

Лекция 3

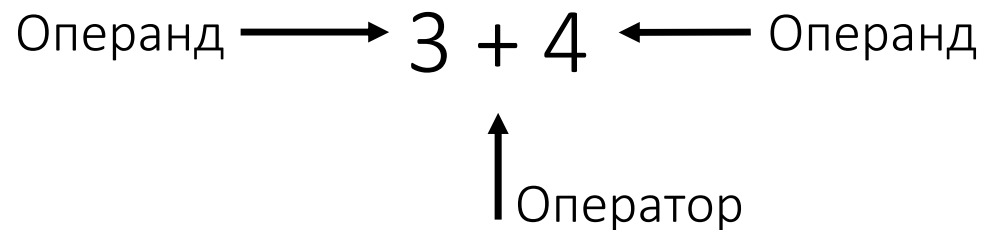
Функции работы с переменными в RHR.
Математические операции.

Операторы и операнды

Для осуществления операций с переменными существуют различные группы операторов.

Оператор - это элемент программного кода, который описывает то или иное действие в выражении. В PHP оператор представляет собой символ, благодаря которому могут производиться различные виды вычислений, сравнений или присваиваний с участием одного или нескольких значений.

Операнд - это то, на что воздействует оператор. Операнды и операторы дают в результате выражение, которое формирует новое значение.

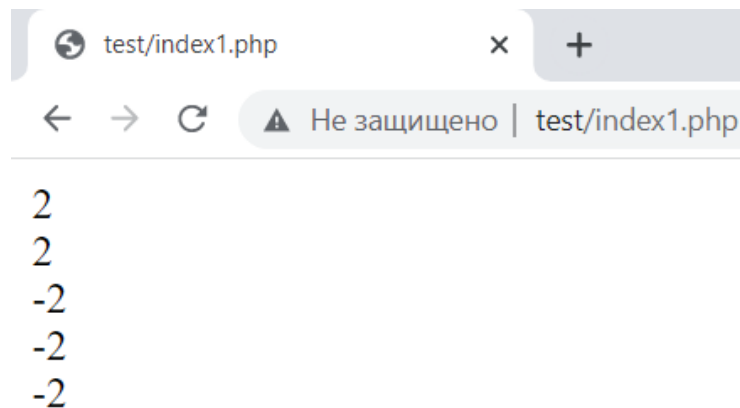


Арифметические операции в РНР

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 $\$b$
$\$a ** \b	Возведение в степень	Возводит значение переменной $\$a$ в степень заданную значением переменной $\$b$

Арифметические операции в PHP

```
<?php
echo (5 % 3) . "<br>";           // выводит 2
echo (5 % -3) . "<br>";          // выводит 2
echo (-5 % 3) . "<br>";           // выводит -2
echo (-5 % -3) . "<br>";          // выводит -2
echo (-5.5 % -3.3) . "<br>";      // выводит -2
?>
```



Арифметические операции в РНР

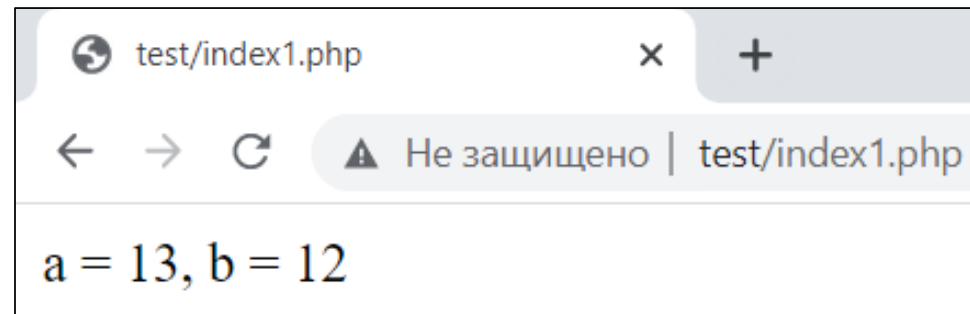
1. Операция деления ("/") возвращает число с плавающей точкой, кроме случая, когда оба значения являются целыми числами (или строками, которые преобразуются в целые числа), которые делятся нацело - в этом случае возвращается целое значение.
2. Операцию вычисления остатка от деления "%" рекомендуется использовать только с целыми числами, так как применение ее к дробным может привести к нежелательному и непредсказуемому результату.
3. Приоритет одних математических операций над другими и изменение приоритетов при использовании скобок в арифметических выражениях соответствуют обычным математическим правилам.

Операции инкремента и декремента

Пример	Название	Действие
<code>++\$a</code>	Префиксный инкремент	Увеличивает <code>\$a</code> на единицу и возвращает значение <code>\$a</code>
<code>\$a++</code>	Постфиксный инкремент	Возвращает значение <code>\$a</code> , а затем увеличивает <code>\$a</code> на единицу
<code>--\$a</code>	Префиксный декремент	Уменьшает <code>\$a</code> на единицу и возвращает значение <code>\$a</code>
<code>\$a--</code>	Постфиксный декремент	Возвращает значение <code>\$a</code> , а затем уменьшает <code>\$a</code> на единицу

Операции инкремента и декремента

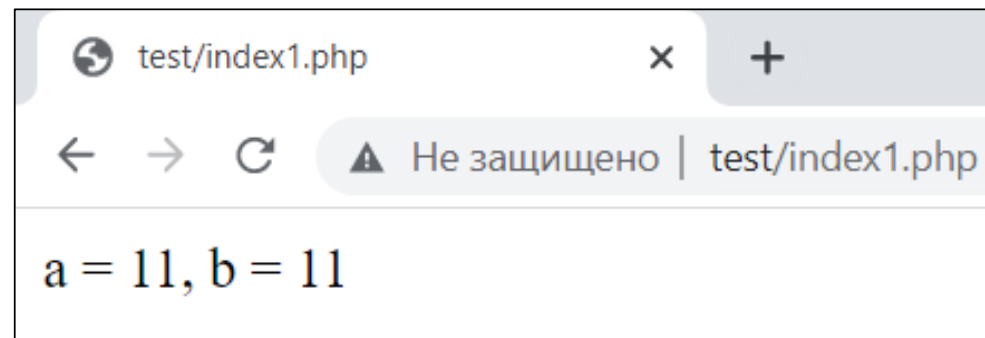
```
$a = 12;  
$b = $a++; // $b равно 12  
echo "a = $a    b = $b";
```



Здесь сначала значение переменной `$a` передается переменной `$b`, а затем происходило увеличение значения переменной `$a`.

Операции инкремента и декремента

```
$a = 12;  
$b = --$a; // $b равно 11  
echo "a = $a    b = $b";
```



Здесь сначала значение переменной `$a` уменьшается на единицу, а затем ее значение присваивается переменной `$b`.


```
<?php
echo "<h3>Постфиксный инкремент</h3>";
$a = 5;
echo "Должно быть 5: " . $a++ . "<br>";
echo "Должно быть 6: " . $a . "<br>";

echo "<h3>Префиксный инкремент</h3>";
$a = 5;
echo "Должно быть 6: " . ++$a . "<br>";
echo "Должно быть 6: " . $a . "<br>";

echo "<h3>Постфиксный декремент</h3>";
$a = 5;
echo "Должно быть 5: " . $a-- . "<br>";
echo "Должно быть 4: " . $a . "<br>";

echo "<h3>Префиксный декремент</h3>";
$a = 5;
echo "Должно быть 4: " . --$a . "<br>";
echo "Должно быть 4: " . $a . "<br>";
?>
```

test/index1.php x +
← → ↻ Не защищено | test/index1.php

Постфиксный инкремент

Должно быть 5: 5
Должно быть 6: 6

Префиксный инкремент

Должно быть 6: 6
Должно быть 6: 6

Постфиксный декремент

Должно быть 5: 5
Должно быть 4: 4

Префиксный декремент

Должно быть 4: 4
Должно быть 4: 4

Операции инкремента с символьными переменными

Операции инкремента можно применять и для символьных переменных. PHP следует соглашениям Perl (в отличие от C) касательно выполнения арифметических операций с символьными переменными.

Например, в PHP и Perl

```
$a = 'Z';
```

```
$a++;
```

присвоит \$a значение 'AA', в то время как в C

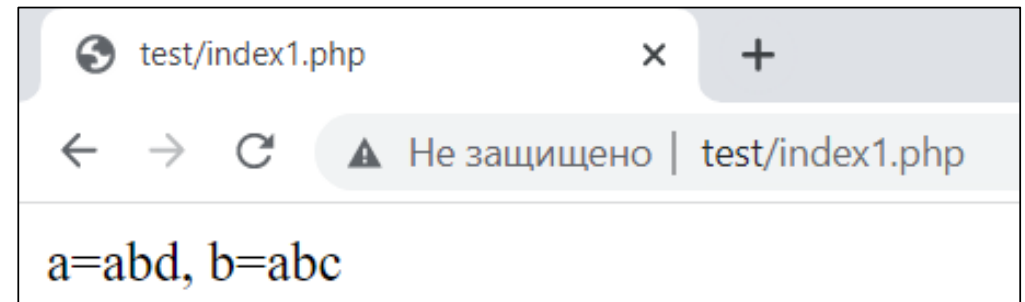
```
a = 'Z';
```

```
a++;
```

присвоит a значение '[' (ASCII-значение 'Z' равно 90, а ASCII-значение '[' равно 91).

