Лабораторная работа №3

# Основы языка веб-программирования PHP. Базовые элементы языка (продолжение)

## Введение

В данной работе продолжают рассматриваться основные элементы языка PHP – массивы и функции пользователя. В ходе выполнения работы требуется создать новый сайт с примерами использования основных конструкций языка PHP, приведенными в данном описании, а также с результатом самостоятельного решения контрольного варианта задания. Для создания страниц учебного сайта необходимо использовать редактор исходных текстов ***SciTE*** и комплект веб-разработчика ***Denwer***.

## 1 Константы

В отличие от переменных, значение константы устанавливается единожды и не подлежит изменению. Константы не начинаются с символа $ и определяются с помощью оператора *define*:

```
1.  <?
2.    define ('MY_NAME', 'Вася');
3.
4.    echo 'Меня зовут ' . MY_NAME;
5.  ?>
```

Константы необязательно называть прописными буквами, но это общепринятое (и удобное) соглашение.

Поскольку имя константы не начинается с какого-либо спецсимвола, внутри двойных кавычек значение константы поместить невозможно (так как нет возможности различить, где имя константы, а где - просто текст).

## 2 Массивы

Массив представляет собой набор переменных, объединенных одним именем. Каждое значение массива идентифицируется индексом, который указывается после имени переменной-массива в квадратных скобках. Комбинацию индекса и соответствующего ему значения называют элементом массива.

```
1.  <?
2.    $i = 1024;
3.    $a[1] = 'abc';
4.    $a[2] = 100;
```

```
5.    $a['test'] = $i - $a[2];
6.
7.    echo $a[1] . "<br>\n";
8.    echo $a[2] . "<br>\n";
9.    echo $a['test'] . "<br>\n";
10  ?>
```

В приведенном примере, в строке три объявляется элемент массива $a с индексом 1; элементу массива присваивается строковое значение 'abc'. Этой же строкой объявляется и массив $a, так как это первое упоминание переменной $a в контексте массива, массив создается автоматически. В строке 4 элементу массива с индексом 2 присваивается числовое значение 100. В строке же 5 значение, равное разности $i и $a[2], присваивется элементу массива $a со строковым индексом 'test'.

Как видите, индекс массива может быть как числом, так и строкой.

В других языках программирования (например, Perl) массивы, имеющие строковые индексы, называются хэшами (hash), и являются отдельным типом данных. В PHP же, по сути, все массивы являются хэшами, однако индексом может служить и строка, и число.

В предыдущем примере массив создавался автоматически при описании первого элемента массива. Но массив можно задать и явно:

```
1.  <?
2.    $i = 1024;
3.    $a = array( 1 => 'abc',  2 => 100,  'test' => $i-100 );
4.    print_r($a);
5.  ?>
```

Созданный в последнем примере массив $a полностью аналогичен массиву из предыдущего примера. Каждый элемент массива здесь задается в виде ***индекс => значение***. При создании элемента 'test' пришлось указать значение 100 непосредственно, так как на этот раз мы создаем массив "одним махом", и значения его элементов на этапе создания неизвестны PHP.

В строке 4 для вывода значения массива мы воспользовались функцией *print_r()*, которая очень удобна для вывода содержимого массивов на экран - прежде всего, в целях отладки.

Строки в выводе функции *print_r* разделяются обычным переводом строки \n, но не тэгом <br>. Для удобства чтения, строку *print_r(..)* можно окружить операторами вывода тэгов <pre>...</pre>:

```
echo '<pre>';
print_r($a);
echo '</pre>';
```

Если явно не указывать индексы, то здесь проявляется свойство массивов PHP, характерное для числовых массовов в других языках: очередной элемент будет иметь порядковый числовой индекс. Нумерация начинается с нуля. Пример:

```
1.  <?
2.    $os = array( 'Windows', 'Linux', 'FreeBSD', 'OS/2');
3.    $os[] = 'MS-DOS';
4.
5.    echo "<pre>";
6.    print_r($os);
7.    echo "</pre>";
8.  ?>
```

Вывод:

```
Array
(
    [0] => Windows
    [1] => Linux
    [2] => FreeBSD
    [3] => OS/2
    [4] => MS-DOS
)
```

