# **Лабораторная работа №4-5. Функции работы с переменными в PHP**

## **Операции с переменными PHP (операторы)**

Для осуществления операций с переменными существуют различные группы операторов.

**Оператором** называется нечто, состоящее из одного или более значений ([выражений](http://www.php.su/learnphp/?expressions), если говорить на жаргоне программирования), которое можно вычислить как новое значение (таким образом, вся конструкция может рассматриваться как выражение). Отсюда следует, что функции или любые другие конструкции, которые возвращают значение (например, [print()](http://www.php.su/functions/?print)) являются операторами, в отличие от всех остальных языковых конструкций (например, [echo()](http://www.php.su/functions/?echo)), которые ничего не возвращают.

### **Арифметические операции в PHP**

Описанные ниже операторы работают в соответствии с логикой стандартных математических операций.

| **Пример** | **Название** | **Результат** |
| --- | --- | --- |
| -$a | Отрицание | Смена знака $a. |
| $a + $b | Сложение | Сумма $a и $b. |
| $a - $b | Вычитание | Разность $a и $b. |
| $a \* $b | Умножение | Произведение $a и $b. |
| $a / $b | Деление | Частное от деления $a на $b. |
| $a % $b | Деление по модулю | Целочисленный остаток от деления $a на $b. |

Например:

**<?php**$a = 10;  
$b = 4;  
$c = $a%$b;  
**echo "Целочисленный остаток от деления a на b равен** $c**"**;  
**?>**

Но есть некоторые ньюансы.

Операция деления ("/") всегда возвращает вещественный тип, даже если оба значения были целочисленными (или строками, которые преобразуются в целые числа). В противном случае результат будет дробным.

Операция вычисления остатка от деления «**%**» работает только с целыми числами, так что применение ее к дробным может привести к нежелательному результату.

Приоритет одних математических операций над другими и изменение приоритетов при использовании скобок в арифметических выражениях соответствуют обычным математическим правилам.

### **Операции инкремента и декремента**

PHP, аналогично C, поддерживает префиксные и постфиксные операторы инкремента и декремента.

| **Пример** | **Название** | **Действие** |
| --- | --- | --- |
| ++$a | Префиксный инкремент | Увеличивает $a на единицу и возвращает значение $a. |
| $a++ | Постфиксный инкремент | Возвращает значение $a, а затем увеличивает $a на единицу. |
| --$a | Префиксный декремент | Уменьшает $a на единицу и возвращает значение $a. |
| $a-- | Постфиксный декремент | Возвращает значение $a, а затем уменьшает $a на единицу. |

### **Постфиксные операторы инкремента и декремента**

Как и в языке Си, эти операторы увеличивают или уменьшают значение переменной, а в выражении возвращают значение переменной **$a** до изменения.

Например:

**<?php**$a = 10;  
$b = $a++;  
**echo "a =** $a**, b =** $b**"**; *// Выводит a=11, b=10***?>**

Как видите, сначала переменной **$b** присвоилось значение переменной **$a**, а уж затем последняя была инкрементирована. Впрочем, выражение, значение которого присваивается переменной **$b**, может быть и сложнее – в любом случае, инкремент **$a** произойдет только после его вычисления.

### **Префиксные операторы инкремента и декремента**

Существуют также операторы инкремента и декремента, которые указываются до, а не после имени переменной. Соответственно, и возвращают они значение переменной уже после изменения.

Пример:

**<?php**$a = 10;  
$b = --$a;  
**echo "a =** $a**, b =** $b**"**; *// Выводит a=9, b=9***?>**

Операции инкремента и декремента на практике применяются очень часто. Например, они встречаются практически в любом [цикле **for**](http://www.php.su/learnphp/cs/?cycles) (будет рассмотрен в последующих лабораторных работах).

Приведем другие примеры операций инкремента и декремента.

