiple lines
    using heredoc syntax.
    EOD;

    echo $str;

    echo <<<EOT
    My name is Amal. I am printing some text.
    This should print a capital 'A': \x41
    EOT;
?>
```

d. Nowdox syntax.

- ✓ **Nowdocs** are to **single-quoted strings** what **heredocs** are to **double-quoted strings**.
- ✓ A **nowdoc is specified similarly to a heredoc**, but no parsing is done inside a nowdoc.
- ✓ A nowdoc is identified with the same <<< sequence used for heredocs,
- ✓ but the **identifier which follows is enclosed in single quotes**, e.g. <<<'EOT'

```php
<?php
    $str = <<<'EOD'
        Example of string
        spanning multiple lines
        using heredoc syntax.
     EOD;

    echo $str;
?>
```

5. array data literals: There are **no data literals for array data type**. Arrays are **created by using the array constructor**.

6. object data literals: There are **no data literals for object data type**. **Objects are created by class constructors**.

7. resource data literals: There are no data literals for resource data type. **Resources** are **created** by **using PHP built-in functions**.

8. null data literal: Data literal for null data type is 1 reserved key words: null. Note that **null is case in-sensitive** - **NULL** is the **same** as **null**.

2017-06

6) Which of the following literal(s) is/are valid in PHP?

(a) `True`	(b) `0x3b2`	(c) `23.2e-2`
(d) `'isn't it'`	(e) `'$a'`	

2019-08

8) Consider following statement about PHP data types.

i) A string is a sequence of characters inside single or double quotes.
ii) 0xab is a valid integer.
iii) 10.36 is a valid integer.

Which of the above statements is/are true?

(a)	i) only	(b)	ii) only
(c)	iii) only	(d)	i)and ii) only
(e)	ii) and iii) only		

❖ PHP Array.

An array is a special variable, which can hold more than one value at a time.

An array can be **created** using the **array ()** language **construct**.

It takes any number of comma-separated **key => value pairs** as arguments.

```
array(
    key  => value,
    key2 => value2,
    key3 => value3,
...
)
```

The key can either be an integer or a string. The value can be of any type.

The **comma after the last array element is optional** and can be omitted.

As of PHP 5.4 you can also use the short array syntax, which replaces *array()* with *[]*.

```
$array = array(
    "foo" => "bar",
    "bar" => "foo",
);

// as of PHP 5.4
$array = [
    "foo" => "bar",
    "bar" => "foo",
];
```

❖ In PHP, there are three types of arrays:

- **Indexed** arrays - Arrays with a numeric index
- **Associative** arrays - Arrays with named keys
- **Multidimensional** arrays - Arrays containing one or more arrays

a. **Indexed** arrays.

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


New value add, replace existing value,
      $cars[3] = “CIVIC” ;
      $cars[0] = “AUDI” ;
Access value,
      $val =$cars[0];
      var_dump($val);
```

2017-10

Which one of the following PHP array declarations is/are valid?

```
(a) $a = array(1,2,3);
(b) $a = array(1='a',2='b',3='c');
(c) $a = array(1=>'a',2=>'b',3=>'c');
(d) $a = array(1->'a',2->'b',3->'c');
(e) $a = array(1=>'a',"2"=>'b',3=>array(1,2));
```

b. Associative arrays.

Associative arrays are arrays that use named keys that you assign to them.

```
$age = array("amal"=>"35", "kamal"=>"37", "nimal"=>"43");


New value add, replace existing value,
     $age['Sunimal'] = "25";
     $age['amal'] = "63";
Access value,
$value =$age["amal"];
var_dump($value);
```

2016-7

Consider the following PHP declaration.

```
$a = array("abc",array(1,2,3),"cde","def");
```

Which of the following statements is/are true?

(a) The value of $a[3] is the string **cde.**	b) The value of $a[3] is the string **def.**
(c) The value of count($a) is 4.	d) The value of count($a) is 6.
(e) The value of count($a) is 0.	

c. **Multidimensional** arrays.

A multidimensional array is an array containing one or more arrays.

PHP supports multidimensional arrays that are two, three, four, five, or more levels deep.

```
$cars = array (
    array("Volvo",22,18),
    array("BMW",15,13),
    array("Saab",5,2),
    array("Land Rover",17,15)
);
```

Now the two-dimensional $cars array contains four arrays, and it has two indices: row and column.

To get access to the elements of the $cars array we must point to the two indices (row and column):

```
echo $cars[0][0].": In stock: ".$cars[0][1].", sold: ".$cars[0][2].".<br>";
echo $cars[1][0].": In stock: ".$cars[1][1].", sold: ".$cars[1][2].".<br>";
echo $cars[2][0].": In stock: ".$cars[2][1].", sold: ".$cars[2][2].".<br>";
echo $cars[3][0].": In stock: ".$cars[3][1].", sold: ".$cars[3][2].".<br>";
```

❖ **Additionally, the following key casts will occur.**

- **Strings containing valid decimal integers, unless the number is preceded by a + sign, will be cast to the integer type**. E.g. the key "8" will actually be stored under 8. On the other hand, **"08" will not be cast**, as it isn't a valid decimal integer.

