Лекция 6
Управляющие конструкции языка PHP

Условные операторы

Группы управляющих конструкций PHP, которые будут рассмотрены:

Условные операторы:

`if`

`else`

`elseif`

Конструкции выбора:

switch-case

Циклы:

`while`

`do-while`

`for`

`foreach`

`break`

`continue`

Условные операторы. Конструкция if

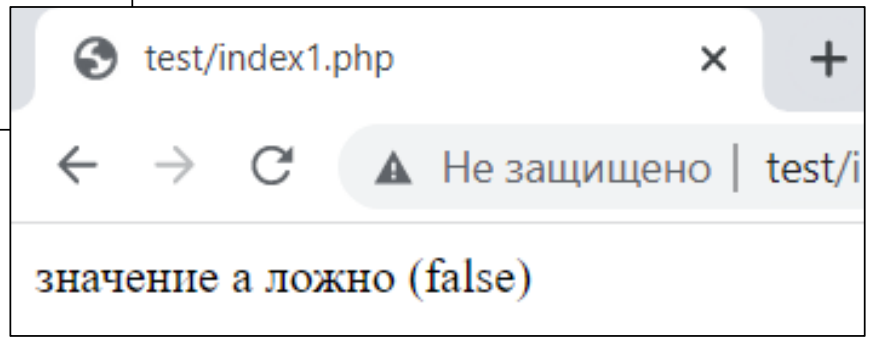
```
<?php
if (логическое выражение) оператор;
?>
```

Согласно выражениям PHP, конструкция if содержит логическое выражение. Если логическое выражение истинно (true), то оператор, следующий за конструкцией if будет исполнен, а если логическое выражение ложно (false), то следующий за if оператор исполнен не будет.

Условные операторы. Конструкция if

```
<?php
if ($a > $b) echo "значение a больше, чем b";
?>
```

```
<?php
$a = false;
if (!$a) echo "значение a ложно (false)";
?>
```

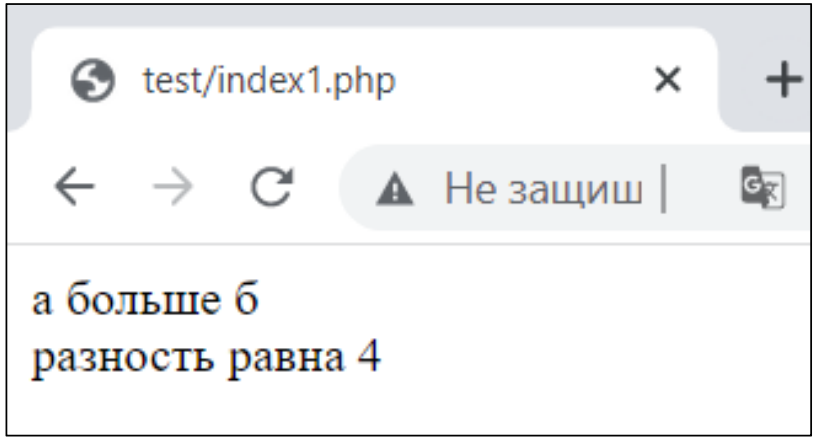


Условные операторы. Конструкция if

Часто вам будет необходим блок операторов, который будет выполняться при определенном условном критерии, тогда эти операторы необходимо поместить в фигурные скобки {...}

Пример:

```php
<?php
    $a = 5;
    $b = 1;
    if ($a > $b) {
        echo "a больше b";
        $c = $a - $b;
        echo "<br>", "разность равна $c";
    }
?>
```



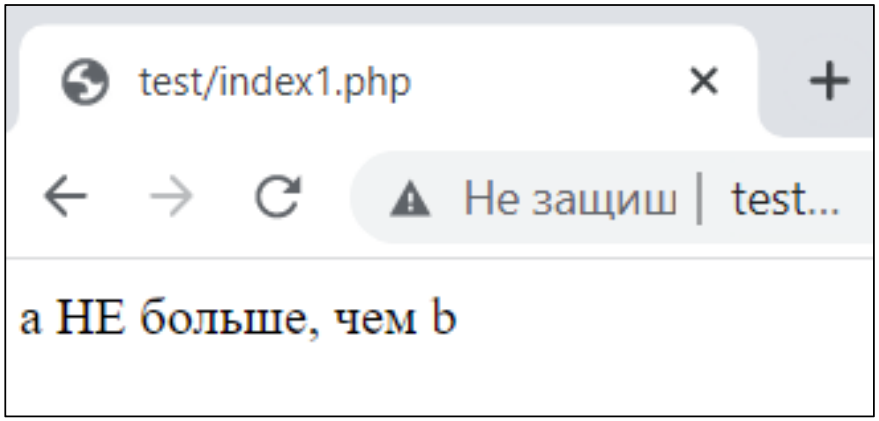
Условные операторы. Конструкция if-else

Часто возникает потребность исполнения операторов не только в теле конструкции **if**, если выполнено какое-либо условие конструкции **if**, но и в случае, если условие конструкции **if** не выполнено. В данной ситуации нельзя обойтись без конструкции **else**. В целом, такая конструкция будет называться конструкцией **if-else**.

```
if (логическое_выражение)
   инструкция_1;
else
   инструкция_2;
```

Условные операторы. Конструкция if-else

```php
<?php
    $a = 5;
    $b = 6;
    if ($a > $b) {
        echo "a больше, чем b";
    } else {
        echo "a НЕ больше, чем b";
    }
?>
```

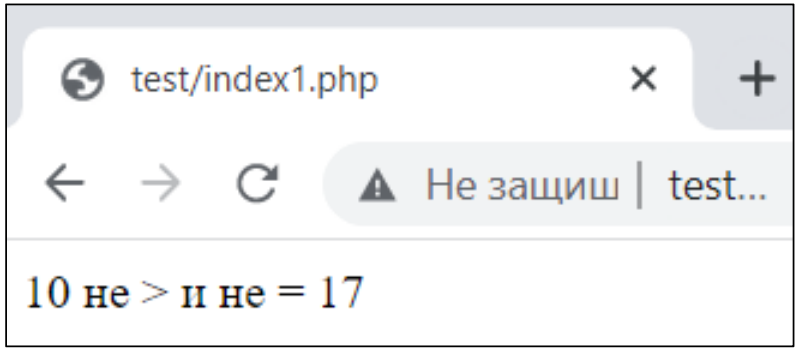


Условные операторы. Конструкция if-else (альтернативный синтаксис)

```
if (логическое_выражение):
   команды;
elseif(другое_логическое_выражение):
   другие_команды;
else:
   иначе_команды;
endif
```

Условные операторы. Конструкция if-else (альтернативный синтаксис)

```php
<?php
    $x = 10;
    $y = 17;
    if($x > $y):
        echo $x." больше, чем ".$y;
    elseif($x == $y):
        echo $x." равно ".$y;
    else:
        echo $x." не > и не = ".$y;
    endif;
?>
```

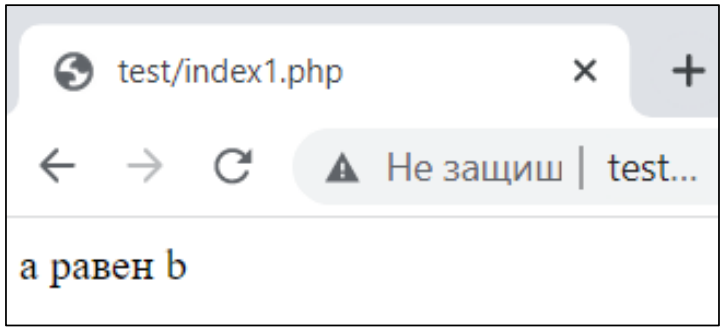


Условные операторы. Конструкция elseif

```
if (логическое_выражение_1)
   оператор_1;
elseif (логическое_выражение_2)
   оператор_2;
else
   оператор_3;
```

Условные операторы. Конструкция elseif

```php
<?php
    $a = 5;
    $b = 5;
    if ($a > $b) {
        echo "a больше, чем b";
    } elseif ($a == $b) {
        echo "a равен b";
    } else {
        echo "a меньше, чем b";
    }
?>
```



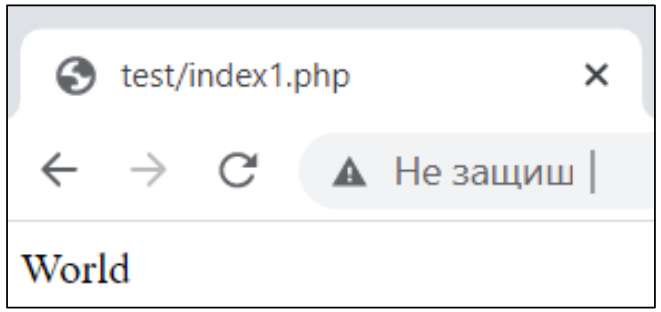
Условные операторы. Сокращенная запись if-else. Тернарный оператор

Тернарный оператор, т.е. с тремя операндами, имеет достаточно простой синтаксис, в котором слева от знака **?** записывается условие, а справа — два оператора разделенные знаком **:**, слева от знака выполняется оператор при истинности условия, а справа от знака **:** выполняется оператор при ложном условии.

```
условие ? оператор1 : оператор2;
```

Условные операторы. Сокращенная запись if-else. Тернарный оператор

```php
<?php
$condition = "0";
$result = $condition ? 'Hello' : 'World';
echo $result;
?>
```



Если переменная **$condition** возвращает **true**, тогда переменной **$result** будет присвоен левый операнд (по левую сторону от двоеточия, т.е., **Hello**). Если условие ложное, тогда будет использоваться правый операнд (по правую сторону от двоеточия, т.е., **World**).

Циклы в PHP

Циклы позволяют повторять определенное (и даже неопределенное - когда работа цикла зависит от условия) количество раз различные операторы. Данные операторы называются телом цикла. Проход цикла называется итерацией.

PHP поддерживает следующие виды циклов:

- Цикл с предусловием (while);
- Цикл с постусловием (do-while);
- Цикл со счетчиком (for);
- Специальный цикл перебора массивов (foreach).

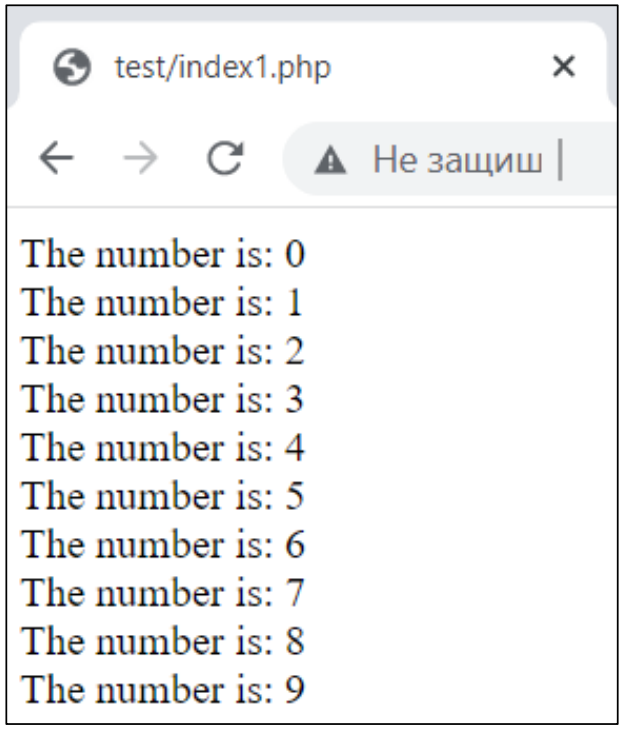
Циклы в PHP.
Цикл с предусловием while

Цикл с предусловием while работает по следующим принципам:
1. Вычисляется значение логического выражения.
2. Если значение истинно, выполняется тело цикла, в противном случае - переходим на следующий за циклом оператор.

```
while (логическое_выражение)
инструкция;
```

