Часть II

Описание языка

В этой части...

ЧАС 4

Строительные блоки

На этом занятии...

- Переменные
- Типы данных
- Операторы и выражения
- Константы

В этом часе вам придется замарать свои руки “кирпичами” и “цементом” языка PHP. Это я к тому, что вам нужно освоить слишком много материала сразу, и новичок в программировании может почувствовать себя неуютно под лавиной информации. Но не стоит беспокоиться — всегда можно перечитать эту главу позже. Следует уделять больше внимания пониманию, а не запоминанию описываемых возможностей.

Опытному программисту желательно, по крайней мере, бегло ознакомиться с материалом этого занятия. Он посвящен нескольким специфическим для PHP особенностям.

Переменные

Переменная — это специальный контейнер, который содержит значение согласно определению пользователя. Переменные являются основой программирования. Без них пришлось бы “жестко” закодировать все числовые значения в своих сценариях. Например, при сложении двух чисел и печати результата получается что-то полезное:

```
print (2 + 4);
```

Однако этот сценарий полезен только для тех, кто хочет знать сумму чисел 2 и 4. Чтобы выйти из положения, можно написать сценарий нахождения суммы двух других чисел, скажем 3 и 5. Этот подход к программированию явно абсурден. Переменные дают возможность создавать шаблоны для операций (сложение двух чисел, например), не заботясь о том, какие именно значения они содержат. Таким образом, переменные определяют значения, используемые при выполнении сценария, которые непосредственно ввел пользователь или которые были получены в результате выполнения запроса к базам данных.

Переменные следует использовать в сценарии, если предполагается, что данные, над которыми выполняются какие-либо операции, изменяются при каждом запуске сценария, либо даже во время выполнения одного сценария.

Переменная определяется именем, выбираемым произвольно, перед которым ставится знак доллара (`$`). Имя переменной может состоять из букв, цифр и символа подчеркивания (_). Имя переменной не может содержать пробелов или каких-либо других символов, и оно должно начинаться с буквы или символа подчеркивания. В следующем примере кода определяются некоторые допустимые переменные.

```
$a;
$a_longish_variable_name;
$_2453;
$sleepyZZZZ;
```

Не забывайте, что для окончания команды PHP-сценария используется точка с запятой (`;`). В предыдущем фрагменте кода точка с запятой не является частью имени переменной.

В **переменной** хранятся данные определенного типа. Она может содержать числа, строки символов, объекты, массивы или логические величины. Значение (или содержание) переменной можно изменить в любой момент.

Как видите, существует огромное количество имен переменных. Для объявления переменной нужно всего лишь включить ее в свой сценарий. Обычно в одной и той же команде объявляют переменную и присваивают ей значение, например:

```
$num1 = 8;
$num2 = 23;
```

В этих двух строках объявляются две переменные, и им назначаются значения с помощью оператора присваивания (`=`). Подробнее операция присваивания описана в разделе “Операторы и выражения” этого часа. После присвоения переменной определенного значения с ней можно работать точно так же, как если бы это было само значение. Другими словами,

```
print $num1;
```

эквивалентно

```
print 8;
```

при условии, что в переменной `$num1` содержится число `8`.

Типы данных

Для размещения в памяти данных разного типа требуется определенное место в памяти, а при обращении к ним из сценария они обрабатываются также по-разному. Поэтому в большинстве языков программирования требуется, чтобы программист заранее объявил, какие типы данных будет содержать переменная. В отличие от них, язык PHP **слабо типизирован**. Это означает, что в нем типы данных определяются по мере присваивания их каждой переменной. Как известно, такой подход имеет как свои преимущества, так и определенные недостатки. С одной стороны, это означает, что переменную можно использовать достаточно гибко, сохраняя в одном месте сценария, скажем, строку, а в другом — целое число. С другой стороны, подобный подход может приводить к возникновению ошибок в больших сценариях. Программист может предполагать, что переменной присвоено значение одного типа, а на самом деле в ней содержится что-то совсем другое. Например, можно создать код, предназначенный для работы с переменной-массивом. Если вдруг окажется, что эта переменная вместо массива будет содержать числовое значение, то при попытке выполнить над этой переменной операции, определенные для массива, возникнет ошибка в сценарии.

В табл. 4.1 приведены стандартные типы данных PHP.

Таблица 4.1. Стандартные типы данных

Тип	*Пример*	*Описание*
`Integer`	`5`	Целое число
`Double`	`3.234`	Число с плавающей точкой
`String`	`"Привет"`	Строка символов
`Boolean`	`true`	Одно из логических значений `true` или `false`
`Object`	Объект	См. час 9, “Объекты”
`Array`	Массив	См. час 7, “Массивы”

Рассмотрение массивов и объектов мы отложим до 7-го и 9-го занятий. В PHP также предусмотрено два специальных типа данных, перечисленных в табл. 4.2.