**<?php  
echo "<h3>Постфиксный инкремент</h3>"**;  
$a = 5;  
**echo "Должно быть 5: "** . $a++ . **"<br />\n"**;  
**echo "Должно быть 6: "** . $a . **"<br />\n"**;  
  
**echo "<h3>Префиксный инкремент</h3>"**;  
$a = 5;  
**echo "Должно быть 6: "** . ++$a . **"<br />\n"**;  
**echo "Должно быть 6: "** . $a . **"<br />\n"**;  
  
**echo "<h3>Постфиксный декремент</h3>"**;  
$a = 5;  
**echo "Должно быть 5: "** . $a-- . **"<br />\n"**;  
**echo "Должно быть 4: "** . $a . **"<br />\n"**;  
  
**echo "<h3>Префиксный декремент</h3>"**;  
$a = 5;  
**echo "Должно быть 4: "** . --$a . **"<br />\n"**;  
**echo "Должно быть 4: "** . $a . **"<br />\n"**;  
**?>**

Операцию декремента нельзя применять для символьных переменных.

Например:

**<?php**$a=**'abc'**;  
$b=$a++;  
**echo "a =** $a**, b =** $b**"**; *// Выводит a=abd, b=abc***?>**

[**Булевые** типы](http://www.php.su/learnphp/datatypes/?boolean) не подлежат инкрементированию и декрементированию.

### **Операции присвоения**

Базовый оператор присвоения обозначается как *=*. На первый взгляд может показаться, что это оператор «равно». На самом деле это не так. В действительности, оператор присвоения означает, что левый операнд получает значение правого выражения, (т.е. устанавливается результирующим значением).

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

**<?php**$b = 4;  
$a = $b + 5; *// результат: $a установлена значением 9, переменной $b присвоено 4.***echo "a =** $a **, b =** $b**"**;  
**?>**

Отметим, что в одной строке кода может быть несколько операций присвоения. В таком случае предыдущий пример будет выглядеть следующим образом.

**<?php**$a = ($b = 4) + 5; *// результат: $a установлена значением 9, переменной $b присвоено 4.***echo "a =** $a **, b =** $b**"**;  
**?>**

В дополнение к базовому оператору присвоения 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:

**<?php**$a = 7;  
$a += 3; *// устанавливает $a значением 10, аналогично записи: $a = $a + 3;***echo "a =** $a**"**;  
**echo "<br>"**;  
$b = **"Привет. "**;  
$b .= **"Как дела."**; *// устанавливает $b строкой "Привет. Как дела", как и $b = $b . "Как дела.";***echo "b =** $b**"**;  
**?>**

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

### **Строковые операции**

В PHP есть два оператора для работы со строками. Первый – оператор конкатенации ('.'), который возвращает объединение левого и правого аргумента.

Второй – оператор присвоения с конкатенацией, который присоединяет правый аргумент к левому. Приведем конкретный пример:

**<?php**$a = **"Hello "**;  
$b = $a . **"World!"**; *// $b содержит строку "Hello World!"***echo** $b;  
**echo "<br>"**;  
$a = **"Hello "**;  
$a .= **"World!"**; *// $a содержит строку "Hello World!"***echo** $a;  
**?>**

Результат обеих операций в данном примере – один и тот же

### **Побитовые операции**

Эти операции предназначены для работы (установки/снятия/проверки) групп битов в целой переменной. Биты целого числа – это не что иное, как отдельные разряды того же самого числа, записанного в двоичной системе счисления. Например, в двоичной системе число 12 будет выглядеть как 1100, а 2 — как 10, так что выражение **12|2** вернет нам число 14 (1110 в двоичной записи).

Если переменная не целая, то она в начале округляется, а уж затем к ней применяются перечисленные ниже операторы.

Для представления одного числа используются 32 бита:

0000 0000 0000 0000 0000 0000 0000 0000 - это ноль;

0000 0000 0000 0000 0000 0000 0000 0001 - это 1;

0000 0000 0000 0000 0000 0000 0000 0010 - это 2;

0000 0000 0000 0000 0000 0000 0000 0011 - это 3;

0000 0000 0000 0000 0000 0000 0000 0100 - это 4;

0000 0000 0000 0000 0000 0000 0000 0101 - это 5;

...

0000 0000 0000 0000 0000 0000 0000 1100 - это 12;

...

0000 0000 0000 0000 0000 0000 0000 1111 - это 15;

...

