Operators

Disclaimer: вы смотрите просто запись лекции,
это **НЕ** специально подготовленный видеокурс!

Main definition

An **operator** is something that takes one or more values (or expressions, in programming jargon) and yields another value (so that the construction itself becomes an expression).

Three types of operators

Unary operators take only one value:

```php
<?php

$someNewValue = !$someOldValue;
```

Binary operators take two values:

```php
<?php

$someVariable = $operandA + $operandB;
```

Ternary operator takes three values:

```php
<?php

$action = (empty($_POST['action'])) ? 'default' : $_POST['action'];
```

Operator precedence

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.

Operator precedence

Do not try to memorize this table, just use ( ) !

Associativity	Operators
(n/a)	clone and new
right	**
(n/a)	+ - ++ -- ~ (int) (float) (string) (array) (object) (bool) @
left	Instanceof
(n/a)	!
left	* / %
left	+ - .
left	<< >>
left	.
non-associative	< <= > >=
non-associative	== != === !== <> <=>
left	&
left	^
left	|
left	&&
left	||
right	??
non-associative	? :
right	= += -= *= **= /= .= %= &= |= ^= <<= >>= ??=
(n/a)	yield from
(n/a)	yield
(n/a)	print
left	and
left	xor
left	or

All operators varieties…

- Arithmetic Operators
- Assignment Operators
- Bitwise Operators
- Comparison Operators
- Error Control Operators
- Execution Operators
- Increment/Decrement Operators
- Logical Operators
- String Operators
- Array Operators
- Type Operators

Arithmetic Operators

Most of these operators work identically in most programming languages:

Example	Name	Result
+$a	Identity	Conversion of $a to int or float as appropriate.
-$a	Negation	Opposite of $a.
$a + $b	Addition	Sum of $a and $b.
$a - $b	Subtraction	Difference of $a and $b.
$a * $b	Multiplication	Product of $a and $b.
$a / $b	Division	Quotient of $a and $b.
$a % $b	Modulo	Remainder of $a divided by $b.
$a ** $b	Exponentiation	Result of raising $a to the $b'th power.

Assignment Operators

The basic assignment operator is "=". The value of an assignment expression is the value assigned. That is, the value of "$a = 3" is 3.

```php
<?php

$someVariable = 5;

$someVariable = $someVariable * 12;

// Do NOT do this! It complicates your code readability!
$someVariable = ($someOtherVariable = 4) + 5;
```

Combined Assignment Operators

In addition to the basic assignment operator, there are "combined operators" for all of the binary arithmetic, array union and string operators that allow you to use a value in an expression and then set its value to the result of that expression.

```php
<?php

$someVariable = 3;
$someVariable += 5; // $someVariable = $someVariable + 5

$someOtherVariable = "Hello ";
$someOtherVariable .= "There!"; // $someOtherVariable = $someOtherVariable . "There!"
```

Assignment by Reference

Assignment by reference means that both variables end up pointing at the same data, and nothing is copied anywhere.

```php
<?php

$basicVariable = 'Test';
$synonymVariable = &$basicVariable;

echo $synonymVariable; // 'Test'

$synonymVariable = 'OK'; // This also changes $basicVariable

echo $basicVariable; // 'OK'
```

Assignment by Reference: pros and cons

Assignment by reference may significantly simplify the code, still it brings some **potential problem**.

```php
<?php

$someArray = [5, 7, 11];

foreach ($someArray as &$value) {
    $value++;
}

print_r($someArray); // [6, 8, 12]

// If you forget this unset, the behavior may drive you crazy:
// unset($value);

foreach ($someArray as $key => $value) {
    echo $key . ' = ' . $value . "\n"; // 0 = 6, 1 = 8, 2 = 8
}

print_r($someArray); // [6, 8, 8]
```

Bitwise Operators

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

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")