Таблица 4.2. Специальные типы данных

Тип	*Описание*
`Resource`	Ссылка на ресурс стороннего производителя (например, базу данных)
`NULL`	Неинициализированная переменная

Данные типа ресурс часто возвращаются функциями, работающими с внешними приложениями или файлами. Вы еще столкнетесь с примерами переменных-ресурсов на страницах этой книги. Тип `NULL` резервируется для переменных, которые не были инициализированы (т.е. им еще не было присвоено значение).

Для определения типа любой переменной можно использовать встроенную функцию `gettype()`. Если поместить имя переменной в скобки при вызове функции,

то она возвратит строку, описывающую тип этой переменной. В листинге 4.1 одной и той же переменной назначаются пять различных типов данных с последующей их проверкой с помощью функции `gettype()`.

Между прочим Подробнее о вызовах функций мы поговорим в часе 6, “Функции”.

Листинг 4.1. Отображение типа переменной

```
<!DOCTYPE html PUBLIC
  "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html>
  <head>
    <title>Листинг 4.1. Отображение типа переменной /title>
  </head>

<body>
 <div>
  <?php
      $testing;  // Объявление переменной без присваивания
      print gettype( $testing ); // NULL
      print "<br />";

      $testing = 5;
      print gettype( $testing ); // integer
      print "<br />";

      $testing = "five";
      print gettype( $testing ); // string
      print "<br />";

      $testing = 5.0;
      print gettype( $testing ); // double
      print "<br />";

      $testing = true;
      print gettype( $testing ); // boolean
      print "<br />";

  ?>
  </div>
</body>
</html>
```

Этот сценарий выводит следующее:

```
NULL
integer
string
```

```
double
boolean
```

При объявлении в строке 13 переменной `$testing`, значение ей не присваивается. Таким образом, когда в строке 14 эта переменная используется впервые в функции `gettype()` для проверки ее типа, выводится строка `"NULL"`. После этого мы назначаем переменной `$testing` несколько значений с помощью оператора присваивания `=` и каждый раз вызываем функцию `gettype()` для отображения ее типа. В строке 18 переменной `$testing` присваивается значение `5`, соответствующее *целому типу* (`integer`). На самом деле это число может также являться и вещественным числом. Поэтому в PHP целыми считаются числа без десятичной точки. В строке 22 переменной `$testing` присваивается текстовая строка `"five"`, соответствующая *строковому типу* (`string`). При работе в сценариях со строками их следует всегда заключать в двойные (`"`) или в одинарные кавычки (`'`). В строке 26 переменной `$testing` присваивается число с плавающей точкой `5.0`, соответствующее типу `double`. В строке 30 переменной `$testing` присваивается значение `true`, соответствующее *логическому типу* (`boolean`). Заметьте, что в строке 30 переменной `$testing` можно присвоить одно из двух специальных значений: `true` или `false`.

Между прочим

Существует разница между двойными и одинарными кавычками при использовании их со строками. Двойные кавычки позволяют анализировать переменные (т.е. подставлять вместо имен переменных их значения). Так, если заключить имя переменной в двойные кавычки, то интерпретатор PHP подставит вместо него значение переменной, например:

```
$name = "john";
print "hello, $name"; // hello, john
```

Если во второй строке этого сценария заменить двойные кавычки одинарными, то значение переменной `$name` не подставляется:

```
print 'hello, $name'; // hello, $name
```

Строки, заключенные в двойные кавычки, также анализируются на наличие управляющих символов . Такие символы начинаются с обратной косой черты (`\`), за которой следует какой-либо знак. Основными среди управляющих символов являются следующие:

- `\n` — символ новой строки;
- `\t` — табуляция;
- `\"` — символ двойной кавычки, находящийся в строке, заключенной в двойные кавычки;
- `\\` — обратная косая черта;
- `\$` — знак доллара (чтобы не путать его с признаком начала имени переменной).

На практике, если нужно отобразить строку точно так же, как она была введена, используйте одинарные кавычки. При этом сценарий будет работать быстрее, поскольку интерпретатору не придется анализировать эту строку. Если в строке предполагается использовать управляющие символы и выполняется подстановка значений переменных, следует пользоваться двойными кавычками.

Между прочим

До PHP 4 логического типа данных не существовало. И хотя значение `true` использовалось, в действительности оно считалось константой (специальный вид переменной, который мы опишем далее в этом часе) с целочисленным значением `1`.
Оба типа — `NULL` и `Resource` — появились в PHP 4.

Отображение информации о типах с помощью функции `var_dump()`

Функция `gettype()` — это специфичное средство. Она возвращает только строку, соответствующую типу переменной. В отличие от нее, функция `var_dump()` выводит как тип переменной, так и ее значение. Более того, для сложных типов переменных, таких как массивы или объекты, функция `var_dump()` выводит информацию обо всех типах, содержащихся в этой переменной, а также о самой этой переменной.

Таким образом, изменяя строку 18 в листинге 4.1, мы можем протестировать работу функции `var_dump()`:

```
$testing = 5;
var_dump( $testing );
```

Этот фрагмент дает следующий результат:

```
int(5)
```

Такая запись говорит о том, что переменная `$testing` содержит целое число и что его значение равно `5`. Отметим, что для функции `var_dump()` нет необходимости использовать ключевое слово `print`. Дело в том, что эта функция выводит свои результаты прямо в браузер или в командную строку.