- **Floats are also cast to integers**, which means that the fractional part will be truncated. E.g. the key 8.7 will actually be stored under 8.

- **Bools are cast to integers**, too, i.e. the key true will actually be stored under 1 and the key false under 0.

- **Null will be cast to the empty string**, i.e. the key null will actually be stored under "".

- **Arrays and objects cannot be used as keys**. Doing so will result in a warning: Illegal offset type.

```php
$array = array(
    1 => "a",
    "1" => "b",
    1.5 => "c",
    true => "d",
);
var_dump($array);
```

Run and Output

```
C:\wamp64\www\test\index.php:8:

array (size=1)
  1 => string 'd'
(length=1)
```

If **multiple elements in the array declaration use the same key**, **only the last one will be used as all others are overwritten.**

Explanation

As all the keys in the above example are cast to 1, the value will be overwritten on every new element and the last assigned value "d" is the only one left over.

- **PHP arrays can contain integer and string keys at the same time as PHP does not distinguish between indexed and associative arrays.**

```php
$array = array(
    "foo" => "bar",
    "bar" => "foo",
    100 => -100,
    -100 => 100,
);
var_dump($array);
```

Run and Output

```
C:\wamp64\www\test\index.php:8:
array (size=4)
  'foo' => string 'bar' (length=3)
  'bar' => string 'foo' (length=3)
  100 => int -100
  -100 => int 100
```

- The key is optional. If it is not specified, PHP will use the increment of the largest previously used integer key.

```php
$array = array("foo", "bar", "hello", "world");
var_dump($array);
```

```
C:\wamp64\www\test\index.php:4:
array (size=4)
  0 => string 'foo' (length=3)
  1 => string 'bar' (length=3)
  2 => string 'hello' (length=5)
  3 => string 'world' (length=5)
```

- It is possible to specify the key only for some elements and leave it out for others:

```php
$array = array(
    "a",
    "b",
    6 => "c",
    "d",
);
var_dump($array);
```

Output

```
C:\wamp64\www\test\index.php:8:
array (size=4)
  0 => string 'a' (length=1)
  1 => string 'b' (length=1)
  6 => string 'c' (length=1)
  7 => string 'd' (length=1)

As you can see the last value "d" was assigned the key 7. This is because the largest integer key before that was 6.
```

As you can see the last value "d" was assigned the key 7. This is because the largest integer key before that was 6.

❖ Accessing array elements with square bracket syntax. Array elements can be accessed using the array[key] syntax.

```php
$array = array(
    "foo" => "bar",
    42    => 24,
    "multi" => array(
         "dimensional" => array(
             "array" => "foo"
         )
    )
);

var_dump($array["foo"]);
var_dump($array[42]);
var_dump($array["multi"]["dimensional"]["array"]);
```

Outputs:

```
C:\wamp64\www\test\index.php:13:string 'bar' (length=3)
C:\wamp64\www\test\index.php:14:int 24
C:\wamp64\www\test\index.php:15:string 'foo' (length=3)
```

Note:
Both square brackets and curly braces can be used interchangeably for accessing array elements.

(e.g. *$array[42]* and *$array{42}* will both do the same thing in the example above).

As of PHP 5.4 it is possible to array dereference the result of a function or method call directly. Before it was only possible using a temporary variable.

AQ-1

What is the output of the following PHP code?

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

Select one or more:

a. I like Volvo and BMW
b. Error
c. I like Toyota and BMW
d. I like BMW and Volvo
e. I like BMW and Toyota

AQ-2

What is the output of the following PHP code?

```php
<?php
    $fruits =array("bananna","pineapple",array("apple","mango"),"guava");
    echo(count($fruits,1))
?>
```

Select one or more:

a. 0
b. 5
c. 4
d. 3
e. 6

AQ-3

What is the output of the following PHP code?

```php
<?php
   $car ="maruti";
   $var =$car[2];
   echo "$var";
?>
```

Select one or more:

a. maruti
b. $var
c. Error
d. r
e. a

❖ Conditional Statements.

a) **if statement** -executes some code only if a specified condition is true.
b) **if...else statement**-executes some code if a condition is true and another code if the condition is false.
c) **if...elseif....else statement**-selects one of several blocks of code to be executed.
d) **switch statement**-selects one of many blocks of code to be executed.