Bitwise Operators

0	&	0	=	0
0	&	1	=	0
1	&	0	=	0
1	&	1	=	1

0	|	0	=	0
0	|	1	=	1
1	|	0	=	1
1	|	1	=	1

0	^	0	=	0
0	^	1	=	1
1	^	0	=	1
1	^	1	=	0

```php
<?php

define('EXECUTE_PERMISSION', 0b0001);
define('WRITE_PERMISSION',   0b0010);
define('READ_PERMISSION',    0b0100);
define('OWNER_PERMISSION',   0b1000);

$userPermissions = 0b0000;

// Add write permission.
$userPermissions = $userPermissions | WRITE_PERMISSION;
echo $userPermissions . "\n"; // 2 (0010)

if ($userPermissions & WRITE_PERMISSION) {
    echo "The user has write permission\n";
}

// Revoke write permission.
$userPermissions = $userPermissions ^ WRITE_PERMISSION;
echo $userPermissions . "\n"; // 0 (0000)
```

Comparison Operators

Comparison operators, as their name implies, allow you to compare two values.

Example	Name	Result
$a == $b	Equal	true if $a is equal to $b after type juggling.
$a === $b	Identical	true if $a is equal to $b, and they are of the same type.
$a != $b	Not equal	true if $a is not equal to $b after type juggling.
$a <> $b	Not equal	true if $a is not equal to $b after type juggling.
$a !== $b	Not identical	true if $a is not equal to $b, or they are not of the same type.
$a < $b	Less than	true if $a is strictly less than $b.
$a > $b	Greater than	true if $a is strictly greater than $b.
$a <= $b	Less than or equal to	true if $a is less than or equal to $b.
$a >= $b	Greater than or equal to	true if $a is greater than or equal to $b.
$a <=> $b	Spaceship	An int less than, equal to, or greater than zero when $a is less than, equal to, or greater than $b, respectively.

Comparison Operators

```php
<?php

// Integers
echo 1 <=> 1; // 0
echo 1 <=> 2; // -1
echo 2 <=> 1; // 1

// Floats
echo 1.5 <=> 1.5; // 0
echo 1.5 <=> 2.5; // -1
echo 2.5 <=> 1.5; // 1

// Strings
echo "a" <=> "a"; // 0
echo "a" <=> "b"; // -1
echo "b" <=> "a"; // 1

echo "a" <=> "aa"; // -1
echo "zz" <=> "aa"; // 1
```

```php
<?php

// Arrays
echo [] <=> []; // 0
echo [1, 2, 3] <=> [1, 2, 3]; // 0
echo [1, 2, 3] <=> []; // 1
echo [1, 2, 3] <=> [1, 2, 1]; // 1
echo [1, 2, 3] <=> [1, 2, 4]; // -1

// Objects
$a = (object)["a" => "b"];
$b = (object)["a" => "b"];
echo $a <=> $b; // 0

$a = (object)["a" => "b"];
$b = (object)["a" => "c"];
echo $a <=> $b; // -1

$a = (object)["a" => "c"];
$b = (object)["a" => "b"];
echo $a <=> $b; // 1

// Not only values are compared, keys must match:
$a = (object)["a" => "b"];
$b = (object)["b" => "b"];
echo $a <=> $b; // 1
```

Mind types juggling!

Mind types juggling! Or simple use ===.

```php
<?php

// With ==
$someVariableOne = '0';
$someVariableTwo = 0;
var_dump($someVariableOne == $someVariableTwo); // bool(true)

// With ===
$someVariableOne = '0';
$someVariableTwo = 0;
var_dump($someVariableOne === $someVariableTwo); // bool(false)
```

Comparison Operators, Ternary Operator

Another conditional operator is the "?:" (or ternary) operator.

```php
<?php

// Ternary Operator
$action = (empty($_POST['action'])) ? 'default' : $_POST['action'];

// The above is identical to this if/else statement
if (empty($_POST['action'])) {
    $action = 'default';
} else {
    $action = $_POST['action'];
}
```

Comparison Operators, Null Coalescing Operator

This operator automates checks for **null**s.

```php
<?php

// Null Coalesce Operator
$action = $_POST['action'] ?? 'default';

// The above is identical to this if/else statement
if (isset($_POST['action'])) {
    $action = $_POST['action'];
} else {
    $action = 'default';
}
```

Comparison Operators, Null Coalescing Operator

Since PHP 8 there is a possibility to automate checks for property existence:

```php
<?php

// This approach...
$name = $session?->user?->getPersonalData()?->name;

// ... is equivalent to this one:
if ($session !== null) {
    $user = $session->user;
    if ($user !== null) {
        $data = $user->getPersonalData();
        if ($address !== null) {
            $name = $data->name;
        }
    }
}
```

PHP and errors

PHP follows the next idea: if an error still allows to continue script execution, let's execute.