Здесь мы явно не указывали индексы: PHP автоматически присвоил числовые индексы, начиная с нуля. При использовании такой формы записи массив можно перебирать с помощью цикла *for*. Количество элементов массива возвращает оператор *count* (или его синоним, *sizeof*):

```
1.  <?
2.    $os = array( 'Windows', 'Linux', 'FreeBSD', 'OS/2');
3.    $os[] = 'MS-DOS';
4.
5.    echo '<table border=1>';
6.    for ($i=0; $i<count($os); $i++) {
7.         echo '<tr><td>' . $i . '</td><td>' .
8.              $os[$i] . '</td></tr>';
9.    }
10.   echo '</table>';
11. ?>
```

Стили записи можно смешивать. Обратите внимание на то, какие индексы автоматически присваиваются PHP после установки некоторых индексов вручную.

```
 1.  <?
 2.    $languages = array(
 3.      1 => 'Assembler',
 4.      'C++',
 5.      'Pascal',
 6.      'scripting' => 'bash'
 7.    );
 8.    $languages['php'] = 'PHP';
 9.    $languages[100] = 'Java';
10.    $languages[] = 'Perl';
11.
12.    echo "<pre>";
13.    print_r($languages);
14.    echo "</pre>";
15.  ?>
```

Вывод:

```
Array
(
    [1] => Assembler
    [2] => C++
    [3] => Pascal
    [scripting] => bash
    [php] => PHP
    [100] => Java
    [101] => Perl
)
```

## 2.1 Цикл foreach

Массив, подобный предыдущему, перебрать с помощью for затруднительно. Для перебора элементов массива предусмотрен специальный цикл ***foreach***:

```
 1.  <?
 2.    $languages = array(
 3.      1 => 'Assembler',
 4.      'C++',
 5.      'Pascal',
 6.      'scripting' => 'bash'
 7.    );
 8.    $languages['php'] = 'PHP';
 9.    $languages[100] = 'Java';
10.    $languages[] = 'Perl';
11.  ?>
12.  <table>
```

```
13.  <tr>
14.    <th>Индекс</th>
15.    <th>Значение</th>
16.  </tr>
17.  <?
18.    foreach ($languages as $key => $value) {
19.      echo '<tr><td>' . $key .
20.           '</td><td>' . $value . '</td></tr>';
21.    }
22.  ?>
23.  </table>
```

Этот цикл работает следующим образом: в порядке появления в коде программы элементов массива *$languages*, переменным *$key* и *$value* присваиваются соответственно индекс и значение очередного элемента, и выполняется тело цикла.

Если индексы нас не интересуют, цикл можно записать следующим образом: `foreach ($languages as $value)`.

## 2.2 Конструкции *list* и *each*

В дополнение к уже рассмотренной конструкции *array*, существует дополняющая ее конструкция *list*, являющаяся своего рода антиподом *array*: если последняя используется для создания массива из набора значений, то *list*, напротив, заполняет перечисленные переменные значениями из массива.

Допустим, у нас есть массив `$lang = array('php', 'perl', 'basic')`. Тогда конструкция `list($a, $b) = $lang` присвоит переменной $a значение 'php', а $b - 'perl'. Соответственно, `list($a, $b, $c) = $lang` дополнительно присвоит $c = 'basic'.

Если бы в массиве $lang был только один элемент, PHP бы выдал замечание об отсутствии второго элемента массива.

А если нас интересуют не только значения, но и индексы? Воспользуемся конструкцией each, которая возвращает пары индекс-значение.

```
 1.  <?
 2.    $browsers = array(
 3.      'MSIE' => 'Microsoft Internet Explorer 6.0',
 4.      'Gecko' => 'Mozilla Firefox 0.9',
 5.      'Opera' => 'Opera 7.50'
 6.    );
 7.
 8.    list($a, $b) = each($browsers);
 9.    list($c, $d) = each($browsers);
10.    list($e, $f) = each($browsers);
```

```
11.    echo $a.':'.$b."<br>\n";
12.    echo $c.':'.$d."<br>\n";
13.    echo $e.':'.$f."<br>\n";
14.  ?>
```

На первый взгляд может удивить тот факт, что в строках 8-10 переменным присваиваются разные значения, хотя выражения справа от знака присваивания совершенно одинаковые. Дело в том, что у каждого массива есть скрытый указатель текущего элемента. Изначально он указывает на первый элемент. Конструкция each же продвигает указатель на один элемент вперед.