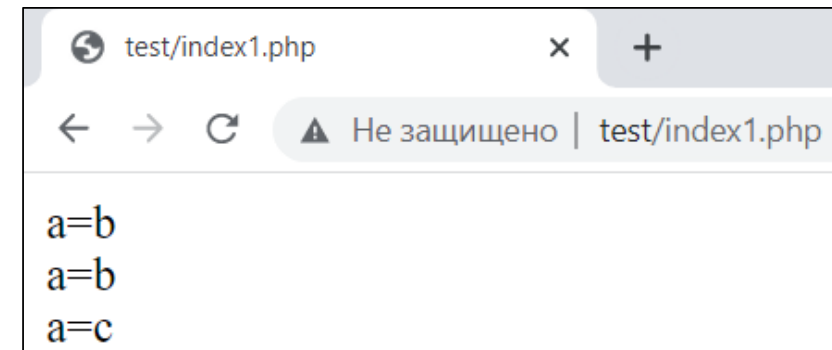
Операции инкремента с символьными переменными

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

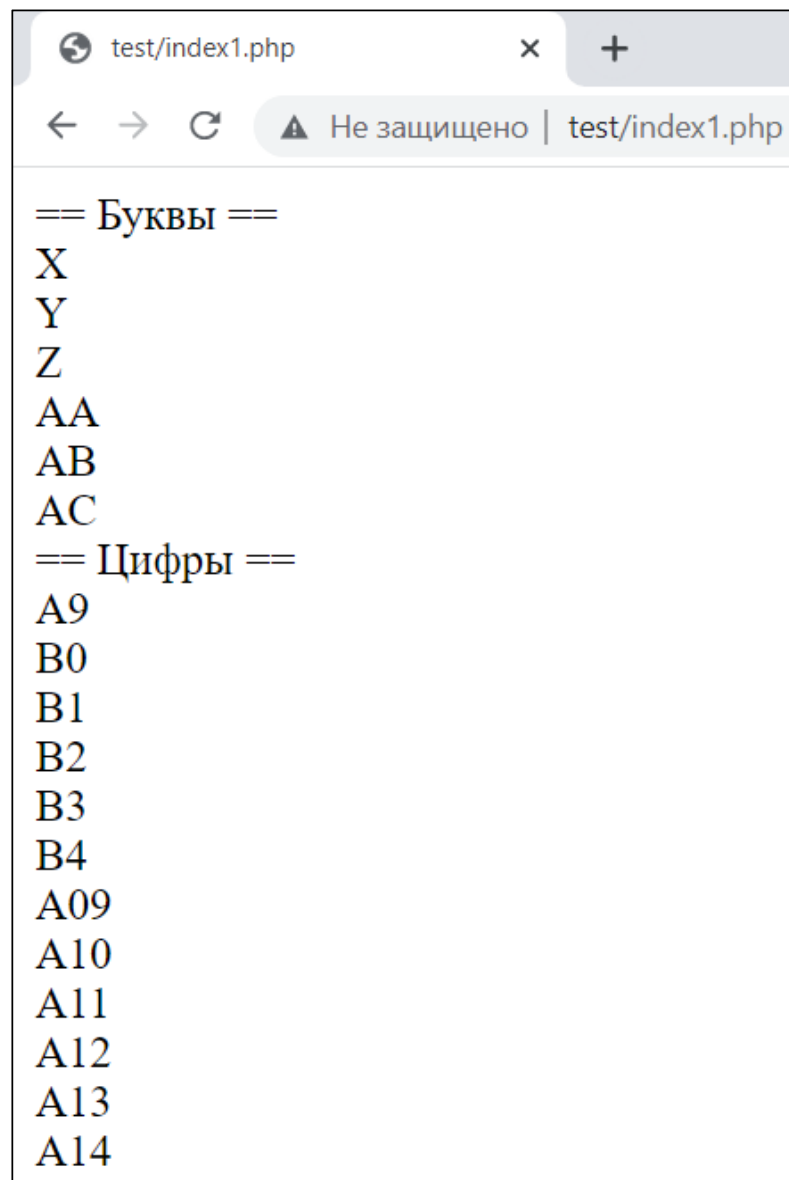


Операции инкремента с символьными переменными

```
<?php
    $a='a';
    echo 'a=' . ++$a, "<br>";
// Выводит a=b
    echo 'a=' . $a++, "<br>";
// Выводит a=b, а затем a увеличивает на 1
    echo 'a=' . $a;
?>
```



```
<?php
echo '== Буквы ==' . "<br>";
$s = 'W';
for ($n=0; $n<6; $n++) {
    echo ++$s . "<br>";
}
// С цифрами несколько по-другому
echo '== Цифры ==' . "<br>";
$d = 'A8';
for ($n=0; $n<6; $n++) {
    echo ++$d . "<br>";
}
$d = 'A08';
for ($n=0; $n<6; $n++) {
    echo ++$d . "<br>";
}
?>
```



The screenshot shows a web browser window with the address bar displaying "test/index1.php". The page content is as follows:

```
== Буквы ==
X
Y
Z
AA
AB
AC
== Цифры ==
A9
B0
B1
B2
B3
B4
A09
A10
A11
A12
A13
A14
```

Операции инкремента с символьными переменными

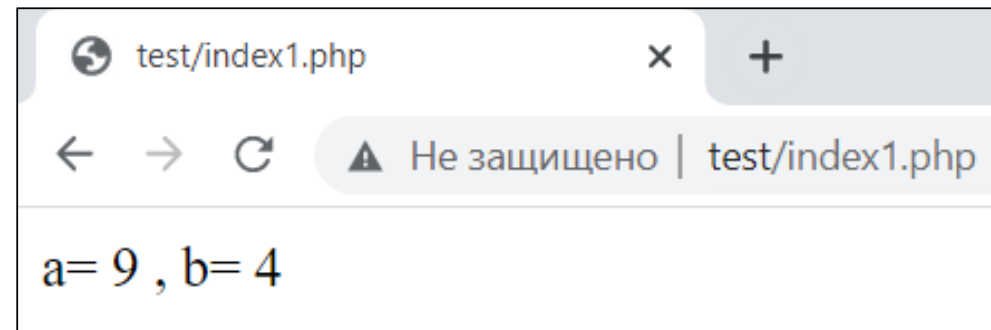
Важно

Булевые типы не подлежат инкрементированию и декрементированию.

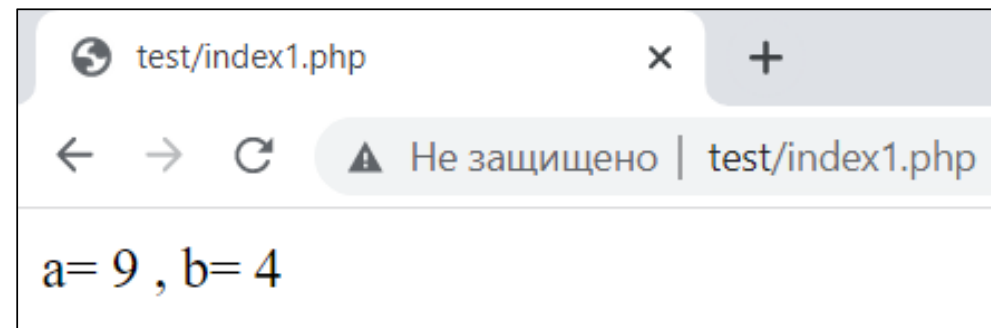
Символьный тип подлежит только операции инкрементирования. Кроме того поддерживается только алфавит ASCII и цифры (a-z, A-Z и 0-9).

Операции присвоения

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

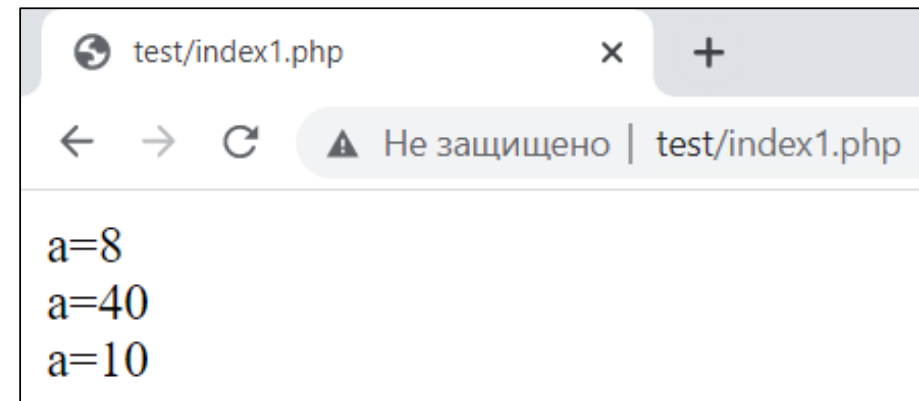


```
<?php
$a = ($b=4) + 5; // результат: $a
                  установлена значением 9,
                  переменной $b
                  присвоено 4.
echo "a= $a , b= $b";
?>
```



Комбинирование арифметических операций и присвоения

```
<?php
$a = 3;
$a += 5; // устанавливает $a значением 8,
аналогично записи: $a = $a + 5;
echo "a=$a";
echo"<br>";
$a *= 5; // устанавливает $a значением 40,
аналогично записи: $a = $a * 5;
echo "a=$a";
echo"<br>";
$a /= 4; // устанавливает $a значением 10,
аналогично записи: $a = $a / 4;
echo "a=$a";
echo"<br>";
?>
```