But the error/warning message will be displayed, and that:

- Looks bad (no user wants to see bunch of warnings/errors spread across the application).
- May help intruders to analyze and hack the application.

So, there is a way to hide error/warning messages.

This is NOT a way to deal with errors (any error situation should be handled), still – it helps in "emergency situations".

Error Control Operator

PHP supports one error control operator: the at sign (@). When prepended to an expression in PHP, any diagnostic error that might be generated by that expression will be suppressed.

```php
<?php

// Imagine that fle does not exist:
$someData = @file('non_existing_file');

// @ also works with expressions
$someValue = @$someArray[$someKey];
```

Error Control Operator, alternative approach

It is a good idea to keep error_reporting option set to 0 in php.ini and/or to use error_reporting(0) in production environment.

```php
<?php

// Turns off all error messages (for this particular script execution)
error_reporting(0);

// No more error message here
$someData = file('non_existing_file');

// No more error message here also
$someValue = $someArray[$someKey];
```

Execution Operator

PHP supports one execution operator: backticks (` `). Note that these are not single-quotes!

```php
<?php

$output = `ver`;
echo $output; // Microsoft Windows [Version 10.0.19044.1415]
```

See also “Program execution Functions”:
https://www.php.net/manual/en/ref.exec.php

Execution Operator

This approach allows you to utilize external tools for miscellaneous operations. Imagine, you want to use ffmpeg fo some video processing…

```php
<?php

// ... Some preparations here ...

$xecutionString =
    "ffmpeg -i \"" . $originalVideoFileName . "\"" .
    " -i \"" . $firstAndLastPartsReplacementFileName . "\"" .
    " -i \"" . $shortLogoReplacementFileName . "\"" .
    " -i \"" . $longLogoReplacementFileName . "\"" .
    " -c:v libx264 -crf 0 -c:a copy -filter_complex" .
    " \"[0][1]overlay=W-w:H-h:enable='between(t,0," . $leadingLogoEndTime . ")'[v1];" .
    " [v1][2]overlay=y=H-h:enable='between(t," . $leadingLogoEndTime . "," . $longLogoStartTime . ")'[v2];" .
    " [v2][3]overlay=y=H-h+5:enable='between(t," . $longLogoStartTime . "," . $longLogoEndTime . ")'[v3];" .
    " [v3][2]overlay=y=H-h:enable='between(t," . $longLogoEndTime . "," . $trailingLogoStartTime . ")'[v4];" .
    " [v4][1]overlay=y=H-h:enable='between(t," . $trailingLogoStartTime . "," . $trailingLogoStartTime + 100 . ")'[v5]\"" .
    " -pix_fmt yuv420p -map \"[v5]\" -map 0:a \"" . $finalVideoFileName . "\"";
}

exec($executionString);
```

Incrementing/Decrementing Operators

PHP supports C-style pre- and post-increment and decrement operators.

Example	Name	Effect
++$a	Pre-increment	Increments $a by one, then returns $a.
$a++	Post-increment	Returns $a, then increments $a by one.
--$a	Pre-decrement	Decrements $a by one, then returns $a.
$a--	Post-decrement	Returns $a, then decrements $a by one.

Incrementing/Decrementing Operators

```php
<?php

// Postincrement
$someVariable = 5;
echo "Should be 5: " . $someVariable++ . "\n";
echo "Should be 6: " . $someVariable . "\n";

// Preincrement
$someVariable = 5;
echo "Should be 6: " . ++$someVariable . "\n";
echo "Should be 6: " . $someVariable . "\n";

// Postdecrement
$someVariable = 5;
echo "Should be 5: " . $someVariable-- . "\n";
echo "Should be 4: " . $someVariable . "\n";

// Predecrement
$someVariable = 5;
echo "Should be 4: " . --$someVariable . "\n";
echo "Should be 4: " . $someVariable . "\n";
```

Incrementing/Decrementing Operators, check your knowledge

What is the result of this script execution?

```php
<?php

$i = 2;
$i += $i++ + ++$i;
echo $i;
```

NEVER do such things in real production code! It completely destroys code readability.

