Арифметические операторы

Оператор	Название	Пример	Результат
+	Сложение	10+3	13
-	Вычитание	10-3	7
*	Умножение	10*3	30
/	Деление	10/3	3.333333333
%	Остаток от деления	10%3	1

Оператор конкатенации

Символ оператора конкатенации — простая точка. Этот оператор объединяет две строки, точнее, присоединяет правую строку к левой. Таким образом, выражение "hello "."world" имеет значение

"hello world"

Независимо от типа своих операндов, оператор конкатенации всегда обрабатывает их как строки и результат его выполнения всегда является строкой.

Дополнительные операторы присваивания

Существует только один оператор присваивания, однако есть несколько операторов, которые представляют собой комбинацию арифметического оператора и оператора присваивания.

Оператор	Пример	Эквивалентная запись
+=	$x += 5	$x = $x + 5
-=	$x -= 5	$x = $x - 5
/=	$x /= 5	$x = $x / 5
*=	$x *= 5	$x = $x * 5
%=	$x %= 5	$x = $x % 5
.=	$x .= "test"	$x = $x."test"

Для каждого арифметического оператора существует соответствующий дополнительный оператор присваивания, который выглядит как пара знаков арифметического оператора и оператора присваивания. Также имеется и дополнительный оператор присваивания для конкатенации.

Операторы сравнения

Оператор	Название	Условие выполнения	Пример	Результат при $x, равном 3
==	Равенство	Левый операнд равен правому	$x == 5	false
!=	Неравенство	Левый операнд не равен правому	$x != 5	true
===	Идентичность	Операнды равны и их типы совпадают	$x === 5	false
>	Больше чем	Левый операнд больше правого	$x > 3	false
>=	Больше или равно	Левый операнд больше правого или равен ему	$x >= 3	true
<	Меньше чем	Левый операнд меньше правого	$x < 3	false
<=	Меньше или равно	Левый операнд меньше правого или равен ему	$x <= 3	true

Логические операторы

Логические операторы предназначены для построения логических выражений. Например, логическое **ИЛИ** возвращает значение ***true***, если хоть один из его операндов имеет значение ***true***. Таким образом, выражение

```
true || false
```

имеет значение ***true***.

Логическое И возвращает ***true***, если оба операнда имеют значение ***true***. Таким образом, выражение

```
true && false
```

имеет значение ***false***. Конечно, маловероятно, что вам понадобится сравнивать логические константы. Гораздо больше смысла в сравнении логических выражений, например,

```
($x > 2) && ($x <15)
```

имеет значение ***true***, если значение $x находится в интервале между 2 и 15. Скобки в этом выражении расставлены просто для того, чтобы выражение легче читалось.

Логические операторы

Оператор	Название	Условие истинности	Пример	Результат
||	ИЛИ	Хотя бы один из операндов истинен	true || false	true
OR	ИЛИ	Хотя бы один из операндов истинен	true OR false	true
XOR	Исключающее ИЛИ	Только один из операндов истинен	true XOR true	false
&&	И	Оба операнда истинны	true && false	false
AND	И	Оба операнда истинны	true AND false	false
!	НЕ	Операнд не истинен	!true	false

Увеличение и уменьшение целой переменной

```
$x = $x + 1;  // увеличение $x на 1
```

То же самое можно сделать с помощью дополнительного оператора присваивания:

```
$x += 1;  // увеличение $x на 1
```

Постфиксный оператор выглядит как два знака «-» или «+» после имени переменной.

```
$x++;  // увеличение $x на 1
$x--;  // уменьшение $x на 1
```

Если такой постфиксный оператор использовать в условном выражении, то значение переменной будет изменено только после вычисления выражения.

```
$x = 3;
$x++ < 4;  // выражение истинно
```

Но в других обстоятельствах вам может понадобиться сделать так, чтобы значение переменной изменялось до того, как будет вычислено все выражение. Для этого вам придется воспользоваться префиксной формой оператора.

```
++$x;  // увеличение $x на 1
--$x;  // уменьшение $x на 1
```

Если такую форму оператора использовать в условном выражении, то значение переменной будет изменено до того, как вычислять все выражение.

```
$x = 3;
++$x < 4;  // выражение ложно
```

В этом случае при сравнении переменной $x с константой 4 значение переменной уже равно 4, т.е. оно не меньше 4, а следовательно, выражение ложно.