Проверка специфических типов данных

Функция `gettype()` полезна при отладке сценария, потому что она позволяет точно узнать тип любой переменной. Однако часто бывает нужно только проверить, содержит ли переменная значение некоторого типа. В PHP для этого предусмотрен набор специальных функций, соответствующих каждому типу данных. Этим функциям передаются имена переменных или значения, а возвращают они значение логического (`boolean`) типа (`true` или `false`). Эти функции перечислены в табл. 4.3.

Таблица 4.3. Функции для проверки типов данных

Функция	*Описание*
`is_array()`	Возвращает `true`, если аргумент массив (`array`)
`is_bool()`	Возвращает `true`, если аргумент логического типа (`boolean`)
`is_double()`	Возвращает `true`, если аргумент вещественное число двойной точности (`double`)

Окончание табл. 4.3

Функция	*Описание*
`is_int()`	Возвращает `true`, если аргумент целое число (`integer`)
`is_object()`	Возвращает `true`, если аргумент объект (`object`)
`is_string()`	Возвращает `true`, если аргумент строка (`string`)
`is_null()`	Возвращает `true`, если аргумент неинициализирован (`null`)
`is_resource()`	Возвращает `true`, если аргумент ресурс (`resource`)

В часе 5, "Управление процессом выполнения программы", будет описан оператор `if`, который позволяет изменить порядок выполнения команд сценария в зависимости от результатов проверки. Поэтому перечисленные выше функции часто используются вместе с оператором `if` для проверки типов переменных, передаваемых функциям или методам объекта.

Изменение типа с помощью функции `settype()`

Для изменения типа переменной в PHP предусмотрена функция `settype()`. При вызове функции `settype()` следует поместить в скобки имя переменной, тип которой нужно изменить, а также описание ее нового типа и разделить их запятыми. В листинге 4.2 показан пример преобразования значения `3.14` типа `double` в четыре других типа, рассматриваемых в этом часе.

Листинг 4.2. Изменение типа переменной с помощью функции `settype()`

```
1:   <!DOCTYPE HTML PUBLIC
2:     "-//W3C//DTD XHTML 1.0 Strict//EN"
3:     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4:   <html>
5:     <head>
6:       <title>Листинг 4.2. Изменение типа переменной
           с помощью функции settype()</title>
7:     </head>
8:
9:    <body>
10:    <div>
11:     <?php
12:       $undecided = 3.14;
13:
14:       print gettype( $undecided );   // double
15:       print " -- $undecided<br />";  // 3.14
16:
17:       settype( $undecided, string );
18:       print gettype( $undecided );   // string
19:       print " -- $undecided<br />";  // 3.14
20:
21:       settype( $undecided, int );
22:       print gettype( $undecided );   // integer
```

```
23:        print " -- $undecided<br />";  // 3
24:
25:        settype( $undecided, double );
26:        print gettype( $undecided );   // double
27:        print " -- $undecided<br />";  // 3.0
28:
29:        settype( $undecided, bool );
30:        print gettype( $undecided);    // boolean
31:        print " -- $undecided<br />";  // 1
32:     ?>
33:    </div>
34:   </body>
35:  </html>
```

Для того чтобы убедиться, что изменение типа произошло, вызывается функция `gettype()`, а затем выводится значение переменной `$undecided` в браузер. Когда мы в строке 21 конвертируем значение `3.14` в целое число, его дробная часть (числа после десятичной точки) отбрасывается. По этой причине значение переменной `$undecided` будет равно `3` после повторного ее преобразования в тип `double` в строке 25. И, наконец, в строке 29 мы преобразуем переменную `$undecided` в логический тип.

При преобразовании в логический тип любое число, отличающееся от `0`, автоматически становится равным `true`. При печати логического типа в PHP значение `true` соответствует `1`, а `false` — пустой строке. Поэтому в строке 31 выводится значение переменной `$undecided`, равное `1`.

Изменение типа с помощью операции приведения

Для того чтобы создать копию значения некоторой переменной и преобразовать ее в указанный тип данных, нужно после знака равно и перед именем исходной переменной поместить в круглых скобках желаемый спецификатор типа.

Принципиальное отличие между функцией `settype()` и явным приведением типа (cast) состоит в том, что вторая операция создает копию исходной переменной заданного типа, не изменяя при этом типа исходной переменной. Это продемонстрировано в листинге 4.3.