Logical Operators

The reason for the two different variations of "and" and "or" operators is that they operate at different precedence. (See “Operator Precedence” earlier.)

Example	Name	Result
$a and $b	And	true if both $a and $b are true.
$a or $b	Or	true if either $a or $b is true.
$a xor $b	Xor	true if either $a or $b is true, but not both.
! $a	Not	true if $a is not true.
$a && $b	And	true if both $a and $b are true.
$a || $b	Or	true if either $a or $b is true.

Logical Operators

The most common case of logical operator usage is compound condition.

```php
<?php

if (((($dayOfWeek >= 6) && ($dayOfWeek <= 7)) || ($dayType == DAY_HOLYDAY)) {
    // ...
}
```

String Operator

PHP has only one string operator – concatenation operator ('.'), which returns the concatenation of its right and left arguments (it also works as concatenating assignment operator ('.=')).

```php
<?php

$someVariable = "Hello ";
$someAnotherVariable = $someVariable . "World!"; // now $someAnotherVariable contains
"Hello World!"

$someVariable = "Hello ";
$someVariable .= "World!";      // now $someVariable contains "Hello World!"
```

Array Operators

While PHP supports several array operators, it is recommended to use corresponding functions instead – just to increase code readability.

Example	Name	Result
$a + $b	Union	Union of $a and $b.
$a == $b	Equality	true if $a and $b have the same key/value pairs.
$a === $b	Identity	true if $a and $b have the same key/value pairs in the same order and of the same types.
$a != $b	Inequality	true if $a is not equal to $b.
$a <> $b	Inequality	true if $a is not equal to $b.
$a !== $b	Non-identity	true if $a is not identical to $b.

Array Operators

```php
<?php

$someArrayOne = array("a" => "apple", "b" => "banana");
$someArrayTwo = array("a" => "pear", "b" => "strawberry", "c" => "cherry");

// Union (one + two)
$someArrayThree = $someArrayOne + $someArrayTwo;
print_r($someArrayThree); // [a] => apple, [b] => banana, [c] => cherry

// Union (two + one)
$someArrayThree = $someArrayTwo + $someArrayOne;
print_r($someArrayThree); // [a] => pear, [b] => strawberry, [c] => cherry

// Union (one = one + two)
$someArrayOne += $someArrayTwo;
print_r($someArrayOne); // [a] => apple, [b] => banana, [c] => cherry

// Comparison
$someArrayOne = array("apple", "banana");
$someArrayTwo = array(1 => "banana", "0" => "apple");

var_dump($someArrayOne == $someArrayTwo); // bool(true)
var_dump($someArrayOne === $someArrayTwo); // bool(false)
```

Type Operator

In PHP **instanceof** is used to determine whether a variable is an instantiated object of a certain class. It also works with class hierarchy and interfaces.

Type Operator

```php
<?php

// The simplest classic usage:

class SomeClass
{
}

class NotSomeClass
{
}
$someObject = new SomeClass;

var_dump($someObject instanceof SomeClass); // bool(true)
var_dump($someObject instanceof NotSomeClass); // bool(false)
```

Type Operator

```php
<?php

// Hierarchy analysis:

class SomeParentClass
{
}

class SomeChildClass extends SomeParentClass
{
}

$someObject = new SomeChildClass();

var_dump($someObject instanceof SomeChildClass); // bool(true)
var_dump($someObject instanceof SomeParentClass); // bool(true)
```

Type Operator

```php
<?php

// Interface implementation analysis:

interface SomeTrickyInterface
{
}

class SomeTrickyClass implements SomeTrickyInterface
{
}

$someObject = new SomeTrickyClass();

var_dump($someObject instanceof SomeTrickyClass); // bool(true)
var_dump($someObject instanceof SomeTrickyInterface); // bool(true)
```

Operators

Disclaimer: вы смотрите просто запись лекции,
это **НЕ** специально подготовленный видеокурс!