Лабораторно упражнение № 3

Оператор за цикъл – for
Операторите за предаване на управление
Операторите include и require

1. *Оператор за цикъл - for*

Структура:

for (израз1; израз2; израз3) {блок_за_изпълнение}

или

for (израз1; израз2; израз3): блок_за_изпълнение endfor;

Първият израз се изпълнява веднъж в началото на изпълнението на цикъла. Вторият израз се изчислява преди всяка итерация. Ако неговата стойност е TRUE, операторите от тялото на цикъла се изпълняват, в противен случай изпълнението на цикъла приключва. Третият израз се изчислява след всяка итерация. Всеки от трите израза може да бъде празен. Ако израз2 е празен, цикълът ще се изпълнява безкрайно, тъй като PHP приема по подразбиране неговата стойност TRUE.

Пример 1. Отпечатва всички четни числа между 1 и 9.

Вариант 1.

```
<?php
    for ($i=1; $i<10; $i++){
       if ($i % 2 == 0){
           print $i;
           print "<br>";
        }
    }
?>
```

Вариант 2.

```
<?php
    for ($i = 1; ; $i++) {
      if ($i >9)
         break;
      if ($i % 2 == 0){
           print $i;
           print "<br>";
      }
    }
?>
```

Вариант 3.

```
<?php
    $i = 1;
    for (; ; ) {
      if ($i > 9)
         break;
      if ($i % 2 == 0){
           print $i;
           print "<br>";
      }
    $i++;
    }
?>
```

Пример 2. Да се отпечатат всички числа между 1 и 9.

В третия израз на конструкция *for* може да запишем няколко прости команди, разделени със запетая. Ако блока съдържа само една команда или не съдържа команди, то фигурни скоби може да няма!!!

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

2. Оператори за предаване на управление

Понякога се изисква незабавно да се излезе от цикъл. За това се използват оператори *break* и *continue*.

Break

Операторът break прекратява изпълнението на операторите за цикъл (*for*, *foreach*, *while*, *do..while)* и оператора *switch*. *Break* може да се използва с числов аргумент.

Continue

Операторът continue прекратява изпълнението на текущата итерация на цикъл, като преминава към проверка на условието за край на цикъла и към началото на нова итерация.

Пример 3. *(break и continue)* Програмата генерира произволно число от 1 до 10 *(функция rand(1,10))*. Ако генерираното число е 5 – прекратяваме цикъла. Ако генерираното число е 7 – итерацията се игнорира.При всяка итерация ще се отпечатва номера на итерацията и генерираното число:
iteration 1 -> number

```
iteration 1-> number generated: 6
Next iteration ...

iteration 2-> number generated: 3
Next iteration ...

iteration 3-> number generated: 7
iteration 3-> number generated: 10
Next iteration ...

iteration 4-> number generated: 5
Number of iterations 4 The cycle stops!
```

```
<?php
    $i=1;
    while ($i){
      $n = rand(1,10); // генерираме произволно число от 1 до 10
      echo "<br>iteration $i-> number generated: $n ";
      if ($n==5) break;
      if ($n==7) continue;
      echo "<br>Next iteration ...<br>";
      $i++;
    }
    echo "<br>Number of iterations $i The cycle stops!";
?>
```

Забележка: Още математически функции: http://www.php.net/manual/bg/ref.math.php.

Пример 4. Програмата генерира произволно число от 1 до 10 *(функция rand(1,10))*. Ако генерираното число е 5 – прекратяваме switch. Ако генерираното число е 10 – прекартяваме switch и while. *Ако след оператора break се укаже число, то се прекъсва изпълнението на именно такова количество вложени цикли.*

```
<?php
    $i=1;
    while ($i) {
    $n = rand(1,10);
      // генерираме произволно число
      // от 1 до 10
      switch ($n)
    {     case 5:
           echo "Изход от switch (n=$n)";
           break 1;// прекратява се работата на switch
         case 10:
           echo "Exit from switch and while (n=$n)";
           break 2;
       // прекратява се работата на switch и while
         default:
           echo "switch works (n=$n), ";
    }
    echo " while works – step $i <br>";
    $i++;
    }
    echo "<br>Number of iterations $i ";
?>
```

