Лабораторная № 6

Знакомство с PHP

Что такое PHP?

PHP — это язык программирования, который давно уже перерос свое название. Дело в том, что первоначально это был просто набор макросов, предназначенных для создания несложных личных web-страниц, и название PHP — не более чем аббревиатура от слов *Personal Home Page* (личная домашняя страница). Но со временем набор макросов превратился в полноценный язык программирования, с помощью которого можно создавать развитые web-узлы, обменивающиеся информацией с современными базами данных.

В настоящее время создатели PHP называют его обработчиком гипертекста (*HyperText Preprocessor*) Это язык программирования, используемый на стороне сервера (*server side scripting language*), конструкции которого вставляются в HTML-текст.

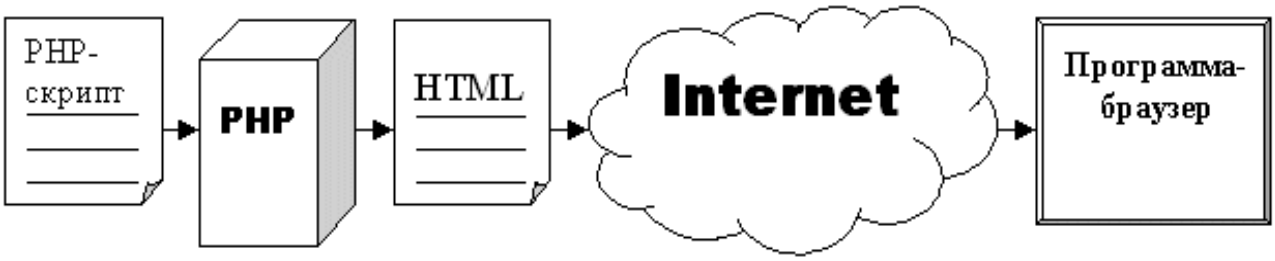

Рисунок 1

В отличие от обычного HTML-текста web-страницы, программа на PHP не передается браузеру, но обрабатывается препроцессором PHP или его модулями (рисунок 1). Фрагменты HTML-текста при этом остаются без изменений, а операторы PHP выполняются и результат их обработки вставляется в HTML-текст, после чего все вместе передается браузеру. Программа на PHP может делать запрос к базе данных, создавать графические изображения, читать и записывать файлы, общаться с внешними серверами, то есть возможности такой программы практически не ограничены.

История PHP

Первая версия PHP была создана программистом по имени Расмус Лердорф (Rasmus Lerdorf) в 1995 году и представляла собой набор макросов (сценарий Perl/CGI) для подсчета количества посетителей сайта, прочитавших его онлайновое резюме. Этот сценарий решал две задачи: регистрацию данных посетителя и вывод количества посетителей на web-странице. Развитие WWW

еще только начиналось, никаких специальных средств для решения этих задач не было, и к автору хлынул поток сообщений с вопросами. Лердорф начал бесплатно раздавать свой инструментарий, названный *Personal Home Page* (PHP) или *Hypertext Processor* (гипертекстовый процессор). В дальнейших разработках Лердорф перешел с Perl на C и добавил программу обработки форм. Расширение существующего инструментария PHP привело к появлению PHP 2.0, или, PHP/FI (*Personal Home Page — Form Interpreter*). Оказавшись привлекательным для пользователей, пакет вскоре привлек внимание разработчиков. Его популярность неуклонно возрастала. В 1997 г. над проектом уже работала целая команда программистов.

В результате таких коллективных усилий появилась следующая версия — PHP 3.0 (1997 год). Это была существенная переработка исходной версии PHP, включавшая в себя новый обработчик текстов, созданный программистами по имени Зив Зураски и Энди Гутманс (Zeev Surasky, Andi Gutmans). Были также внесены некоторые изменения в синтаксис языка и добавлены новые функции. Новая версия оказалась самым лучшим на то время языком программирования на стороне сервера, и популярность продукта стала просто поразительной.

В течение следующих двух лет стремительное развитие PHP продолжалось. В язык добавлялись новые функции, в частности — поддержка регулярных выражений, мощные средства работы с массивами, объектно-ориентированная методология. Работа шла быстро и завершилась 22 мая 2000 годв выпуском PHP версии 4.0 Создание интерфейса для работы с базой данных MySQL и сервером Apache еще больше укрепило позиции PHP. Сервер Apache в настоящее время является самым распространенным web-сервером в мире, и PHP может быть скомпилирован в виде модуля для сервера Apache. MySQL — это современная система управления базой данных, распространяемая бесплатно, поэтому в PHP включены функции для работы с этой СУБД. Приходится признать, что на сегодня объединение Apache, MySQL и PHP просто не имеет конкурентов.

Но из сказанного не следует, что PHP не может работать в другой среде и с другими базами данных. Эта технология прекрасно работает с большим количеством баз данных и веб-серверов.

