

# Unit-1 PHP

The INSB IITMS BCA College, Idar

# PHP Constants

→ A constant is an identifier (name) for a simple value. The value cannot be changed during the script.

→ A valid constant name starts with a letter or underscore (no \$ sign before the constant name).

**Note:** Unlike variables, constants are automatically global across the entire script.

→ PHP Constant Syntax: use the define() function

→ define(name, value, case sensitive)

```
define("abc", "Welcome to BCA College idar!");  
echo abc;
```



# Echo & Print

- Echo has no return value While print is more like PHP function so it return value, Which is always set 1.
- Echo can take multiple parameters while print can take one argument.
- echo and print are more or less the same. They are both used to output data to the screen.
- echo is marginally faster than print.



# Datatype

→ Variables can store data of different types, and different data types can do different things.

PHP supports the following data types:

- 1) String
- 2) Integer
- 3) Float (floating point numbers - also called double)
- 4) Boolean
- 5) Array
- 6) Object
- 7) NULL



# PHP Operators

→ Operators are used to perform operations on variables and values.

PHP divides the operators in the following groups:

- 1) Arithmetic operators
- 2) Assignment operators
- 3) Comparison operators
- 4) Increment/Decrement operators
- 5) Logical operators
- 6) String operators
- 7) Conditional assignment operators
- 8) Bitwise operators
- 9) Precedence and associativity



# Arithmetic Operator

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

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

# Assignment Operator

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

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

# Comparison Operator

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

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



# Increment / Decrement Operator

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

Operator	Same as...	Description
<code>++\$x</code>	Pre-increment	Increments <code>\$x</code> by one, then returns <code>\$x</code>
<code>\$x++</code>	Post-increment	Returns <code>\$x</code> , then increments <code>\$x</code> by one
<code>--\$x</code>	Pre-decrement	Decrements <code>\$x</code> by one, then returns <code>\$x</code>
<code>\$x--</code>	Post-decrement	Returns <code>\$x</code> , then decrements <code>\$x</code> by one



# Logical Operator

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

Operator	Name	Example	Result
and	And	<code>\$x and \$y</code>	True if both <code>\$x</code> and <code>\$y</code> are true
or	Or	<code>\$x or \$y</code>	True if either <code>\$x</code> or <code>\$y</code> is true
xor	Xor	<code>\$x xor \$y</code>	True if either <code>\$x</code> or <code>\$y</code> is true, but not both
&&	And	<code>\$x &amp;&amp; \$y</code>	True if both <code>\$x</code> and <code>\$y</code> are true
	Or	<code>\$x    \$y</code>	True if either <code>\$x</code> or <code>\$y</code> is true
!	Not	<code>!\$x</code>	True if <code>\$x</code> is not true



# String Operator

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

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



# Conditional Assignment Operators

→ The PHP conditional assignment operators are used to set a value depending on conditions.

Operator	Name	Example	Result
<code>?:</code>	Ternary	<code>\$x = expr1 ? expr2 : expr3</code>	Returns the value of <code>\$x</code> . The value of <code>\$x</code> is <code>expr2</code> if <code>expr1 = TRUE</code> . The value of <code>\$x</code> is <code>expr3</code> if <code>expr1 = FALSE</code>
<code>??</code>	Null coalescing	<code>\$x = expr1 ?? expr2</code>	Returns the value of <code>\$x</code> . The value of <code>\$x</code> is <code>expr1</code> if <code>expr1</code> exists, and is not <code>NULL</code> . If <code>expr1</code> does not exist, or is <code>NULL</code> , the value of <code>\$x</code> is <code>expr2</code> . Introduced in PHP 7



# Bitwise Operators

→ Bitwise operators are used on (binary) numbers.

Operator	Name	Description	Example
&	AND	Sets each bit to 1 if both bits are 1	<code>x &amp; y</code>
	OR	Sets each bit to 1 if one of two bits is 1	<code>x   y</code>
^	XOR	Sets each bit to 1 if only one of two bits is 1	<code>x ^ b</code>
<<	Zero fill left shift	Shift left by pushing zeros in from the right	<code>x &lt;&lt; 2</code>
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off	<code>x &gt;&gt; 2</code>



# Operator Precedence and Associativity

→ The precedence of an operator specifies how "tightly" it binds two expressions together. For example, in the expression  $1 + 5 * 3$ , the answer is 16 and not 18 because the multiplication (" $*$ ") operator has a higher precedence than the addition (" $+$ ") operator. Parentheses may be used to force precedence, if necessary. For instance:  $(1 + 5) * 3$  evaluates to 18.