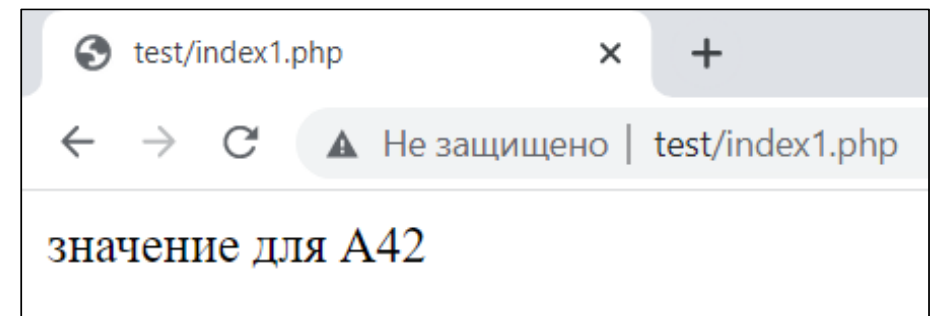
Операции присвоения

Начиная с PHP 7, добавился новый оператор «??» (null coalescing), который позволяет присваивать значение по умолчанию. Приведем простой пример.

```
<?php
$nullValue = NULL;
$emptyText = ""; // ложноподобное
$someNumber = 42;

$valA = $nullValue ?? "значение для A";
$valB = $emptyText ?? "значение для B";
$valC = $someNumber ?? 0;

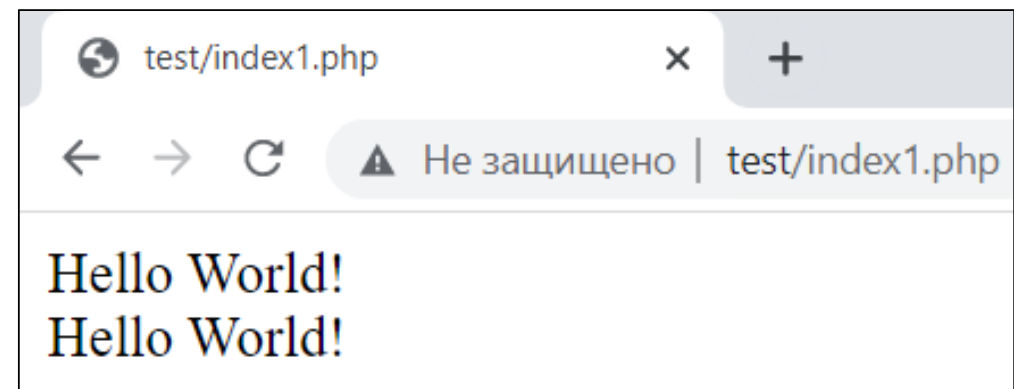
echo $valA;    // "значение для A"
echo $valB;    // "" (поскольку пустая
               строка не приравнивается к null или
               undefined)
echo $valC;    // 42
?>
```



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.

```
<?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;
?>
```



Побитовые операции

Побитовые операторы выполняют низкоуровневые манипуляции с битами в двоичных представлениях чисел. Биты целого числа — это не что иное, как отдельные разряды того же самого числа, записанного в двоичной системе счисления. 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 операции

1 байт (8 бит)										
Установленное значение	128	64	32	16	8	4	2	1		
	1	1	1	1	1	1	1	1		
	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0		
	128	64	32	16	8	4	2	1	=	255

1 байт (8 бит)										
Установленное значение	128	64	32	16	8	4	2	1		
	0	1	0	1	1	1	0	1		
	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0		
	0	64	0	16	8	4	0	1	=	93

Побитовые операции

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 .
$a \wedge b$	Исключающее 'ИЛИ'	Устанавливаются только те биты, которые установлены либо только в a , либо только в b .
$\sim a$	Отрицание	Устанавливаются те биты, которые в a не установлены, и наоборот.
$a \ll b$	Сдвиг влево	Все биты переменной a сдвигаются на b позиций влево (каждая позиция подразумевает 'умножение на 2')
$a \gg b$	Сдвиг вправо	Все биты переменной a сдвигаются на b позиций вправо (каждая позиция подразумевает 'деление на 2')

Побитовые операции

1. Побитовое «И»:

И	0011
	0101
	0001

, т.е. в десятичной форме 1.

2. Побитовое «ИЛИ»:

ИЛИ	0011
	0101
	0111

, т.е. в десятичной форме 7.

3. Побитовое исключающее «ИЛИ»:

Искл. ИЛИ	0011
	0101
	0110

, т.е. в десятичной форме 6.

Побитовые операции

4. Сдвиг влево (не забываем, что все числа в rhr 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.

13<<2 → 0000 0000 0000 0000 0000 0000 0000 1101<<2 →
→ 0000 0000 0000 0000 0000 0000 0011 0100, т.е. в десятичной форме 52.

13>>2 → 0000 0000 0000 0000 0000 0000 0000 1101>>2 →
→ 0000 0000 0000 0000 0000 0000 0000 0011, т.е. в десятичной форме 3.

```
<?php
```

```
$a = 12; //в двоичной системе 12 - это 1100
```

```
$b=$a|6;
```

```
echo "12|6 = $b", "<br>"; // результат = 14
```

```
$b=$a&6;
```

```
echo "12&6 = $b", "<br>"; // результат = 4
```

```
$b=$a^6;
```

```
echo "12^6 = $b", "<br>"; // результат = 10
```

```
$b=$a<<2;
```

```
echo "$a<<2 = $b", "<br>"; // результат = 48
```

```
$b=++$a<<2;
```

```
echo "$a<<2 = $b", "<br>"; // результат = 52
```

```
$b=$a>>2;
```

```
echo "$a>>2 = $b", "<br>"; // результат = 3
```

```
$b=$a>>3;
```

```
echo "$a>>2 = $b"; // результат = 1
```

```
?>
```

test/index1.php

x

+



Не защищено | test/index1.php

12|6 = 14

12&6 = 4

12^6 = 10

12<<2 = 48

13<<2 = 52

13>>2 = 3

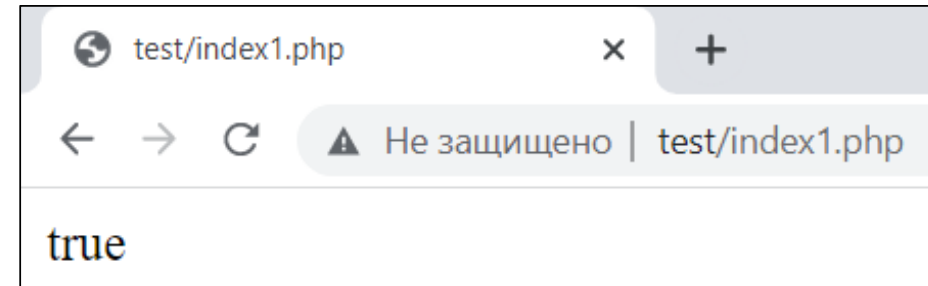
13>>2 = 1

Операции 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
$\$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 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 $\$a$ <u>is</u> меньше или равно $\$b$
$\$a >= \b	Больше или равно	TRUE если $\$a$ больше или равно $\$b$
$\$a <=> \b	<u>spaceship</u>	Число типа <u>integer</u> меньше, больше или равное нулю, когда $\$a$ соответственно меньше, больше или равно $\$b$ (доступно с PHP 7).

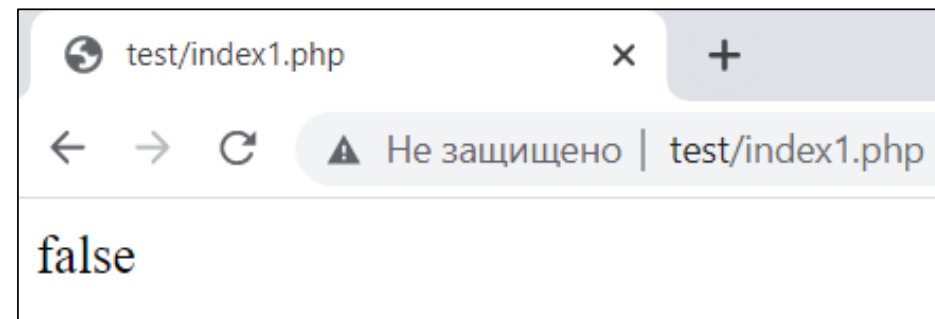
Операции сравнения

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



Операции сравнения

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

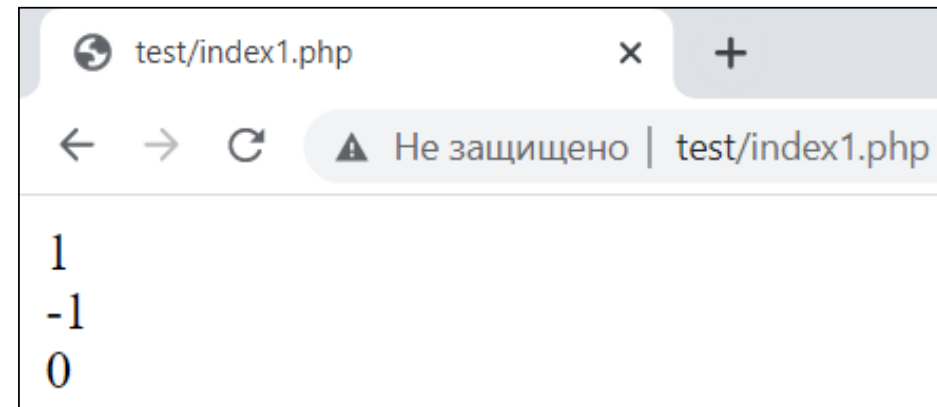


Операции сравнения

```
<?php
$a = 12;
$b=11;
echo ($a<=>$b) ;
echo "<br>";

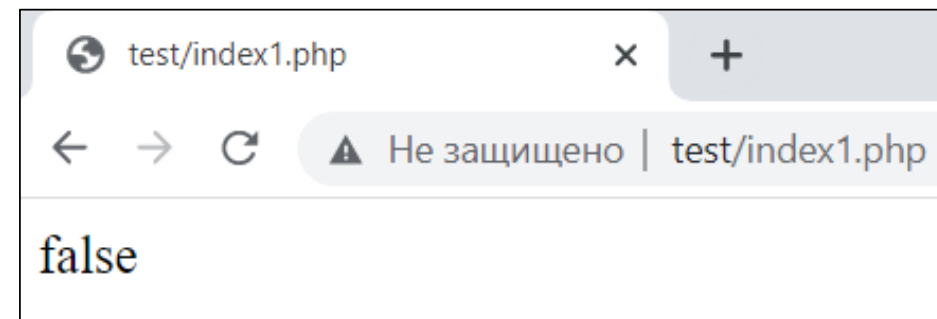
$a = 11;
$b=12;
echo ($a<=>$b) ;
echo "<br>";

$a = 11;
$b=11;
echo ($a<=>$b) ;
echo "<br>";
?>
```



