Урок 3

Циклы и массивы

Знакомство с различными типами циклов в программировании и их реализацией в PHP. Знакомство с понятием и реализацией массивов.

Понятие цикла

Цикл - разновидность управляющей конструкции в высокоуровневых языках программирования, предназначенная для организации многократного исполнения набора инструкций. Говоря простым языком, цикл – это конструкция, которая заставляет определённую группу действий повторяться до наступления нужного условия. Например:

- опрашивать клиентов, пока не наберётся 100 успешных ответов;
- читать строчки из файла, пока не доберёмся до конца самого файла.

Набор этих инструкций называется телом цикла. Каждое выполнение тела цикла называется итерацией. Условие, которое определяет, делать ли следующую итерацию или завершить цикл, называется условием выхода. Очень удобно знать, сколько итераций уже сделано – т.е. хранить их число в переменной. Такая переменная называется счётчиком итераций цикла, но очевидно, что она не является обязательной для всех циклов.

Каждая итерация состоит из следующих шагов:

1. Инициализация переменных цикла.
2. Проверка условия выхода.
3. Исполнение тела цикла.
4. Обновление счётчика итераций.

Кроме того, в большинстве языков программирования есть инструменты досрочного прерывания всего цикла или же выполнения текущей итерации.

Циклы в PHP

Циклы в PHP следуют всем вышеобозначенным правилам и делятся на несколько типов:

1. Цикл с предусловием - цикл, который выполняется пока истинно некоторое условие, указанное перед его началом. Поскольку условие проверяется до выполнения самой первой итерации, вполне может не выполниться ни одной итерации, если условие изначально ложно.
2. Цикл с постусловием - цикл, в котором условие проверяется после выполнения тела цикла. Отсюда следует, что тело всегда выполняется хотя бы один раз.
3. Цикл со счётчиком - цикл, в котором счётчик итераций изменяет своё значение от заданного начального значения до конечного значения с некоторым шагом, и для каждого значения счётчика тело цикла выполняется один раз.
4. Совместный цикл - цикл, задающий выполнение некоторой операции для объектов из заданного множества, без явного указания порядка перечисления этих объектов. Говоря проще, этот цикл проходит по всем элементам переданного множества.

Циклы while и do…while

Цикл while является примером цикла с предусловием. Выглядит он следующим образом:

```
<?php
while( Условие ) {
  Операторы;
}
?>
```

Нетрудно догадаться, что тело цикла, содержащее операторы, будет выполняться до тех пор, пока истинно условие, указанное в начале цикла. Алгоритм такого цикла в сравнении с базовым будет выглядеть так:

1. Проверка условия.
2. Выполнение тела, если условие истинно. Выход, если условие ложно.

Для управления циклом обычно требуется одна или несколько переменных. К примеру, некое значение boolean, которое обращается в false при достижении некоего граничного условия. Или же целочисленное значение, каждый раз увеличиваемое на единицу, пока не достигнет определённого значения. На практике:

```
<?php
$n = 10;
$i = 1;
while ($i <= $n) {
  echo "$i<br/>";
  $i++;
}
?>
```

Обычно цикл while используется для реализации программ, работающих постоянно (так называемые «демоны», не совсем характерные для языка PHP). Хороший пример демона – программа, которая «слушает» входящие соединения до тех пор, пока не придёт команда на разрыв соединения.

Цикл do…while – это уже цикл с постусловием, работающий по алгоритму:

1. Выполнение блока операторов.
2. Проверка условия.
3. Выход, если условие ложно.

В 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.

Цикл for

Цикл for относится к третьему типу циклов – это цикл со счётчиком. Является классическим примером красивой организации кода и имеет наиболее схожий вид в большинстве языков программирования.

В PHP он пришёл из языка C. Конструкция позволяет одной строкой полностью определить поведение цикла.

```
<?php
for(выражение1; выражение2; выражение3){
  операторы;
}
?>
```

Выражение1 вычисляется перед началом цикла. Обычно в нем инициализируется управляющая 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.
Сам цикл выполняется согласно следующему алгоритму:

1. Выполнение выражения1.
2. Проверка на истинность выражения2.
3. Если выражение2 истинно, то выполняется тело цикла.
4. Выполнение выражения3.