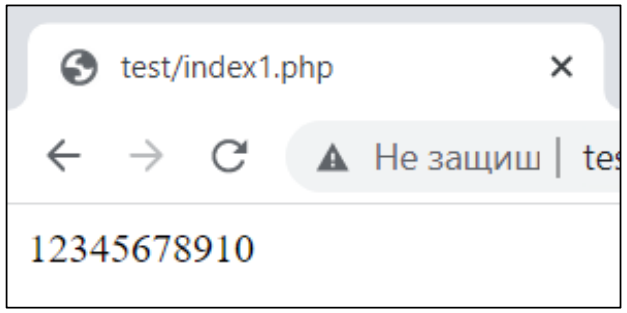
Циклы в PHP.
Цикл с предусловием while

```php
<?php
$a = 0;
while($a <= 9) {
  echo "The number is: $a <br>";
  $a++;
}
?>
```



Циклы в PHP.
Цикл с предусловием while

```php
<?php
$x=0;
while ($x++<10) echo $x;
// Выводит 12345678910
?>
```

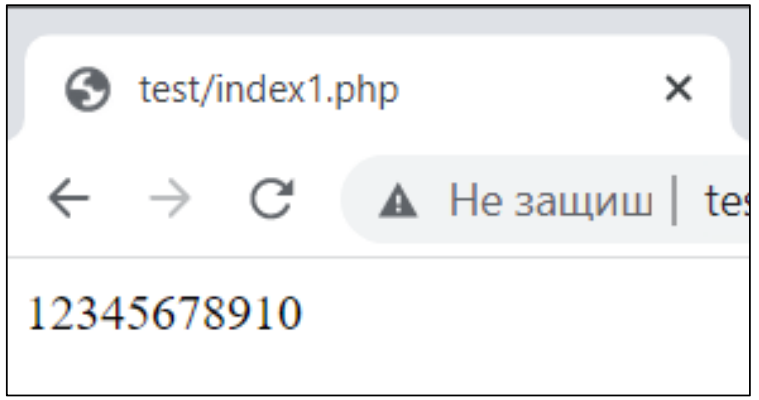


Обратите внимание на последовательность выполнения операций условия *$x++<10*. Сначала проверяется условие, а только потом увеличивается значение переменной. Если мы поставим операцию инкремента перед переменной (++$x<10), то сначала будет выполнено увеличение переменной, а только затем - сравнение. В результате мы получили бы строку 123456789.

Циклы в PHP.
Цикл с предусловием while

Этот же цикл можно было бы записать по-другому:

```php
<?php
    $x=0;
    while ($x<10)
    {
        $x++; // Увеличение счетчика
        echo $x;
    }
    // Выводит 12345678910
?>
```

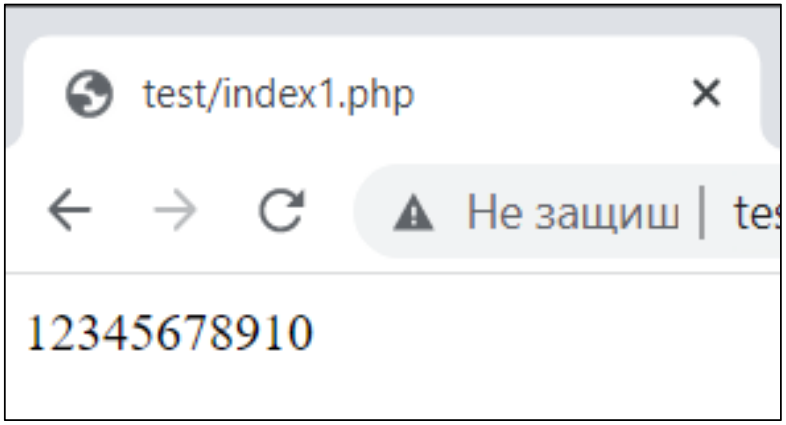


Если мы увеличим счетчик после выполнения оператора **echo**, мы получим строку *0123456789*.
В любом случае, мы имеем *10* итераций. Итерация - это выполнение операторов внутри тела цикла.

Циклы в PHP. Цикл с предусловием while (альтернативная форма записи)

```
while (логическое_выражение):
инструкция;
...
endwhile;
```

```php
<?php
    $x = 1;
    while ($x <= 10):
          echo $x;
          $x++;
endwhile;
?>
```



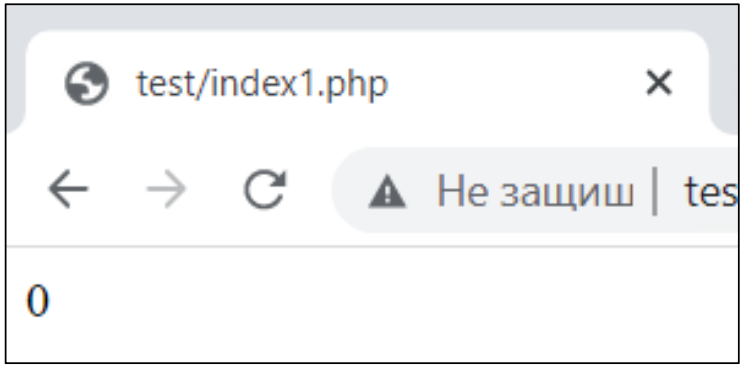
Циклы в PHP. Цикл с постусловием do while

В отличие от цикла while, этот цикл проверяет значение выражения не до, а после каждого прохода (итерации). Таким образом, тело цикла выполняется хотя бы один раз.

```
do
{
тело_цикла;
}
while (логическое_выражение);
```

Циклы в PHP. Цикл с постусловием do while

```php
<?php
$i = 0;
do {
    echo $i;
} while ($i > 0);
?>
```



В примере цикл будет выполнен ровно один раз, так как после первой итерации, когда проверяется истинность выражения, она будет вычислена как FALSE (*$i* не больше 0) и выполнение цикла прекратится.

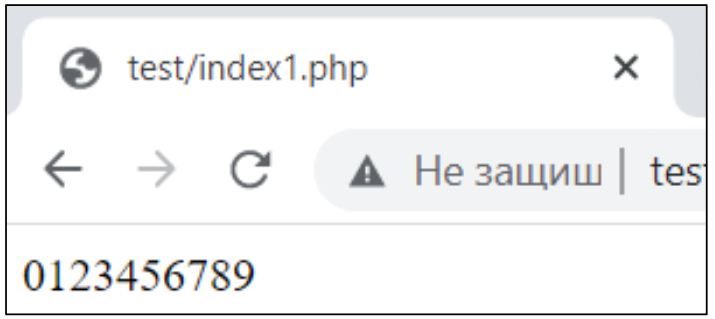
Циклы в PHP. Цикл со счетчиком for

Цикл со счетчиком используется для выполнения тела цикла определенное число раз. С помощью цикла **for** можно (и нужно) создавать конструкции, которые будут выполнять действия совсем не такие тривиальные, как простая переборка значения счетчика.

```
for (инициализирующие_команды; условие_цикла; команды_после_итерации)
{ тело_цикла; }
```

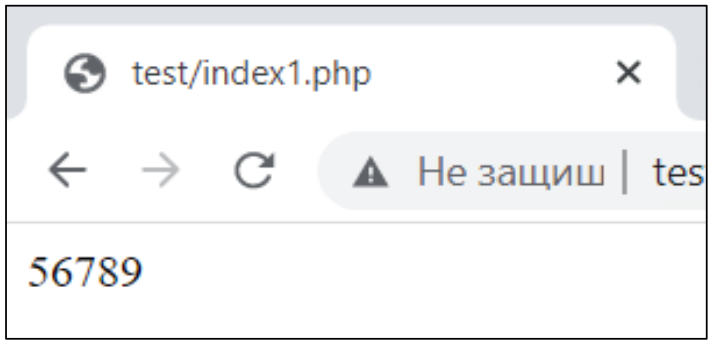
Циклы в PHP. Цикл со счетчиком for

```php
<?php
  for ($x=0; $x<10; $x++) echo $x;
?>
```



Объявление цикла for может опускать отдельный части. Например, опустить определение счетчика (он может быть определен вне цикла):

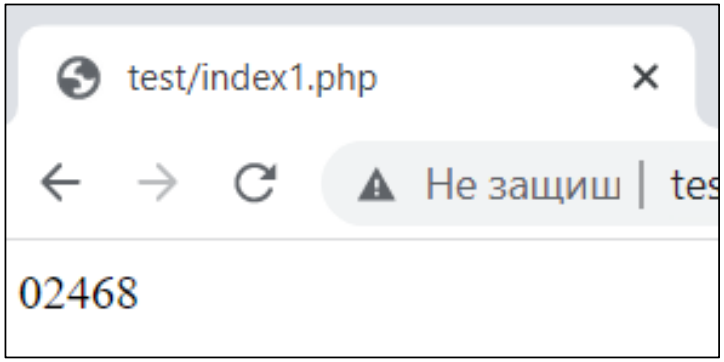
```php
<?php
$i = 5;
for (; $i < 10; $i++) echo $i;
?>
```



Циклы в PHP. Цикл со счетчиком for

Можно опустить изменение значения счетчика и изменять его внутри цикла:

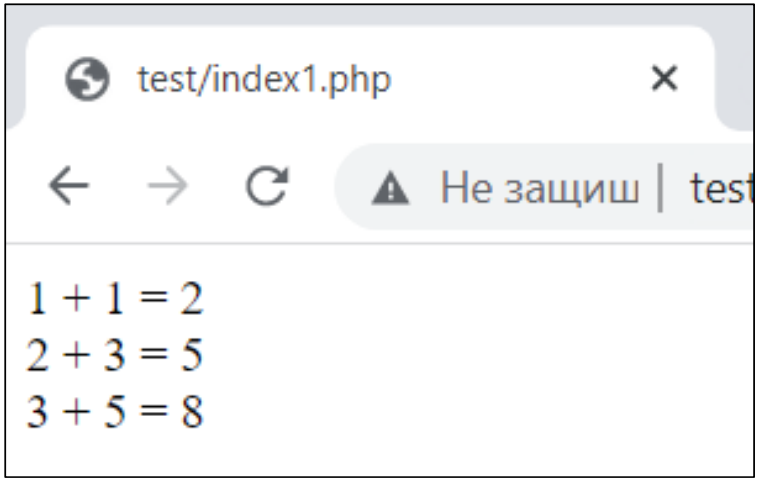
```php
<?php
$i = 0;
for (; $i < 10;)
{
    echo $i;
    $i += 2;
}
?>
```



Циклы в PHP. Цикл со счетчиком for

Можно в объявлении цикла определять и использовать сразу несколько переменных:

```php
<?php
for ($i=1, $j=1; $i + $j < 10; $i++, $j+=2)
{
    echo "$i + $j = " . $i + $j . "<br>";
?>
```

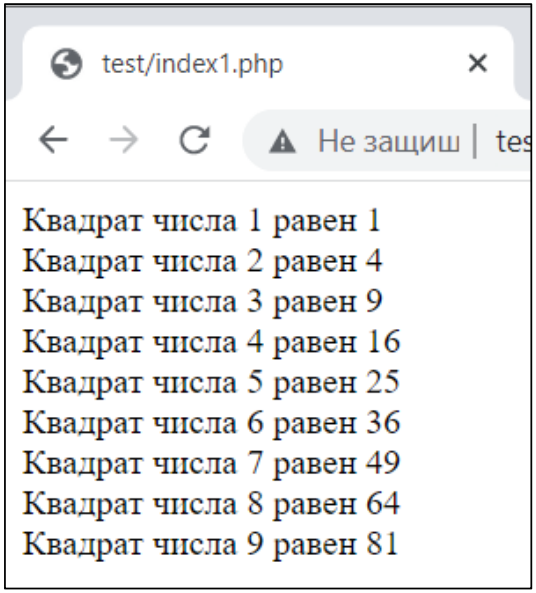


В данном случае в объявлении цикла определяются две переменных: $i и $j. При каждой итерации переменная $i увеличивается на 1, а $j - на 2. При этом цикл продолжается, пока сумма двух переменных не достигнет 10.