Примерен резултат от работата на този скрипт:

```
switch works (n=3), while works – step 1
switch works (n=8), while works – step 2
switch works (n=3), while works – step 3
switch works (n=8), while works – step 4
switch works (n=4), while works – step 5
switch works (n=7), while works – step 6
switch works (n=3), while works – step 7
Изход от switch (n=5) while works – step 8
Изход от switch (n=5) while works – step 9
Exit from switch and while (n=10)
Number of iterations 10
```

continue

Понякога е нужно не напълно да се прекрати работата на цикъла, а само да се започне негова нова итерация. Оператор *continue* позволява да се пропуснат останалите инструкции от блока_за_изпълнение.

continue може да се използва с числов аргумент, указващ колко съдържащи го управляващи конструкции трябва да завършат работа.

Пример 5. Да заменим в пример 4 оператор *break* с *continue*. Още – да ограничим броя на стъпки на цикъла на 3.

```
<?php
    $i=1;
    while ($i<4) {
       $n = rand(1,10);
      // генерираме произволно число от 1 до 10
       echo "$i:$n ";
       // извеждаме номер на итерация и генерираното число
       if ($n==5) {
          echo "New iteration <br>";
          continue;
        /* Ако се генерира число 5,
         то започва нова итерация,
         $i не се увеличава */
       }
       echo "The cycle works<br>";
       $i++;
    }
    --$i;
    echo "<br>Number of iterations $i ";
?>
```

Примерен резултат от работата на този скрипт:

```
1:4 The cycle works
2:6 The cycle works
3:5 New iteration
3:2 The cycle works
4:6 The cycle works
Number of iterations 4
```

Същият резултат може да се постигне и ако напишем:

```
<?php
    $i=1;
    while ($i<4) {
       $n = rand(1,10);
      // генерираме произволно число
       // от 1 до 10
       if ($n!==5) {
          echo "$i:$n The cycle works <br>";
      // извеждаме номер на итерация
       // и генерираното число
       $i++;
       }
    }
 ?>
```

Примерен резултат от работата на този скрипт:

1:8 The cycle works
2:3 The cycle works
3:7 The cycle works

В PHP съществува една особеност на оператор *continue* – в конструкция *switch* той работи така както и *break*. Ако *switch* се намира вътре в цикъла и е нужно да започнем нова итерация на цикъла, следва да използваме continue 2.

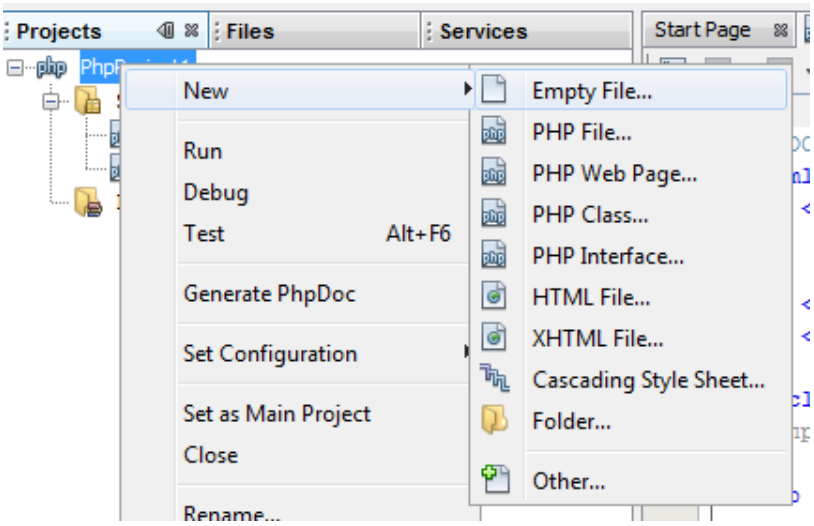
3. Оператори за включване

include

Оператор *include* позволява включване на код, съдържащ се в указан файл, и изпълнението му толкова пъти, колкото пъти програмата среща този оператор. Включването може да стане по един от изброените начини:

include 'име_на_файла';
Пример: include 'params.php'.
include $file_name;
Пример: $file_name='params.php';
include $file_name ;
include ("име_на_файла");
Пример: include ("params.inc");