Рост популярности PHP совпал по времени с изменением подхода к созданию web-страниц и узлов. В середине 90-х годов стало обычным явление, когда даже довольно крупные узлы состояли из сотен статических страниц, написанных полностью на HTML. Но теперь времена изменились. Разработчики получили в свое распоряжение средства, позволяющие web-страницам общаться с базами данных, обрабатывать формы и менять свое содержание в зависимости от обстоятельств и реакции пользователя.

Использование баз данных для хранения информации и программ для доступа к этой информации становится еще более актуальным по мере возрастания

потребности в передаче данных на устройства разной природы, такие как мобильные телефоны, цифровое телевидение и пр.

Принимая во внимание все сказанное, не стоит удивляться, что такая гибкая и эффективная технология, как PHP, все больше завоевывает мир.

Почему следует предпочесть PHP?

Существует несколько серьезных причин для того, чтобы выбрать PHP в качестве средства для разработки web-приложений. Почти всегда окажется, что скорость продвижения проектов будет гораздо выше, чем при использовании других языков и средств разработки. Поскольку PHP — это открытый продукт, он обеспечен технической поддержкой талантливой команды разработчиков и проверен широкой аудиторией пользователей. Кроме того, PHP может работать почти на любой операционной системе и с любым сервером.

Скорость разработки. Благодаря тому, что PHP позволяет отделить HTML-текст от выполнимой части, вы можете добиться значительного снижения времени, затраченного на разработку проекта. Во многих случаях, при удачном управлении проектом, вам удастся отделить программную часть проекта от собственно разработки web-страниц. Это облегчает жизнь не только программисту, но и дизайнеру, так как позволяет получить большую гибкость.

PHP — это открытый продукт. Для многих слова «открытый продукт» означают просто то, что данный продукт бесплатный. Безусловно, само по себе это уже неплохо, но далеко не все. Использование открытых программных продуктов дает дополнительные преимущества. Вы как бы присоединяетесь к сообществу разработчиков, которые делятся с вами накопленным опытом. Вполне возможно, что стоящая перед вами в данный момент проблема уже кем-то решена, но даже если это и не так, то вы можете спросить совета по электронной почте и получить квалифицированную консультацию.

Кроме того, вы можете быть уверены, что вам сообщат обо всех обнаруженных ошибках и обеспечат исправленной версией, а также вы будете получать все обновления и расширения данного продукта.

У открытого продукта, будь то сервер или операционная система, нет конкретного владельца, и вы можете вносить изменения в программы по своему усмотрению, для того чтобы настроить эти программы по своим потребностям.

Переносимость. Технология PHP разработана таким образом, чтобы обеспечить эффективную переносимость. В результате этого PHP может работать на многих операционных системах и почти на всех серверах. Вы можете разработать продукт для операционной системы UNIX, а потом без труда перенести его на систему Windows NT и, кроме того, испытывать свою работу на персональном web-сервере, а затем установить ее на сервере Apache.

Платформы, серверы и базы данных. PHP работает на самых разных платформах. Это может быть Windows, многие из версий UNIX, в том числе Linux, и даже Macintosh. PHP поддерживает широкий набор серверов, среди которых Apache (сам по себе являющийся открытым программным продуктом), Microsoft Internet Information Server, WebSite Pro, iPlanet Web Server и Microsoft Personal Web Server. Последний сервер особенно полезен тогда, когда вы хотите проверить свои программы под Windows, хотя нужно заметить, что и сервер Apache работает под управлением системы Windows.

Интерпретатор PHP можно скомпилировать так, чтобы он был независимым приложением. В таком случае вы сможете вызывать его просто из командной строки. Здесь в основном будет говориться о создании web-приложений, но не следует недооценивать PHP как язык программирования общего назначения, сравнимый с языком Perl.

При создании языка PHP учитывалось требование высокой интеграции с базами данных. Это стало одной из причин того, что PHP так популярен при создании развитых web-приложений. Многие базы данных непосредственно поддерживаются языком PHP, и среди них такие, как Adabas D, InterBase, Solid, dBase, mSQL, Sybase, Empress, MySQL, Velocis, FilePro, Oracle, UNIX dbm, Informix и др. Кроме того, PHP поддерживает стандарт ODBC.

1. Основные конструкции PHP

Рассмотрим конструкции языка PHP на примере простейшей PHP-программы. Как и HTML-документы, PHP-программы состоят из простого текста, поэтому писать их можно с помощью любого текстового редактора — Блокнота, если вы работаете в Windows, VI или Emacs в системе UNIX. Популярные HTML-редакторы имеют встроенную поддержку для редактирования PHP-текста.

Введите текст из листинга 1.1 и сохраните файл, дав ему имя `ls1-1.php`.

Листинг 1.1. Наша первая PHP-программа

```
<?php
     print "Hello Web!";
?>
```

У этого файла должно быть правильное расширение, потому что на основании этого сервер распознает файл как PHP-программу и запустит интерпретатор. По умолчанию в PHP расширение файлов программ должно быть `.php`.