и т.д.

Побитовые операторы:

| **Пример** | **Название** | **Результат** |
| --- | --- | --- |
| $a & $b | Побитовое 'И' | Устанавливаются только те биты, которые установлены и в $a, и в $b. |
| $a | $b | Побитовое 'ИЛИ' | Устанавливаются те биты, которые установлены либо в $a, либо в $b. |
| $a ^ $b | Исключающее ИЛИ | Устанавливаются только те биты, которые установлены либо только в $a, либо только в $b |
| ~ $a | Отрицание | Устанавливаются те биты, которые в $a не установлены, и наоборот. |
| $a << $b | Сдвиг влево | Все биты переменной $a сдвигаются на $b позиций влево (каждая позиция подразумевает 'умножение на 2') |
| $a >> $b | Сдвиг вправо | Все биты переменной $a сдвигаются на $b позиций вправо (каждая позиция подразумевает 'деление на 2') |

Приведем примеры побитовых операций.

1. Побитовое И:

1100

12 & 6 **→** И **→** 0100, т.е. в десятичной форме 4.

0110

1. Побитовое ИЛИ:

1100

12 | 6 **→** И **→** 1110, т.е. в десятичной форме 14.

0110

1. Побитовое исключающее ИЛИ:

1100

12 ^ 6 **→** И **→** 1010, т.е. в десятичной форме 10.

0110

1. Сдвиг влево (не забываем, что все числа в php кодируются 32 битами):

12<<2 **→** 0000 0000 0000 0000 0000 0000 0000 1100<<2 **→**

**→** 0000 0000 0000 0000 0000 0000 0011 0000, т.е. в десятичной форме 48.

**<?php**$a = 12; *//в двоичной системе 12 - это 1100*$b = $a|6;  
**echo "12|6 =** $b**"**; *// результат = 14***echo "<br>"**;  
  
$b = $a&6;  
**echo "12&6 =** $b**"**; *// результат = 4***echo "<br>"**;  
  
$b = $a^6;  
**echo "12^6 =** $b**"**; *// результат = 10***echo "<br>"**;  
  
$b = $a<<2;  
**echo "**$a **<<2 =** $b**"**; *// результат = 48***echo "<br>"**;  
**?>**

### **Операции сравнения**

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

Это в своем роде уникальные операции, потому что независимо от типов своих аргументов они всегда возвращают одно из двух: **false** или **true**. Операции сравнения позволяют сравнивать два значения между собой и, если условие выполнено, возвращают **true**, а если нет – **false**.

В PHP разрешается сравнивать только скалярные переменные. Массивы и объекты в PHP сравнивать нельзя. Их даже нельзя сравнивать на равенство (при помощи оператора ==), но при выполнении такой операции PHP не выдает предупреждения. Так что удивившись как-то раз, почему два совершенно разных массива при сравнении их с помощью **==** оказываются вдруг одинаковыми, вспомните, что перед сравнением оба операнда преобразуются в слово [**array**](http://www.php.su/functions/?array), которое потом и сравнивается.

Операторы сравнения:

| **Пример** | **Название** | **Результат** |
| --- | --- | --- |
| $a == $b | Равно | **TRUE** если $a равно $b. |
| $a === $b | Тождественно равно | **TRUE** если $a равно $b и имеет тот же тип. (Добавлено в PHP 4 и старше) |
| $a != $b | Не равно | **TRUE** если $a не равно $b. |
| $a <> $b | Не равно | **TRUE** если $a не равно $b. |
| $a !== $b | Тождественно не равно | **TRUE** если $a не равно $b или в случае, если они разных типов (Добавлено в PHP 4 и старше) |
| $a < $b | Меньше | **TRUE** если $a строго меньше $b. |
| $a > $b | Больше | **TRUE** если $a строго больше $b. |
| $a <= $b | Меньше или равно | **TRUE** если $a меньше или равно $b. |
| $a >= $b | Больше или равно | **TRUE** если $a больше или равно $b. |

Приведем несколько примеров.

