Лабораторная работа № 3 "Циклы и массивы"

1. Циклы в PHP

Циклы позволяют многократно выполнять блок кода. Это необходимо для решения множества задач. Например, перебор записей, полученных из базы данных, построчное чтение файла или обход элементов массива. В PHP есть четыре типа циклов: *while*, *do..while*, *for* и *foreach*.Первые три описываются ниже, а *foreach* -при обсуждениимассивов далее в этом уроке.

1.1 Цикл while

Конструкция *while* представляет собой простейший оператор цикла. Блок операторов выполняется пока верно условие.

```
while (условие)
{
операторы
}
```

Рассмотрим работу цикла подробнее:

1. проверка условия
2. если оно истинно, выполнение операторов; если ложно – выход из цикла
3. переход к шагу 1

Для управления циклом обычно требуется одна или несколько переменных. Например, целочисленное значение, каждый раз увеличиваемое на единицу. Эти переменные так и называются – управляющие переменные цикла.
Рассмотрим простой пример вывода чисел от 1 до N:

```
<?php
$n = 10;
$i = 1;

while ($i <= $n)
{
echo "$i<br/>";
$i++;
}
```

```
?>
```

Иногда управляющая переменная цикла бывает булевой. Например, при чтении в цикле строк файла можно использовать переменную булевского типа, определяющую конец файла.

1.2 Цикл do..while

Циклы *do..while* аналогичны циклам *while*, за исключением того, что условие проверяется не в начале, а в конце каждой итерации. Это означает, что цикл будет выполнен, по крайней мере, один раз.

```
do
{
операторы

} while (условие)
```

Рассмотрим работу цикла:

1. выполнение блока операторов
2. проверка условия
3. если оно истинно, переход к шагу 1; если ложно – выход из цикла

1. следующем примере единица будет выведена, даже если N=0:

```
<?php
$n = 10;
$i = 1;

do
{
echo "$i<br/>";
$i++;
} while ($i <= $n) ?>
```

Цикл *do..while* используют достаточно редко ввиду его громоздкости и плохой читаемости. Лучше слегка изменить алгоритм (от чего он, скорее всего, выиграет) и применить *for* или *while*.

1.3 Цикл for

Цикл *for* – шедевр лаконичной организации кода, пришедший из языка Си. Конструкция позволяет одной строкой полностью определить поведение цикла.

```
```

for

```
(выражение1; выражение2; выражение3)
{
операторы
}
```

Выражение1 вычисляется перед началом цикла.Обычно в нем инициализируетсяуправляющая переменная. *Выражение2* вычисляется в начале каждой итерации цикла. Это выражение ведет также, как условие цикла *while*, если значением *Выражения2* оказывается *true*, цикл продолжается, иначе – останавливается. *Выражение3* вычисляется в конце каждой итерации и, как правило, используется для изменения значения управляющей переменной цикла.

Ход выполнения цикла:

1. выполнение *Выражения1*
2. проверка *Выражения2*
3. если оно истинно, переход к шагу 4; если ложно – выход из цикла
4. выполнение блока операторов
5. выполнение *Выражения3*
6. переход к шагу 2

Рассмотрим тот же пример вывода чисел от 1 до N:

```
<?php
$n = 10;

for ($i = 1; $i <= $n; $i++)
{
echo "$i<br/>";
}
?>
```

Пример демонстрирует наиболее частое применение конструкции *for*: инициализировать управляющую переменную, сравнить переменную со значением, инкрементировать или декрементировать значение. Однако *for* может использоваться и другими способами.

Так же, как в Си и других языках, допускается не указывать одно или более выражений (если не указано *Выражение2*, считается, что оно *true*).
while всегда можно заменить на for. Следующие две конструкции эквивалентны.

1:

```
while (условие)
{
операторы
}
```

2:

```
for ( ; условие; )
{
операторы
}
```

1.4 Бесконечный цикл, операторы выхода из цикла и прерывания итерации цикла

Бесконечным называется цикл такого вида:

```
while (true)
{
…
}
```

или такого (что одно и то же):

```
for (;;)
{
…
}
```

Для выхода из цикла можно использовать оператор break:

```
<?php
$n = 10;
$i = 1;

while (true)
```

```
{
echo "$i<br/>";
$i++;
if ($i > $n)
break;
}
?>
```

Оператор break мгновенно завершает выполнение цикла. Если же нужно закончить текущую итерацию цикла и вернуться к проверке его условия, можно воспользоваться оператором continue. Следующий пример демонстрирует вывод нечетных чисел от 1 до N:

```
<?php
$n = 10;

for ($i = 1; $i <= $n; $i++)
{
if ($i % 2 == 0)
continue;

echo "$i<br/>";
}
?>
```

Остаток от деления на два четного числа равен нулю, такие числа мы пропускаем в приведенном примере.

Правильнее считается не использовать операторы прерывания цикла, а возлагать логику управления на условие. Старайтесь организовывать циклы именно так.

2. Массивы

2.1 Что такое массив

Массив - именованный набор однотипных переменных. Он состоит из элементов. Обращение к каждому элементу происходит по его номеру. Нумерация начинается с нуля, то есть у первого элемента индекс 0, у второго элемента индекс 1 и т. д.