Эта особенность позволяет перебирать массив с помощью обычных циклов *while* и *for*. Конечно, ранее рассмотренный цикл *foreach* удобнее, и стоит предпочесть его, но конструкция с использованием *each* довольно распространена, и вы можете ее встретить во множестве скриптов в сети.

```
1.  <?
2.    $browsers = array(
3.      'MSIE' => 'Microsoft Internet Explorer 6.0',
4.      'Gecko' => 'Mozilla Firefox 0.9',
5.      'Opera' => 'Opera 7.50'
6.    );
7.
8.    while (list($key,$value)=each($browsers)) {
9.      echo $key . ':' . $value . "<br>\n";
10.   }
11.
12.  ?>
```

После завершения цикла, указатель текущего элемента указывает на конец массива. Если цикл необходимо выполнить несколько раз, указатель надо принудительно сбросить с помощью оператора *reset: reset($browsers)*. Этот оператор устанавливает указатель текущего элемента в начало массива.

## 2.3 Многомерные массивы

Поскольку значение массива может быть чем угодно, им также может быть другой массив. Таким образом вы можете создавать рекурсивные и многомерные массивы.

```
<?
$fruits = array ( "фрукты" => array ( "a" => "апельсин",
                                      "b" => "банан",
                                      "c" => "яблоко"
                                    ),
                  "числа"  => array ( 1,
                                      2,
```

```
                                3,
                                4,
                                5,
                                6
                            ),
                "дырки"  => array (      "первая",
                                5 => "вторая",
                                     "третья"
                            )
            );

// Несколько примеров доступа к значениям предыдущего массива
echo $fruits["дырки"][5];    // напечатает "вторая"
echo $fruits["фрукты"]["a"]; // напечатает "апельсин"
unset($fruits["дырки"][0]);  // удалит "первая"

// Создаст новый многомерный массив
$juices["яблоко"]["зеленое"] = "хорошее";
?>
```

Обратите внимание, что при присваивании массива всегда происходит копирование значения. Чтобы копировать массив по ссылке, вам нужно использовать оператор ссылки.

```
<?
     $arr1 = array(2, 3);
     $arr2 = $arr1;
     $arr2[] = 4; // $arr2 изменился,
                  // $arr1 по прежнему array(2,3)

     $arr3 = &$arr1;
     $arr3[] = 4; // теперь $arr1 и $arr3 эквивалентны
?>
```

В данной работе рассмотрены только основы работы с массивами. В PHP существует множество разнообразных функций работы с массивами; их подробное описание находится в соответствующем разделе документации.

# 3 Функции, определяемые пользователем

Функции являются одной из фундаментальных конструкций PHP и предназначены для многократного использования повторяющихся программных блоков. Вместо того чтобы дублировать код, его можно определить в виде функции и вызывать ее по мере необходимости, передавая функции разные аргументы.

Функция это процедура PHP — набор операторов, выполняющих определённую задачу. Чтобы использовать функцию необходимо сначала определить её; после чего скрипт может её вызвать.

Функции объявляются с помощью оператора *function*, после которого

идет желаемое имя. Имя функции может быть любым, но на него распространяются те же правила, что на идентификаторы.

```
function имя_функции (аргументы) {

   тело функции

}
```

Аргументы, если они необходимы, заключаются в скобки и перечисляются через запятую. Само тело функции обязательно должно размещаться между фигурными скобками:

```
<?
     function square($number) {
          return $number * $number;
     }
?>
```

Функция *square* принимает один аргумент — *$number*. Функция состоит из одного оператора, который возвращает квадрат аргумента функции. Оператор *return* специфицирует значение, возвращаемое функцией.

Определение функции ещё не вызывает её выполнения. Определение функции просто именует её и специфицирует действия функции при её вызове. Вызов функции выполняет специфицированные действия с указанными параметрами, например, для определенной ранее функции:

```
square(5)
```

Здесь функция вызывается с аргументом 5. Функция выполняет свои операторы и возвращает значение 25.

Функция может быть рекурсивной, то есть может вызывать сама себя. Например, функция вычисления факториала:

```
function factorial($n) {
   if ($n > 1)
      return $n * factorial($n-1);
   return 1
   }
```

Внутри функции можно использовать любой корректный PHP-код, в том числе другие функции и даже объявления классов.

В PHP 3 функции должны были быть определены прежде, чем они будут использованы. Начиная с PHP 4 такого ограничения нет, исключая тот случай, когда функции определяются условно.

PHP не поддерживает перегрузку функции, также отсутствует возможность переопределить или удалить объявленную ранее функцию.

Имена функций регистронезависимы, тем не менее, более предпочтительно вызывать функции так, как они были объявлены.

## 3.1 Аргументы функции

Функция может принимать информацию в виде списка аргументов, который является списком разделенных запятыми переменных и/или констант.