Порядок вычисления операторов

++ --
/ * %
+ -
< <= => >
== === !=
&&
||
= += -= /= *= %= .=
AND
XOR
OR

$x **&&** $y || $z ≡ ($x **AND** $y) **OR** $z

Константы

Константа — именованная величина, которая не изменятся во время выполнения программы.

define("PI", "3.141592");

Создание константы

```
<html> <head>
<title> Создание константы </title>
</head> <body>
<?php
define ("PI", "3.141592");
print "Число Пи равно ".PI."<br>";
$pi2 = PI * PI;
print "Пи в квадрате равно $pi2";
?>
</body> </html>
```

Условные операторы

Оператор ***if*** с блоком ***else***

```
if (выражение) {
// этот фрагмент выполняется, если выражение истинно
}
else {
// этот фрагмент выполняется, если выражение ложно
}
```

```
<html> <head>
<title> Оператор if с блоком else </title>
</head> <body>
<?php
$var = "плохо";
if ($var == "хорошо"){
print "Я в хорошем настроении!";
}
else {
print "Мне $var";
}
?>
</body> </html>
```

Блок ***elseif*** оператора ***if***

```
if (выражение_1) {
// этот фрагмент выполняется, если выражение истинно
}
elseif (выражение_2) {
// этот фрагмент выполняется, если выражение_1 ложно,
// а выражение_2 истинно
}
else {
// этот фрагмент выполняется во всех остальных случаях
}
```

```
<html> <head>
<title> Использование блоков else и elseif
</title> </head> <body>
<?php
$var = "плохо";
if ($var == "хорошо") {
print "Я в хорошем настроении!";
}
elseif ($var == "плохо") {
print "Не отчаиваться!";
}
else {
print "Непонятно, просто $var";
}
?>
</body> </html>
```

Оператор switch

```
switch (выражение)
{
case значение_1:
// выполняется, если выражение равно значение_1
break;
case значение_2:
// выполняется, если выражение равно значение_2
break;
default:
// выполняется, если выражение не приняло
// ни одного из перечисленных значений
}
```

```
<html> <head>
<title> Оператор switch </title>
</head> <body>
<?php
$var = "плохо";
switch ($var) {
case "хорошо":
    print "Я в хорошем настроении!";
    break;
case "плохо":
    print "Не отчаиваться!";
    break;
default:
    print "Непонятно, просто $var";
}
?>
</body> </html>
```

Оператор ?

Оператор ? возвращает значение одного из двух выражений, разделенных знаком двоеточия. Какое из двух выражений сформирует возвращаемое значение — зависит от истинности тестового выражения.

(тестовое_выражение) ? выражение_1: выражение_2;

```
<html> <head>
<title> Оператор ? </title>
</head> <body>
<?php
$var = "плохо";
$text = ($var == "хорошо") ? "Хорошо!" : "Мне $var";
print "$text";
?>
</body> </html>
```

Цикл while

```
while (выражение) {
        // тело цикла
}
```

```
<html> <head>
<title> Цикл while </title>
</head> <body>
<?php
$count = 1;
while ($count <= 10){
   print "$count умножить на 2 будет ".($count*2)."<br>";
   $count++;
}
?>
</body> </html>
```

Цикл do … while

```
do {
    // тело цикла
} while (выражение);
```

```
<html> <head>
<title> Цикл do..while </title>
</head> <body>
<?php
$num = 1;
do {
    print "Номер прохода: $num<br>\n";
    $num++;
} while ($num > 200 && $num < 400);
?>
</body> </html>
```

Цикл for

```
for (инициализация; условие;
приращение)
{
    //тело цикла
}
```

```
<html> <head>
<title> Цикл for </title>
</head> <body>
<?php
for ($count = 1;  $count <=10;  $count ++
){
print "$count умножить на 2 будет
".($count*2)."<br>";
}
?>
</body> </html>
```

