ГЛАВА 3

Введение в PHP

В главе 1 о PHP говорилось как о языке, заставляющем сервер генерировать динамическую, потенциально разную выходную информацию при каждом запросе браузером веб-страницы. В данной главе начнется изучение этого простого, но мощного языка, которое продолжится в следующих главах и завершится в главе 7.

Я призываю вас выполнять разработку кода PHP в одной из интегрированных сред разработки (IDE), упомянутых в главе 2, или в хорошем редакторе кода.

Многие из этих программ позволяют запускать код, рассматриваемый в этой главе, и изучать производимую им выходную информацию. Вы также узнаете, как создавать PHP-код, чтобы иметь представление о внешнем виде выходной информации на веб-странице (то есть о том виде, в каком она в итоге предстанет перед пользователями). Но этот шаг, каким бы захватывающим он ни был поначалу, на данном этапе не имеет особого значения.

В процессе создания веб-страницы будут представлять собой комбинацию PHP, HTML, JavaScript, инструкций MySQL и форматирования с помощью CSS. Кроме того, каждая страница может привести на другие страницы, предоставляя пользователям возможность щелкать на ссылках и заполнять формы. Хотя при изучении этих языков можно обойтись и без таких сложностей. На данном этапе нужно сконцентрироваться исключительно на написании PHP-кода и на достижении предсказуемости содержимого выходной информации или по крайней мере на умении разбираться в характере этой информации.

Включение PHP в HTML

По умолчанию в конце имен PHP-документов ставится расширение PHP. Когда веб-сервер встречает в запрашиваемом файле это расширение, он автоматически передает файл PHP-процессору. Веб-серверы имеют довольно широкий диапазон настроек, и некоторые веб-разработчики выбирают такой режим работы, при

котором для разбора PHP-процессору принудительно передаются также файлы с расширениями HTM или HTML. Обычно это связано с тем, что разработчики хотят скрыть факт использования PHP.

Программа на PHP отвечает за возвращение файла в чистом виде, пригодном для отображения в браузере. В простейшем случае на выходе документа PHP будет получаться только код HTML. Чтобы убедиться в этом, можно взять любой HTML-документ и сохранить его в качестве PHP-документа (например, сохраняя файл `index.html` под именем `index.php`), и он будет отображаться точно так же, как исходный файл.

Для запуска команд PHP нужно изучить новый тег. Его открывающая часть имеет следующий вид:

```
<?php
```

Первое, что может броситься в глаза, — незавершенность тега. Это обусловлено тем, что внутри тега могут помещаться целые фрагменты кода PHP. Они заканчиваются, только когда встречается закрывающая часть тега такого вида:

```
?>
```

Небольшая PHP-программа Hello World может иметь вид, показанный в примере 3.1.

Пример 3.1. Вызов PHP

```
<?php
  echo "Hello world";
?>
```

Этот тег очень гибок в использовании. Некоторые программисты открывают его в начале документа, а закрывают в самом конце и выводят любой код HTML путем непосредственного использования команды PHP.

Другие программисты предпочитают помещать в эти теги как можно меньшие фрагменты кода PHP и именно в тех местах, где нужно воспользоваться динамическими сценариями, а весь остальной документ составлять из стандартного кода HTML.

Сторонники последнего метода программирования зачастую аргументируют свой выбор тем, что такой код выполняется быстрее, а сторонники первого метода утверждают, что увеличение скорости настолько мизерное, что оно не может оправдать дополнительные сложности многочисленных вставок PHP в отдельно взятый документ.

По мере изучения языка вы, несомненно, определитесь в своих стилевых предпочтениях при создании разработок на PHP, но для упрощения примеров, приводимых в этой книге, я свел количество переходов между PHP и HTML к минимуму, в среднем к одному-двум переходам на один документ.

Кстати, существует и несколько иной вариант синтаксиса PHP. Если поискать примеры PHP-кода в интернете, то можно встретить код, где используется следующий синтаксис открывающего и закрывающего тегов:

```
<?
  echo "Hello world";
?>
```

Несмотря на то что здесь неочевиден вызов PHP-парсера, это вполне приемлемый альтернативный синтаксис, который, как правило, также работает. Но я не советую его использовать, поскольку он несовместим с XML и в настоящее время его применение не приветствуется (это значит, что он больше не рекомендуется и его поддержка может быть удалена в будущих версиях).

Если в файле содержится только код PHP, то закрывающий тег ?> можно опустить. Именно так и нужно делать, чтобы гарантировать отсутствие в файлах PHP лишнего пустого пространства (что имеет особую важность при написании объектно-ориентированного кода).

