Тема 3.2
«Операторы PHP»

Вступление

Оператором называется «нечто», состоящее из одного или более значений (выражений), что можно вычислить как новое значение (таким образом, вся конструкция может рассматриваться как выражение).

Иными словами, оператор представляет собой символическое обозначение некоторого действия, выполняемого с операндами в выражении.

Следует помнить, что PHP выполняет автоматическое переключение типов операндов на основании типа оператора, – в других языках программирования это происходит не всегда.

Виды операторов в PHP

Операторы бывают трёх видов:

Унарные операторы, которые работают только с одним аргументом, например, ! (оператор отрицания) или ++ (инкремент):

```
$b=!$b;
$i++;
```

Бинарные операторы – большинство поддерживаемых в PHP операторов:

```
$c=$a+$b;
```

Тернарный оператор ?:. Он используется для условного выбора между двумя операторами, в зависимости от результата вычисления третьего оператора:

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

Аналогично:

```
if (empty($_POST['action'])) $action = 'default'; else $action = $_POST['action'];
```

Приоритет и ассоциативность операторов

Приоритет является характеристикой операторов, определяющей порядок выполнения действий с окружающими операндами.

Для арифметических операторов PHP применяет математические правила приоритета операций:

```
$c = $a + $b * 0.5;
```

Эквивалентно:

```
$c = $a + ($b * 0.5);
```

Для неарифметических операторов тоже есть свои правила, определённые «ассоциативностью операторов».

Приоритет и ассоциативность операторов

Ассоциативность операторов определяет последовательность выполнения операторов с одинаковым приоритетом. Выполнение может, как правило (т.к. есть «неассоциативные операторы»), происходить в двух направлениях: либо слева направо, либо справа налево.

Поскольку таблица ассоциативности достаточно велика, и её непросто сразу запомнить, есть золотое правило: «сомневаетесь – используйте скобки».

Приоритет и ассоциативность операторов

Оператор	Ассоциативность	Значение оператора
( )	Неассоциативный	Изменение приоритета
new	Неассоциативный	Создание экземпляра класса
! ~	Правая	Логическое и поразрядное отрицание
++--	Правая	Инкремент и декремент
@	Правая	Маскировка сообщений об ошибках
/ * %	Левая	Деление, умножение, «деление по модулю» (остаток)
+- .	Левая	Сложение, вычитание, конкатенация строк
<< >>	Левая	Поразрядный сдвиг влево и вправо
< <= > >=	Неассоциативный	Меньше, меньше или равно, больше, больше или равно
== != <> === !==	Неассоциативный	Равно, не равно, не равно; равно и не равно по типу и значению
& ^ |	Левая	Поразрядные AND, XOR, OR
&& AND || OR XOR	Левая	Логические AND, OR, XOR
?:	Правая	Тернарный оператор
= += *= /= .=	Правая	Присваивание и присваивание с выполнением действия
%= &= |= ^= <<= >>=	Правая	Присваивание с выполнением действия

Оператор присваивания

Оператор присваивания обозначается как =

Результатом выполнения оператора присваивания является само присвоенное значение. Таким образом, результат выполнения $a = 3 будет равен 3. Это позволяет использовать конструкции вида:

```
$a = ($b = 4) + 5; // $a присвоено 9, $b присвоено 4.
```

Внимание! Так делать можно, но НЕ НУЖНО! Такой код читается с трудом, что повышает вероятность допустить и/или не обнаружить ошибку.

Оператор присваивания активно используется для инициализации переменных (а в PHP, как мы помним, это будет ещё и объявлением переменных):

```
$a=10;
$b="Minsk";
```

Оператор присваивания

В дополнение к базовому оператору присваивания имеются «комбинированные операторы» для всех бинарных, арифметических и строковых операций, которые позволяют использовать некоторое значение в выражении, а затем установить его как результат данного выражения.

Например:

```
$a =3;
$a += 5; // $a равно, аналогично записи: $a = $a + 5;

$b = "Hello ";
$b .= " world!"; // $b равно "Hello world!“: $b = $b . “ world!";
```

Оператор присваивания

Обратите внимание, что присваивание копирует оригинальную переменную в новую (присваивание по значению) и, таким образом, все последующие изменения одной из переменных на другой никак не отражаются.