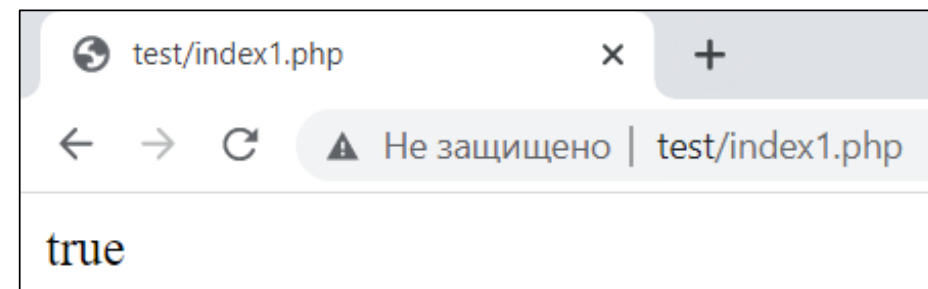
Операции сравнения

```
<?php
$a = 1/3; //переменная типа float
$b=0.3333333333333333; //переменная типа
float (14 знаков после запятой)
if ($a==$b) {
    echo 'true';
}
else {
    echo 'false';
}
echo"<br>";
?>
```



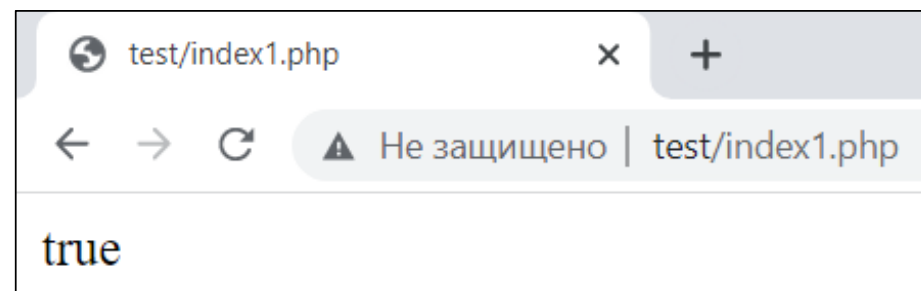
Операции сравнения

```
<?php
$a = 1/3; //переменная типа float
$b=0.3333333333333333; //переменная типа
float (16 знаков после запятой)
if ($a==$b) {
    echo 'true';
}
else {
    echo 'false';
}
echo "<br>";
?>
```



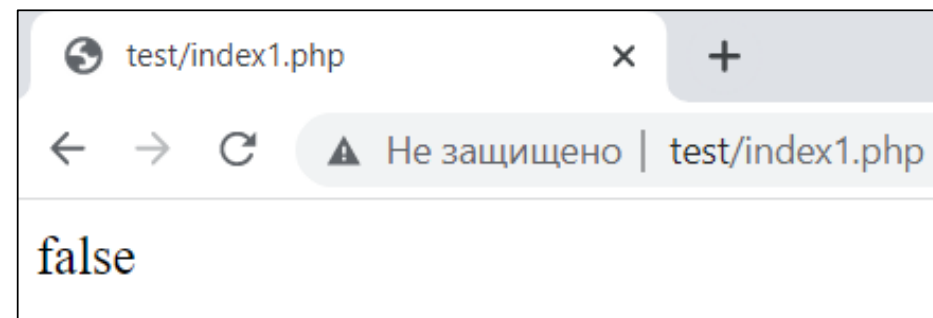
Операции сравнения

```
<?php
$a = 0;
$b = "";
if ($a==$b) {
    echo 'true';
}
else {
    echo 'false';
}
echo"<br>";
?>
```



Операции сравнения

```
<?php
$a = 0;
$b = "";
if ($a=== $b) {
    echo 'true';
}
else {
    echo 'false';
}
echo "<br>";
?>
```



Логические операции

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 (инвертирует логическое значение операнда).
$\$a$ && $\$b$	Логическое 'И'	TRUE если и $\$a$, и $\$b$ TRUE.
$\$a$ $\$b$	Логическое 'ИЛИ'	TRUE если или $\$a$, или $\$b$ TRUE.

Отметим, что “| |” имеет больший приоритет, чем “OR”, а “&&” – чем “AND”.

Логические операции

Следует заметить, что вычисление логических выражений, содержащих представленные в таблице 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. Рассмотрим следующий пример.

```
<?php
```

```
    $a = 10;
```

```
    $b=-5;
```

```
    echo 'Результат условия ($a<0 and ++$b<0) : ';
```

```
    var_dump(($a<0 and ++$b<0));
```

```
    echo "<br>";
```

```
    var_dump($b);
```

```
    echo "<br>";
```

```
    echo 'Результат условия ($a>0 and ++$b<0) : ';
```

```
    var_dump(($a>0 and ++$b<0));
```

```
    echo "<br>";
```

```
    var_dump($b);
```

```
?>
```

test/index1.php



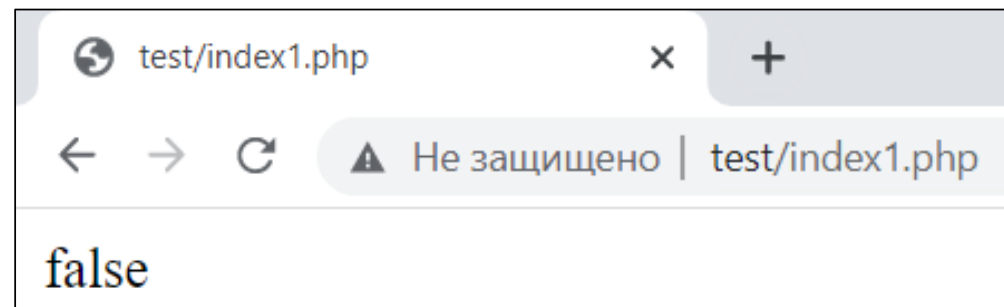
Не защищено | test/index1.php

Результат условия (\$a<0 and ++\$b<0) : bool(false)
int(-5)

Результат условия (\$a>0 and ++\$b<0) : bool(true)
int(-4)

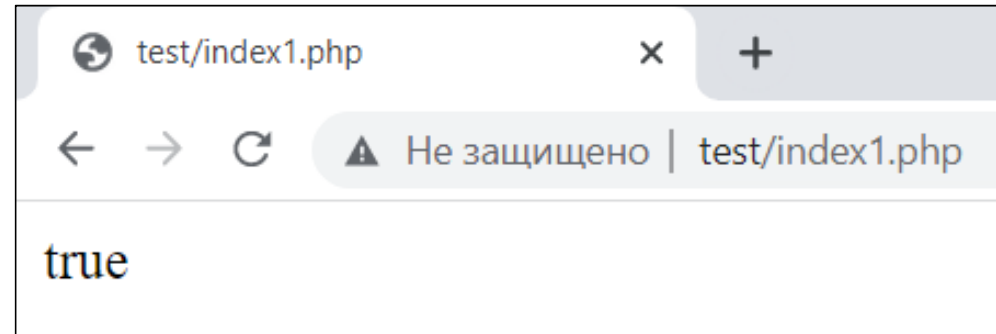
Логические операции

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



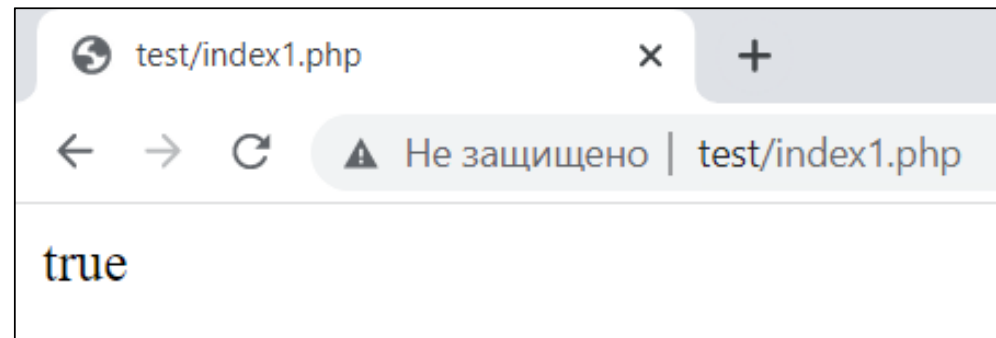
Логические операции

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



Логические операции

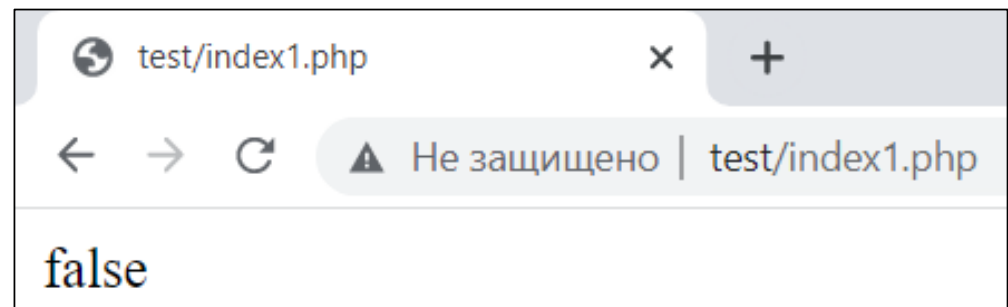
```
<?php
$a = 10;
$b=-5;
if ($a>0 xor $b>0)
{
    echo 'true';
} else
{
    echo 'false';
}
?>
```



Оператор **исключающее ИЛИ** обозначается как XOR. Он возвращает значение true, если один и только один из операндов имеет значение true. Если оба операнда имеют значение true, оператор вернет значение false.