Массивы в PHP – чрезвычайно мощный инструмент. По сути это не даже массив, а словарь. Словарь (он же хеш-таблица) – такая структура данных, которая хранит множество пар ключ-значение. Ключи не могут повторяться. В качестве ключа может использоваться как целое число, так и строка.

Есть несколько методов инициализации переменной массива. Один из них состоит просто

1. том, чтобы начать присваивать значения элементам переменной массива. Приводимый ниже код создает массив с именем *$aLanguages* из трех элементов. Поскольку индексы не указаны, PHP по умолчанию присваивает числовые индексы 0, 1 и 2:

```
$aLanguages[] = "Arabic";

$aLanguages[] = "German";

$aLanguages[] = "Korean";

echo($aLanguages[2]); // Выводит "Korean"
```

Чтобы явно указать индекс, заключите его в квадратные скобки:

```
$aLanguages[0] = "Arabic";

$aLanguages[1] = "German";

$aLanguages[2] = "Korean";

echo($aLanguages[2]); // Выводит "Korean"
```

Элементы массива не обязательно должны объявляться последовательно. Следующий код создает массив элементов с индексами 100, 400, 300 и 401:

```
$aLanguages[100] = "Arabic";

$aLanguages[400] = "German";

$aLanguages[300] = "Korean";

$aLanguages[] = "Tagalog";

echo($aLanguages[300]); // Выводит "Korean"

echo($aLanguages[401]); // Prints "Tagalog"
```

Поскольку индекс последнего элемента не был задан, PHP присвоил ему первый доступный индекс после самого большого использованного до сих пор индекса: 401.

Конструкция *array()* дает альтернативный способ определения массивов. *array()*

принимает разделенный запятыми список значений, подлежащих помещению в массив:

```
$aLanguages = array("Arabic", "German", "Korean", "Tagalog"); echo($aLanguages[2]); // Выводит "Korean"
```

1. снова, поскольку индексы не были заданы, они присваиваются элементам массива по умолчанию. Для явного указания индексов в конструкции *array()* применяется оператор

=>:

```
$aLanguages = array("Arabic", 3 => "German", "Korean", "Tagalog"); echo($aLanguages[0]); // Выводит "Arabic" echo($aLanguages[3]); // Выводит "German" echo($ai_anguages[4]); // Выводит "Korean" echo($aLanguages[5]); //Выводит "Tagalog"
```

Как упоминалось выше, индексы массива могут быть строками:

```
$aLanguages = array(
"ar" => "Arabic",
"de" => "German",
"tl" => "Tagalog"
);
echo($aLanguages["tl"]); // Выводит "Tagalog"
$aLanguages["ko"] = "Korean";
echo($aLanguages["ko"]); // Выводит "Korean"
```

2.2 Обход массивов в цикле

Самый простой способ обойти все элементы массива:

```
for ($i = 0; $i < count($arr); $i++)
{
…
}
```

Однако способ годится только тогда, когда в качестве ключа используется порядковый номер элемента (начиная с нуля).
Для обхода любых массивов (в общем случае – словарей) существует специальный оператор foreach. Его синтаксис прост:

```
foreach (массив as [$key =>] $value)
```

```
{
операторы
}
```

Оператор foreach проходит каждый элемент массива по одному разу. В каждом проходе в переменную $key помещается индекс этого элемента, а в переменную $value - значение. Имена этих двух переменных произвольны.

```
foreach ($aLanguages as $sIndex => $sVal)
{
echo("$sIndex is $sVal <br/>");
}
```

Переменная для ключа необязательна, поскольку не всегда нужна внутри цикла. В данном примере переменная $key опущена, а вместо $value используется $sLang:

```
echo("Available Languages: <br/><ul>");

foreach ($aLanguages as $sLang)
{
echo( "<li>$sLang</li>" ) ;
}

echo("</ul>");
```

2.3 Функции для работы с массивами

PHP предлагает массу функций, облегчающих работу с массивами. Ряд полезных функций мы приводим ниже. Полный список можно найти в электронной документации на английском языке: *http://www.php.net/manual/en/ref.array.php*.

count()

int count(mixed var)

Функция count() принимает в качестве аргумента массив и возвращает количество элементов в нем. Если переменная не установлена или не содержит элементов, возвращается ноль.

in_array()

boolean in_array(mixed needle, array haystack [, bool strict])

Эта функция ищет в массиве haystack значение needle и возвращает true, если оно найдено, и false в противном случае.

sort()

void sort(array array [, int sort_flags])

Эта функция применяется для сортировки значений в массиве. Необязательный второй параметр sort_flags указывает, как должны сортироваться данные. Допустимыми значениями являются SORT_REGULAR, SORT_NUMERIC, устанавливающие сравнение значений как чисел, и SORT_STRING, устанавливающее сравнение значений как строк. PHP содержит много функций сортировки, синтаксис которых очень близок к sort(). Эти функции по-разному ведут себя, предоставляя разные варианты процедуры сортировки, включая направление сортировки, обработку ключей и алгоритмы сравнения. Подробнее смотрите в документации такие функции как arsort(), asort(), ksort(), natsort(), natcasesort(), rsort(), usort(), array_multisort() и uksort().