Пример 6. Нека във файл params.php се съхранява набор от параметри и функции. Всеки път, когато са ни нужни тези параметри (функции), ние слагаме команда include 'params.php'.



params.php

```
<?php
$user = "Alla";
$today = date("d.m.y");
/* функция date() връща дата
и време (тук – датата е във формат ден.месец.год) */
?>
```

include.php

```
<?php
include ("params.php");
/* променливи $user и $today са зададени в params.php.*/
echo "Hello, $user!<br>";    // извежда "Hello, Alla!"
echo "Today is $today";
?>
```

Резултат от работата на този скрипт е:
Hello, Alla!
Today is 10.02.18

Търсенето на файла за включване става по следния начин: Първо се търси в include_path текущата директория (текуща е директорията в която се намира вашия основен файл include.php, например www), ако файла, който ще включваме е в поддиректория my_dir на текущата директория (www/my_dir/params.php) то първо търсим там. Ако файла params.php не се намери, то търсенето продължава в директорията на текущия скрипт (www).

Ако един файл е включен с помощта на include, то съдържащия се в него код наследява областта на видимост на променливите на стринга, в който се намира include. Съответно, ако include е вътре във функция, то кодът, съдържащ се в извикания файл, ще се държи така, както е определен вътре във функцията (наследява областта на видимост на функцията).

Пример 7 Нека файл params.php не се променя, а include.php се промени на:

```
<?php
    function Footer(){
       // функция Footer
       include ("params.php");
    /* включваме файл params.php.
      Сега параметрите могат да се използват само във функцията */
       $str = "Today: $today <br>";
       $str .= "THE PAGE IS CREATED BY $user";
       echo "$str";
    }
    Footer();
    echo "$user, $today";
    // вижда се, че тези променливи са видни само във функцията
?>
```

Резултат от работата на този скрипт е:
Today: 10.02.18
THE PAGE IS CREATED BY Alla
Notice: Undefined variable: user in C:\xampp\htdocs\PhpProject36\index.php on line 24
Notice: Undefined variable: today in C:\xampp\htdocs\PhpProject36\index.php on line 24

Освен локалните файлове, с помощта на include може да включим и външни файлове, указвайки тейния url-адрес. Тази възможност се контролира с директивата url_fopen_wrappers във файла с настройки на PHP и по подразбиране, като правило, е включена след версии на PHP от PHP 4.3.0 нагоре.

Забележка: include() е специална езикова конструкция, ето защо при използване в условни блокове трябва да се поставя във фигурни скоби.

```
<?php
    /*  Това е грешен запис.Очаква се грешка. */
    if ($condition) include("first.php");
    else include("second.php");
    // Сега е правилно.
    if ($condition){ include("first.php"); }
    else { include("second.php"); }
?>
```

Забележка: *При използване на include са възможни два вида грешки – грешка при включване (например, не намира файла и т.н.) или при изпълнение (ако грешката е вътре във включения файл).*

require- включва даден файл в текущия скрипт, изпълнява кода, който той съдържа, и връщат резултата от това изпълнение.

Операторите include и require са напълно идентични в действието си, с изключение на начина, по който обработват грешките. Операторът include издава предупреждение за грешка (warning) и продължава изпълнението на скрипта, докато require издава съобщение за грешка (fatal error) и прекратява изпълнението на скрипта. Тази разлика обуславя и прилагането на двата оператора: require трябва да се използва когато липсата на файла, който се включва трябва да прекрати изпълнението на скрипта и обратно, когато трябва изпълнението да продължи докрай независимо от това дали файлът е намерен, трябва да се използва include.