Циклы в PHP. Цикл со счетчиком for (альтернативный синтаксис)

```
for(инициализирующие_команды; условие_цикла; команды_после_итерации):
операторы;
endfor;
```

```
<?php
for ($i = 1; $i < 10; $i++):
    echo "Квадрат числа $i равен " .
$i * $i . "<br/>";
endfor;
?>
```

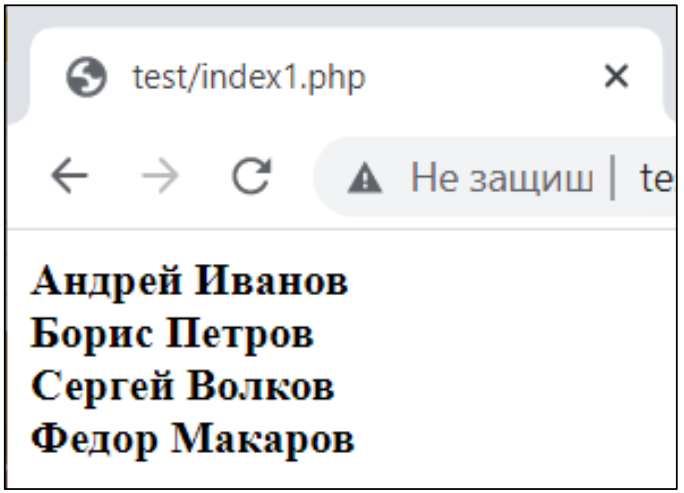


Циклы в PHP.
Цикл перебора массивов foreach

Данный цикл предназначен специально для перебора массивов. Синтаксис цикла foreach выглядит следующим образом.

```
foreach (массив as $ключ=>$значение)
команды;
```

```php
<?php
    $names["Иванов"] = "Андрей";
    $names["Петров"] = "Борис";
    $names["Волков"] = "Сергей";
    $names["Макаров"] = "Федор";
    foreach ($names as $key => $value) {
    echo "<b>$value $key</b><br>";
    }
?>
```



Циклы в PHP.
Цикл перебора массивов foreach

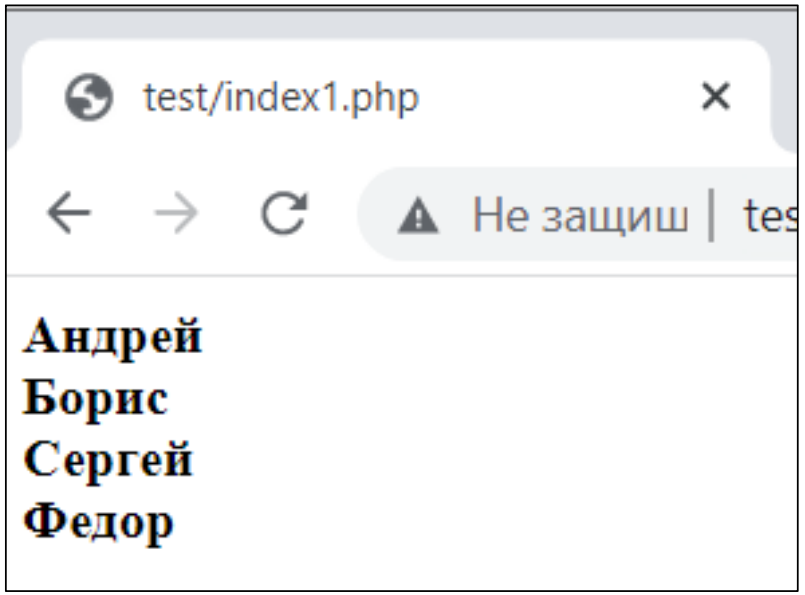
У цикла foreach имеется и другая форма записи, которую следует применять, когда нас не интересует значение ключа очередного элемента. Выглядит она так:

```
foreach (массив as $значение)
команды;
```

Важно: цикл foreach оперирует не исходным массивом, а его копией. Это означает, что любые изменения, которые вносятся в массив, не могут быть "видны" из тела цикла. Что позволяет, например, в качестве массива использовать не только переменную, но и результат работы какой-нибудь функции, возвращающей массив (в этом случае функция будет вызвана всего один раз - до начала цикла, а затем работа будет производиться с копией возвращенного значения).

Циклы в PHP.
Цикл перебора массивов foreach

```php
<?php
$names[] = "Андрей";
$names[] = "Борис";
$names[] = "Сергей";
$names[] = "Федор";
foreach ($names as $value)
{
echo "<b>$value</b><br>";
}
?>
```



Циклы в PHP.
Конструкция break

Очень часто для того, чтобы упростить логику какого-нибудь сложного цикла, удобно иметь возможность его прервать в ходе очередной итерации (к примеру, при выполнении какого-нибудь особенного условия). Для этого и существует конструкция **break**, которая осуществляет немедленный выход из цикла. Она может задаваться с одним необязательным параметром - числом, которое указывает, из какого вложенного цикла должен быть произведен выход. По умолчанию используется 1, т. е. выход из текущего цикла, но иногда применяются и другие значения. Синтаксис конструкции **break**:

```
break; // По умолчанию
break(номер_цикла);
// Для вложенных циклов (указывается номер прерываемого цикла
```

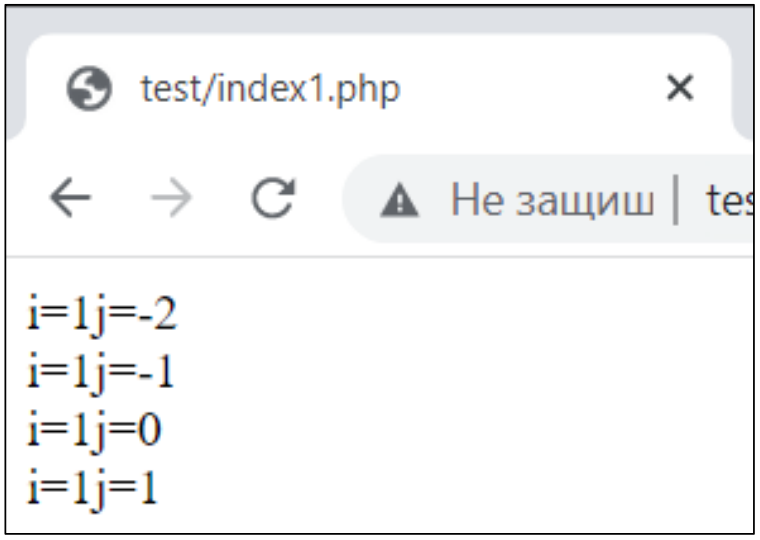
Циклы в PHP.
Конструкция break

Если нам нужно прервать работу определенного (вложенного) цикла, то нужно передать конструкции break параметр - номер_цикла, например, break(1). Нумерация циклов выглядит следующим образом:

```
for (...) // Третий цикл
{
  for (...) // Второй цикл
  {
     for (...) // Первый цикл
     {
     }
  }
}
```

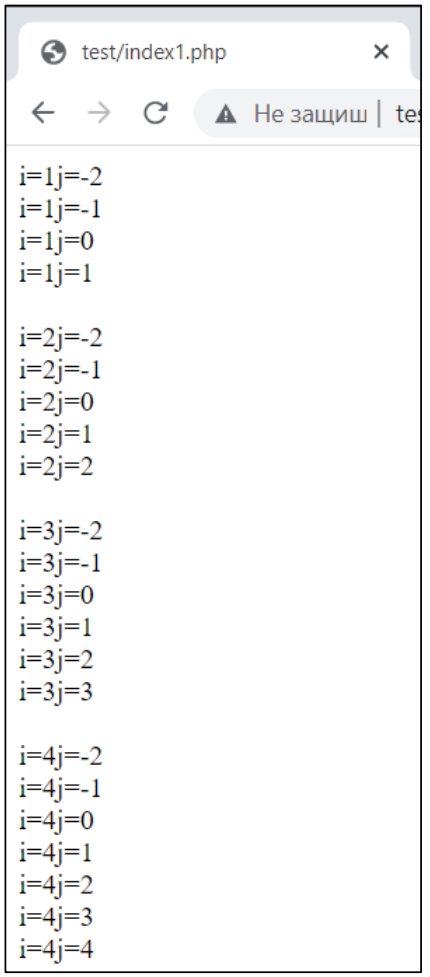
Циклы в PHP.
Конструкция break

```php
<?php
    for ($i=1; $i<=4; $i++){
        for ($j=-2; $j<=15; $j++){
            echo "i={$i}","j=$j","<br>";
            if ($j>=$i) break(2);
        }
    echo "<br>";
    }
?>
```



Циклы в PHP.
Конструкция break

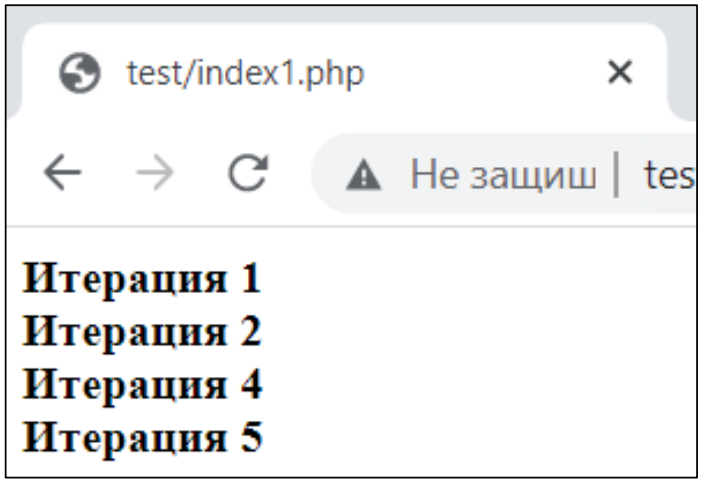
```php
<?php
    for ($i=1; $i<=4; $i++){
        for ($j=-2; $j<=15; $j++){
            echo "i={$i}","j=$j","<br>";
            if ($j>=$i) break(1);
        }
    echo "<br>";
    }
?>
```



Циклы в PHP.
Конструкция continue

Конструкция continue так же, как и break, работает как правило "в паре" с циклическими конструкциями. Она немедленно завершает текущую итерацию цикла и переходит к новой (конечно, если выполняется условие цикла для цикла с предусловием). Точно так же, как и для break, для continue можно указать уровень вложенности цикла, который будет продолжен по возврату управления.

```php
<?php
$x=0;
while ($x++<5) {
if ($x==3) continue;
echo "<b>Итерация $x</b><br>";
      }
// Цикл прервется только на третьей итерации
?>
```



Циклы в PHP.
Конструкции выбора

Часто вместо нескольких расположенных подряд инструкций if-else целесообразно воспользоваться специальной конструкцией выбора switch-case. Данная конструкция предназначена для выбора действий в зависимости от значения указанного выражения. Конструкция switch-case чем-то напоминает конструкцию if-else, который, по сути, является ее аналогом. Конструкцию выбора можно использовать, если предполагаемых вариантов много, например, более 5, и для каждого варианта нужно выполнить специфические действия. В таком случае, использование конструкции if-else становится действительно неудобным.

```
switch(выражение) {
   case значение1: команды1; [break;]
   case значение2: команды2; [break;]
   . . .
   case значениеN: командыN; [break;]
   [default: команды_по_умолчанию;
   [break]]
}
```

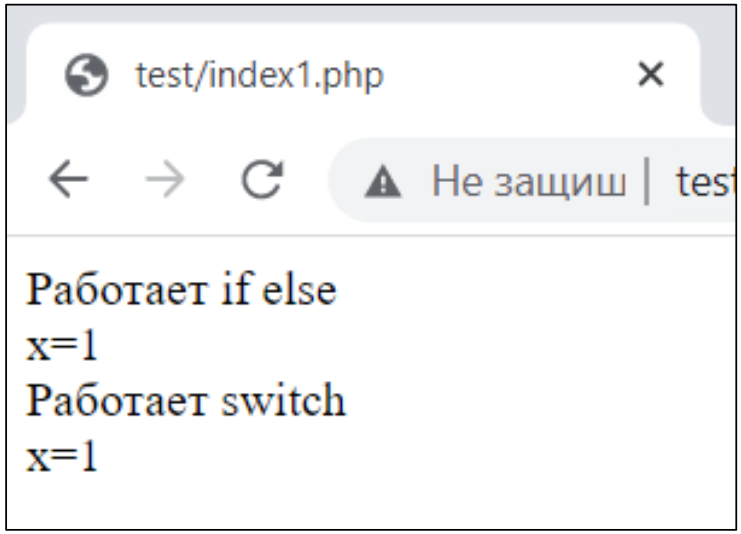
Циклы в PHP.
Конструкции выбора

Принцип работы конструкции switch-case такой:
1. Вычисляется значение выражения;

2. Просматривается набор значений. Пусть значение1 равно значению выражения, вычисленного на первом шаге. Если не указана конструкция (оператор) break, то будут выполнены команды i, i+1, i+2, ... , N. В противном случае (есть break) будет выполнена только команда с номером i.