→ When operators have equal precedence their associativity decides how the operators are grouped. For example " $-$ " is left-associative, so  $1 - 2 - 3$  is grouped as  $(1 - 2) - 3$  and evaluates to -4. " $=$ " on the other hand is right-associative, so  $\$a = \$b = \$c$  is grouped as  $\$a = (\$b = \$c)$ .

→ Operators of equal precedence that are non-associative cannot be used next to each other, for example  $1 < 2 > 1$  is illegal in PHP. The expression  $1 <= 1 == 1$  on the other hand is legal, because the `==` operator has a lower precedence than the `<=` operator.



Associativity	Operators	Additional Information
(n/a)	clone new	<a href="#">clone</a> and <a href="#">new</a>
right	**	<a href="#">arithmetic</a>
(n/a)	+ - ++ -- ~ (int) (float) (string) (array) (object) (bool) @	<a href="#">arithmetic</a> (unary + and -), <a href="#">increment/decrement</a> , <a href="#">bitwise</a> , <a href="#">type casting</a> and <a href="#">error control</a>
left	instanceof	<a href="#">type</a>
(n/a)	!	<a href="#">logical</a>
left	* / %	<a href="#">arithmetic</a>
left	+ - .	<a href="#">arithmetic</a> (binary + and -), <a href="#">array</a> and <a href="#">string</a> (. prior to PHP 8.0.0)
left	<< >>	<a href="#">bitwise</a>
left	.	<a href="#">string</a> (as of PHP 8.0.0)
non-associative	< <= > >=	<a href="#">comparison</a>
non-associative	== != === !== <> <=>	<a href="#">comparison</a>
left	&	<a href="#">bitwise</a> and <a href="#">references</a>
left	^	<a href="#">bitwise</a>
left		<a href="#">bitwise</a>



# Variable Handling Functions

Function	Description
<u>boolval()</u>	Returns the boolean value of a variable
<u>debug_zval_dump()</u>	Dumps a string representation of an internal zend value to output
<u>doubleval()</u>	Alias of <u>floatval()</u> .
<u>empty()</u>	Checks whether a variable is empty
<u>floatval()</u>	Returns the float value of a variable
<u>get_defined_vars()</u>	Returns all defined variables, as an array
<u>get_resource_type()</u>	Returns the type of a resource
<u>gettype()</u>	Returns the type of a variable
<u>intval()</u>	Returns the integer value of a variable
<u>is_array()</u>	Checks whether a variable is an array
<u>is_bool()</u>	Checks whether a variable is a boolean
<u>is_callable()</u>	Checks whether the contents of a variable can be called as a function
<u>is_countable()</u>	Checks whether the contents of a variable is a countable value
<u>is_double()</u>	Alias of <u>is_float()</u> .
<u>is_float()</u>	Checks whether a variable is of type float



<u>is_long()</u>	Alias of <u>is_int()</u>
<u>is_null()</u>	Checks whether a variable is NULL
<u>is_numeric()</u>	Checks whether a variable is a number or a numeric string
<u>is_object()</u>	Checks whether a variable is an object
<u>is_real()</u>	Alias of <u>is_float()</u>
<u>is_resource()</u>	Checks whether a variable is a resource
<u>is_scalar()</u>	Checks whether a variable is a scalar
<u>is_string()</u>	Checks whether a variable is of type string
<u>isset()</u>	Checks whether a variable is set (declared and not NULL)
<u>print_r()</u>	Prints the information about a variable in a human-readable way
<u>serialize()</u>	Converts a storable representation of a value
<u>settype()</u>	Converts a variable to a specific type
<u>strval()</u>	Returns the string value of a variable
<u>unserialize()</u>	Converts serialized data back into actual data
<u>unset()</u>	Unsets a variable
<u>var_dump()</u>	Dumps information about one or more variables
<u>var_export()</u>	Returns structured information (valid PHP code) about a variable

# PHP Conditional Statements

→ Very often when you write code, you want to perform different actions for different conditions. You can use conditional statements in your code to do this.