И снова в цикле выведем числа от 1 до 10, но уже с применением цикла for:

```
<?php
$n = 10;
for ($i = 1; $i <= $n; $i++){
  echo "$i<br/>";
}
?>
```

Это самый частый вариант применения for: мы создаём управляющую переменную-счётчик, устанавливаем ей границу роста и инкрементируем её с каждой итерацией. Но это не единственный способ использования for.
Совершенно необязательно указывать все три выражения для расчёта. Таким образом, for может с лёгкостью заменить, к примеру, while:

```
<?php
for ( ; условие; ){
  операторы
}
?>
```

Что означает эта запись? Первое выражение не выполняется, на каждом шаге будет проверяться условие, т.к. нам надо знать, когда выходить из цикла. После каждой итерации мы не производим никаких неявных манипуляций. Т.е. мы имеем цикл только с управляющим предусловием, что эквивалентно циклу while.

Бесконечный цикл и выход из шагов, цикла

Настоящий бич неопытных программистов – это бесконечные циклы. Ведь если в него войти, то с каждой итерацией скорее всего будет потребляться больше и больше памяти, и, в лучшем случае, «упадёт» программа, а не весь сервер. Но в умелых руках бесконечный цикл можно обернуть на пользу.

Бесконечными считаются циклы видов:

```
<?php
for (;;){
  …
}
While(true){
  …
}
?>
```

Такие циклы используются для программ-демонов, о которых мы поговорили в начале занятия. Но как же ими управлять, если условие выхода никогда не вернёт false?
Для выхода из всего процесса цикла в PHP используется оператор break.

```
<?php
$n = 10;
$i = 1;
while (true){
  echo "$i<br/>";
  $i++;
  if ($i > $n)
    break;
}
?>
```

В данном примере мы выводим числа от 1 до 10, но при помощи бесконечного цикла. Каждый раз в конце цикла проверяется состояние переменной i. Как только она станет равна 11, вызовется оператор break, который мгновенно завершает выполнение цикла и приступает к выполнению инструкций, идущих после объявления цикла в коде.
Не всегда нужно останавливать весь цикл. Иногда нужно пропустить итерацию и вернуться к проверке условия, но уже перед следующим выполнением тела цикла. Для этого используется оператор continue.

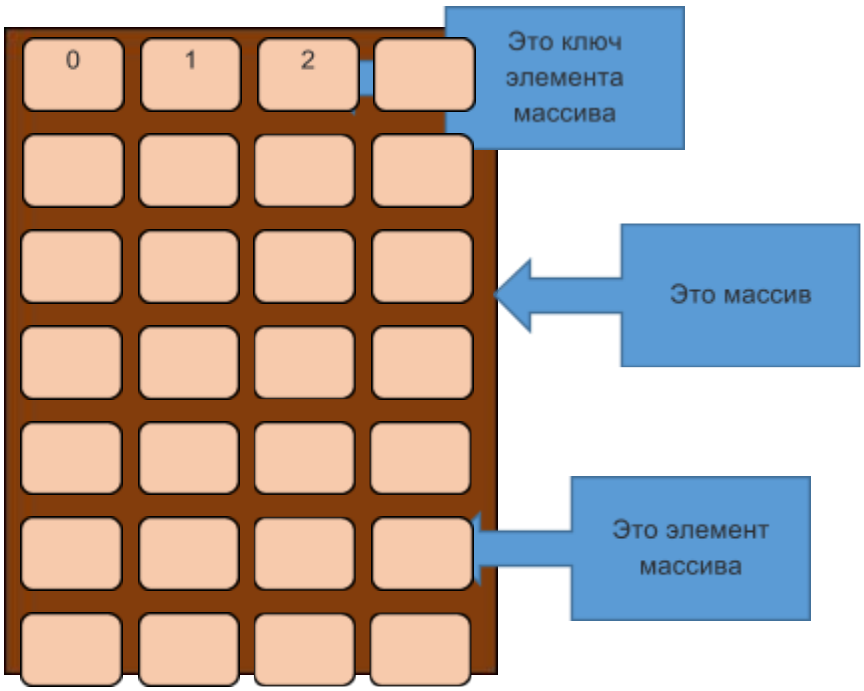
```
<?php
$n = 10;
for ($i = 1; $i <= $n; $i++){
if ($i % 2 == 0)
  continue;

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

В данном примере мы выводим все нечётные числа, пропуская завершение итераций для чётных чисел. Так или иначе, правильнее считается не использовать операторы прерывания цикла, а возлагать логику управления на условие. Старайтесь организовывать циклы именно так.

Массивы

Массив – это именованный набор однотипных переменных. Определение не очень понятно, поэтому гораздо проще обрисовать массив в виде большого шкафа со множеством ящиков. Сам шкаф – это массив, а ящики в этом шкафу – это элементы массива.



В классическом виде нумерация элементов (ящиков) начинается с нуля. Ключи не могут повторяться, они уникальны.

Таким образом, массив может быть определён как переменная, которая может хранить внутри себя не одно, а множество значений.

Массивы в PHP

В языке PHP массив устроен чуть сложнее, чем просто набор элементов – это хэш-таблица, хранящая пары ключ-значение. Продолжая аналогию со шкафом, хэш-таблица – это справочник, лежащий перед нашим шкафом, указывающий, в какой из ячеек лежат носки, а в какой – футболки.

Какое в этом преимущество? В PHP в роли ключа элемента массива может храниться не только число, но и строка. Такие массивы называются ассоциативными и являются одной из отличительных черт языка PHP.

Перейдём к делу. Массивы в языке PHP создаются следующим образом:

```
<?php
// PHP < 5.3
$oldArrayStyle = array();
// PHP >= 5.4
$myArray = [];
?>
```

В данном примере указано, что начиная с версии 5.4 языка PHP массивы можно создавать при помощи сокращённой записи «[]». Но будьте аккуратны – в 5.3 такая запись работать не будет.

В языках со строгой типизацией данных мы можем создать массив строк, массив чисел, массив объектов одинакового типа, и так далее. PHP – язык с динамической типизацией, поэтому в каждую ячейку массива мы можем записать переменную любого типа.

Давайте наполним наш массив.

```
<?php
$myArray = [];
$someVar = true;
$myArray[] = 1;
$myArray[] = 'Hello, world!';
$myArray[] = $someVar;
var_dump($myArray);
?>
```

Квадратные скобки после переменной массива указывают интерпретатору, что необходимо добавить элемент в массив с номером на 1 больше максимального ключа в массиве. Если Ваш массив ассоциативный и имеет только строковые ключи, то PHP создаст записи с ключами от первого доступного индекса после самого большого использованного до сих пор индекса. В нашем случае это 0,1,2 и т.д.

К конкретному элементу можно обратиться по его индексу (ключу).

```
<?php
$myArray = [];
$someVar = true;
$myArray[] = 1;
$myArray[] = 'Hello, world!';
$myArray[] = $someVar;
echo $myArray[1];
?>
```

Будет выведен второй элемент массива. Обращение к несуществующему элементу массива приведёт к некритической ошибке, но, в зависимости от дальнейших инструкций, ошибка может стать весьма существенной. Рекомендуется проверять наличие элемента с помощью функции isset().

Разумеется, в ячейки массива можно класть не только простые переменные, но и другие массивы. Создадим индексный массив, содержащий в каждой своей ячейке по ассоциативному массиву.

```php
<?php
$users = [];
$users[0] = [
  'name' => 'Alex',
  'email' => 'alex@example.com'
];
$users[1] = [
  'name' => 'George',
  'email' => 'george@domain.ru',
  'additionalData' => 'Всем привет!'
];
$users[3] = [
  'name' => 'Michael',
  'email' => 'mich@test.com'
];
?>
```

С таким массивом удобно работать, если знаешь, какие ключи у его элементов. Но что делать в противном случае?

Применение циклов для работы с массивами

Если мы хотим обработать каждый элемент массива, то мы совершаем так называемый обход массива в цикле. Самый простой способ обойти массив – использовать цикл for:

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

Всё очень здорово ровно до тех пор, пока мы не начинаем работу с ассоциативными массивами. Тогда числовой счётчик становится неприменим. Для обхода массивов произвольных типов существует специальный оператор цикла - foreach.