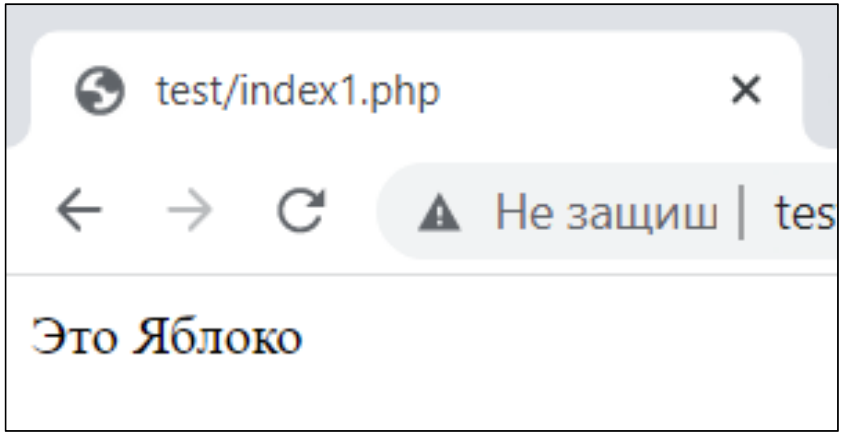
3. Если ни одно значение из набора не совпало со значением выражения, тогда выполняется блок default, если он указан.

```php
<?php
    $x=1;
    echo "Работает if else", "<br>";
    if ($x == 0) {
        echo "x=0<br>";
    } elseif ($x == 1) {
        echo "x=1<br>";
    } elseif ($x == 2) {
        echo "x=2<br>";}

    echo "Работает switch", "<br>";
    switch ($x) {
        case 0:
            echo "x=0<br>";
            break;
        case 1:
            echo "x=1<br>";
            break;
        case 2:
            echo "x=2<br>";
            break;}
?>
```



Циклы в PHP.
Конструкции выбора

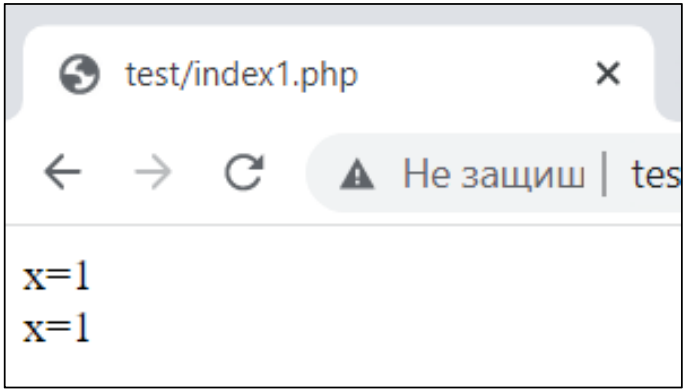
```php
<?php
    $x="Яблоко";
    switch ($x) {
    case "Яблоко":
        echo "Это Яблоко";
        break;
    case "Груша":
        echo "Это Груша";
        break;
    case "Арбуз":
        echo "Это Арбуз";
        break;
    }
?>
```



Циклы в PHP.
Конструкции выбора

Конструкция switch выполняется поэтапно. Сперва никакой код не исполнен. Только, когда конструкция case найдена со значением, которое соответствует значению выражения switch, PHP начинает исполнять конструкции. PHP продолжает исполнять конструкции до конца блока switch, пока не встречается оператор break. Если не использовать конструкции (операторы) break, скрипт будет выглядеть так:

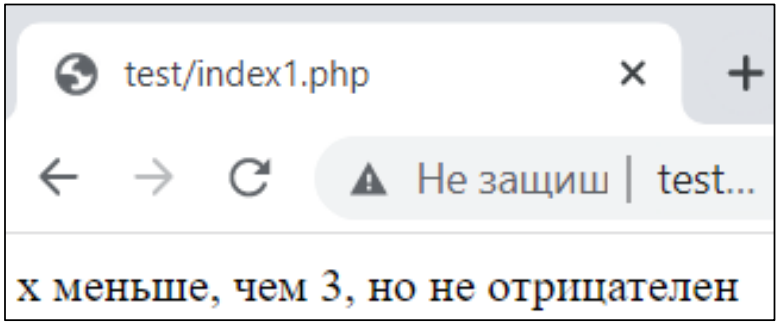
```php
<?php
    $x=1;
    switch ($x) {
        case 0:
            echo "x=$x<br>";
        case 1:
            echo " x=$x<br>";
        case 2:
            echo " x=$x<br>";
    }
?>
```



Циклы в PHP.
Конструкции выбора

Операторный список для **case** может быть также пуст, он просто передает управление в операторный список до следующей конструкции **case**:

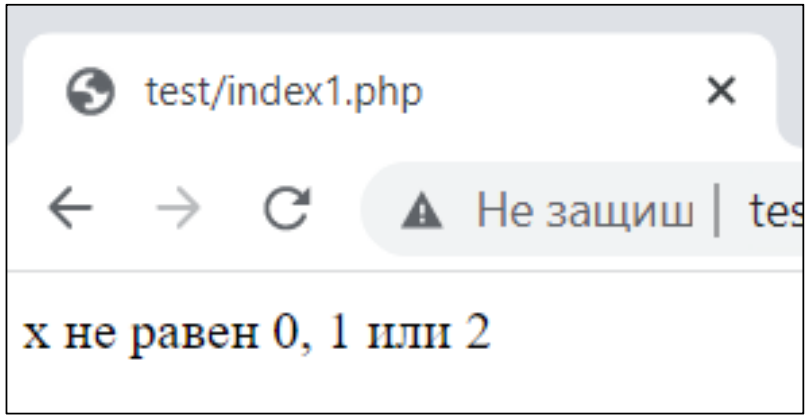
```
<?php
$x = 2;
switch ($x) {
case 0:
case 1:
case 2:
     echo "x меньше, чем 3, но не отрицателен";
     break;
case 3:
     echo "x=3";
}
?>
```



Циклы в PHP. Конструкции выбора

Когда ни одно значение из набора не совпало со значением выражения, тогда выполняется блок default, если он указан, например:

```php
<?php
$x = 3;
switch ($x) {
case 0:
     echo "x=0";
     break;
case 1:
     echo "x=1";
     break;
case 2:
     echo "x=2";
     break;
default:
     echo "x не равен 0, 1 или 2";
}
?>
```

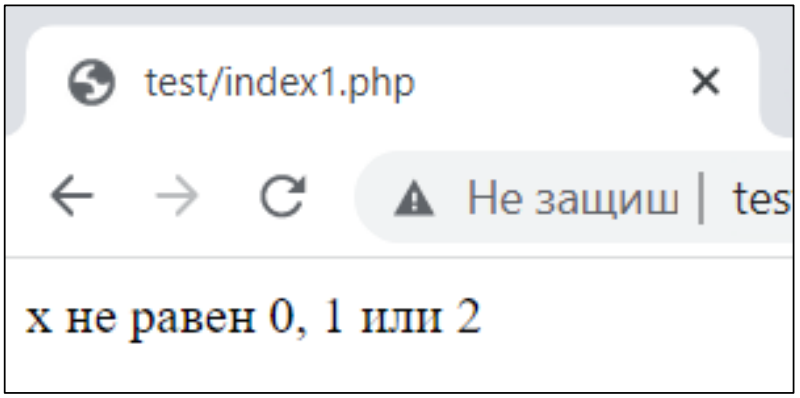


Циклы в PHP.
Конструкции выбора (альтернативный вариант)

```
switch(выражение):
   case значение1: команды1; [break;]
   . . .
   case значениеN: командыN; [break;]

   [default: команды_по_умолчанию; [break];]

endswitch;
```

Циклы в PHP.
Конструкции выбора (альтернативный вариант)

```php
<?php
    $x=3;
    switch ($x):
    case 0:
        echo "x=0";
        break;
    case 1:
        echo "x=1";
        break;
    case 2:
        echo "x=2";
        break;
    default:
        echo "x не равен 0, 1 или 2";
    endswitch;
?>
```



Конструкции возврата значений

Конструкция return

1. Конструкция rerurn возвращает значения, преимущественно из пользовательских функций, как параметры функционального запроса. При вызове return исполнение пользовательской функции прерывается, а конструкция return возвращает определенные значения.

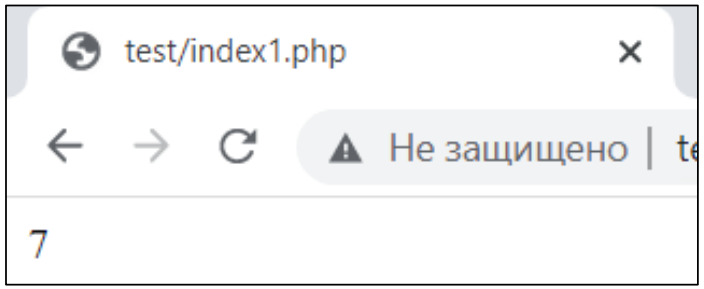
2. Если конструкция return будет вызвана из глобальной области определения (вне пользовательских функций), то скрипт также завершит свою работу, а return также возвратит определенные значения.

3. Возвращаемые значения могут быть любого типа, в том числе это могут быть списки и объекты. Возврат приводит к завершению выполнения функции и передаче управления обратно к той строке кода, в которой данная функция была вызвана.

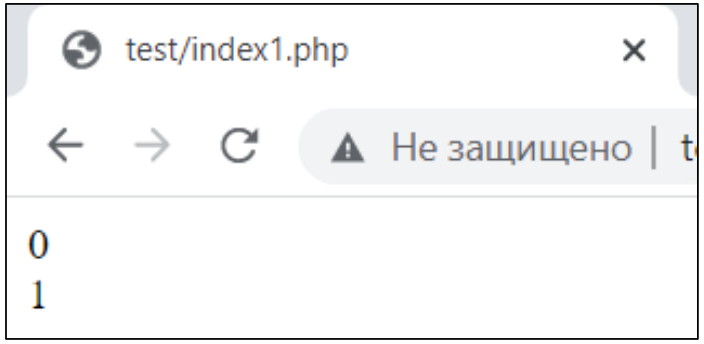
Конструкции возврата значений

Конструкция return

```php
<?php
function retfunct()
{
     return 7;
}
echo retfunct(); // выводит '7'
?>
```




```php
<?php
function numbers()
{
  return array (0, 1, 2);
}
list ($zero, $one, $two) = numbers();
echo $zero, "<br>";
echo $one, "<br>";
?>
```



Конструкции включений в PHP

Конструкция включений require

Конструкция require позволяет включать файлы в сценарий PHP до исполнения сценария PHP. Общий синтаксис require такой:

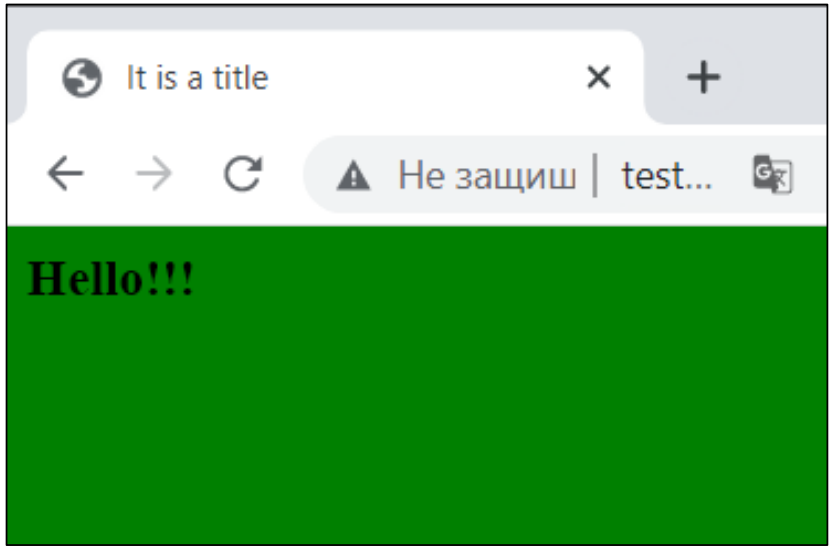
```
require имя_файла;
```

При запуске (именно при запуске, а не при исполнении!) программы интерпретатор просто заменит инструкцию на содержимое файла имя_файла (этот файл может также содержать сценарий на PHP, обрамленный, как обычно, тэгами <? и?>).