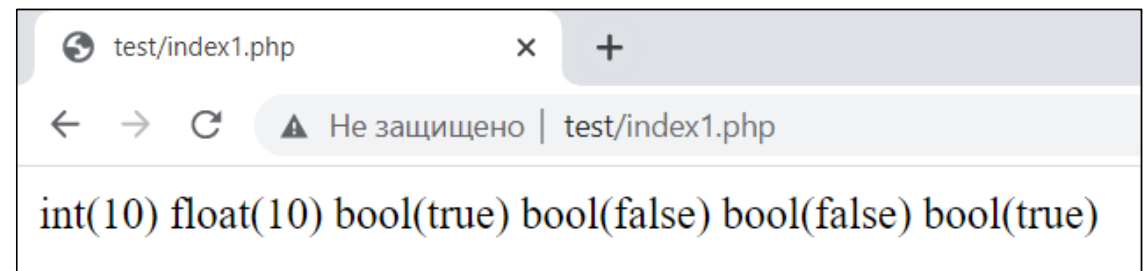
Логические операции

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



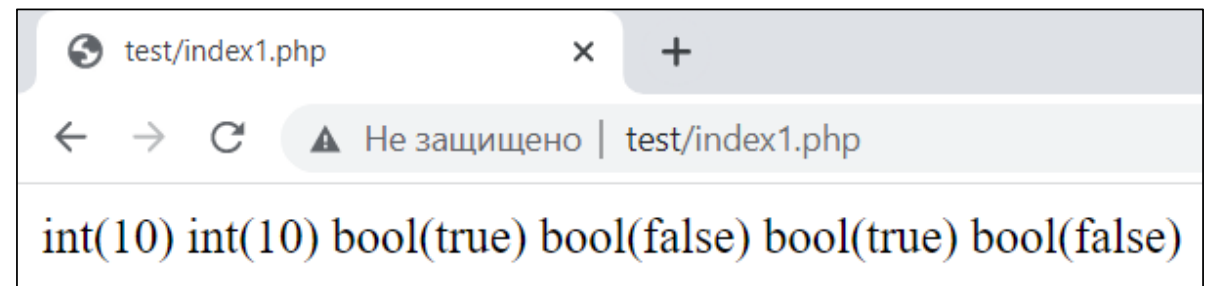
Логические операции

```
<?php
$a = 10;
var_dump($a);
$b=55/5.5;
var_dump($b);
var_dump($a==$b);
var_dump(!$a==$b);
var_dump($a=== $b);
var_dump(!$a=== $b);
?>
```



Логические операции

```
<?php
$a = 10;
var_dump($a);
$b=50/5;
var_dump($b);
var_dump($a==$b);
var_dump(!$a==$b);
var_dump($a=== $b);
var_dump(!$a=== $b);
?>
```



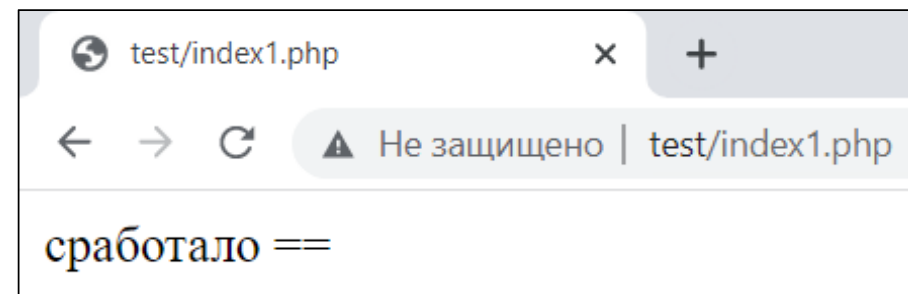
Операторы, работающие с массивами

Пример	Название	Результат
$\$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 пары 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 $\$b$.

Операторы, работающие с массивами

```
<?php
$a = array("apple", "banana");
$b = array(1 => "banana", 0 => "apple");

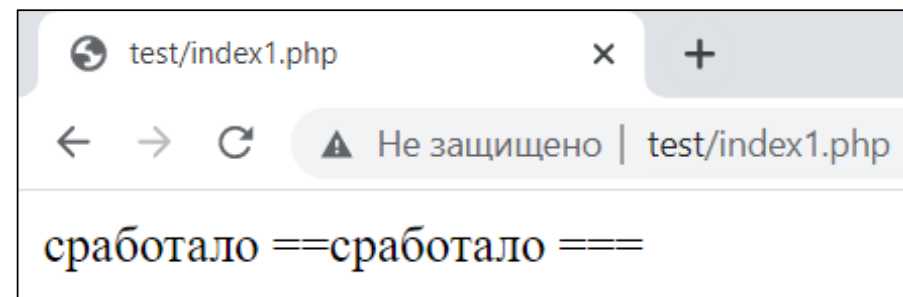
if ($a == $b) echo "сработало ==";
// bool(true)
if ($a === $b) echo "сработало ===";
// bool(false)
?>
```



Операторы, работающие с массивами

```
<?php
$a = array("apple", "banana");
$b = array(0 => "apple", 0 => "banana");

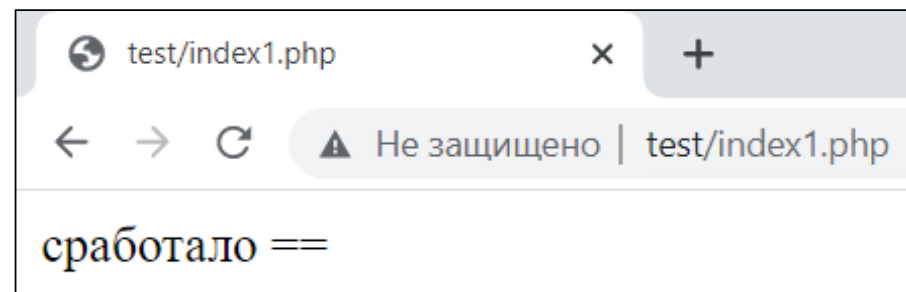
if ($a == $b) echo "сработало ==";
// bool(true)
if ($a === $b) echo "сработало ===";
// bool(false)
?>
```



Операторы, работающие с массивами

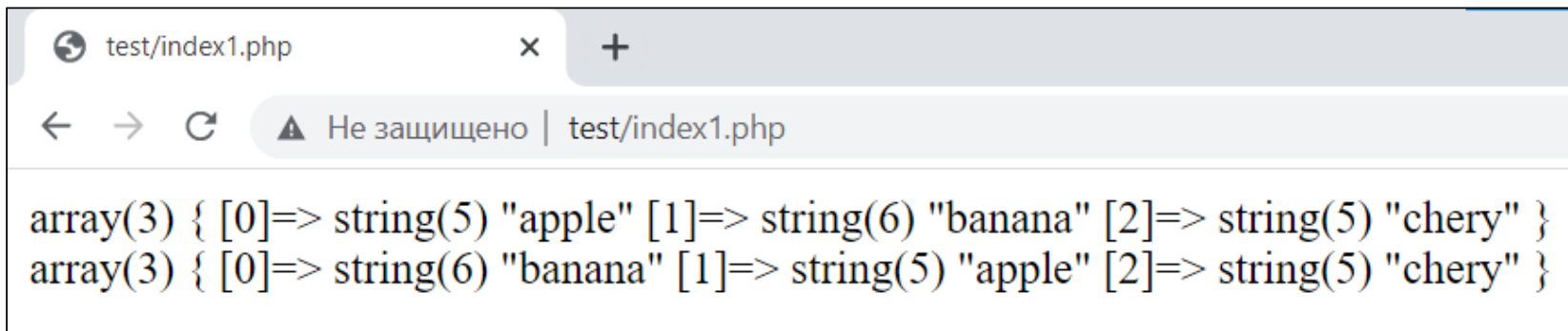
```
<?php
$a = array(1, 2, 3);
$b = array("1", "2", "3");

if ($a == $b) echo "сработало ==";
// bool(true)
if ($a === $b) echo "сработало ===";
// bool(false)
?>
```



Объединение массивов

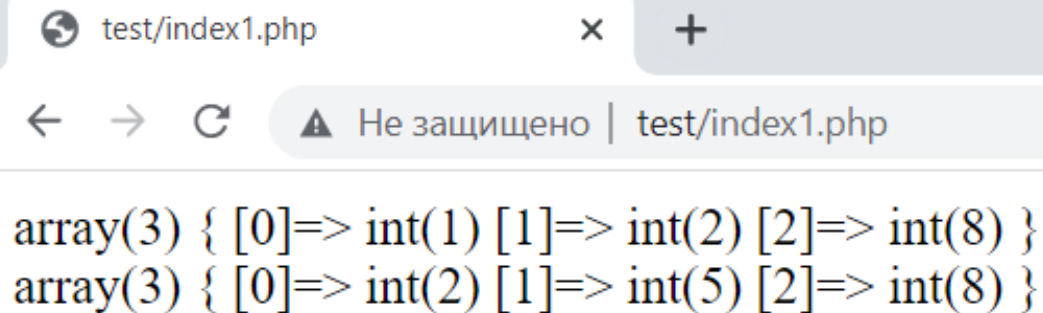
```
<?php
$a = array("apple", "banana");
$b = array("banana", "apple", "chery");
$c=$a+$b;
var_dump($c);
echo "<br>";
$c=$b+$a;
var_dump($c);
?>
```



```
array(3) { [0]=> string(5) "apple" [1]=> string(6) "banana" [2]=> string(5) "chery" }
array(3) { [0]=> string(6) "banana" [1]=> string(5) "apple" [2]=> string(5) "chery" }
```