Начиная с PHP 4, также поддерживается присваивание по ссылке:

```
$var = &$othervar;
```

Присваивание по ссылке означает, что обе переменные указывают на одни и те же данные (один и тот же адрес в памяти) и никакого копирования не происходит, а изменения одной переменной автоматически приводят к таким же изменениям другой переменной.

Внимание! НЕ НАДО использовать присваивание по ссылке, если у вас для этого нет очень веской аргументации. Такой подход – чудесный источник сложнообнаружимых ошибок.

Арифметические операторы

Операция деления / всегда возвращает вещественный (дробный) тип, даже если оба операнда были целочисленными (или строками, которые преобразуются в целые числа).

Остаток $a % $b будет отрицательным для отрицательных значений $a.

Оператор	Действие	Пример
$a= - $a;	Смена знака	$a=5; $a=-$a; // -5
$c=$a + $b;	Сложение	$c=3+$x;
$c=$a - $b;	Вычитание	$c=17.6-$z;
$c=$a * $b;	Умножение	$z=$n*$x*76;
$c=$a / $b;	Деление	$x=$n/$y;
$c=$a % $b;	Деление по модулю (остаток)	$c=5%2; // 1

Поразрядные операторы

Эта группа операторов работает с битовыми представлениями значений целочисленных операндов. В основном эти операторы применяются для создания т.н. «битовых масок», для решения задач криптографии и при генерации изображений.

Оператор	**Действие**
$c=$a & $b;	Поразрядное И (AND)
$c=$a | $b;	Поразрядное ИЛИ (OR)
$c=$a ^ $b;	Поразрядное исключающее ИЛИ (XOR)
$c= ~ $a;	Поразрядное отрицание (NOT)
$c=$a << 1;	Поразрядный сдвиг влево
$c=$a >> 2;	Поразрядный сдвиг вправо

Поразрядное И (AND)

Результат равняется 1, если в обоих операндах в этом месте бит равняется 1. В противном случае результат равняется 0.

Таблица истинности AND (коньюнкция):

A	B	A & B
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

Пример: $a=5; $b=7; $c=$a & $b;

$a	1	0	1
$b	1	1	1
$c	**1**	**0**	**1**

Задание для самопроверки: как определить, что в некоторой переменной N-й бит равен единице?

Поразрядное ИЛИ (OR)

Результат равняется 1, если хотя бы в одном операнде в этом месте бит равняется 1. В противном случае результат равняется 0.

Таблица истинности OR (дизъюнкция):

A	B	A & B
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**1**

Пример: $a=1; $b=5; $c=$a | $b;

$a	0	0	1
$b	1	0	1
$c	**1**	**0**	**1**

Поразрядное исключающее ИЛИ (XOR)

Результат равняется 1, если ТОЛЬКО в одном операнде в этом месте бит равняется 1. В противном случае результат равняется 0.

Таблица истинности XOR (сложение по модулю 2):

A	B	A & B
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**0**

Пример: $a=1; $b=5; $c=$a xor $b;

$a	0	0	1
$b	1	0	1
$c	**1**	**0**	**0**

Задание для самопроверки (для «продвинутых»): написать скрипт, выполняющий шифрование текста «по методу XOR».

Поразрядное отрицание (NOT)

Биты операнда инвертируются.

Пример: $a=5; $b=~$a;

$a	1	0	1
$b	**1**	**1**	**0**

Поразрядный сдвиг влево и вправо

Биты операнда сдвигаются на указанное количество позиций влево или вправо соответственно.

Пример1: $a=5; $b=$a << 1;

Пример2: $a=5; $b=$a >> 2;

$a	1	0	1
$b	**0**	**1**	**0**

$a	1	0	1
$b	**0**	**0**	**1**

Логические операторы

Эта группа операторов работает с битовыми представлениями значений целочисленных операндов. В основном эти операторы применяются для создания т.н. «битовых масок», для решения задач криптографии и при генерации изображений.