Конструкции включений в PHP

Конструкция включений require

```
//Файл header.html:
<html>
<head><title>It is a title</title></head>
<body bgcolor=green>

//Файл footer.html:
<h3>Hello!!!</h3>
</body></html>

//Файл script.php
<?php
require "header.html";
// Сценарий выводит само тело документа
require "footer.html";
?>
```



Конструкции включений в PHP

Конструкция включений include

Конструкция include также предназначена для включения файлов в код сценария PHP.
В отличие от конструкции require конструкция include позволяет включать файлы в код PHP скрипта во время выполнения сценария. Синтаксис конструкции include выглядит следующим образом:

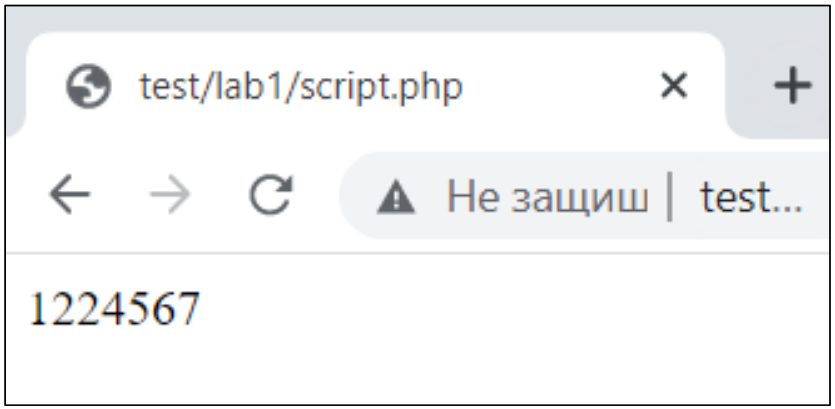
```
include имя_файла;
```

Конструкции включений в PHP

Конструкция включений include

Создадим 7 файлов с именами 1.txt, 2.txt и так далее до 7.txt, содержимое этих файлов - просто десятичные цифры 1, 2 ..... 7 (по одной цифре в каждом файле). Создадим такой сценарий PHP:

```php
<?php
    // Создаем цикл, в теле которого
    конструкция include
    for($i=1; $i<=7; $i++) {
    include "$i.txt";
    }
    // Включили десять файлов: 1.txt,
    2.txt, 3.txt ... 7.txt
    // Результат - вывод 1234567
?>
```



Конструкции включений в PHP

Отличия require от include

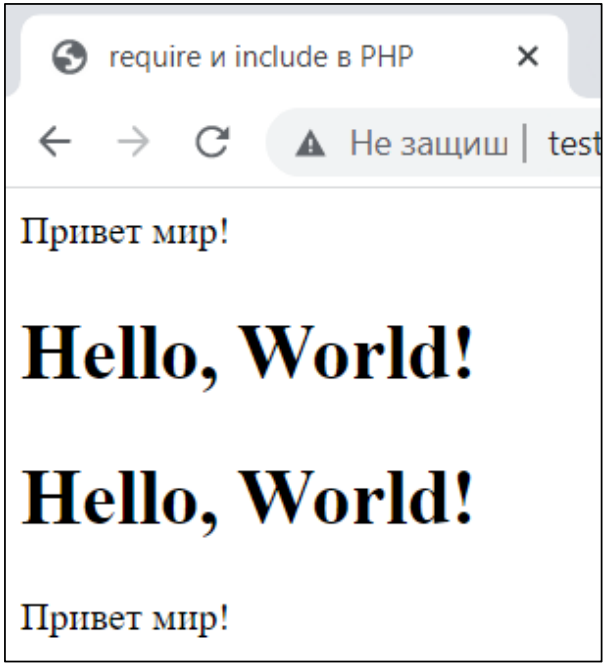
Отличие require от include, заключается в том, что при возникновении ошибки, например, файл находится в другой директории и не доступен:
--- при подключении с помощью include - выполнится весь наш код в данном файле, а файл, который не найден, не подключится, и будет при этом выведена ошибка подключения файла. Но вместе с тем, код нашей страницы отработал. И полученная ошибка не фатальная, не критичная - это просто предупреждение. Это значит, что код продолжит работать, но есть какая-то незначительная ошибка, которая не влияет на его работоспособность.

--- при подключении с помощью require мы получим уже фатальную ошибку, после которой php прерывает выполнение скрипта.
include - вызывает ошибку уровня Warning,
require - вызывает ошибку уровня Fatal error.

Конструкции включений в PHP

Отличия require от include

```
//Файл script.php:
<h1>Hello, World!</h1>
```

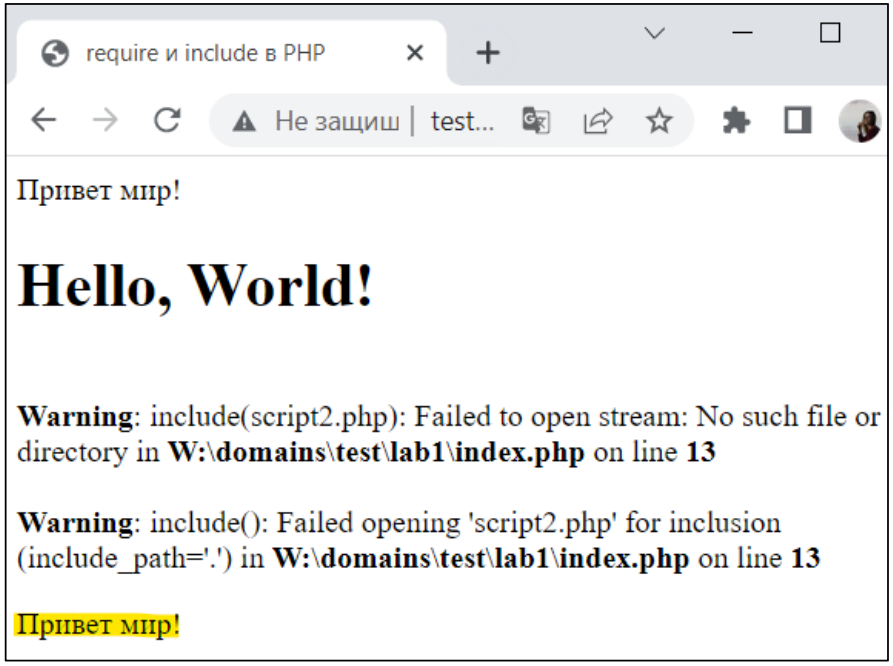



```
//Файл index.php:
<html>
<head>
<meta charset="UTF-8">
<title>require и include в PHP</title>
</head>
<body>
<p>Привет мир!</p>
<!-- подключаем файл script.php с помощью include -->
<?php include 'script.php' ?>
<!-- подключаем файл script.php второй раз с помощью include -->
<?php include 'script.php' ?> <!-- это сработает -->
<p>Привет мир!</p>
</body>
</html>
```

Конструкции включений в PHP

Отличия require от include

```
//Файл script.php:
<h1>Hello, World!</h1>
```

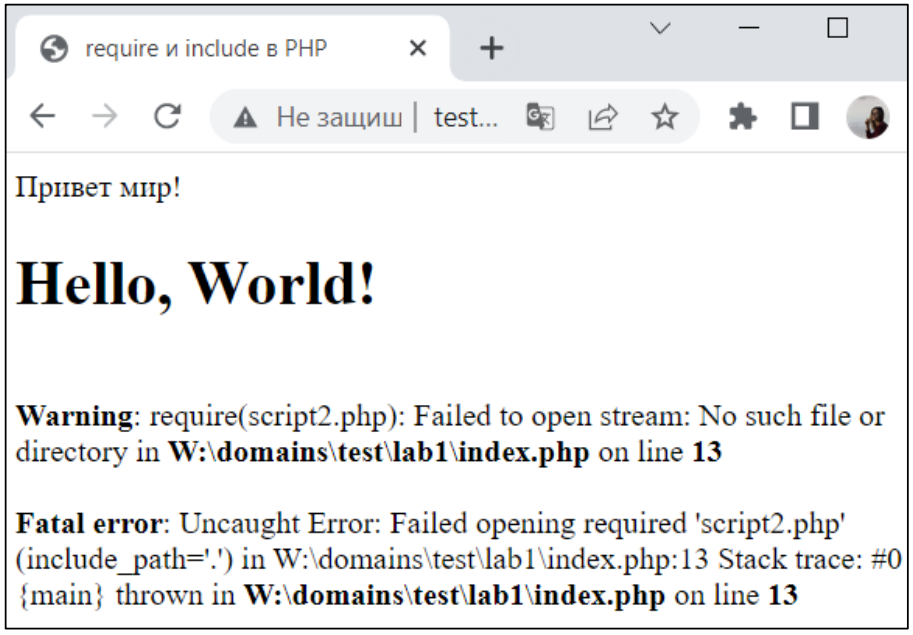



```
//Файл index.php:
<html>
<head>
<meta charset="UTF-8">
<title>require и include в PHP</title>
</head>
<body>
<p>Привет мир!</p>
<!-- подключаем файл script.php с помощью include -->
<?php include 'script.php' ?>
<!-- подключаем файл script2.php, которого нет-->
<?php include 'script2.php' ?>
<p>Привет мир!</p>
</body>
</html>
```

Конструкции включений в PHP

Отличия require от include

```
//Файл script.php:
<h1>Hello, World!</h1>
```




```
//Файл index.php:
<html>
<head>
<meta charset="UTF-8">
<title>require и include в PHP</title>
</head>
<body>
<p>Привет мир!</p>
<!-- подключаем файл script.php с помощью include -->
<?php include 'script.php' ?>
<!-- подключаем файл script2.php, которого нет-->
<?php require 'script2.php' ?>
<p>Привет мир!</p>
</body>
</html>
```

Конструкции включений в PHP

Отличия require от include

Если у нас должен подключаться файл, от которого зависит дальнейшая работа скрипта, например, подключение к базе данных, то желательно использовать - require.

Если же файл, который подключается, не ключевой, и если он не подключится - ничего страшного не будет, тогда можно использовать - include.

Конструкции включений в PHP

Конструкции однократного включения require_once и include_once

Работают конструкции однократного включения **require_once** и **include_once** так же, как и require и include соответственно. Разница в их работе лишь в том, что перед включением файла интерпретатор проверяет, включен ли указанный файл ранее или нет. Если да, то файл не будет включен вновь.

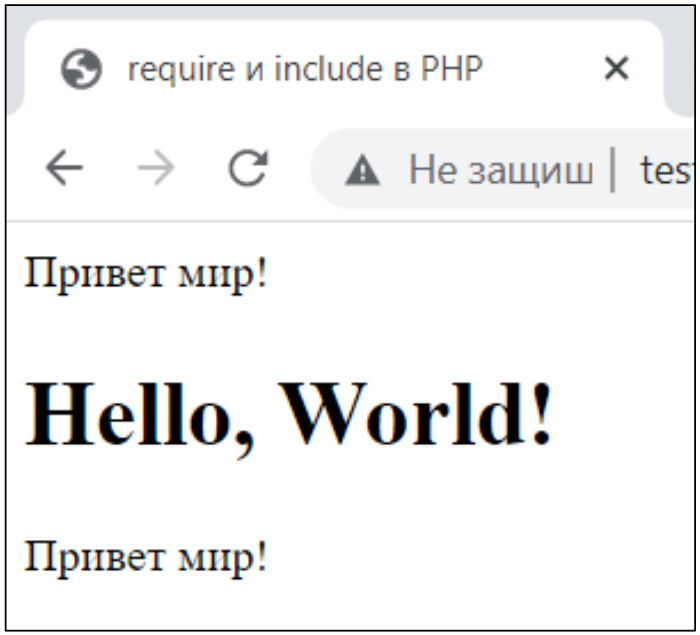
Конструкции однократных включений require_once и include_once также позволяют включать удаленные файлы, если такая возможность включена в конфигурационном файле PHP.