Объединение массивов

```
<?php
$a = array(1, 2);
$b = array(2, 5, 8);
$c=$a+$b;
var_dump($c);
echo"<br>";
$c=$b+$a;
var_dump($c);
?>
```



test/index1.php × +

← → ↻ ⚠ Не защищено | test/index1.php

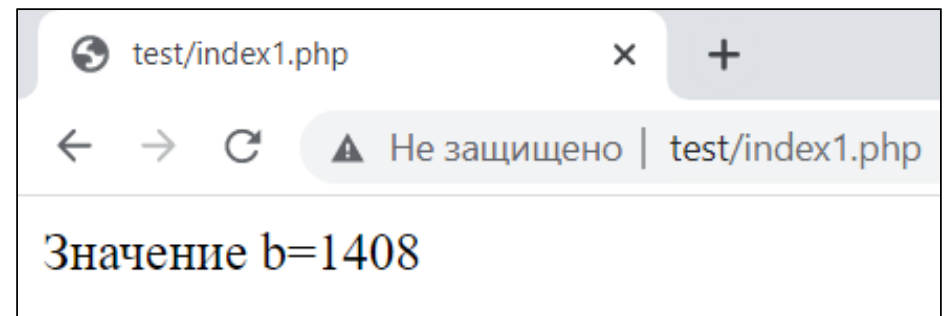
```
array(3) { [0]=> int(1) [1]=> int(2) [2]=> int(8) }
array(3) { [0]=> int(2) [1]=> int(5) [2]=> int(8) }
```

Приоритеты операторов

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

Приоритеты операторов

```
<?php
$a=11;
$b=$a<<2+5;
echo 'Значение b=' . $b;
?>
```

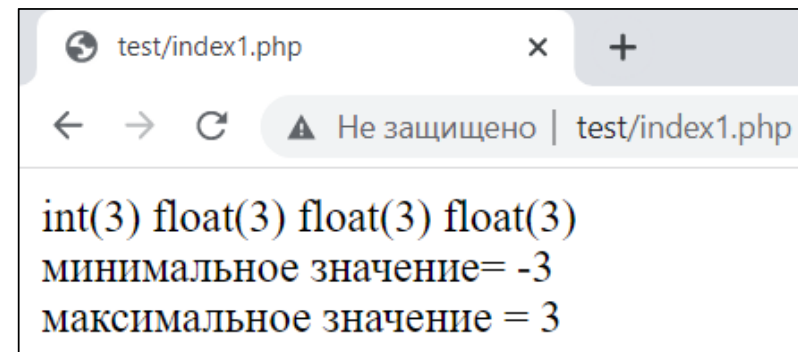


Простые математические функции

Функция	Описание
<u>floor()</u>	Принимает единственный фактический параметр (как правило, число с плавающей точкой двойной точности) и возвращает наибольшее целое число, которое меньше или равно этому фактическому параметру (округление в меньшую сторону)
<u>ceil()</u>	Имя этой функции представляет собой сокращение от слова <u>ceiling</u> (потолок). Функция принимает единственный фактический параметр (как правило, число с плавающей точкой) и возвращает наименьшее целое число, которое больше или равно этому фактическому параметру (округление в большую сторону)
<u>round()</u>	Принимает единственный фактический параметр (как правило, число с плавающей точкой двойной точности) и возвращает ближайшее целое число
<u>abs()</u>	Модуль числа. Если единственный числовой фактический параметр имеет отрицательное значение, то функция возвращает соответствующее положительное число; если фактический параметр является положительным, то функция возвращает сам фактический параметр
<u>min()</u>	Принимает любое количество числовых фактических параметров (но не менее одного) и возвращает наименьшее из всех значений фактических параметров
<u>max()</u>	Принимает любое количество числовых фактических параметров (но не менее одного) и возвращает наибольшее из всех значений фактических параметров
<u>sqrt()</u>	Вычисляет квадратный корень

Простые математические функции

```
<?php
var_dump (abs(-3));
var_dump (round(2.7));
var_dump (ceil(2.3));
var_dump (floor(3.9));
$result = min(2, abs(-3), -3, ceil(2.3) );
echo 'минимальное значение= '.$result;
$result = max(2, abs(-3), -3, ceil(2.3) );
echo "<br>";
echo 'максимальное значение = '.$result;
?>
```



Математические константы

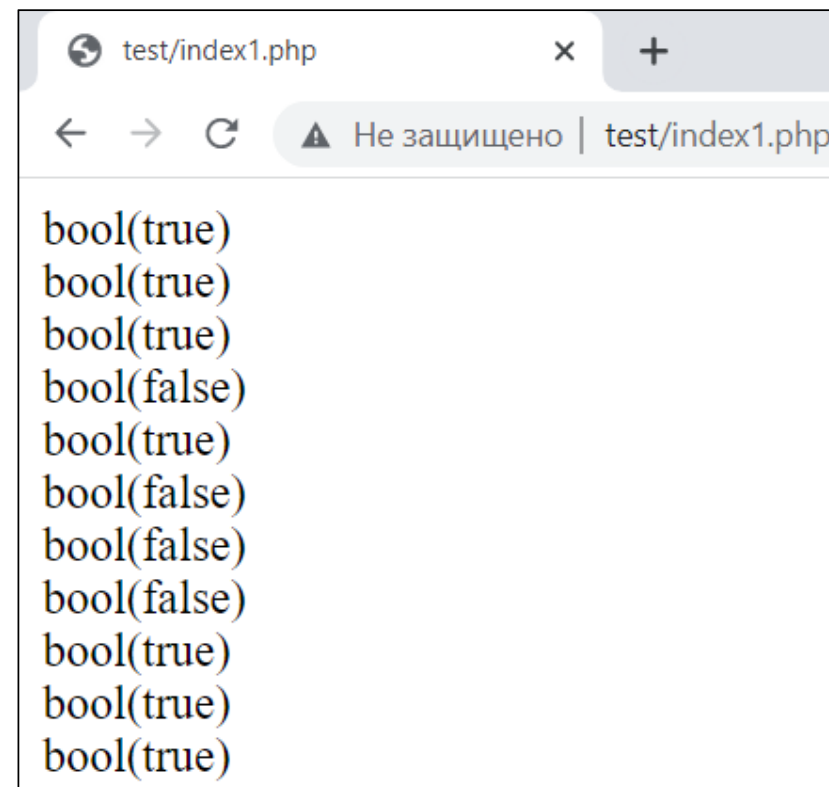
Константа	Описание
M_PI	π
M_PI_2	$\pi / 2$
M_PI_4	$\pi / 4$
M_1_PI	$1 / \pi$
M_2_PI	$2 / \pi$
M_2_SQRTPI	$2 / \sqrt{\pi}$
M_E	e
M_SQRT2	$\sqrt{2}$
M_SQRT1_2	$1 / \sqrt{2}$
M_LOG2E	$\log_2(e)$
M_LOG10E	$\lg(e)$
M_LN2	$\log_e(2)$
M_LN10	$\log_e(10)$

Проверка формата чисел

1. Первая и наиболее простая проверка заключается в использовании функции `is_numeric()`. Как и при осуществлении большинства других таких проверок, функция `is_numeric` возвращает булев результат - *true*, если переданный ей параметр представляет собой числовые данные любого типа (со знаком или без знака, целочисленные или с плавающей точкой).
2. С помощью функций `is_int()` и `is_float()` можно определить является ли число целым или дробным.
3. `is_finite()` - является ли переданная переменная (значение) допустимым конечным числом на данной платформе. `is_infinite()` возвращает `TRUE`, если `val` является бесконечностью (положительной или отрицательной), например, как результат вычисления `log(0)` или любые другие значения, слишком большие, чтобы уместиться в `float` на данной платформе.

Проверка формата чисел

```
<?php
    var_dump(is_numeric(4));           // true
    var_dump(is_numeric(25 - 6));      // true
    var_dump(is_numeric("25"));        // true
    var_dump(is_numeric("25 - 6"));    // false
    var_dump(is_int(4));               // true
    var_dump(is_int(4.2));             // false
    var_dump(is_int("4"));             // false
    // false - данная проверка строже, чем
    // проверка с помощью функции is_numeric()
    var_dump(is_float(4));             // false
    var_dump(is_float(4.0));           // true
    var_dump(is_float(M_PI));          // true
    var_dump(is_float(M_PI));          // true
?>
```



test/index1.php x +

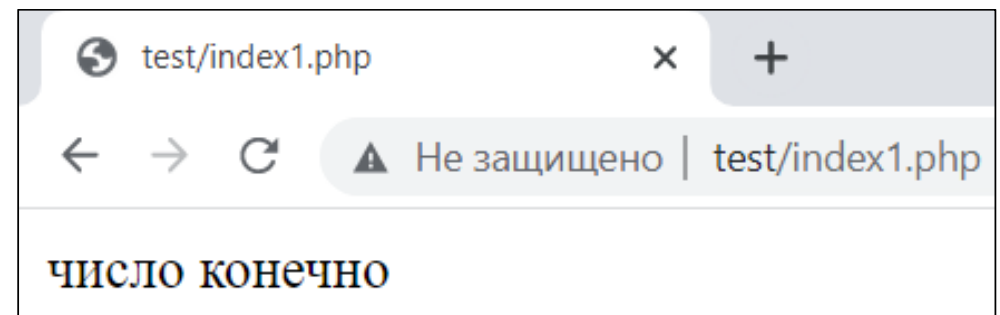
← → ↻ ⚠ Не защищено | test/index1.php

bool(true)
bool(true)
bool(true)
bool(false)
bool(true)
bool(false)
bool(false)
bool(false)
bool(true)
bool(true)
bool(true)

Проверка формата чисел

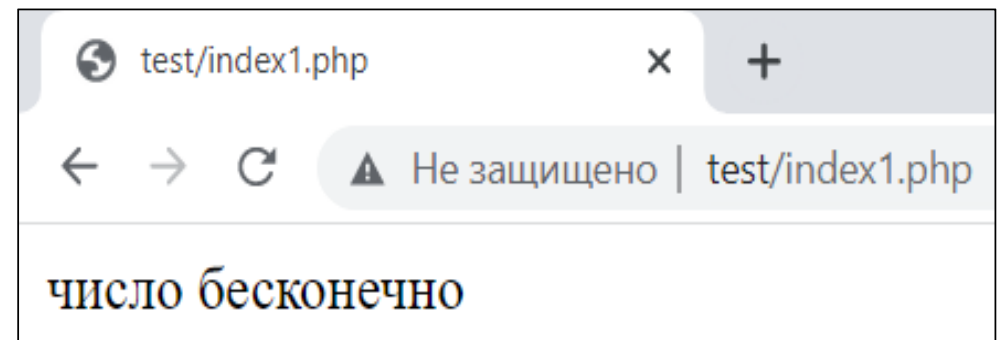
```
<?php
    if (is_infinite(log(1))===true)
    { echo 'число бесконечно'; }
    else
    { echo 'число конечно'; }

?>
```



```
<?php
    if (is_infinite(log(0))===true)
    { echo 'число бесконечно'; }
    else
    { echo 'число конечно'; }

?>
```