→ In PHP we have the following conditional statements:

**1) If statement** - executes some code if one condition is true

Ex. if (*condition*)

{

*// code to be executed if condition is true;*

}

→ if (5 > 3) { echo "Have a good day!"; }



# PHP Conditional Statements

2) **If...else statement** - executes some code if a condition is true and another code if that condition is false

Syntax:-

```
if (condition)
```

```
{
```

```
    // code to be executed if condition is true;
```

```
} else
```

```
{
```

```
    // code to be executed if condition is false;
```

```
}
```

→ `If($a==$b) { echo"True"; } else { echo " False"; }`



# PHP Conditional Statements

## 3) If...else If...else statement -

The if...elseif...else statement executes different codes for more than two conditions.

Syntax:-


```
if (condition)  
{  
    // code to be executed if condition is true;  
} else if (condition)  
{  
    // code to be executed if condition is false and this  
condition is true;  
} else {  
    // code to be executed if all conditions are false;  
}
```

---

4) **Switch statement** - The switch statement is used to perform different actions based on different conditions.

Syntax:-

```
switch(expression) {  
    case label1:  
        //code block  
        break;  
    case label2:  
        //code block;  
        break;  
    case label3:  
        //code block  
        break;  
    default:  
        //code block }  
}
```



# PHP Loops and Branching Statements

→ when you write code, you want the same block of code to run over and over again a certain number of times. So, instead of adding several almost equal code-lines in a script, we can use loops.

Loops are used to execute the same block of code again and again, as long as a certain condition is true.

In PHP, we have the following loop types:

- 1) **While** - loops through a block of code as long as the specified condition is true
- 2) **Do...while** - loops through a block of code once, and then repeats the loop as long as the specified condition is true
- 3) **for- loops**- through a block of code a specified number of times
- 4) **foreach**- loops through a block of code for each element in an array
- 5) **Break**-The break statement can be used to jump out of a for loop.
- 6) **Continue**- The continue statement stops the current iteration in the for loop and continue with the next.

# PHP Loops and Branching Statements

→ 1) **While** - loops through a block of code as long as the specified condition is true.

Syntax:-

```
while(condition)
{
    //code to be executed
}
```

For ex.

```
$i = 1;
while ($i < 6)
{
    echo $i;
    $i++;
}
```

**Alternative Syntax:**

```
while(condition):
    code to be executed
```

---

▶ Endwhile;

2) **Do...while** - loops through a block of code once, and then repeats the loop as long as the specified condition is true.

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

For ex.    \$i = 1;  
              do { echo \$i;  
                      \$i++;  
                      } while (\$i < 6);

Ex.

```
$i = 1;  
do {  
  if ($i == 3) break;  
  echo $i;  
  $i++;  
} while ($i < 6);
```

---





# PHP Loops and Branching Statements

3) **for- loops**- The for loop is used when you know how many times the script should run.

**Syntax:-** for (expression1, expression2, expression3)  
{  
    // code block  
}

**Ex.**

```
for ($n = 1; $n <= 10; $n++)  
{  
    echo "$n ";  
}
```

// Output: 1 2 3 4 5 6 7 8 9 10

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

---



# PHP Loops and Branching Statements

4) **foreach**- The foreach loop - Loops through a block of code for each element in an array or each property in an object. The most common use of the foreach loop, is to loop through the items of an array.

## Syntax:

```
foreach ($array as $var) {  
    //code to be executed  
}
```

Ex.

```
$colors = array("red", "green", "blue", "yellow");  
    foreach ($colors as $x)  
    {  
        echo "$x <br>";  
    }
```



# PHP Loops and Branching Statements

5) **Break**-The break statement can be used to jump out of a for loop.  
The break statement can be used to jump out of a for loop.

Ex.

```
for ($x = 0; $x < 10; $x++)  
{  
    if ($x == 5)  
    {  
        break;  
    }  
    echo "The number is: $x <br>";  
}
```



# PHP Loops and Branching Statements

6) **Continue**- The continue statement stops the current iteration in the for loop and continue with the next.

Ex.

```
$x = 0;
while($x < 10)
{
    if ($x == 4)
    {
        continue;
    }
    echo "The number is: $x <br>";
    $x++;
}
```