Листинг 4.3. Явное приведение типов переменных

```
1:   <!DOCTYPE HTML PUBLIC
2:     "-//W3C//DTD XHTML 1.0 Strict//EN"
3:     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4:
5:   <html>
6:    <head>
7:     <title>Листинг 4.3. Явное приведение типов переменных </title>
8:    </head>
9:
10:   <body>
```

```
11:    <div>
12:     <?php
13:       $undecided = 3.14;
14:
15:       $holder = ( double ) $undecided;
16:       print gettype( $holder );     // double
17:       print " -- $holder<br />";    // 3.14
18:
19:       $holder = ( string ) $undecided;
20:       print gettype( $holder );     // string
21:       print " -- $holder<br />";    // "3.14"
22:
23:       $holder = ( integer ) $undecided;
24:       print gettype( $holder );     // integer
25:       print " -- $holder<br />";    // 3
26:
27:       $holder = ( boolean ) $undecided;
28:       print gettype( $holder );     // boolean
29:       print " -- $holder<br />";    // 1
30:
31:       print gettype( $undecided );  // double
32:       print " -- $undecided<br />"; // 3.14
33:     ?>
34:    </div>
35:   </body>
36:  </html>
```

В этом примере тип переменной `$undecided` не изменяется и повсюду в сценарии остается `double`. Мы проиллюстрировали это в строке 31, используя для вывода типа переменной `$undecided` функцию `gettype()`.

Фактически в результате операции приведения типа создается копия значения переменной `$undecided` указанного типа, которое затем присваивается новой переменной `$holder` в строках 15, 19, 23 и 27. Поскольку мы работаем с копией переменной `$undecided`, ее тип и значение никогда не изменяются, в отличие от строк 17, 21, 25 и 29 листинга 4.2.

Теперь, когда мы уже умеем изменять тип содержимого переменной, используя либо функцию `settype()`, либо операцию явного приведения типов, следует выяснить, для чего это может понадобиться. Конечно, явное приведение типов используется в реальных сценариях не часто, поскольку интерпретатор PHP автоматически выполняет нужное приведение, когда того требует контекст. Однако операция автоматического приведения типа является временной, тогда как в сценарии может понадобиться, чтобы переменная хранила конкретный тип данных постоянно.

Числа, вводимые пользователем в поля формы HTML, передаются в сценарий в виде строк. Если попытаться сложить две строки, содержащие числа, PHP услужливо сначала преобразует эти строки в числа, а затем складывает их. Таким образом:

```
"30см" + "40см"
```

даст в результате целое число `70`. При приведении этих строк к целым числам интерпретатор PHP игнорирует нечисловые символы. Однако вам может потре-

боваться самостоятельно проанализировать введенные пользователем данные. Предположим, что пользователь получил приглашение ввести число. Эту ситуацию можно смоделировать, объявляя переменную и присваивая ей некоторое значение, например:

```
$test = "30см";
```

Данная ситуация напоминает случай, когда пользователь ошибочно указал единицу измерения после числа. Однако, как можно непосредственно убедиться, введенная пользователем информация корректно приводится к числовому виду:

```
$test = (integer) $test;
print "Вы ввели $test см";
```

Другие способы изменения типа

Выше были описаны два способа преобразования типов данных: с помощью функции `settype()` или с помощью операции явного приведения типа. Кроме этого, в PHP предусмотрены функции для преобразования значений в целые числа, числа с плавающей точкой и в строки. Этим функциям передаются значения любого типа, кроме массивов или объектов, и они возвращают преобразованное значение нужного типа. В табл. 4.4 представлен список этих функций.

Таблица 4.4. Функции для преобразования типов данных

Функция	*Описание*
`doubleval()`	Преобразует передаваемое значение в тип с плавающей точкой
`intval()`	Преобразует передаваемое значение в целый тип
`strval()`	Преобразует передаваемое значение в строку символов

Зачем нужно контролировать типы переменных

Для чего в сценарии может понадобиться контролировать типы переменных? При создании сценариев часто приходится обрабатывать данные, полученные из разных источников. Например, в часе 6, “Функции”, рассказано, как создавать функции в сценариях. Функциям может передаваться какая-либо информация из вызывающего кода в виде аргументов. Чтобы функция корректно работала с предоставленными данными, сначала нужно убедиться, что передаваемые данные имеют правильный тип. Например, функция, которой в качестве параметра должен быть передан ресурс, при получении строки работать не будет.

Операторы и выражения

Теперь вы можете присваивать переменным разные значения и даже определять и изменять их типы. Однако любой язык программирования бесполезен, если в нем не будет предусмотрена возможность манипуляции хранимыми данными. Поэтому

для создания нового значения в сценариях используются специальные символы — **операторы**. Величина, на которую действует оператор, называется **операндом**.

Оператор — это символ или набор специальных символов, который выполняет некоторое действие над указанными значениями, в результате чего, как правило, получается новое значение.

Операнд — это значение, используемое при выполнении оператора. Обычно один оператор имеет два операнда.

А теперь объединим два операнда с помощью оператора + и получим новое значение:

```
4 + 5
```

Цифры 4 и 5 — это операнды, на которые действует оператор сложения (+), что дает в результате число 9. Операторы почти всегда размещаются между двумя операндами, хотя далее в этом часе приведено несколько исключений.