Примеры в этой книге

Чтобы вы не тратили время на набор примеров, приводимых в книге, все они хранятся на GitHub. Можете скачать архив на свой компьютер, посетив сайт https://github.com/RobinNixon/lpmj6.

Примеры перечислены в соответствии с номерами глав (допустим, `example3-1.php`), но для некоторых из них могут потребоваться и конкретные имена файлов. Поэтому копии примеров сохраняются также с использованием имени файла в той же папке, как в показанном далее примере 3.4, который нужно будет сохранить в файле `test1.php`.

Структура PHP

В этом разделе будет рассмотрено довольно много основных положений. Разобраться во всем этом несложно, но я рекомендую проработать материал как можно тщательнее, поскольку он служит основой для понимания остальных глав. Как всегда, в конце главы будут заданы вопросы, с помощью которых можно будет проверить, насколько глубоко усвоен материал.

Комментарии

Существует три способа добавления комментариев к коду PHP. Первый, предусматривающий размещение в начале строки двух прямых слешей, превращает в комментарий отдельную строку:

```
// Это комментарий
```

Он хорошо подходит для временного исключения из программы строки кода, являющейся источником ошибок. Например, такой способ комментирования можно применить для того, чтобы скрыть строку кода до тех пор, пока в ней не возникнет необходимость:

```
// echo "X equals $x";
```

Такой комментарий можно также вставить сразу же после строки кода, чтобы описать ее действие:

```
$x += 10; // Увеличение значения $x на 10
```

Так же работает вариант с символом # в начале строки:

```
# Это тоже комментарий
```

Когда вам понадобится использовать несколько строк, нужно будет прибегнуть к третьему способу комментирования, который показан в примере 3.2.

Пример 3.2. Многострочный комментарий

```
<?php
/* Это область
   многострочного комментария,
   которая не будет
   подвергаться интерпретации */
?>
```

Для открытия и закрытия комментария можно воспользоваться парами символов /* и */ практически в любом произвольно выбранном месте кода. Если не все, то большинство программистов используют эту конструкцию для временного превращения в комментарий целого неработоспособного раздела кода или такого раздела, который по тем или иным причинам нежелательно интерпретировать.

Типичная ошибка — применение пар символов /* и */ для того, чтобы закомментировать большой фрагмент кода, уже содержащий закомментированную область, в которой используются эти же пары символов. Комментарии не могут быть вложенными друг в друга, поскольку PHP-интерпретатор не поймет, где заканчивается комментарий, и выведет на экран сообщение об ошибке. Но если вы используете редактор программ или интегрированную среду разработки с подсветкой синтаксиса, то ошибку такого рода нетрудно будет заметить.

Основной синтаксис

PHP — очень простой язык, уходящий своими корнями в языки C и Perl (надеюсь, вы когда-нибудь встречались с ними), но все же больше похожий на Java. Он очень гибок, но существует несколько правил, относящихся к его синтаксису и структуре, которые следует изучить.

Точки с запятыми

В предыдущих примерах можно было заметить, что команды PHP завершаются точкой с запятой:

```
$x += 10;
```

Одна из наиболее часто встречающихся причин ошибок, с которыми мы сталкиваемся в работе с PHP, — забывчивость. Если не поставить эту точку с запятой, PHP вынужден будет рассматривать в качестве одной сразу несколько инструкций, при этом он не сможет разобраться в ситуации и выдаст ошибку синтаксического разбора — `Parse error`.

Символ $

Символ `$` используется в разных языках программирования в различных целях. Например, в языке BASIC символ `$` применялся в качестве завершения имен переменных, чтобы показать, что они относятся к строкам.

А в PHP символ `$` должен ставиться перед именами всех переменных. Это нужно для того, чтобы PHP-парсер работал быстрее, сразу же понимая, что имеет дело с переменной. К какому бы типу ни относились переменные — к числам, строкам или массивам, все они должны выглядеть так, как показано в примере 3.3.

Пример 3.3. Три разновидности присваивания значений переменным

```
<?php
  $mycounter = 1;
  $mystring = "Hello";
  $myarray = array("One", "Two", "Three");
?>
```

Вот, собственно, и весь синтаксис, который нужно усвоить. В отличие от языков, в которых отношение к способам отступа текста программы и размещения кода очень строгое (например, от Python), PHP дает полную свободу использования (или игнорирования) любых отступов и любого количества пробелов на ваше усмотрение. В действительности же разумное использование того, что называется свободным пространством, обычно поощряется (наряду с подробным комментированием), поскольку помогает разобраться в собственном коде, когда к нему приходится возвращаться по прошествии некоторого времени. Это помогает и другим программистам, вынужденным поддерживать ваш код.