Разница между require_once и include_once та же, что и между require и include - в уровне ошибки.

Конструкции включений в PHP

Отличия require от include

```
//Файл script.php:
<h1>Hello, World!</h1>
```




```
//Файл index.php:
<html>
<head>
<meta charset="UTF-8">
<title>require и include в PHP</title>
</head>
<body>
<p>Привет мир!</p>
<!-- подключаем файл script.php с помощью
include_once -->
<?php include_once 'script.php' ?>
<?php include_once 'script.php' ?>
<p>Привет мир!</p>
</body>
</html>
```

Ссылки в PHP

1. Хотя в PHP нет такого понятия, как указатель, все же существует возможность создавать ссылки на другие переменные. Существует две разновидности ссылок: **жесткие** и **символические**.

2. Ссылки в PHP – это средство доступа к содержимому одной переменной под разными именами. В PHP имя переменной и её содержимое – это разные вещи, поэтому одно содержимое может иметь разные имена.

Ссылки в PHP

Жесткие ссылки в PHP

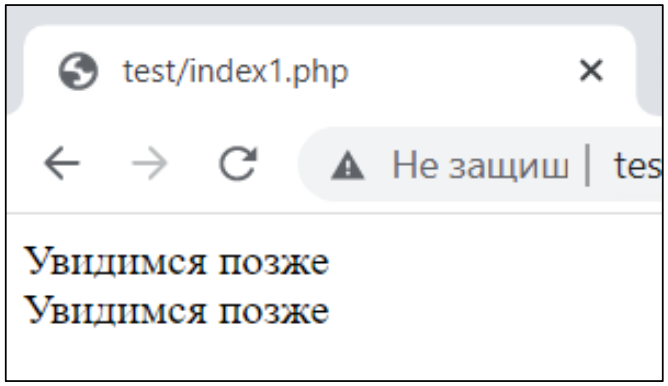
Жесткая ссылка представляет собой просто переменную, которая является синонимом другой переменной. Многоуровневые ссылки (то есть, ссылка на ссылку на переменную, как это можно делать, например, в Perl) не поддерживаются.

Чтобы создать жесткую ссылку, нужно использовать оператор & (амперсанд). Например:

Ссылки в PHP

Жесткие ссылки в PHP

```php
<?php
$myVar = "Привет!";
$Var = & $myVar;
$Var = "Увидимся позже";
echo $myVar;       // Выведет "Увидимся позже"
echo $Var;         // Выведет "Увидимся позже"
?>
```



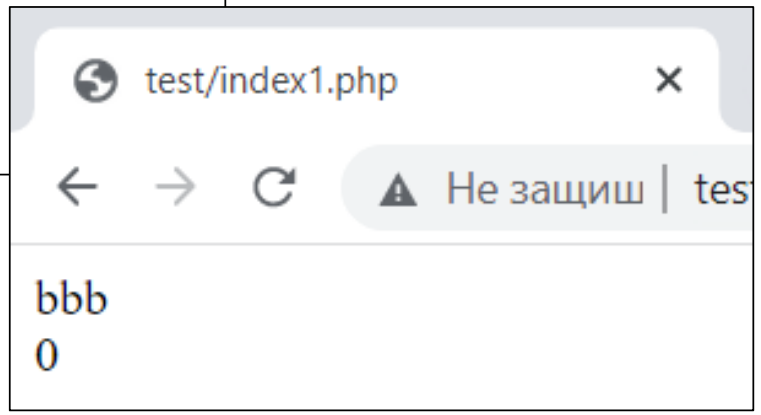
Можно видеть, что *$myVar* также изменен на «Увидимся позже»! Почему это произошло? Вместо того, чтобы присвоить значение переменной *$myVar* переменной *$Var* — которые просто создают две независимых копии одного и того же значения — мы сделали переменную *$Var ссылкой* на значение *$myVar*. Другими словами, *$myVar* и *$Var* оба указывают на одно и то же значение. Таким образом, когда мы присвоили новое значение переменной *$Var ,* значение переменной *$myVar* также изменилось.

Ссылки в PHP

Жесткие ссылки в PHP

Ссылаться можно не только на переменные, но и на элементы массива (этим жесткие ссылки выгодно отличаются от символических). Например:

```php
<?php
$A = array('a' => 'aaa', 'b' => 'bbb');
$b = &$A['b'];
// теперь $b — то же, что и элемент с индексом 'b' массива
echo $A['b'],"<br>"; // Выводит bbb
$b = 0;              // на самом деле $A['b']=0;
echo $A['b'];        // Выводит 0
?>
```

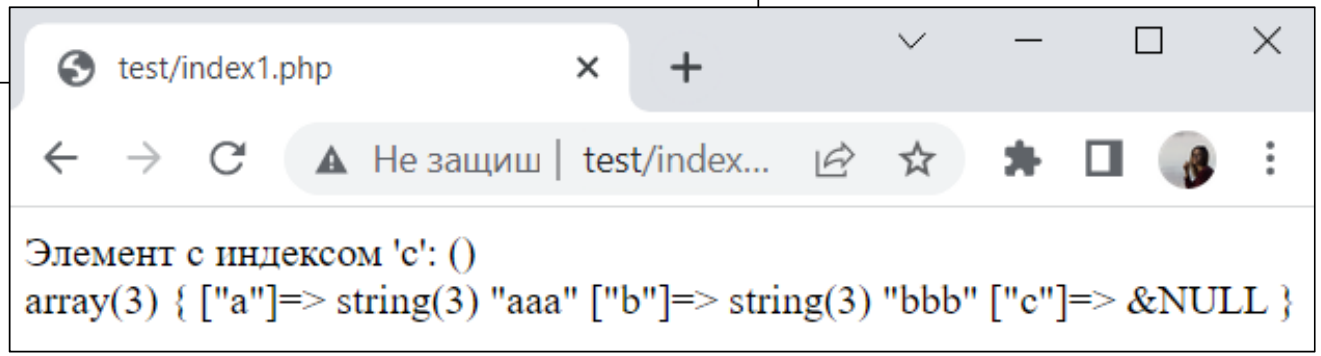


Ссылки в PHP

Жесткие ссылки в PHP

Ссылаться можно не только на переменные, но и на элементы массива (этим жесткие ссылки выгодно отличаются от символических). Например:

```php
<?php
$A = array('a' => 'aaa', 'b' => 'bbb');
$b = &$A['c'];
// теперь $b — то же, что и элемент с индексом 'c' массива
echo "Элемент с индексом 'c': (".$A['c'].")";
var_dump($A);
?>
```

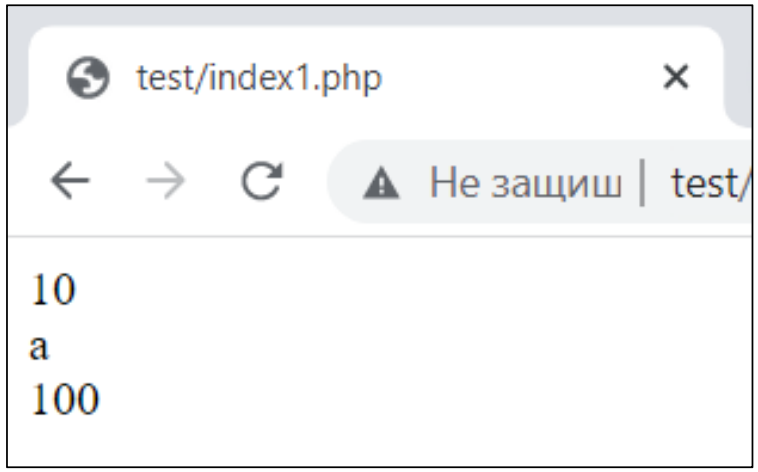


Ссылки в PHP

Символические ссылки (переменные переменные)

Символическая ссылка — это всего лишь строковая переменная, хранящая имя другой переменной. Чтобы добраться до значения переменной, на которую ссылается символическая ссылка, необходимо применить дополнительный знак $ перед именем ссылки. Рассмотрим пример:

```php
<?php
$a = 10;
$b = 20;
$c = 30;
$p = "a"; // или $p="b" или $p="c"
(присваиваем $p имя другой переменной)
echo $$p,"<br>"; // выводит переменную, на
которую ссылается $p, т. е. $a
echo $p,"<br>";
$$p = 100; // присваивает $a значение 100
echo $a;
?>
```

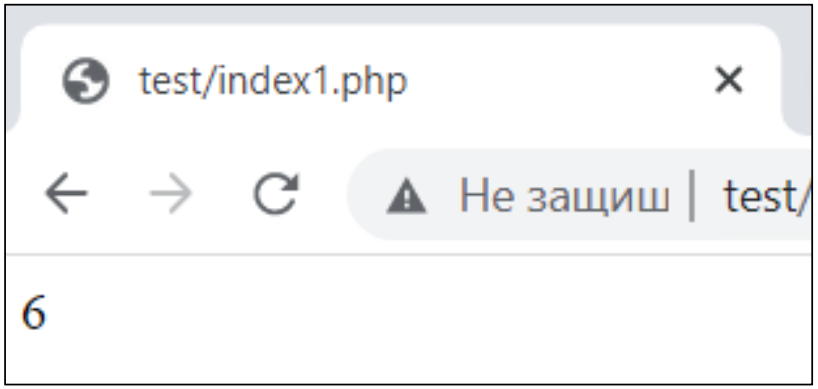


Ссылки в PHP

Передача значений по ссылке с использованием функций

Можно передавать переменные в пользовательскую функцию по ссылке, если необходимо разрешить функции модифицировать свои аргументы. В таком случае, пользовательская функция сможет изменять аргументы.

```php
<?php
function foo(&$var)
{
      $var++;
}
$a = 5;
foo($a);
echo $a;
// $a здесь равно 6
?>
```

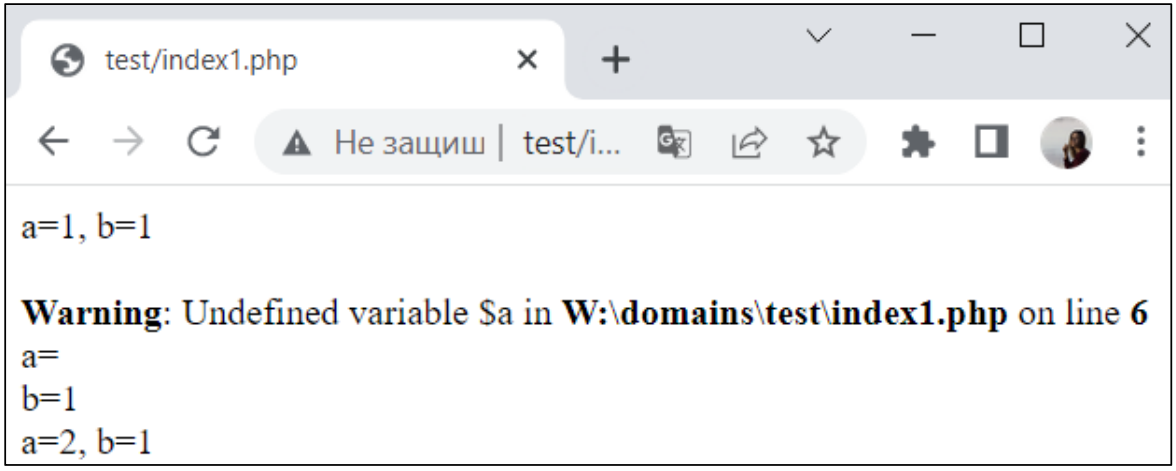


Ссылки в PHP

Удаление ссылок (сброс ссылок)

При удалении ссылки, просто разрывается связь имени и содержимого переменной. Это не означает, что содержимое переменной будет разрушено. Например:

```php
<?php
$a = 1;
$b = &$a;
echo "a=$a",", b=$b";
unset($a);
echo "<br>","a=$a";
echo "<br>","b=$b";
$a = 2;
echo "<br>","a=$a",", b=$b";
?>
```



Пользовательские функции в PHP

Подпрограмма – это специальным образом оформленный фрагмент программы, к которому можно обратиться из любого места внутри программы. Подпрограммы существенно упрощают жизнь программистам, улучшая читабельность исходного кода, а также сокращая его, поскольку отдельные фрагменты кода не нужно писать несколько раз.