Объединение операндов с помощью оператора, дающее результат, называется **выражением**. И хотя в большинстве выражений чаще всего используются операторы, само по себе выражение не обязательно должно содержать какой-либо оператор. По сути в PHP *выражение* определяется как нечто, что можно использовать как значение. К выражениям относятся целые константы, как, например, число 654, переменные (например, `$user`) и функции, такие как `gettype()`. Следовательно, `(4+5)` — это тоже выражение, состоящее из двух выражений и оператора. Если выражение возвращает какое-либо значение, то говорят, что это значение *вычисляется*. Иными словами, когда вычислены все значения некоторого подвыражения, то это выражение может рассматриваться так, как если бы это значение было непосредственно указано в сценарии.

Выражением называют любое сочетание функций, значений и операторов, дающее в результате некоторое значение. На практике, если что-то можно использовать как значение, то это что-то является выражением.

Теперь, когда мы разобрались с принципами, пришло время познакомиться с несколькими простыми операторами PHP.

Оператор присваивания

Вы уже встречались с оператором присваивания при инициализации переменных. Он состоит из единичного символа "=". Оператор присваивания берет значение своего правого операнда и присваивает его левому операнду.

```
$name = "matt";
```

Переменная `$name` теперь содержит строку `"matt"`. Интересно, что эта конструкция тоже является выражением. На первый взгляд может показаться, что оператор присваивания просто изменяет значение переменной `$name` и не возвращает никакого значения. Но, фактически, при использовании в выражении оператора присваивания всегда возвращается значение его правого операнда. Таким образом,

```
print ( $name = "matt" );
```

выводит в браузер строку `"matt"`, а также присваивает значение `"matt"` переменной `$name`.

Арифметические операторы

Арифметические операторы делают в точности то, чего вы от них ожидаете. Они перечислены в табл. 4.5. Оператор сложения прибавляет правый операнд к левому, тогда как оператор вычитания вычитает из левого операнда правый. Оператор деления делит левый операнд на правый, а оператор умножения умножает левый операнд на правый. Оператор получения остатка от деления (деление по модулю) возвращает остаток от деления левого операнда на правый.

Таблица 4.5. Арифметические операторы

Оператор	*Название*	*Пример*	*Результат*
+	Сложение	10+3	13
-	Вычитание	10-3	7
/	Деление	10/3	3.3333333333333
*	Умножение	10*3	30
%	Остаток от деления	10%3	1

Оператор объединения

Оператором объединения (concatenation) служит единичная точка (.). При его использовании подразумевается, что оба операнда являются строками. Он добавляет содержимое правого операнда к левому. Таким образом,

```
"Здравствуй," . " мир!"
```

эквивалентно

```
"Здравствуй, мир!"
```

Независимо от типа данных операндов они считаются строками и результатом операции всегда есть строка. Оператор объединения часто используется в примерах из этой книги, когда нужно добавить некоторую строку к результатам вычисления выражения. Вот пример:

```
$centimeters = 212;
print "Ширина равна " . ($centimeters/100) . " метров";
```

Комбинированные операторы присваивания

Хотя в действительности существует только один оператор присваивания, в PHP предусмотрено еще несколько его вариаций — комбинированных операторов, которые влияют на значение левого оператора, а также возвращают результат. Как правило, до выполнения оператора значение его операндов не изменяется. Однако при использовании комбинированного оператора присваивания это правило нарушается. Данный оператор присваивания состоит из стандартного символа одного из операторов, за которым следует знак равенства. Комбинированные операторы присваивания позволяют в некоторых случаях избежать использования классического оператора присваивания с двумя дополнительными операторами. Например, вместо

```
$x = 4;
$x = $x + 4; // $x теперь равно 8
```

можно написать

```
$x = 4;
$x += 4; // $x теперь равно 8
```

Для каждого из арифметических операторов, а также для оператора объединения существует свой комбинированный оператор присваивания. В табл. 4.6 перечислены некоторые из самых распространенных операторов такого рода.

Таблица 4.6. Некоторые комбинированные операторы присваивания

Оператор	*Пример*	*Эквивалент*
`+=`	`$x += 5`	`$x = $x + 5`
`-=`	`$x -= 5`	`$x = $x - 5`
`/=`	`$x /= 5`	`$x = $x / 5`
`*=`	`$x *= 5`	`$x = $x * 5`
`%=`	`$x %= 5`	`$x = $x % 5`
`.=`	`$x .= " test"`	`$x = $x . " test"`

Каждый из примеров в табл. 4.6 изменяет значение переменной `$x`, используя значение правого операнда.

Операторы сравнения

Операторы сравнения проверяют значения своих операндов. Они возвращают логическое значение `true`, если проверка прошла успешно, и `false` в противном случае. Этот вид выражений особенно полезен в управляющих структурах, таких как операторы `if` и `while`. Они будут описаны в часе 5.

Например, чтобы проверить, что значение переменной `$x` меньше `5`, используется оператор меньше:

```
$x < 5
```

Если `$x` равно `3`, это выражение эквивалентно `true`. Если `$x` равно `7`, то вычисление этого выражения даст `false`.

В табл. 4.7 перечислены операторы сравнения PHP.