Переменные

Понять, что такое переменные PHP, поможет простая метафора. Думайте о них как о небольших (или больших) спичечных коробках! Именно как о спичечных коробках, которые вы раскрасили и на которых написали некие имена.

Строковые переменные

Представьте, что у вас есть коробок, на котором написано слово username (имя пользователя). Затем вы пишете на клочке бумаги *Fred Smith* и кладете эту бумажку в коробок (рис. 3.1). Этот процесс похож на присваивание переменной строкового значения:

```
$username = "Fred Smith";
```

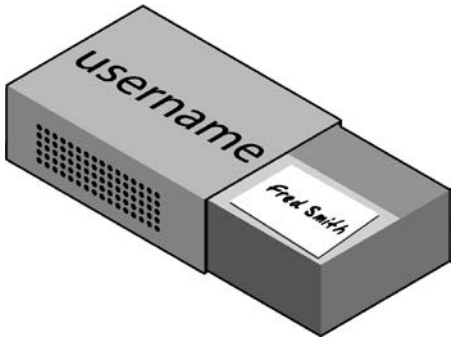

Рис. 3.1. Переменные можно представить в виде спичечного коробка, содержащего какие-то предметы

Кавычки служат признаком того, что `Fred Smith` является *строкой* символов. Каждую строку нужно заключать либо в двойные, либо в одинарные кавычки (апострофы). Между этими двумя видами кавычек есть весьма существенное различие, которое будет рассмотрено далее.

Когда хочется посмотреть, что находится внутри коробка, вы его открываете, вынимаете бумажку и читаете, что на ней написано. В PHP подобное действие (выводящее содержимое переменной) выглядит следующим образом:

```
echo $username;
```

Можно также присвоить содержимое другой переменной (сделать ксерокопию бумажки и поместить ее в другой коробок):

```
$current_user = $username;
```

Пример 3.4. Ваша первая PHP-программа

```
<?php // test1.php
  $username = "Fred Smith";
  echo $username;
  echo "<br>";
  $current_user = $username;
  echo $current_user;
?>
```

Теперь эту программу можно запустить, введя в адресную строку браузера следующий адрес:

```
http://localhost/test1.php
```

В маловероятном случае, предполагающем, что в ходе установки веб-сервера (рассмотренной в главе 2) вы изменили назначенный серверу порт на какой-нибудь другой, отличающийся от порта 80, вы должны поместить номер этого порта в URL в данном и всех последующих примерах из этой книги. Например, если вы изменили порт на 8080, то предыдущий URL приобретет следующий вид:

```
http://localhost:8080/test1.php
```

Не забудьте об этом при тестировании других примеров из книги или при написании собственного кода.

Результатом запуска этого кода будет двойное появление имени `Fred Smith`, первое — в результате выполнения команды `echo $username`, а второе — в результате выполнения команды `echo $current_user`.

Числовые переменные

Переменные могут содержать не только строки, но и числа. Если вернуться к аналогии со спичечным коробком, сохранение в переменной `$count` числа 17 будет эквивалентно помещению, скажем, 17 бусин в коробок, на котором написано слово `count`:

```
$count = 17;
```

Можно также использовать числа с плавающей точкой (содержащие десятичную точку). Синтаксис остается прежним:

```
$count = 17.5;
```

Чтобы узнать о содержимом коробка, его нужно просто открыть и посчитать бусины. В PHP можно присвоить значение переменной `$count` другой переменной или вывести его с помощью браузера на экран, воспользовавшись командой `echo`.

Массивы

Массивы можно представить в виде нескольких склеенных вместе спичечных коробков. Например, нам нужно сохранить имена пяти футболистов одной команды в массиве `$team`. Для этого мы склеим вместе боковыми сторонами пять коробков, запишем имена всех игроков на отдельных клочках бумаги и положим каждый клочок в свой коробок.

Вдоль всей верхней стороны склеенных вместе коробков напишем слово `team` (рис. 3.2). В PHP эквивалентом этому действию будет следующий код:

```
$team = arrayCBill', 'Mary', 'Mike', 'Chris', 'Anne');
```

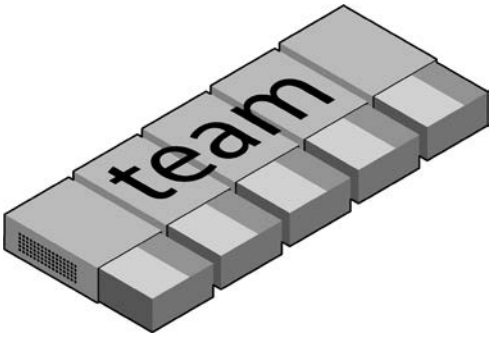

Рис. 3.2. Массив похож на несколько склеенных вместе спичечных коробков

Этот синтаксис несколько сложнее рассмотренных ранее инструкций. Код создания массива представляет собой следующую конструкцию:

```
array();
```

с пятью строками внутри круглых скобок. Каждая строка заключена в одинарные или двойные кавычки, и строки должны быть отделены друг от друга запятыми.