В PHP такими подпрограммами являются **пользовательские функции**.

Пользовательские функции в PHP

Создание пользовательских функций

Пользовательская функция может быть объявлена в любой части программы (скрипта), до места ее первого использования. И не нужно никакого предварительного объявления.

Дойдя до определения пользовательской функции, транслятор проверит корректность определения и выполнит трансляцию определения функции во внутреннее представление, но транслировать сам код он не будет. И это правильно - зачем транслировать код, который, возможно, вообще не будет использован. Синтаксис объявления функций следующий:

```
function Имя аргумент1[=значение1],...,аргумент1[=значение1])
{
тело_функции
}
```

Пользовательские функции в PHP

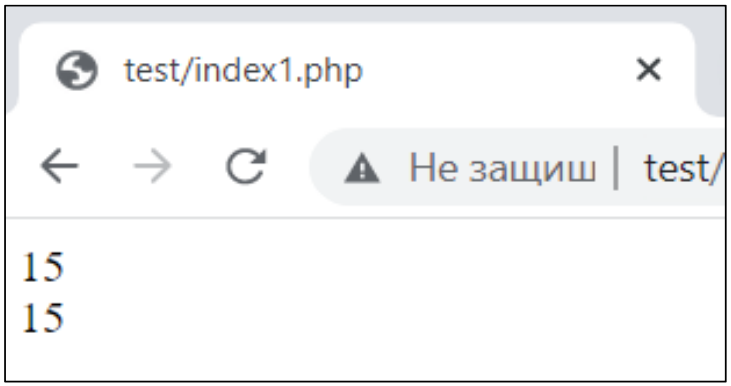
Требования, предъявляемые к именам функций:

1. Имена функций могут содержать русские буквы, но давать функциям имена, состоящие из русских букв не рекомендуется;
2. Имена функций не должны содержать пробелов;
3. Имя каждой пользовательской функции должно быть уникальным. При этом, необходимо помнить, что регистр при объявлении функций и обращении к ним не учитывается. То есть, например, функции funct() и FUNCT() имеют одинаковые имена;
4. Функциям можно давать такие же имена, как и переменным, только без знака $ в начале имен.

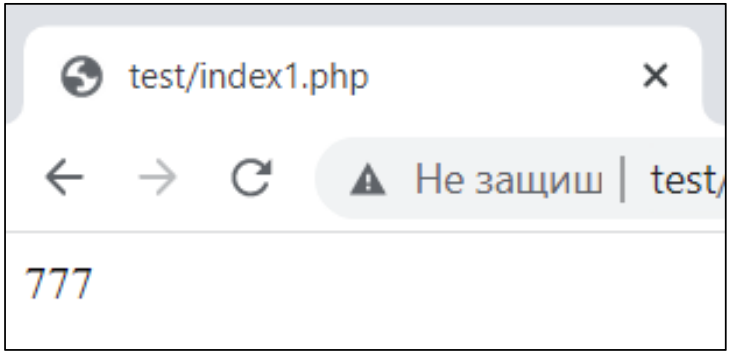
Типы значений, возвращаемые пользовательскими функциями, могут быть любыми. Для передачи результата работы пользовательских функций в основную программу (скрипт) используется конструкция **return.**

Пользовательские функции в PHP

```php
<?php
function getSum($a, $b) {
  return $a + $b;
}
echo getSum(7, 8);
echo GETSUM(7, 8);
?>
```




```php
<?php
function funct() {
        $number = 777;
        return $number;
}
$a = funct();
echo $a;
?>
```



Пользовательские функции в PHP

Особенности пользовательских функций PHP

Доступны параметры по умолчанию. Есть возможность вызывать одну и ту же функцию с переменным числом параметров;

Пользовательские функции могут возвращать любой тип;

Область видимости переменных внутри функции является иерархической (древовидной);

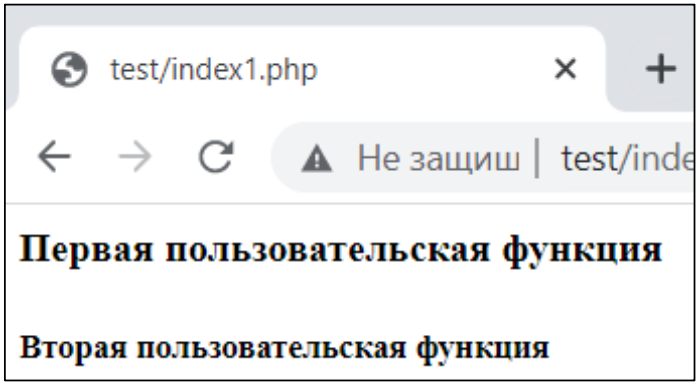
Есть возможность изменять переменные, переданные в качестве аргумента.

Пользовательские функции в PHP

Недостатки пользовательских функций PHP

1. Невозможность объявления локальных функций. В PHP вы не можете объявить локальную функцию, как это можно сделать в других языках программирования. Попросту говоря, вы не можете создать функцию внутри другой функции таким образом, чтобы первая (вложенная) функция была видна только во второй функции. В PHP вложенная функция будет доступна всей программе (скрипту), а значит не будет локальной.

```php
<?php
function first_function() {
    echo "<h4>Первая пользовательская функция</h4>";
    function second_function() {
        echo "<h5>Вторая пользовательская функция</h5>";
    }
}
first_function();
second_function();
?>
```



Пользовательские функции в PHP

Недостатки пользовательских функций PHP

2. Второй недостаток пользовательских функций PHP связан с областью видимости функций. **Для PHP все объявленные и используемые в функции переменные по умолчанию локальны для функции**. То есть, по умолчанию нет возможности изменить значение глобальной переменной в теле функции.

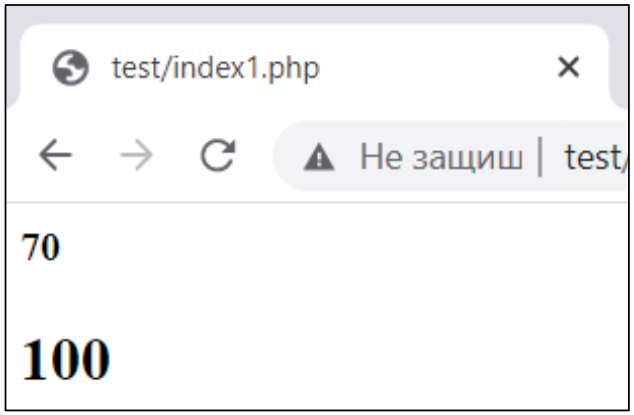
Глобальные переменные - это переменные, которые доступны всей программе, включая подпрограммы (функции).
Локальные переменные - переменные, определенные внутри подпрограммы (функции). Они доступны только внутри функции, в которой они определены.

Если вы в теле пользовательской функции будете использовать переменную с именем, идентичным имени глобальной переменной (находящейся вне пользовательской функции), то никакого отношения к глобальной переменной эта локальный переменная иметь не будет. В данной ситуации в пользовательской функции будет создана локальная переменная с именем, идентичным имени глобальной переменной, но доступна данная локальная переменная будет только внутри этой пользовательской функции.

Пользовательские функции в PHP

Недостатки пользовательских функций PHP

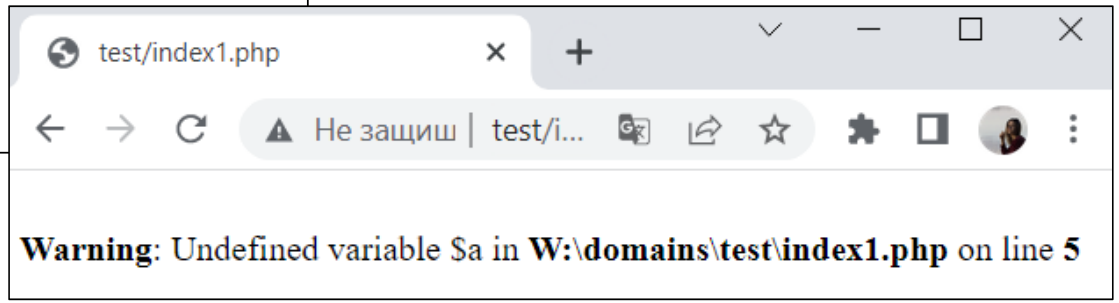
```php
<?php
    $a = 100; /* глобальная область
    видимости */

    function funct() {
        $a = 70; /* ссылка на переменную
    локальной области видимости */
        echo "<h4>$a</h4>";
    }
    funct();
    echo "<h2>$a</h2>";
?>
```



Пользовательские функции в PHP

Основное достоинство локальных переменных — отсутствие непредвиденных побочных эффектов, связанных со случайной или намеренной модификацией глобальной переменной.

```php
<?php
    $a = 1; /* глобальная область видимости */
    function Test()
    {
        echo $a; /* ссылка на переменную
    локальной области видимости */
    }
    Test();
?>
```

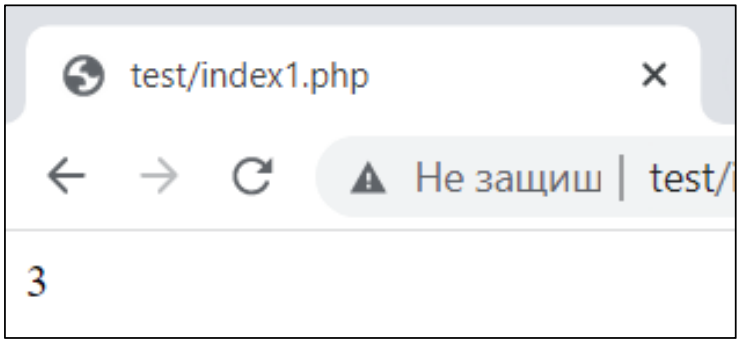


Этот скрипт не сгенерирует никакого вывода, поскольку выражение echo указывает на локальную переменную $a, а в пределах локальной области видимости ей не было присвоено значение.

Пользовательские функции в PHP

В PHP существует специальная инструкция global, позволяющая пользовательской функции работать с глобальными переменными.

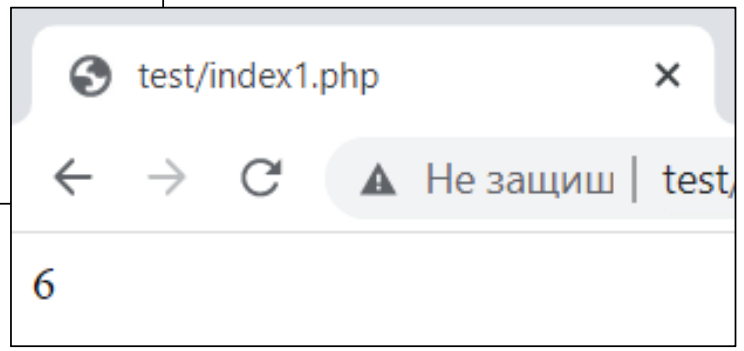
```php
<?php
    $a = 1;
    $b = 2;

    function Sum()
    {
        global $a, $b;
        $b = $a + $b;
    }
    Sum();
    echo $b;
?>
```



Пользовательские функции в PHP

Второй способ доступа к переменным глобальной области видимости – использование специального, определяемого PHP массива $GLOBALS.

```php
<?php
$a = 1;
$b = 5;

function Sum()
{
     $GLOBALS["c"] = $GLOBALS["a"] + $GLOBALS["b"];
}
Sum();
echo $c;
?>
```



Пользовательские функции в PHP

Передача аргументов пользовательским функциям

При объявлении функции можно указать список параметров, которые могут передаваться функции, например:

```
<?php
   function funct($a, $b, /* ..., */ $z)
   { ... };
?>
```

При вызове функции funct() нужно указать все передаваемые параметры, поскольку они являются обязательными. В PHP пользовательские функции могут обладать необязательными параметрами или параметрами по умолчанию.