Таблица 4.7. Операторы сравнения

Оператор	*Название*	*Возвращает true, если…*	*Пример ($x равно 4)*	*Результат*
`==`	Равенство	Левая часть равна правой	`$x == 5`	`false`
`!=`	Неравенство	Левая часть не равна правой	`$x != 5`	`true`

Окончание табл. 4.7

`===`	Идентичность	Левая часть равна правой и обе они одного типа	`$x === 5`	`false`
`>`	Больше	Левая часть больше правой	`$x > 5`	`false`
`>=`	Больше или равно	Левая часть больше или равна правой	`$x >=4`	`true`
`<`	Меньше	Левая часть меньше правой	`$x <4`	`false`
`<=`	Меньше или равно	Левая часть меньше или равна правой	`$x <= 4`	`true`

Эти операторы чаще всего используются с целыми числами или с числами с плавающей точкой, хотя оператор равенства используется и для сравнения строк.

Между прочим

Начиная с PHP 5 оператор идентичности `===` можно использовать для проверки того, содержат ли две переменные один и тот же объект. В PHP 4 оператор `===` реализован несколько иначе. Он сравнивает свойства двух объектов и возвращает `true, если все свойства совпадают и` оба объекта являются экземплярами одного класса. Подобная работа оператора идентичности в корне отличает две версии PHP при проверке двух различных объектов одного типа, содержащих один и тот же набор свойств. В PHP 4 такие объекты считаются эквивалентными, в то время как в PHP 5 эти объекты друг другу не соответствуют.
Подробно объекты будут рассмотрены в часе 9, “Объекты”, и в часе 17, “Дополнительная информация об объектах”.

Создание сложных проверочных выражений с помощью логических операторов

Логические операторы используются для тестирования логических значений. Оператор “или” (два символа конвейерной обработки (`||`) либо просто символы `or`) возвращает `true`, если либо правый, либо левый операнд содержит `true`. Таким образом,

```
true || false
```

возвращает `true`.

Оператор “и” (два символа амперсанда (`&&`) либо символы `and`) возвращает `true`, только если и правый, и левый операнды равны `true`. Таким образом,

```
true && false
```

возвращает `false`. Однако маловероятно, что вы будете использовать логический оператор для тестирования логических констант. Имеет гораздо больший смысл тестировать два или больше выражений логического типа. Например,

```
($x > 2) && ($x < 15)
```

возвращает `true`, если значение переменной `$x` больше 2, но меньше 15. Скобки показаны только для большей удобочитаемости кода. В табл. 4.8 перечислены все логические операторы PHP.

Таблица 4.8. Логические операторы

Оператор	*Название*	*Возвращает* `true`*, если…*	*Пример*	*Результат*
`||`	ИЛИ	Левая или правая часть равна `true`	`true || false`	`true`
`or`	ИЛИ	Левая или правая часть равна `true`	`true || false`	`true`
`xor`	Исключающее ИЛИ	Левая или правая часть равна `true`, но не обе одновременно	`true xor true`	`false`
`&&`	И	Левая и правая часть равны `true`	`true && false`	`false`
`and`	И	Левая и правая часть равны `true`	`true && false`	`false`
`!`	НЕ	Единственный операнд не равен `true`	`! true`	`false`

Почему предусмотрены две версии операторов И и ИЛИ? Это связано с приоритетом операторов, о чем мы поговорим немного позже.

Операторы инкремента и декремента целой переменной

При написании PHP-сценария часто требуется выполнить инкремент (увеличить на единицу) или декремент (уменьшить на единицу) значения целой переменной. Обычно эти операции используются для изменения значения счетчика цикла. Выше уже было показано два способа, как это сделать. Можно увеличить значение целой переменной `$x` с помощью оператора сложения.

```
$x = $x + 1; // $x увеличено на 1
```

Или, например, так, используя комбинированный оператор присваивания:

```
$x += 1; // $x увеличено на 1
```

В обоих случаях результирующее целое значение присваивается переменной `$x`. Из-за распространенности выражений этого вида в PHP предусмотрены специальные операторы, которые добавляют единицу или вычитают ее от целой переменной и присваивают последней результат операции. Речь идет о постфиксных операторах инкремента и декремента. Постфиксный оператор инкремента состоит из двух символов “плюс”, присоединенных к имени переменной, как показано ниже:

```
$x++; // $x увеличено на 1
```

Эта команда увеличивает значение переменной `$x` на единицу. Используя два символа “минус” по аналогии, можно уменьшить на единицу значение переменной:

```
$x--; // $x уменьшено на 1
```

Если постфиксные операторы инкремента и декремента используются вместе с условным оператором, то операнд модифицируется только *после* окончания проверки:

```
$x = 3;
$x++ < 4; // true
```

В этом примере при выполнении проверки переменная `$x` равна `3` и при сравнении с числом `4` оператор "меньше" возвращает `true`. После выполнения проверки значение переменной `$x` увеличивается на `1`.