**<?php**$a = 10.1 + 0.9; *//переменная типа float*$b = 11; *//переменнаям типа integer* **if** ($a==$b) {  
 **echo 'true'**;  
 }  
 **else** {  
 **echo 'false'**;  
} *// результат true***echo "<br>"**;  
**?>**

**<?php**$a = 10.1 + 0.9; *//переменная типа float*$b = 11; *//переменнаям типа integer* **if** ($a===$b) {  
 **echo 'true'**;  
 }  
 **else** {  
 **echo 'false'**;  
} *// результат false***echo "<br>"**;  
**?>**

**<?php**$a = 10.1 + 0.9; *//переменная типа float*$b=11; *//переменнаям типа integer* **if** ($a!==$b) {  
 **echo 'true'**;  
 }  
 **else** {  
 **echo 'false'**;  
} *// результат true***echo "<br>"**;  
**?>**

Работа операторов эквивалентности, например, тождественно равно, будет далее рассмотрено более подробно.

### **Логические операции**

Логические операторы предназначены исключительно для работы с логическими выражениями и также возвращают **false** или **true**.

Приведем таблицу логических операторов PHP:

| **Пример** | **Название** | **Результат** |
| --- | --- | --- |
| $a and $b | Логическое 'и' | **TRUE** если и $a, и $b **TRUE**. |
| $a or $b | Логическое 'или' | **TRUE** если или $a, или $b **TRUE**. |
| $a xor $b | Исключающее 'или' | **TRUE** если $a, или $b **TRUE**, но не оба. |
| ! $a | Отрицание | **TRUE** если $a не **TRUE**. |
| $a && $b | Логическое 'и' | **TRUE** если и $a, и $b **TRUE**. |
| $a || $b | Логическое 'или' | **TRUE** если или $a, или $b **TRUE**. |

**<?php**$a = 10;  
$b = -5;  
 **if** ($a > 0 **and** $b > 0) {  
 **echo 'true'**;  
} **else** {  
 **echo 'false'**;  
}  
**?>**

Следует заметить, что вычисление логических выражений, содержащих такие операторы, идет всегда слева направо, при этом, если результат уже очевиден (например, **false&&что-то** всегда дает **false**), то вычисления обрываются, даже если в выражении присутствуют вызовы функций. Например, в операторе **$logic = 0&&(time()>100)**; стандартная функция [**time()**](http://www.php.su/functions/?time) никогда не будет вызвана.

Будьте осторожны с логическими операциями – не забывайте про удваивание символа. Обратите внимание, что, например, **|** и **||** – два совершенно разных оператора, один из которых может потенциально возвращать любое число, а второй – только **false** и **true**.

Операторы инкремента (++) и декремента (--) не работают с логическими переменными.

### **Операторы эквивалентности**

В PHP, начиная с PHP4, есть оператор тождественного сравнения – тройной знак равенства **===**, или оператор проверки на эквивалентность. PHP довольно терпимо относится к тому, что строки неявно преобразуются в числа, и наоборот.

Например, следующий код выведет, что значения переменных равны:

**<?php**$a = 10;  
$b = **"10"**;  
 **if**($a==$b) **echo "a и b равны"**; *// Выводит "a и b равны"***?>**

И это несмотря на то, что переменная **$a** представляет собой число, а **$b** – строку. Теперь рассмотрим несколько другой пример:

**<?php**$a = 0; *// ноль*$b = **""**; *// пустая строка* **if**($a==$b) **echo "a и b равны"**; *// Выводит "a и b равны"***?>**

Хотя **$a** и **$b** явно неравны даже в обычном понимании этого слова, скрипт заявит, что они совпадают. Почему так происходит? Дело в том, что если один из операндов логического оператора может трактоваться как число, то оба операнда трактуются как числа. При этом пустая строка превращается в **0**, который затем и сравнивается с нулем. Неудивительно, что оператор **echo** срабатывает.