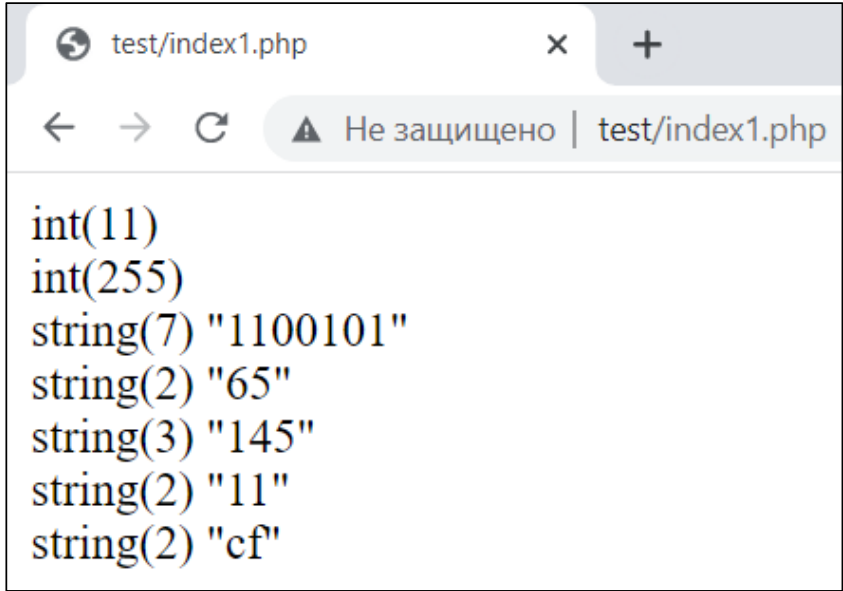


Преобразование систем счисления

Функция	Описание
<u><i>BinDec()</i></u>	Принимает единственный строковый параметр, представляющий собой двоичное целое число (число по основанию 2), и возвращает строковое представление этого числа по основанию системы счисления 10
<u><i>DecBin()</i></u>	Аналогична <u><i>BinDec()</i></u> , но преобразует из основания системы счисления 10 в основание системы счисления 2
<u><i>OctDec()</i></u>	Аналогична <u><i>BinDec()</i></u> , но преобразует из основания системы счисления 8 в основание системы счисления 10
<u><i>DecOct()</i></u>	Аналогична <u><i>BinDec()</i></u> , но преобразует из основания системы счисления 10 в основание системы счисления 8
<u><i>HexDec()</i></u>	Аналогична <u><i>BinDec()</i></u> , но преобразует из основания системы счисления 16 в основание системы счисления 10
<u><i>DecHex()</i></u>	Аналогична <u><i>BinDec()</i></u> , но преобразует из основания системы счисления 10 в основание системы счисления 16
<u><i>base_convert()</i></u>	Принимает строковый параметр (представляющий целое число, которое подлежит преобразованию) и два целочисленных параметра (исходное и желаемое основание). Возвращает строку, представляющую преобразованное число. В этой строке цифры старше, чем 9 (от 10 до 35), представлены символами a-z. И исходное, и желаемые основания должны находиться в пределах 2-36

Преобразование систем счисления

```
<?php
    var_dump (bindec('1011'));
    var_dump (hexdec('FF'));
    var_dump (decbin('101'));
    var_dump (dechex('101'));
    var_dump (decoct('101'));
    var_dump (base_convert('1011',2,10));
    var_dump (base_convert('FF',16, 20));
?>
```



test/index1.php x +

← → ↻ ⚠ Не защищено | test/index1.php

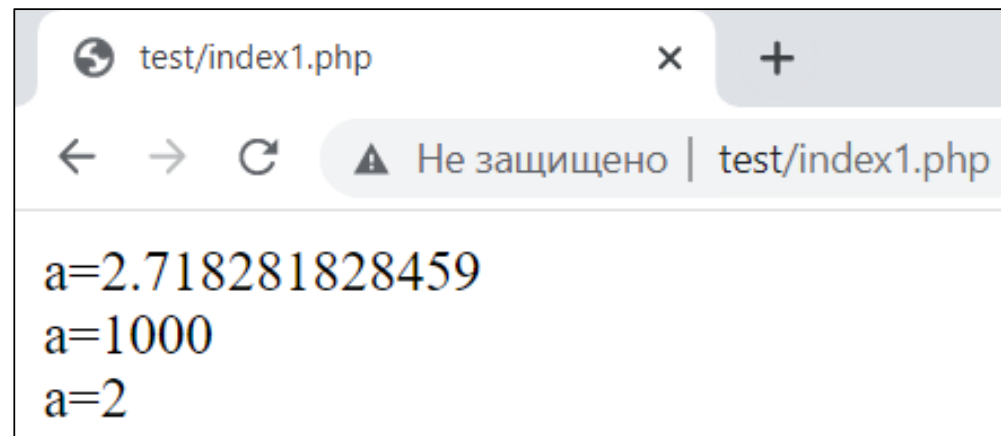
```
int(11)
int(255)
string(7) "1100101"
string(2) "65"
string(3) "145"
string(2) "11"
string(2) "cf"
```

Экспоненты и логарифмы

Функция	Описание
<u>pow()</u>	Принимает два числовых параметра и возвращает первый параметр, возведенный в степень, равную второму параметру. Значение выражения <u>pow(\$x, \$y)</u> равно <u>x^y</u>
<u>exp()</u>	Принимает единственный параметр и возводит число e в степень, равную этому показателю степени. Значение выражения <u>exp(\$x)</u> равно <u>$e^x$</u>
<u>log()</u>	Функция натурального логарифма (<u>ln</u>). Принимает единственный параметр и возвращает его логарифм по основанию e
<u>log10()</u>	Принимает единственный параметр и возвращает его десятичный логарифм (<u>lg</u>)

Экспоненты и логарифмы

```
<?php
    $a=exp(1);
    echo 'a=' . $a, '<br>';
    $a=pow(10,3);
    echo 'a=' . $a, '<br>';
    $a=log10(100);
    echo 'a=' . $a, '<br>';
?>
```



Тригонометрические функции

Функция	Описание
<u><i>pi()</i></u>	Не принимает параметров и возвращает приближенное значение числа π (3.1415926535898). Может использоваться как взаимозаменяемая с константой M_PI
<u><i>sin()</i></u>	Принимает числовой параметр в радианах и возвращает синус параметра в виде числа с плавающей точкой двойной точности
<u><i>cos()</i></u>	Принимает числовой параметр в радианах и возвращает косинус параметра в виде числа с плавающей точкой двойной точности
<u><i>tan()</i></u>	Принимает числовой параметр в радианах и возвращает тангенс параметра в виде числа с плавающей точкой двойной точности
<u><i>asin()</i></u>	Принимает числовой параметр и возвращает арксинус параметра в радианах. Входные данные должны находиться в пределах от -1 до 1 (получение функцией входных данных, выходящих за пределы этого диапазона, приводит к получению результата NAN). Результаты находятся в диапазоне от $-\pi/2$ до $\pi/2$
<u><i>acos()</i></u>	Принимает числовой параметр и возвращает арккосинус параметра в радианах. Входные данные должны находиться в пределах от -1 до 1 (получение функцией входных данных, выходящих за пределы этого диапазона, приводит к получению результата NAN). Результаты находятся в диапазоне от 0 до π
<u><i>atan()</i></u>	Принимает числовой параметр и возвращает арктангенс параметра в радианах. Результаты находятся в диапазоне от $-\pi/2$ до $\pi/2$
<u><i>deg2rad()</i></u>	Преобразует значение из градусов в радианы
<u><i>rad2deg()</i></u>	Преобразует значение из радианов в градусы

Выработка случайных чисел

В PHP есть две группы функций по работе со случайными числами. Чисто внешне их можно отличить по префиксу `mt_` у всех функций одной из групп.

В языке PHP применяются два генератора случайных чисел - `rand()` и `mt_rand()`. С каждым из этих генераторов связаны по три функции одинакового назначения:
функция задания начального значения: `srand()` и `mt_srand()`,
сама функция получения случайного числа,
функция, осуществляющая выборку наибольшего целого числа, которое может быть возвращено генератором `getrandmax()` и `mt_getrandmax()`.

Функции `getrandmax()` и `mt_getrandmax()` возвращают значение наибольшего числа, которое может быть возвращено функцией `rand()` или `mt_rand()`.

Выработка случайных чисел

В PHP есть две группы функций по работе со случайными числами. Чисто внешне их можно отличить по префиксу `mt_` у всех функций одной из групп.