В некоторых случаях нужно увеличить или уменьшить значение переменной на `1` *до* выполнения проверки. Для этих целей в PHP предусмотрены префиксные операторы инкремента и декремента. Во многом они аналогичны постфиксным операторам и записываются с помощью двух символов "плюс" или "минус", предшествующих имени переменной:

```
++$x; // $x увеличена на 1
--$x; // $x уменьшена на 1
```

Если префиксные операторы инкремента и декремента используются вместе с условным оператором, то операнд модифицируется *до* выполнения проверки, например:

```
$x = 3;
++$x < 4; // false
```

В этом фрагменте кода значение переменной `$x` увеличивается до выполнения его сравнения с числом `4`.

Приоритет операторов

При использовании операторов интерпретатор PHP обычно вычисляет значения выражений слева направо. Однако для сложных выражений, в которых используются несколько операторов, все немного усложняется. Рассмотрим сначала простой случай:

```
4 + 5
```

Здесь все очевидно: интерпретатор PHP просто добавляет 5 к 4. А как насчет следующего фрагмента?

```
4 + 5 * 2
```

Здесь ситуация неоднозначна. Эта запись может означать, что сначала вычисляется сумма `4` и `5`, а затем полученный результат умножается на `2`, что дает в результате `18`. Она также может означать, что сначала `5` умножается на `2` и к полученному результату добавляется `4`, что дает в результате `14`. Если бы операторы просто выполнялись слева направо, то справедлив был бы первый результат. На деле же в PHP всем операторам назначен различный *приоритет*. Из-за того, что оператор умножения имеет более высокий приоритет, чем оператор сложения, верным будет второй результат.

Заставить интерпретатор PHP выполнить сложение перед умножением можно с помощью скобок:

```
(4 + 5) * 2
```

Какой бы ни был приоритет операторов в сложном выражении, лучше использовать круглые скобки, чтобы сделать код более ясным и избежать скрытых ошибок. В табл. 4.9 перечислены операторы, описанные на этом занятии, в порядке убывания приоритета.

Таблица 4.9. Порядок выполнения некоторых операторов PHP

Оператор
`++ -- (приведение типа)`
`/ * %`
`* -`
`< <= => >`
`== === !=`
`&&`
`||`
`= += -= /= *= %= .=`
`and`
`xor`
`or`

Как видите, оператор `or` имеет более низкий приоритет, чем `||`, а `and` — более низкий, чем `&&`. Это позволяет использовать логические операторы более низкого приоритета для изменения способа вычисления сложного логического выражения. Однако подобный подход не всегда хорош. Следующие два выражения эквивалентны, но второе намного легче читать:

```
$x and $y || $z
$x && ( $y || $z )
```

Как видите приоритет операторов является единственной причиной, по которой в PHP ввели операторы `&&` и `and`. То же самое справедливо и относительно операторов `||` и `or`. Однако чтобы сделать код более понятным и уменьшить количество ошибок из-за неправильной трактовки приоритета выполнения операторов, рекомендуется в сложных выражениях использовать круглые скобки. В примерах из этой книги используются более привычные операторы `||` и `&&`.

Константы

Переменные являются очень удобным средством для хранения данных, потому что в любой момент можно изменить их значение, а также тип сохраняемых данных. Однако если предполагается, что некоторые значения не будут изменяться на протяжении всей работы сценария, удобнее определить константу (или постоянную величину). Для создания константы используется функция PHP `define()`. После определения значение константы изменить невозможно. При вызове функции

define() имя константы и значение, которое ей нужно присвоить, следует поместить в круглые скобки и разделить запятой, например:

```
define( "ИМЯ_КОНСТАНТЫ", 42 );
```

Вновь создаваемой константе можно присвоить либо числовое значение, либо строку символов. По соглашению имя константы пишется прописными буквами. Доступ к значению константы выполняется по ее имени. Никаких знаков доллара в имени константы использовать не нужно. В листинге 4.4 показан пример определения и использования констант.

Листинг 4.4. Определение и использование константы

```
1:  <!DOCTYPE HTML PUBLIC
2:    "-//W3C//DTD XHTML 1.0 Strict//EN"
3:    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4:
5:  <html>
6:    <head>
7:      <title>Листинг 4.4. Определение и использование
      константы </title>
8:    </head>
9:
10:  <body>
11:   <div>
12:     <?php
13:       define ( "USER", "Сергей" );
14:       print "Добро пожаловать, " . USER;
15:     ?>
16:   </div>
17:  </body>
18: </html>
```

Обратите внимание на строку 14 этого сценария. В ней использован оператор объединения для добавления значения, содержащегося в константе, к строке "Добро пожаловать, ". Дело в том, что интерпретатор PHP не различает константы и строки, находящиеся в кавычках.