Оператор	**Действие**
$c=$a and $b;	Логическое И (AND)
$c=$a or $b;	Логическое ИЛИ (OR)
$c=$a xor $b;	Логическое исключающее ИЛИ (XOR)
$c= ! $a;	Логическое отрицание (NOT)
$c=$a && $b;	Логическое И (AND)
$c=$a || $b;	Логическое ИЛИ (OR)

Логические операторы

Смысл двух разных вариантов для операторов «and» и «or» в том, что они работают с различными приоритетами (приоритетность: or > xor > and > || > && > !).

Операторы этой группы, в отличие от поразрядных, работают с логическими переменными (boolean) и интенсивно используются в управляющих конструкциях.

Логические переменные, имеют лишь два значения: true (TRUE) и false (FALSE). В выражениях true можно заменить на любое отличное от 0 число (чаще всего – на 1), а false – на 0. Возможность замены true и false на 1 и 0 – это «наследство» языка C, в котором не было логического типа данных.

Множество примеров с применением логических операторов мы рассмотрим в теме «управляющие конструкции PHP».

Операторы сравнения

Операторы сравнения, как это видно из их названия, позволяют сравнивать между собой два значения.

В случае, если вы сравниваете целое со строкой, строка будет преобразована к числу. В случае, если вы сравниваете две «числовые строки», они сравниваются как целые числа.

Эти правила также распространяются на оператор switch (будет рассмотрен в теме «управляющие конструкции PHP»).

Оператор	**Название**	**Результат**
$a == $b	Равно	TRUE, если $a равно $b
$a === $b	Равно по типу и значению	TRUE, если $a равно $b по типу и значению
$a != $b $a <> $b	Не равно	TRUE, если $a НЕ равно $b
$a !== $b	Не равно по типу или значению	TRUE, если $a НЕ равно $b по типу ИЛИ значению
$a < $b	Меньше	TRUE, если $a < $b
$a > $b	Больше	TRUE, если $a > $b
$a <= $b	Меньше либо равно	TRUE, если $a <= $b
$a >= $b	Больше либо равно	TRUE, если $a >= $b

Операторы сравнения

Особый интерес представляют операторы === и !==.

Поскольку PHP является нестроготипизированным языком с переключением типов, он «не отличает» схожие значения разных типов данных. Так, например:

число 10 будет равно строке «10»;

любое отличное от нуля число будет равно TRUE;

пустая строка и строка «0» будет равна FALSE;

и т.д.

Чтобы избежать подобных недоразумений, в тех случаях, когда нам нужно учитывать типы данных при сравнении, следует применять операторы === и !==. Оператор === сразу возвращает FALSE, если типы операндов не совпадают. Оператор !== вернёт TRUE, если операнды не равны по типу или по значению.

Операторы сравнения

Рассмотрим пример с использованием операторов === и !==:

```
$a=1;
$b="1";
if ($a==$b) echo "OK"; // выведется OK
if ($a===$b) echo "OK"; // ничего не выведется
if ($a!==$b) echo "Aga!"; // выведется Aga!
```

Лирическое отступление

Когда-то на bash.org.ru проскакивала такая «программистская шутка»:

```
function check_boolean ($boolean)
{
if ($boolean==1)
 {
  echo “TRUE”;
 }
  elseif ($boolean==0)
  {
   echo “FALSE”;
  }
   elseif ($boolean==2)
    {
     echo “Нифига себе! O_o”;
    }
     else
      {
       echo “He не не я не, Дэвид Блэйн! Прекрати свои демонские фокусы!”
      }
}
```

Однако для PHP эта шутка приобретает горький привкус правды. Отсюда следует вывод: если вы надеетесь, что та или иная переменная окажется нужного вам типа, вы рискуете получить непредвиденное поведение программы, а потому – **ПРОВЕРЯЙТЕ типы переменных!**

Оператор маскировки сообщений об ошибках

Сколь бы надёжной ни была программа, она может оказаться в условиях, когда то или иное действие приведёт к возникновению ошибки. В веб-приложениях появление «сообщений от транслятора» неприятно по двум причинам:

- оно сильно портит имидж разработчика;
- оно может дать злоумышленнику информацию для взлома сайта.