PHP поддерживает передачу аргументов по значению (по умолчанию), передачу аргументов по ссылке, и значения по умолчанию. Списки аргументов переменной длины поддерживаются, начиная с PHP 4.

### 3.1.1 Передача аргументов по ссылке

По умолчанию аргументы в функцию передаются по значению (это означает, что если вы измените значение аргумента внутри функции, то вне ее значение все равно останется прежним). Если вы хотите разрешить функции модифицировать свои аргументы, вы должны передавать их по ссылке.

Если вы хотите, что бы аргумент всегда передавался по ссылке, вы можете указать амперсанд (&) перед именем аргумента в описании функции:

```
<?
     function add_some_extra(&$string) {
         $string .= 'and something extra.';
     }
     $str = 'This is a string, ';
     add_some_extra($str);
     echo $str;// Выведет 'This is a string, and something extra.'
?>
```

### 3.1.2 Значения аргументов по умолчанию

Функция может определять значения по умолчанию в стиле C++ для скалярных аргументов, например:

```
<?
     function makecoffee($type = "cappuccino") {
         return "Making a cup of $type.\n";
     }
     echo makecoffee();
     echo makecoffee("espresso");
?>
```

Результат работы приведенного выше кода будет таким:

```
Making a cup of cappuccino.
```

```
Making a cup of espresso.
```

Обратите внимание, что все аргументы, для которых установлены значения по умолчанию, должны находится правее аргументов, для которых значения по умолчанию не заданы, в противном случае ваш код может работать не так, как вы этого ожидаете.

### 3.1.3 Списки аргументов переменной длины

PHP 4 и выше поддерживает списки аргументов переменной длины для функций, определяемых пользователем. Реализация этой возможности достаточно прозрачна и заключается в использовании функций *func_num_args()*, *func_get_arg()* и *func_get_args()*.

Необходимости в специфическом синтаксисе нет, при этом список аргументов также может быть указан явно и будет обладать тем же поведением.

## 3.2 Возврат значений

Значения возвращаются при помощи необязательного оператора возврата. 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. Для получения более детальной информации ознакомьтесь с описанием функции ***return()***.

Функция не может возвращать несколько значений, но аналогичного результата можно добиться, возвращая массив.

```
<?
     function small_numbers() {
          return array (0, 1, 2);
     }
     list ($zero, $one, $two) = small_numbers();
?>
```

## 3.3 Обращение к функциям через переменные

PHP поддерживает концепцию переменных функций. Это означает, что если к имени переменной присоединены круглые скобки, PHP ищет функцию с тем же именем, что и результат вычисления переменной, и пытается ее выполнить. Эту возможность можно использовать для реализации обратных вызовов, таблиц функций и множества других вещей.

Переменные функции не будут работать с такими языковыми конструкциями как *echo()*, *print()*, *unset()*, *isset()*, *empty()*, *include()*, *require()* и другими подобными им операторами.

```
<?php
     function foo() {
          echo "In foo()<br />\n";
     }

     function bar($arg = '') {
          echo "In bar(); argument was '$arg'.<br />\n";
     }

     $func = 'foo';
     $func();        // Вызывает функцию foo()

     $func = 'bar';
     $func('test');  // Вызывает функцию bar()
?>
```

# 4 Практическая часть

## 4.1 Общая часть задания

1) Задайте двумерный массив (хэш), содержащий сведения о студентах группы (№пп, ФИО, возраст, пол).

2) Создайте скрипт, выводящий все элементы массива в виде HTML–таблицы (используйте для этого цикл *foreach)*:

| № пп | ФИО | возраст | пол |
|---|---|---|---|
| | | | |
| | | | |

3) дополните программу скрипта в соответствии с требованиями индивидуального варианта задания.

## 4.2 Варианты индивидуальных заданий

1) Создайте функцию поиска в массиве студентов самого молодого студента.

2) Напишите функцию формирования списка всех ребят группы.

3) Реализуйте функцию поиска в массиве информации о студенте с заданной фамилией.

4) Напишите функцию формирования списка студентов, возраст которых находится в заданных пределах.

5) Напишите функцию, подсчитывающую процент студентов и студенток в группе.

6) Создайте функцию, формирующую список студентов в прямом или обратном порядке, в зависимости от требований пользователя.

7) Напишите программу, формирующую список в котором строки, соответствующие ребятам, отображались бы синим цветом, а соответствующие девчонкам — зеленым.