У функции define() существует и третий (необязательный) параметр, определяющий, должно ли имя константы зависеть от регистра букв. По умолчанию константы чувствительны к регистру, но, передавая значение true в функцию define(), можно изменить это соглашение. Таким образом, если определить константу USER так:

```
define( "USER", "Сергей", true );
```

то все приведенные ниже выражения эквиваленты.

```
print User;
print usEr;
print USER
```

Поступая подобным образом, вы облегчите жизнь и себе и другими людям, которые впоследствии будут сопровождать ваш сценарий. Им не нужно будет точно запоминать регистр символов для констант, которые были определены в сцена-

рии. С другой стороны, сам факт, что часть констант зависит от регистра символов, а часть — нет, может сбить с толку любого нормального программиста. Поэтому, если у вас нет веских причин поступать иначе, старайтесь всегда определять имена констант в верхнем регистре и делать их зависимыми от регистра символов. Подобное соглашение по именованию констант запомнить легче всего.

Предопределенные константы

В PHP автоматически определяются несколько встроенных констант, которые вы можете использовать в собственных сценариях. Например, константа `__FILE__` содержит имя файла, с которым в настоящий момент работает интерпретатор PHP, а `__LINE__` — номер текущей строки этого файла. Эти константы полезны для формирования сообщений об ошибках. Также с помощью константы `PHP_VERSION` можно выяснить, какая версия интерпретатора PHP обрабатывает ваш сценарий. Тем самым можно ограничить выполнение сценария определенными версиями интерпретатора PHP.

Резюме

На этом занятии были рассмотрены основные свойства языка PHP. Вы изучили переменные и способы присваивания им значений с помощью оператора присваивания; ознакомились с операторами и узнали, как использовать их в выражениях. Наконец, вы узнали, как определять константы и использовать их в сценариях.

Теперь, освоив основы PHP, в следующем часе вы действительно сядете за руль автомобиля. Вы изучите, как с помощью переменных, выражений и операторов пишутся сценарии, принимающие решения и выполняющие повторяющиеся действия.

Вопросы и ответы

Почему иногда полезно знать тип данных, которые содержит переменная?

Часто от типа данных переменной зависят выполняемые над нею действия. Например, прежде чем использовать переменную в математических вычислениях, нужно убедиться, что она содержит целое число или число с плавающей точкой.

Должен ли я придерживаться каких-либо соглашений относительно имен переменных?

Вашей целью всегда должно быть написание кода, легкого для чтения и понимания. Переменная наподобие `$ab12345` ничего не скажет об ее роли в сценарии, кроме того, при ее наборе легко ошибиться. Придерживайтесь коротких и наглядных имен.

С другой стороны, переменная с именем `$f`, вероятно, ничего не скажет вам, если вы обратитесь к своему сценарию через месяц или более. Имя `$filename` имеет куда больше смысла.

Следует ли мне запомнить таблицу приоритетов выполнения операторов наизусть?

Нет причин, чтобы этого не сделать, но я бы поберег силы для более полезных задач. Используя в выражениях круглые скобки, можно сделать код не только не

зависящим от приоритета выполнения операторов, но и существенно более легким для чтения и понимания.

Семинар

Контрольные вопросы

1. Какие из следующих имен переменных недопустимы?

```
$a_value_submitted_by_a_user
$666666xyz
$xyz666666
$_____counter_____
$the first
$file-name
```

2. Что выводит следующий фрагмент кода?

```
$num = 33;
(boolean)$num;
print $num;
```

3. Что выводит следующая команда?

```
print gettype("4");
```

4. Какой результат выводит следующий фрагмент кода?

```
$test_val = 5.4566;
settype( $test_val, "integer" );
print $test_val;
```

5. В какой из перечисленных ниже команд не используется выражение?

```
4;
gettype(44);
5/12;
```

6. В какой из команд из предыдущего вопроса используется оператор?
7. Какое значение возвратит следующее выражение и какой тип данных будет иметь возвращаемое значение?

```
5 < 2
```

Ответы

1. Имя переменной `$666666xyz` недопустимо потому, что оно не начинается с буквы или символа подчеркивания. Имя переменной `$the first` неправильно потому, что оно содержит пробел. `$file-name` также недопустимо, потому что оно содержит не буквенно-цифровой символ.
2. Этот фрагмент выведет число 33. Операция приведения к логическому типу создает копию значения переменной `$num`. Она не меняет значение и тип исходной переменной.
3. Эта команда выводит строку `"string"`.

4. Этот код выводит значение `5`. Когда тип с плавающей точкой преобразовывается в целый, цифры после десятичной точки отбрасываются.
5. Это все выражения, потому что все они при вычислении дают значения.
6. Команда `5/12` содержит оператор деления.
7. Это выражение возвратит логическое значение `false`.

Упражнения

1. Создайте сценарий, содержащий, по меньшей мере, пять различных переменных. Присвойте им значения различного типа и используйте функцию `gettype()`, чтобы отобразить каждый тип в окне браузера.
2. Присвойте значения двум переменным. Используйте оператор сравнения для проверки, является ли первое значение:
   - таким же, как второе;
   - меньше второго;
   - больше второго;
   - меньше второго или равным ему.

   Отобразите результат каждой проверки в окне браузера. Измените значения, присвоенные тестовым переменным, и снова запустите сценарий.