a. if statement

```
if (<condition>) {

}
```

b. if...else statement

```
if (<condition>) {

}

else {

}
```

```
$color="Red";

if ($color =="Red") {
    echo "Please STOP";
} else {
    echo "You can GO";
}
```

c. if...elseif....else statement

```
if (condition) {


} elseif(condition) {


} else {


}
```

```
$color="Red";

if ($color =="Red") {
    echo "Please Stop" ;
} elseif($color =="Yellow") {
    echo "Get ready " ;
} else {
    echo "You can GO";
}
```

d. switch statement

```
switch ($favcolor) {

  case "red":echo "Your favorite color is red!";
      break;

  case "blue":echo "Your favorite color is blue!";
      break;

  case "green":
      echo "Your favorite color is green!";
      break;

  default:echo "Your favorite color is neither
red, blue, or green!";
}
```

2011-13

Consider the following PHP code

```php
$x = $y = $z = -1;

switch ($y) {
    case 1:
        echo "Number 1";
        break;
    case 2:
        echo "Number 2";
        break;
    default:
        echo "Number is not between 1 and 2";
}
```

The output is

(a) Number 2.
(b) Number 1.
(c) Number which is not between 1 and 2.
(d) Does not display anything.
(e) Error.

2015-08

Consider the following PHP script.,

```php
$light= "red";

switch ($light) {
    case "red":
        echo "Stop!!";
    case "yellow":
        echo "Get ready";
    break;
    case "green":
         echo "Go..";

}
```

What would be the output of the script when it is executed?

(a)	Stop!!	(b)	Stop!!Get ready
(c)	Stop!!Get readyGo..	(d)	Stop!!Go..
(e)	Error Message		

2016-08

Consider the following PHP script.

```
$light = array("red","yellow","green");
$colourNo= 1;

switch ($light[1]) {
    case "red":
        echo "Stop!! ";
        break;
    case "yellow":
        echo "Get ready ";
    case "green":
        echo "Go.. ";
        break;
    default:
        echo "Color not defined";
}
```

What would be the output of the script when it is executed?

(a) Stop!!	(b) Color not defined
(c) Go..	(d) Get ready
(e) Get ready Go..	

2011-15

Consider the following code fragment.

```
$x = "May";
$months = array("Apr","Nov","Dec", "May");

foreach $value in $months {

    if ($value == $x)
            print $value;
}
```

The output is

(a) An error message.
(b) Apr.
(c) Nov.
(d) Dec.
(e) May.

❖ Loops.

- When you need the same block of code to be executed over and over again.
- In PHP, we have the following looping statements:

e. while

loops through as long as the given condition is true.

```
while (condition is true) {
  //Do this;
}
```

```php
$i=1;
 while($i<=5) {
    echo "Number: $i<br>";
    $i++;
 }
```

f. do...while

loops through the code once, and then repeats the loop as long as the given condition is true

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

```php
$i=1;
do {
    echo "Number: $i<br>";
    $i++;
} while ($i<=5);
```

g. for

loops through the code a given number of times.

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

h. foreach

loops through the code for each element in a collection.

```
foreach ($array as $value)
{
 //Do this
}
```

```php
$person = array("Nimal","Kamal","Sunil","Amal");

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

2019-13

Consider the following PHP script:

```php
$a = array(1,2,3,4,5);
$x = 0;
$i = 1;

while ($i < 4){
    if ($i % 2 == 0){
        $x = $x + $a[$i];
    }
    $i++;
}
echo $x;
```

What would be the output of the script when it is executed?

(a) 3	(b) 5	(c) 6
(d) 9	(e) 14	

2019-15

Consider the following,

```php
<?php
    $a = array(1=>"a",2=>"b",3=>"c",4=>"d");
    $x = 0;
    $y = "";

    foreach ($a as $i=>$j){
        $x += $i;
        $y .= $j;
    }
    echo $x,"-",$y;
?>
```

What would be the output of the script when it is executed?

(a) 1234	(b) abcd	(c) 1234-abcd
(d) 10-abcd	(e) 1-1	

2018-9

Which of the following PHP script(s) display the value 12 when executed?

(a)

```php
<?php
    $a = array(1,2,3,4,5,6);
    $sum = 0;$i = 0;
        while ($i < count($a)){
            if ($i % 2){
                $sum = $sum + $i;
            }
            $i++;
        }
    echo $sum;
?>
```

(b)

```php
<?php
$a = array(1,2,3,4,5,6);
$sum = 0;
    foreach ($a as $key=>$val){
        if ($val % 2){
            $sum = $sum + $key;
        }
    }
echo $sum;
?>
```

(c)

```php
<?php
$a = array(1,2,3,4,5,6);
$sum = 0;$i = 0;
    while ($i < count($a)){
        if ($i % 2){
            $sum = $sum +$a[$i];
        }
        $i++;
    }
echo $sum;
?>
```

(d)

```php
<?php
    $a = array(1,2,3,4,5,6);
    $sum = 0;
        foreach ($a as $key=>$val){
           if ($key % 2){
              $sum = $sum + $val;
           }
        }
    echo $sum;
?
```

(e)

```php
<?php
    $a = array(1,2,3,4,5,6);
    $sum = array_sum($a);
    echo $sum;
?>
```