Пользовательские функции в PHP

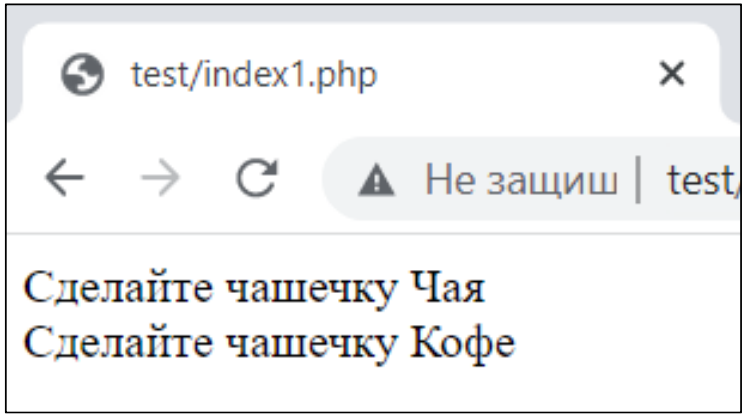
Параметры по умолчанию

При программировании часто возникает необходимость создания функции с переменным числом параметров. Тому есть две причины:

- Параметров слишком много. При этом нет смысла каждый раз указывать все параметры;
- Функции должны возвращать значения разных типов в зависимости от набора параметров.

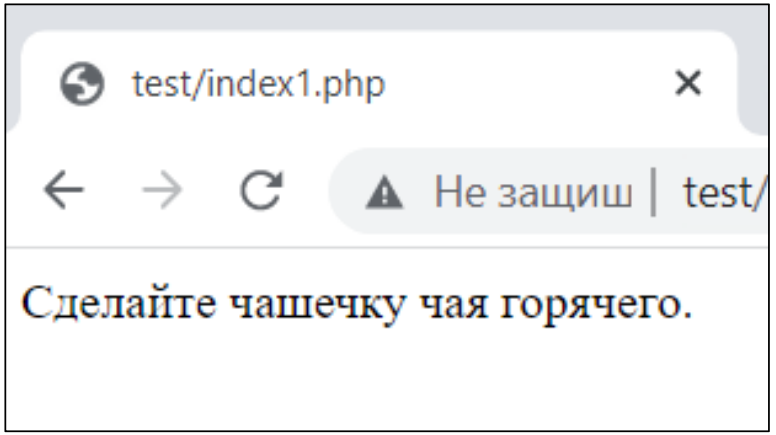
В PHP функции могут возвращать любые значения в зависимости от переданных им параметров.

```php
<?php
    function makecup($type = "Чая")
    {
        return "Сделайте чашечку $type";
    }
    echo makecup();
    echo "<br>";
    echo makecup("Кофе");
?>
```



Пользовательские функции в PHP

```php
<?php
    function makecup($cond, $type = "чая")
    {
        return "Сделайте чашечку $type $cond.";
    }

    echo makecup("горячего");
?>
```



Пользовательские функции в PHP

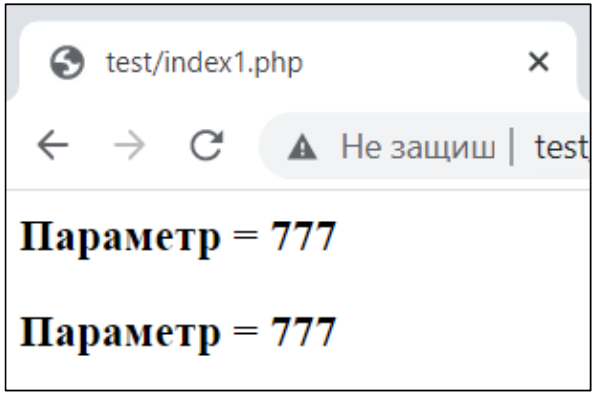
Передача аргументов по ссылке

Согласно сложившимся традициям, во всех языках программирования есть два вида аргументов функций:

- параметры-значения;
- параметры-переменные.

Функции не могут изменить параметр-значение, то есть он доступен функции "только для чтения" - она может его использовать, но не более. В качестве параметра-значения необязательно указывать переменную, можно указать само значение, отсюда название - параметр-значение.

```php
<?php
    function funct($string)
    {
        echo "<h3>Параметр = $string </h3>";
    }
    $str = 777;
    funct(777);
    funct($str);
?>
```

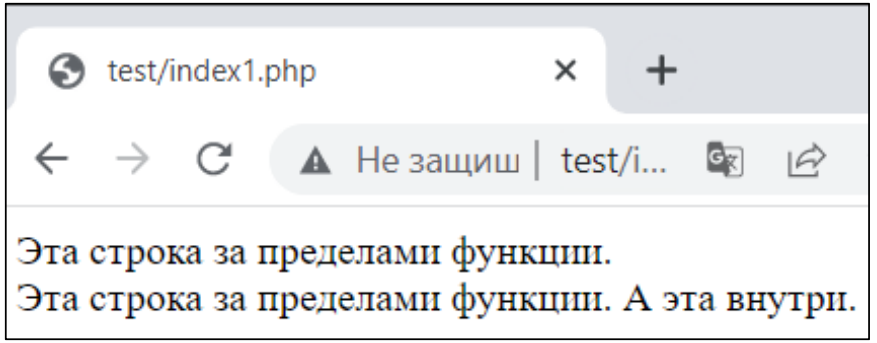


Пользовательские функции в PHP

Передача аргументов по ссылке

В отличие от параметров-значений, параметры-переменные могут быть изменены в процессе работы функции. Тут уже нельзя передавать значение, нужно обязательно передать переменную. В PHP для объявления параметров-переменных используется механизм передачи переменной по ссылке.

```php
<?php
    function funct(&$string)
    {
        $string .= ' А эта внутри.';
    }
    $str = 'Эта строка за пределами функции. ';
    echo $str,"<br>";
    funct($str);
    echo $str;
?>
```



Пользовательские функции в PHP

Переменное число аргументов в функциях

Иногда изначально точно не известно, сколько параметров будет передано функции. Специально для такого случая разработчики PHP предусмотрели возможность использования переменного числа аргументов.

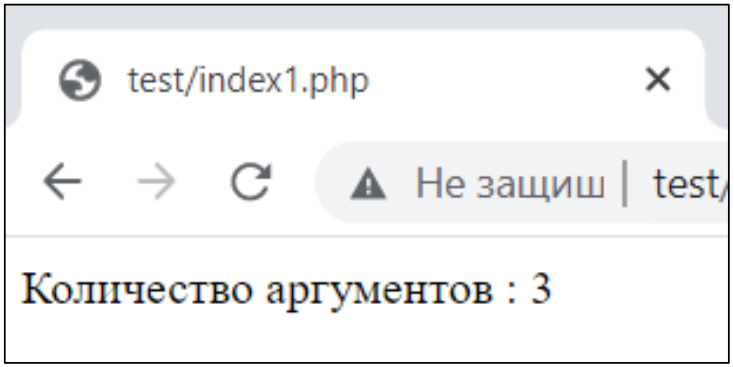
Реализация этой возможности достаточно прозрачна и заключается в использовании функций func_num_args(), func_get_arg() и func_get_args().

Пользовательские функции в PHP

Переменное число аргументов в функциях

Стандартная функция func_num_args() возвращает количество аргументов, переданных пользовательской функции:

```php
<?php
    function funct()
    {
        $numargums = func_num_args();
        echo "Количество аргументов : $numargums";
    }
    funct(1, 2, 3);
    // Скрипт выведет 'Количество аргументов: 3'
?>
```

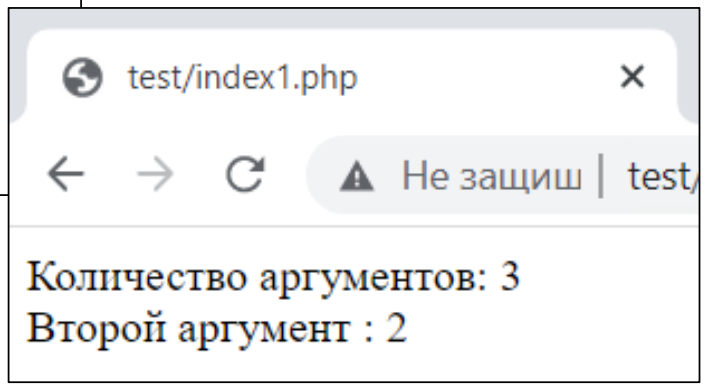


Пользовательские функции в PHP

Переменное число аргументов в функциях

Стандартная функция func_get_arg() возвращает элемент из списка переданных пользовательской функции аргументов:

```
<?php
    function funct()
    {
            $numargs = func_num_args();
            echo "Количество аргументов: $numargs<br>";
            if ($numargs >= 2) {
            echo "Второй аргумент : ".func_get_arg(1)."<br>";
            }
    }
    funct(1, 2, 3);
?>
```

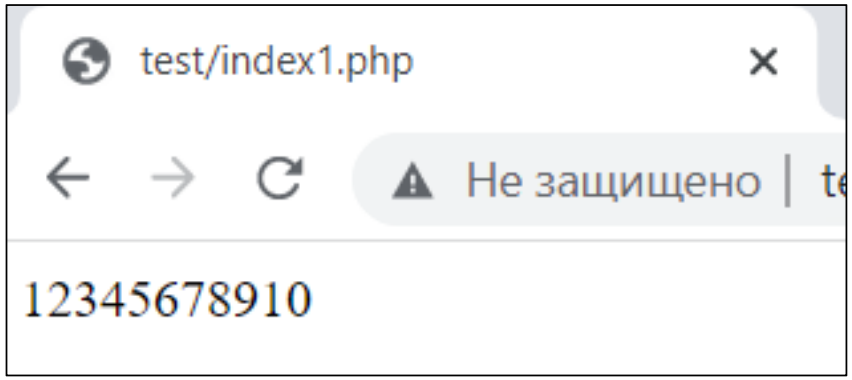


Пользовательские функции в PHP

Статические переменные

Помимо локальных и глобальных переменных, в PHP существует еще один тип переменных – статические переменные.
Если в теле пользовательской функции объявлена статическая переменная, то компилятор не будет ее удалять после завершения работы функции. Пример работы пользовательской функции, содержащей статические переменные:

```php
<?php
    function funct()
    {
        static $a;
        $a++;
        echo "$a";
    }
    for ($i = 0; $i++<10;) funct();
?>
```



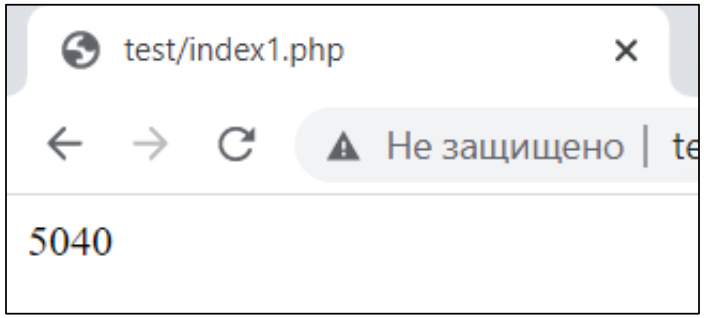
Пользовательские функции в PHP

Рекурсивные функции

Рекурсивные функции - это функции, вызывающие самих себя. Такой вызов называется рекурсивным.
Рекурсия бывает:

- прямая;
- непрямая.

```php
<?php
    function factorial($x) {
        if ($x === 0) return 1;
        else return $x*factorial($x-1);
    }
    echo factorial(7);
?>
```



В рассмотренном примере пользовательская функция factorial() вызывает сама себя, что является прямой рекурсией.
Непрямая рекурсия возникает, когда первая функция вызывает вторую, а вторая - первую.
При создании рекурсивных функций необходимо соблюдать осторожность, стараясь избегать зацикливания.