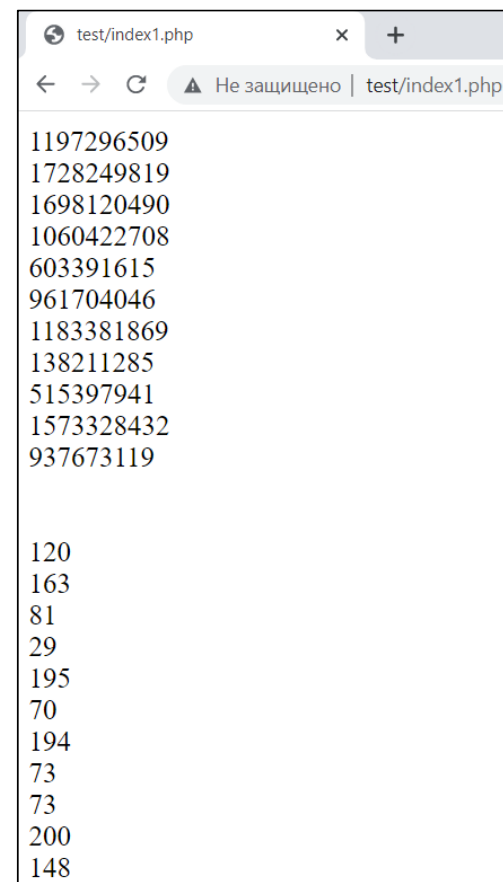
В языке PHP применяются два генератора случайных чисел - `rand()` и `mt_rand()`. С каждым из этих генераторов связаны по три функции одинакового назначения:
функция задания начального значения: `srand()` и `mt_srand()`,
сама функция получения случайного числа,
функция, осуществляющая выборку наибольшего целого числа, которое может быть возвращено генератором `getrandmax()` и `mt_getrandmax()`.

Функции `getrandmax()` и `mt_getrandmax()` возвращают значение наибольшего числа, которое может быть возвращено функцией `rand()` или `mt_rand()`.

Выработка случайных чисел

```
<?php
for ($i=0; $i<=10; $i++)
{
    echo rand();
    echo "<br>";
}
echo "<br><br>";

for ($i=0; $i<=10; $i++)
{
    echo rand(10, 200);
    echo "<br>";
}
?>
```



test/index1.php x +

← → ↻ ⚠ Не защищено | test/index1.php

1197296509
1728249819
1698120490
1060422708
603391615
961704046
1183381869
138211285
515397941
1573328432
937673119

120
163
81
29
195
70
194
73
73
200
148

Выработка случайных чисел

```
<?php
for ($i=0; $i<=10; $i++)
{
    echo mt_rand();
    echo "<br>";
}
echo "<br><br>";

for ($i=0; $i<=10; $i++)
{
    echo mt_rand(10, 200);
    echo "<br>";
}
?>
```

test/index1.php x +

← → ↻ ⚠ Не защищено | test/index1.php

1959869366
809262741
591248771
1953586553
1643179425
51253181
871297413
1727445634
1889481616
1121771518
187650290

186
72
134
10
117
55
38
164
40
15
53

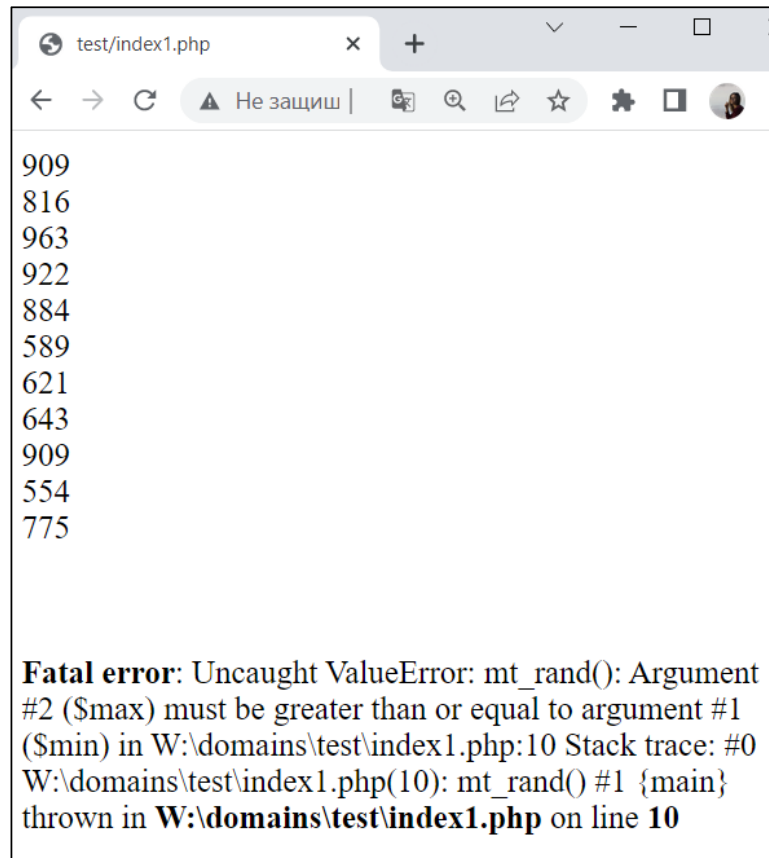
Выработка случайных чисел

Начиная с PHP 7.1.0, `rand()` использует тот же алгоритм получения случайных чисел, что и `mt_rand()`. Для сохранения обратной совместимости, функция `rand()` позволяет задавать параметр `max` меньше, чем параметр `min`. Функция `mt_rand()` в такой ситуации будет возвращать ошибку.

Выработка случайных чисел

```
<?php
for ($i=0; $i<=10; $i++)
{
    echo rand(1000, 500);
    echo "<br>";
}
echo "<br><br>";

for ($i=0; $i<=10; $i++)
{
    echo mt_rand(1000, 500);
    echo "<br>";
}
?>
```



The screenshot shows a web browser window with the address bar displaying "test/index1.php". The page content consists of a list of 11 numbers: 909, 816, 963, 922, 884, 589, 621, 643, 909, 554, and 775. Below the numbers, a red error message is displayed: "Fatal error: Uncaught ValueError: mt_rand(): Argument #2 (\$max) must be greater than or equal to argument #1 (\$min) in W:\domains\test\index1.php:10 Stack trace: #0 W:\domains\test\index1.php(10): mt_rand() #1 {main} thrown in W:\domains\test\index1.php on line 10".

```
test/index1.php x +
← → ↻ ⚠ Не защищ | 📄 🔍 📄 ☆ ⚙ □ 👤
909
816
963
922
884
589
621
643
909
554
775

Fatal error: Uncaught ValueError: mt_rand(): Argument
#2 ($max) must be greater than or equal to argument #1
($min) in W:\domains\test\index1.php:10 Stack trace: #0
W:\domains\test\index1.php(10): mt_rand() #1 {main}
thrown in W:\domains\test\index1.php on line 10
```

Выработка случайных чисел

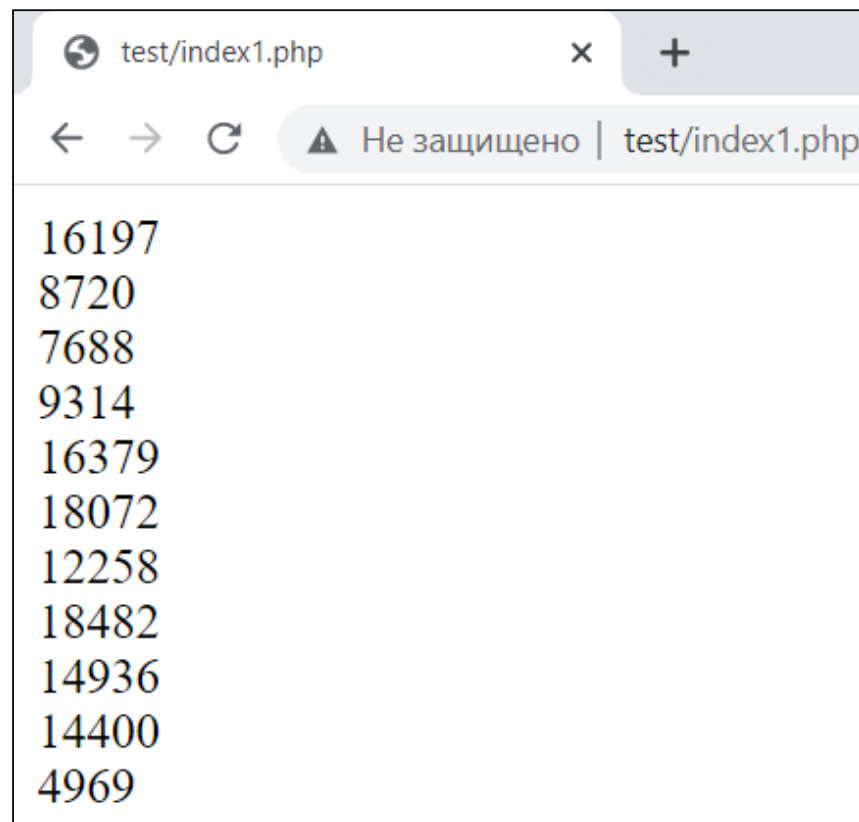
При вызове без параметров `min` и `max`, `rand()` и `mt_rand()` возвращают псевдослучайное целое в диапазоне от 0 до `getrandmax()` / `mt_getrandmax()`.

Начиная с PHP версии 7.0 можно генерировать криптографически безопасные псевдослучайные целые числа (т.е., числа, для которых случайность результата очень критична). Это делается с помощью функции `random_int()`, которая также принимает параметры `min` и `max`.

Выработка случайных чисел

```
<?php
for ($i=0; $i<=10; $i++)
{
    echo random_int(1000, 20000);
    echo "<br>";
}
echo "<br><br>";

for ($i=0; $i<=10; $i++)
{
    echo mt_rand(10, 200);
    echo "<br>";
}
?>
```



```
test/index1.php x +
← → ↻ ⚠ Не защищено | test/index1.php
16197
8720
7688
9314
16379
18072
12258
18482
14936
14400
4969
```

Выработка случайных чисел

Очевидно, что указанные функции выработки псевдослучайных чисел возвращают только целые числа, но случайное целое число из заданного диапазона можно легко преобразовать в соответствующее число с плавающей точкой (скажем, в число из диапазона от 0.0 до 1.0 включительно) с помощью выражения наподобие `rand() / getrandmax()`. После этого указанный диапазон можно масштабировать и сдвигать по мере необходимости.

Выработка случайных чисел

```
<?php
```

```
for ($i=0; $i<=10; $i++){  
    $random = 100.0 + 20.0 * mt_rand() / mt_getrandmax();  
    echo (int)$random;  
    echo "<br>";  
}  
?>
```