```php
<?php
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. Но элемента $key может и не быть. Иногда достаточно работы только со значением.

```
<?php
foreach ($myArray as $my_key => $my_value)
{
  echo("$my_key is $my_value <br/>");
}
?>
```

А как же быть с массивом, который мы определили, как смешанный? Для такого массива используются вложенные циклы. Поскольку на верхнем уровне в нём индексный массив, то можно было бы использовать цикл for. Однако видно, что элемента с индексом 2 в нём нет, а это значит, что for выбросит предупреждение. Таким образом, мы можем использовать вложенные foreach:

```
<?php
foreach ($myArray as $my_value)
{
  foreach ($my_value as $internal_value){
    …
  }
}
?>
```

Основные функции работы с массивами

PHP имеет очень много встроенных функций, облегчающих работу с массивами. Ряд полезных функций мы обсудим сейчас. Полный список можно найти в электронной документации.

count()

```
count()
int count(mixed var)
```

Функция count() принимает в качестве аргумента массив и возвращает количество элементов в нём. Если переменная не установлена или не содержит элементов, возвращается ноль. Очень часто эта функция применяется для ограничения циклов.

in_array()

```
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()

```
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:

```
<?php
// Превратить массив в строку, с разделителем - точкой с запятой:
$sLangString = implode(';', $aLanguages);
echo($sLangString);
$sSentence = 'Per aspera ad astra';
// Превратить предложение в массив отдельных слов:
$aWords = explode(' ', $sSentence);
?>
```

Суперглобальные массивы

Суперглобальные массивы — массивы, имеющиеся в самом PHP, которые можно использовать в любой области видимости. Как правило, эти массивы содержат переменные окружения, веб-сервера, переданные пользователем данные.
Всего их 9:

1. $GLOBALS - содержит ссылки на все переменные глобальной области видимости.
2. $_SERVER - информация о сервере и среде исполнения.
3. $_GET - GET-переменные HTTP. В массив $_GET приходят параметры, перечисленные в адресной строке после знака ? и разделённые знаком &. (localhost/index.php?param1=somevalue&param2=hello&test=world)
4. $_POST - ассоциативный массив данных, переданных скрипту через HTTP метод POST.
5. $_FILES - переменные файлов, загруженных по HTTP.
6. $_COOKIE - ассоциативный массив значений, переданных скрипту через HTTP Куки.
7. $_SESSION - ассоциативный массив, содержащий переменные сессии, которые доступны для текущего скрипта.
8. $_REQUEST - ассоциативный массив, который по умолчанию содержит данные переменных $_GET, $_POST и $_COOKIE.
9. $_ENV - переменные окружения.

Домашнее задание

1. С помощью цикла while вывести все числа в промежутке от 0 до 100, которые делятся на 3 без остатка.
2. С помощью цикла do…while написать 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. Объявить массив, в котором в качестве ключей будут использоваться названия областей, а в качестве значений – массивы с названиями городов из соответствующей области.

Вывести 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)

4. Объявить 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’).

Написать функцию транслитерации строк.

5. Написать функцию, которая заменяет в строке пробелы на подчеркивания и возвращает видоизмененную строчку.
6. В имеющемся шаблоне сайта заменить статичное меню (ul - li) на генерируемое через PHP. Необходимо представить пункты меню как элементы массива и вывести их циклом. Подумать, как можно реализовать меню с вложенными подменю? Попробовать его реализовать.
7. *Вывести с помощью цикла for числа от 0 до 9, НЕ используя тело цикла. То есть выглядеть должно так:

```
for(…){// здесь пусто}
```

8. *Повторить третье задание, но вывести на экран только города, начинающиеся с буквы «К».
9. *Объединить 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).

Дополнительные материалы

1. https://ru.wikipedia.org/wiki/Цикл_(программирование)
2. http://php.net/manual/ru/reserved.variables.globals.php

Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. Котеров Д.: PHP 5 в подлиннике.
2. Head First PHP and MySQL.