Когда потребуется узнать, кто является игроком номер 4, можно воспользоваться следующей командой:

```
echo $team[3]; // Эта команда отображает имя Chris
```

Использование в предыдущем примере числа 3, а не 4 обусловлено тем, что первый элемент PHP-массива является, как правило, нулевым, поэтому номера игроков распределяются в интервале от 0 до 4.

Двумерные массивы

Диапазон использования массивов очень широк. Например, вместо выстраивания одномерных рядов коробков из них можно построить двумерную матрицу, а массивы могут иметь три и более измерения.

Чтобы привести пример двумерного массива, представим, что нужно отслеживать ход игры в крестики-нолики, для чего требуется структура данных, состоящая из девяти клеток, сгруппированных в квадрат 3 × 3. Чтобы представить это в виде спичечных коробков, вообразите себе девять коробков, склеенных друг с другом в матрицу, состоящую из трех строк и трех столбцов (рис. 3.3).

Рис. 3.3. Многомерный массив, смоделированный с помощью коробков

Теперь для каждого хода можно класть в нужные коробки клочки бумаги с крестиком или ноликом. Чтобы сделать это в коде PHP, необходимо создать массив, содержащий три других массива, как в примере 3.5, в котором массив создается для отображения уже ведущейся игры.

Пример 3.5. Определение двумерного массива

```
<?php
  $oxo = array(array('x', ' ', 'o'),
               array('o', 'o', 'x'),
               array('x', 'o', ' '));
?>
```

Мы сделали еще один шаг к усложнению, но смысл его нетрудно понять, если усвоен основной синтаксис массива. Здесь три конструкции `array()` вложены во внешнюю по отношению к ним конструкцию `array()`. Мы заполнили каждую строку массивом, состоящим только из одного символа: `x`, `o` или пробела. (Мы воспользовались пробелом для того, чтобы все ячейки при отображении были одинаковой ширины.)

Для возвращения в дальнейшем третьего элемента во второй строке этого массива можно воспользоваться следующей PHP-командой, которая отобразит символ «x»:

```
echo $oxo[1][2];
```

Не забывайте о том, что отсчет индексов массива (указателей на элементы внутри массива) начинается с нуля, а не с единицы, поэтому в предыдущей команде индекс [1] ссылается на второй из трех массивов, а индекс [2] — на третью позицию внутри этого массива. Эта команда вернет содержимое третьего слева и второго сверху коробка.

Как уже упоминалось, поддерживаются даже массивы с большей размерностью, получаемые путем простого создания большего количества вложенных друг в друга массивов. Но в этой книге массивы с размерностью больше двух рассматриваться не будут.

Подробнее о массивах мы поговорим в главе 6.

Правила присваивания имен переменным

При создании PHP-переменных следует придерживаться четырех правил.

- Имена переменных, после знака доллара, должны начинаться с буквы или с символа _ (подчеркивания).
- Имена переменных могут содержать только символы: латинские буквы a–z и A–Z, 0–9 и _ (подчеркивание).
- Имена переменных не должны включать в себя пробелы. Если имя переменной нужно составить более чем из одного слова, то в качестве разделителя рациональнее всего будет использовать символ подчеркивания (например, `$user_name`).
- Имена переменных чувствительны к регистру символов. Переменная `$High_Score` отличается от переменной `$high_score`.

Чтобы позволить использование ASCII-символов, включающих диакритические знаки, PHP также поддерживает в именах переменных байты от 127 и до 255. Но пока ваш код не будет поддерживаться только теми программистами, которые знакомы с такими символами, от их применения лучше отказаться, поскольку программисты, использующие английские раскладки клавиатуры, будут испытывать трудности при доступе к таким символам.

Операторы

Операторы позволяют указать выполняемые математические операции, такие как сложение, вычитание, умножение и деление. Но существует также и ряд других типов операторов, например операторы работы со строками, проведения сравнений и выполнения логических операций. Математические операции в PHP во многом похожи на обычные арифметические записи. Например, в результате работы следующего оператора выводится число 8:

```
echo 6 + 2;
```

Перед тем как приступить к изучению возможностей PHP, следует уделить немного внимания рассмотрению предоставляющих эти возможности различных операторов.