Прерывание циклов командой break

```
<html> <head>
<title> Использование команды break
</title> </head> <body>
<?php
$count = -4;
for ( ; $count <=10; $count ++ ) {
if ($count == 0)
    break;
    $temp = 4000/$count;
    print "4000 разделить на $count будет $temp<br>";
   }
?>
</body> </html>
```

Пропуск итераций с помощью команды continue

```
<html> <head>
<title> Использование команды continue
</title> </head> <body>
<?php
$count = -4;
for (  ;  $count <= 10;  $count++ )  {
    if ($count == 0)
       continue;
    $temp = 4000/$count;
    print "4000 разделить на $count будет $temp <br>";
 }
?>
</body> </html>
```

Вложенные циклы

В листинге ниже приведен пример использования вложенных циклов, которые выводят на экран браузера таблицу умножения в таком виде:

1	2	3	4	5
2	4	6	8	10
3	6	9	12	15
4	8	12	16	20
5	10	15	20	25

```
<html> <head>
<title> Вложенные циклы for </title>
</head> <body>
<?php
print "<table style='border: 1px'>\n";
for ($y=1;  $y <= 5;  $y++) {
      print "<tr>\n";
      for ($x=1;  $x <= 5;  $x++) {
           print "\t<td>";
           print ($x*$y);
           print "</td>\n";
      }
      print "</tr>\n";
}
print "</table>";
?>
</body> </html>
```

Функции

Функция — это блок команд, который не исполняется немедленно, но может быть вызван программой в случае необходимости. Функции бывают встроенные и созданные пользователем. Они могут требовать для своей работы некоторые данные и возвращать полученное значение.

Первая программа состояла из единственного вызова функции

```
print "Hello, Web!";
```

print() — не совсем обычная функция. Она не требует, чтобы ее аргумент был заключен в скобки. Таким образом,

```
print "Hello, Web!";
```

и

```
print("Hello, Web!");
```

являются эквивалентными конструкциями. Но это единственное исключение. Все остальные функции требуют открывающей и закрывающей скобок после своего имени, даже если им не передаются никакие аргументы.

Если у функции аргументов несколько, то они должны быть разделены запятыми.

```
name_function($argument_1, $argument_2);
```

Функция *print()*, как и положено, возвращает вам некоторое значение. Большинство функций, закончив свою работу, возвращают значение, хотя бы просто для того, чтобы показать, успешно ли была выполнена данная работа. Функция *print()* возвращает для этого булево значение.

Функция *abs()* вычисляет абсолютную величину числа; для этого она принимает число со знаком, а возвращает неотрицательное число:

```
print(abs(-321));
```

Создание функции

```
Function Primer($argument_1, $argument_2)
 {
  // тело функции
 }
```

```
<html> <head>
<title> Определение функции с аргументами
</title> </head> <body>
<?php
function PrintBR($txt) { print ("$txt<br>\n"); }
PrintBR("Это строка");
PrintBR("Это следующая строка");
PrintBR("Это еще одна строка");
?>
</body> </html>
```

Создание функции, возвращающей значение

```
<html> <head>
<title> Функция, возвращающая значение
</title> </head> <body>
<?php
function AddNums($firstnum, $secondnum) {
        $result = $firstnum + $secondnum;
        return $result;
}
print AddNums(3,5); //будет выведено 8
?>
</body> </html>
```

Можно сократить текст функции, обойдясь без переменной $result, а записав следующим образом:

```
{ return ($firstnum + $secondnum); }
```

Типизация функций

Чтобы указать тип для аргумента, напишите желаемый тип перед именем аргумента.

```
function printString(string $first) {
    print($first);
}
```

Чтобы указать тип возвращаемого значения, укажите его через двоеточие после объявления функции.

```
function returnTrue(): bool {
    return true;
}
```