Проблему решает оператор эквивалентности **===** (тройное равенство). Он не только сравнивает два выражения, но также их типы. Перепишем наш пример с использованием этого оператора:

**<?php**$a = 0; *// ноль*$b = **""**; *// пустая строка* **if**($a===$b) **echo "a и b равны"**; *// Ничего не выводит***?>**

Вот теперь ничего выведено не будет. Но возможности оператора эквивалентности идут далеко за пределы сравнения простых переменных. С его помощью можно сравнивать также и массивы, объекты и т. д. Это бывает иногда очень удобно.

| **Операторы, работающие с массивами** | | |
| --- | --- | --- |
| **Пример** | **Название** | **Результат** |
| $a + $b | Объединение | Объединение массива *$a* и массива *$b*. |
| $a == $b | Равно | **TRUE** в случае, если *$a* и *$b* содержат одни и те же пары ключ/значение. |
| $a === $b | Тождественно равно | **TRUE** в случае, если *$a* и *$b* содержат одни и те же паты ключ/значение в том же самом порядке и того же типа. |
| $a != $b | Не равно | **TRUE**, если массив *$a* не равен массиву *$b*. |
| $a <> $b | Не равно | **TRUE**, если массив *$a* не равен массиву *$b*. |
| $a !== $b | Тождественно не равно | **TRUE**, если массив *$a* не равен тождественно массиву *$b*. |

Вот пример:

**<?php**$a = **array**(**"apple"**, **"banana"**);  
$b = **array**(1 => **"banana"**, **"0"** => **"apple"**);  
  
**if** ($a == $b) **echo "сработало =="**; *// bool(true)***if** ($a === $b) **echo "сработало ==="**; *// bool(false)***?>**

Если запустить представленный код, то будет выведено первое сообщение, но не второе. Произойдет это по той причине, что, как мы уже говорили, операнды-массивы преобразуются в строки **array**, которые затем и будут сравниваться. Оператор **===** лишен этого недостатка, поэтому работает верно.

Для оператора **===** существует и его антипод — оператор **!===** .

### **Операции с символьными переменными**

PHP следует соглашениям Perl (в отличие от С) 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.

**<?php**$i = **'W'**;  
**for**($n = 0; $n < 6; $n++){

**echo** ++$i;  
 **echo "<br>"**;  
}  
  
*/\*  
 Результат работы будет следующий:  
  
X  
Y  
Z  
AA  
AB  
AC  
  
\*/***?>**

### **Приоритеты операторов PHP**

Операторы с более высоким уровнем приоритета выполняются в первую очередь:

|  |  |  |
| --- | --- | --- |
| **Приоритет** | **Оператор** | **Порядок выполнения** |
| 13 | (постфикс)++ (постфикс)-- | слева направо |
| 12 | ++(префикс) --(префикс) | справа налево |
| 11 | \* / % | слева направо |
| 10 | + - | слева направо |
| 9 | << >> | слева направо |
| 8 | < <= > >= | слева направо |
| 7 | == != | слева направо |
| 6 | & | слева направо |
| 5 | ^ | слева направо |
| 4 | | | слева направо |
| 3 | && | слева направо |
| 2 | || | слева направо |
| 1 | = += -= \*= /= %= >>= <<== &= ^= |= | справа налево |

# **Задание**

1. Написать php-скрипты, использующие математические операции над переменными (отрицание, сложение, вычитание, деление, умножение, деление по модулю).
2. Написать php-скрипты, использующие операции инкремента и декремента (постфиксные и префиксные) для числовых и символьных переменных. Пояснить полученные результаты.
3. Написать php-скрипты, использующие операции присваивания (в том числе и комбинированные) для различных типов переменных. Пояснить полученные результаты.
4. Написать php-скрипты, использующие операции конкатенации, присваивания с конкатенацией, сложения для символьных переменных. Пояснить полученные результаты.
5. Написать php-скрипты, использующие побитовые операции (‘И’, ‘ИЛИ’, исключающееся ‘ИЛИ’, сдвиг влево <<, сдвиг в право >>) на примере числовых переменных. Результаты просчитать с помощью калькулятора в двоичной системе в ручную и сравнить с полученными резельтатами в программе. Пояснить полученные результаты.
6. Написать php-скрипты, использующие операции сравнения (в том числе и операции эквивалентности). Пояснить полученные результаты.
7. Написать php-скрипты, использующие логические операции. Пояснить полученные результаты.