Потому рекомендуется:

1) Использовать функцию error_reporting(0) для отключения каких бы то ни было сообщений об ошибках и предупреждений в готовом выложенном в Интернет проекте. К слову, в php.ini есть параметр error_reporting, которому на стадии разработки рекомендуется выставить значение E_ALL, чтобы быть в курсе всего, что «не понравится» PHP.

2) Использовать оператор маскировки сообщений об ошибках @ для подавления вывода соответствующих сообщений.

Оператор маскировки сообщений об ошибках

PHP поддерживает один оператор маскировки сообщений об ошибках: знак @. В случае, если он предшествует какому-либо выражению в PHP-коде, любые сообщения об ошибках, генерируемые этим выражением, не будут показаны. Пример:

```
// Маскировка ошибки при работе с файлами
$my_file = @file ('non_existent_file');

// @ работает и для выражений, а не только для функций
$value = @$arr[$k];
// В случае если ключа $k нет в массиве $arr, сообщение об ошибке не будет отображено
```

Если произвольная языковая конструкция возвращает значение, вы можете использовать предшествующий ей оператор @. Например, вы можете использовать @ перед именем переменной, произвольной функцией или вызовом include(), константой и так далее. В то же время вы не можете использовать этот оператор перед определением функции или класса, условными конструкциями, такими как if или foreach. Оператор @ не подавляет вывод ошибок, возникающих на стадии синтаксического разбора скрипта.

Оператор исполнения

PHP поддерживает оператор исполнения: обратные кавычки ` `. Обратите внимание, что это не одинарные кавычки, а именно обратные (на клавиатуре расположены под клавишей Esc).

PHP пытается выполнить строку, заключенную в обратные кавычки, как консольную команду, и возвращает полученный вывод (т.е. он не просто выдаётся на выходе а, например, может быть присвоен переменной).

Пример:

```
$output = `dir`;
echo "<pre>$output</pre>";
```

Оператор исполнения

Подобного эффекта можно достичь использованием функции system (string command [, int &return_var]). Эта функция сразу же выводит результат выполнения команды в выходной поток или помещает его в переменную, которая может быть задана вторым аргументом.

Внимание! Оператор исполнения ` ` и функция system() чудесно работают в posix-совместимых операционных системах. В Windows с их помощью можно выполнять ТОЛЬКО «консольные команды» типа «dir» и т.п. Попытка с помощью этого способа запустить GUI-приложение приводит, как правило, к зависанию Apache.

Операторы инкремента и декремента

PHP, аналогично C, поддерживает префиксные и постфиксные операторы инкремента и декремента числовых переменных. Инкрементирование или декрементирование логических переменных не приводит ни к какому результату.

Оператор	Название	Результат
++$a;	Префиксный инкремент	Увеличивает значение переменной на 1 и возвращает её значение (новое)
--$a;	Префиксный декремент	Уменьшает значение переменной на 1 и возвращает её значение (новое)
$a++;	Постфиксный инкремент	Возвращает значение переменной (старое) и увеличивает значение переменной на 1
$a--;	Постфиксный декремент	Возвращает значение переменной (старое) и уменьшает значение переменной на 1

Операторы инкремента и декремента

Рассмотрим пример кода:

```
// постфиксный инкремент
$a = 5;
echo "Должно быть 5: ". $a++;
echo "Должно быть 6: ". $a;

// префиксный инкремент
$a = 5;
echo "Должно быть 6: ". ++$a;
echo "Должно быть 6: ". $a;

// постфиксный декремент
$a = 5;
echo "Должно быть 5: ". $a--;
echo "Должно быть 4: ". $a;

// префиксный декремент
$a = 5;
echo "Должно быть 4: ". --$a;
echo "Должно быть 4: ". $a;
```

Операторы инкремента и декремента

PHP следует соглашениям Perl (в отличие от C) касательно выполнения арифметических операций с символьными переменными. Например в Perl 'Z'+1 будет вычислено как 'AA', в то время как в C 'Z'+1 будет вычислено как '[' (ord('Z') == 90, ord('[') == 91).