explode() и implode()

Эти две функции официально считаются строковыми, но они касаются массивов. explode() расщепляет строку на отдельные элементы, помещаемые в массив, используя разделитель, переданный в качестве аргумента. implode() осуществляет противоположную операцию: она сжимает элементы массива в одну строку, соединяя их с помощью переданного аргумента:

1. Превратить массив в строку, с разделителем - точкой с запятой: $sLangString = implode(';', $aLanguages); echo($sLangString);

$sSentence = 'Per aspera ad astra';

1. Превратить предложение в массив отдельных слов:

```
$aWords = explode(' ', $sSentence);
```

2.4 Многомерные массивы

Многомерный массив возникает, когда элементы некоторого массива сами содержат массивы (которые, в свою очередь, могут содержать массивы и т. д.). Для инициализации многомерных массивов используются те же средства, включая вложенные конструкции array():

```
$aLanguages = array(

"Slavic" => array("Russian", "Polish", "Slovenian"), "Germanic" => array("Swedish", "Dutch", "English"), "Romance" => array("Italian", "Spanish", "Romanian")

);
```

Для доступа к элементам многомерных массивов, вложенным глубоко внутрь, применяются дополнительные скобки. Таким образом, $aLanguages["Germanic"] указывает на массив, содержащий

германские языки, а $aLanguages["Germanic"][2] указывает на третий элемент ("English") вложенного массива.

Обход многомерных массивов может осуществляться с помощью вложенных циклов:

```
<?php

foreach ($aLanguages as $sKey => $aFamily)

{
```

1. Вывести название семейства языков: **echo**(

```
"<h2>$sKey</h2>" . "<ul>"
);
```

1. Теперь перечислить языки в каждом семействе: **foreach** ($aFamily **as** $sLanguage)

```
{
echo("<li>$sLanguage</li>");

}
```

1. Завершить список:

```
echo("</ul>");
}
?>
```

При каждом проходе внешнего цикла переменной $sKey присваивается в качестве значения название семейства языков, а переменной $aFamily - соответствующий внутренний массив. Внутренний цикл обходит массив $aFamily, помещая значение каждого элемента в переменную $sLanguage.

2.5 Предопределенные массивы

$GLOBALS

Содержит ссылку на каждую переменную, доступную в данный момент в глобальной области видимости скрипта. Ключами этого массива являются имена глобальны переменных.

$_SERVER

Переменные, установленные веб-сервером либо напрямую связанные с окружением выполнения текущего скрипта.

$_GET

Переменные, передаваемые скрипту через HTTP GET.

$_POST

Переменные, передаваемые скрипту через HTTP POST.

$_COOKIE

Переменные, передаваемые скрипту через HTTP cookies.

$_FILES

Переменные, передаваемые скрипту через HTTP post-загрузку файлов.

$_ENV

Переменные, передаваемые скрипту через окружение.

$_SESSION

Переменные, зарегистрированные на данный момент в сессии скрипта.

$_REQUEST

Переменные, передаваемые скрипту через механизмы ввода GET, POST и COOKIE. Не рекомендуется для использования. Следует обращаться к конкретному массиву.

Задание

1. С помощью цикла while выведите все числа в промежутке от 0 до 100, которые делятся на 3 без остатка.

2. С помощью цикла do…while напишите функцию для вывода чисел от 0 до 10, чтобы результат выглядел так:

0 – это ноль
1 – нечетное число
2 – четное число
3 – нечетное число
…

10 – четное число

3. Выведите с помощью цикла for числа от 0 до 9, НЕ используя тело цикла. То есть выглядеть должно вот так:

for(…){// здесь пусто}

4. Объявите массив, где в качестве ключей будут использоваться названия областей, а в качестве значений – массивы с названиями городов из соответствующей области. Выведите в цикле значения массива, чтобы результат был таким:

Московская область:
Москва, Зеленоград, Клин
Ленинградская область:

Санкт-Петербург, Всеволожск, Павловск, Кронштадт Рязанская область

…
(названия городов можно найти на maps.yandex.ru)

5. Повторите предыдущее задание, но выводите на экран только города, начинающиеся с буквы «К».

6. Объявите массив, индексами которого являются буквы русского языка, а значениями – соответствующие латинские буквосочетания (‘а’=> ’a’, ‘б’ => ‘b’, ‘в’

=> ‘v’, ‘г’ => ‘g’, …, ‘э’ => ‘e’, ‘ю’ => ‘yu’, ‘я’ => ‘ya’).
Напишите функцию транслитерации строк.

7. Напишите функцию, которая заменяет в строке пробелы на подчеркивания и возвращает видоизмененную строчку.

8. Объедините две ранее написанные функции в одну, которая получает строку на русском языке, производит транслитерацию и замену пробелов на подчеркивания (аналогичная задача решается при конструировании url-адресов на основе названия статьи в блогах).