Типизация поможет писать код без ошибок. Так, если вы попробуете передать функцию, аргумент такого типа, что его нельзя привести к ожидаемому, вы получите ошибку:

```
function sayHelloTo(string $name)
                                    {
                                        print("Hello, {$name}");
                                    }
                                    sayHelloTo([]);
// PHP Fatal error:  Uncaught TypeError: Argument 1 passed to sayHelloTo() must be of the type string, array given
```

Аналогичная ошибка появится и при попытке вернуть из функции значение, не приводимое к указанному типу.

```
                                    function returnTrue(): int {
                                        return [];
                                    }
// Return value of returnTrue() must be of the type int, array returned
```

Функции-переменные

Одной из интересных конструкций PHP являются функции-переменные: имя функции можно присвоить некоторой строковой переменной, а затем обращаться с этим именем точно так же, как с самой функцией.

```
<html> <head>
<title> Функция-переменная </title>
</head> <body>
<?php
// Приветствие на русском языке
   function Russian() { print "<p>Здравствуйте! </p>"; }
// Приветствие на английском языке
   function English() { print "<p>Hello! </p>"; }
$language = "Russian"; // Выбрали русский язык
$language();          // Выполнение функции-переменной
?>
</body> </html>
```

Область видимости переменных

Переменная, созданная в некоторой функции, становится локальной по отношению к данной функции.

Иногда может понадобиться обратиться к некоторой важной переменной, объявленной вне функции, а передавать ее в виде аргумента неудобно. В таком случае используется команда global.

```
<html> <head>
<title> Доступ к глобальной переменной
</title> </head> <body>
<?php
$day = "воскресенье";
function FreeDay()
   {
    global $day;
    print "<p>Выходной: $day </p>";
   }
FreeDay();
?> </body> </html>
```

Запоминание состояния функции между вызовами

```
<html> <head>
<title>  Сохранение значения переменной
        между вызовами </title> </head> <body>
<?php
$num_of_calls = 0;
function ListItem($txt) {
    global $num_of_calls;
    $num_of_calls++;
    print "<b>$num_of_calls: $txt</b>";
}
ListItem("Видеокамеры");
print("<p>Sony, Panasonic</p>");
ListItem("Фотоаппараты");
print("<p>Canon, Casio</p>");
?>
</body> </html>
```

Результат работы программы:

1: Видеокамеры

Sony, Panasonic

2: Фотоаппараты

Canon, Casio

Запоминание состояния функции между вызовами

Использование команды static для запоминания значения переменной

```
<html> <head>
<title> Использование команды static
</title> </head> <body>
<?php
function ListItem($txt) {
    static $num_of_calls = 0;
    $num_of_calls++;
    print "<b>$num_of_calls. $txt</b>";
   }
ListItem("Видеокамеры ");
print("<p>Sony, Panasonic</p>");
ListItem("Фотоаппараты");
print("<p>Canon, Casio</p>");
?> </body> </html>
```

Значения аргументов по умолчанию

Функция с необязательным аргументом

```
<html> <head>
    <title> Функция с необязательным аргументом
    </title> </head> <body>
<?php
function FontSize($txt, $size=4) {
    print "<div style='font-size: {$size}px'>$txt</div>";
}
FontSize("<p>Крупный шрифт</p>",25);
FontSize("<p>Нормальный шрифт, первая строка</p>");
FontSize("<p>Нормальный шрифт, вторая строка</p>");
?>
</body></html>
```

Передача аргумента по ссылке

Передача аргумента по ссылке при вызове функции

```
<html> <head>
<title>  Передача аргумента по ссылке
        при вызове функции </title> </head> <body>
<?php
function AddFive($num) {
    $num +=5;
}
$var = 10;
AddFive(&$var);
print $var; // выводится 15, без & выведется 10
?>
</body> </html>
```

Передача аргумента по ссылке

Передача аргумента по ссылке
при определении функции

```
<html> <head>
<title> Передача аргумента по ссылке
        при определении функции </title> </head> <body>
<?php
function AddFive(&$num){
$num += 5;
}
$var = 10;
AddFive($var);
print $var;
?>
</body> </html>
```