Следует учесть, что к символьным переменным можно применять операцию инкремента, в то время как операцию декремента применять нельзя:

```
$i = 'W';
for ($n=0; $n<6; $n++) echo ++$i . ", ";
```

Результат работы будет следующим:

X, Y, Z, AA, AB, AC,

Задача для самопроверки

Что получится в результате выполнения такого кода:

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

Ответ: 10

Почему:

1) $i==2
2) Срабатывает префиксный инкремент ++$i, выражение принимает вид:
   $i += $i++ + 3;
3) Происходит подстановка первого операнда, выражение принимает вид:
   $i += 3 + 3; или: $i += 6;
4) Происходит «разворачивание» «сокращённой записи сложения и присваивания» +=, выражение принимает вид:
   $i=3 + 6;
5) Выполнение выражения (п. 4) даёт $i==9;
6) Срабатывает постфиксный инкремент $i++, который увеличивает значение $i на единицу, т.е. теперь $i==10;

P.S. Если когда-нибудь вы в реальном коде встретите такие конструкции – увольте того, кто их писал, без сожаления. Т.к. читаемость такого кода близка к нулю.

Строковые операторы

В PHP есть два оператора для работы со строками.

Первый – оператор конкатенации – точка, который возвращает объединение левого и правого аргумента.

Второй – оператор присвоения с конкатенацией, который присоединяет правый аргумент к левому:

```
$a = "Hello ";
$b = $a . " world!";
// $b содержит строку "Hello world!“
```

```
$a = "Hello ";
$a .= “ world!";
// $a содержит строку "Hello world!"
```

Операторы работы с массивами

Подробнее о массивах мы поговорим в отдельной теме, а сейчас просто рассмотрим операторы работы с массивами, чтобы знать, что такие существуют:

Оператор	Название	Результат
$a + $b	Объединение массивов	Объединение массивов
$a == $b	Проверка на равенство массивов	TRUE, если массивы содержат одинаковые элементы
$a === $b	Проверка на тождественное равенство массивов	TRUE, если массивы содержат одинаковые элементы в том же порядке
$a != $b $a <> $b	Проверка на неравенство массивов	TRUE, если массивы содержат неодинаковые элементы
$a !== $b	Проверка на тождественное неравенство массивов	TRUE, если массивы содержат неодинаковые элементы или порядок элементов отличается

Операторы работы с массивами

Абстрагировавшись от вопросов объявления массивов (его мы затронем позже), рассмотрим несколько примеров:

```
$arr1=array(10 => ‘A’, 20 => ‘B’, 30 => ‘C’);
$arr2=array(10 => ‘aaa’, 20 => ‘bbb’, 90 => ‘ccc’);
$arr3=$arr1+$arr2;
print_r($arr3);
```

Результат:

```
Array ( [10] => A [20] => B [30] => C [90] => ccc )
```

Обратите внимание, что элементы первого массива с ключами, совпадающими с ключами элементов второго массива, не были перезаписаны.

Операторы работы с массивами

```
$arr1=array(10 => 'A', 20 => 'B', 30 => 'C');
$arr2=array(10 => 'A', 20 => 'B', 30 => 'C');
if ($arr1==$arr2) echo "OK";
// выведется OK

$arr1=array(10 => 'A', 20 => 'B', 30 => 'C');
$arr2=array(10 => 'A', 20 => 'B', 900 => 'C');
if ($arr1==$arr2) echo "Yes";
// ничего не выведется

$arr1=array(10 => 'A', 20 => 'B', 30 => 'C');
$arr2=array(10 => 'A', 20 => 'B', 900 => 'C');
if ($arr1===$arr2) echo "Yes";
// ничего не выведется
```

Операторы работы с массивами

На практике операторы работы с массивами почти не применяются в силу их «нечеловекоочевидности» и наличия значительно более простых способов работы с массивами, которые мы рассмотрим в соответствующей теме.

Оператор проверки принадлежности классу

Оператор instanceof используется для определения того, является ли текущий объект экземпляром указанного класса.

Оператор instanceof был добавлен в PHP 5. До этого использовалась конструкция is_a(), которая на данный момент не рекомендуется к применению, более предпочтительно использовать оператор instanceof.

Пример:

```
class A { }
class B { }
$thing = new A;
if ($thing instanceof A) echo 'A'; // выведется A
if ($thing instanceof B) echo 'B'; // ничего не выведется
```