Если вы не работаете непосредственно на том компьютере, который будет обрабатывать PHP-программы, то придется воспользоваться сервисом FTP, для того чтобы скопировать файл программы на сервер.

Когда документ будет скопирован в положенное ему место, вы сможете обратиться к нему с помощью браузера. Если все у вас работает правильно, то на экране браузера вы увидите вывод программы. Если на вашем сервере не установлен PHP или расширение файла распознано неправильно, то, скорее всего, вы увидите на экране браузера текст листинга 1.1. Если такое произошло, первым делом проверьте расширение файла программы. Если с расширением файла у вас все в порядке, то проверьте, правильно ли установлен PHP и правильно ли сконфигурирован сервер, т.е. указано ли ему то самое расширение, которое вы использовали для своей программы. Если ваша программа скопирована на сервер и все работает правильно, давайте посмотрим на ее текст немного внимательнее.

Обрамление блока PHP-команд

При написании PHP-программы вы должны сообщить интерпретатору, как отличить команды, которые он должен обрабатывать, от простого HTML-текста. В противном случае команды будут приняты за HTML-текст и переданы браузеру. В таблице 1.1 перечислены четыре способа обрамления PHP-команд.

Таблица 1.1

Вид тегов	Открывающий тег	Закрывающий тег
Стандартные	<?php	?>
Короткие	<?	?>
ASP	<%	%>
Программные	<SCRIPT LANGUAGE="php">	</SCRIPT>

Из перечисленных в таблице 1.1 тегов только стандартные и программные гарантированно работают в любой конфигурации PHP. Использование коротких тегов и тегов ASP должно быть явно разрешено в файле `php.ini`.

После того как вы отредактировали файл `php.ini`, можете пользоваться в своих программах любым из перечисленных видов тегов. Это в значительной степени вопрос вкуса или привычки, но если собираетесь писать свои страницы на языке XML, вам следует запретить использование коротких тегов (`<? ?>`) и пользоваться стандартными (`<?php ?>`).

Открывающие и закрывающие теги могут располагаться в той же строке, что и команды, т.е. можно написать и так:

```
<? print "Hello Web!"; ?>
```

Теперь, когда вы знаете, что такое блок команд PHP, давайте рассмотрим программу из листинга 2.1 внимательнее.

Функция `print()`

Функция `print()` предназначена для вывода данных в окно браузера. Поскольку в приведенном выше примере аргументом функции `print()` является строка символов, то она обязательно должна быть заключена в кавычки — двойные или одинарные.

В общем случае после имени функции должны находиться скобки, независимо от того, передаются ей какие-то аргументы или нет. Функция `print()` — это исключение из правила, и вы не обязаны заключать в скобки строку, которую вы хотите вывести в окно браузера. Поэтому мы будем опускать скобки при вызове функции `print()`.

Единственный оператор в листинге 1.1 заканчивается точкой с запятой. Это сделано для того, чтобы сообщить интерпретатору об окончании команды. Точка с запятой обязательно должна стоять в конце каждой команды. Исключением из этого может быть оператор, завершающий блок команд. Но в

большинстве случаев пропуск точки с запятой сбивает интерпретатор с толку и приводит к ошибке.

Взаимодействие HTML и PHP

Программа в листинге 1.1 состоит только из команд PHP. Однако можно создать смешанный документ, добавив теги HTML перед открывающим и после закрывающего тегов PHP. Пример этого приведен в листинге 1.2.

Листинг 1.2. Документ, содержащий PHP-команды и HTML-текст

```
<html> <head>
<title> Листинг 1-2. Документ, содержащий PHP-команды
        и HTML-текст </title>
</head> <body>

<p><i>Проверка</i>

<?php
    print "<p style=\"color:blue\">PHP работает!</p>";
?>

</body> </html>
```

Интерпретатор игнорирует все, что находится вне тегов PHP. Если вы отобразите этот документ на экране браузера, то увидите слова «*Проверка*» и «PHP работает!», выделенные, соответственно, курсивом и синим цветом. Обратите внимание на обратную косую черту перед двойными кавычками. Это так называемое *экранирование*, оно применяется для того, чтобы кавычки, стоящие в HTML-тегах внутри аргумента функции `print()`, не обрабатывались интерпретатором PHP, а выводились просто как символы. Если вы посмотрите в браузере исходный текст этой страницы, он будет выглядеть точно так же, как обычный HTML-документ, уже без команд PHP.

В документ можно включать столько блоков PHP-команд, сколько потребуется для формирования web-страницы. Несколько блоков команд в одном документе образуют единую программу. Это означает, что все, что вы определите в первом блоке (переменные, функции или классы), будет доступно программе в следующих блоках.

Комментарии в PHP-программе

Когда вы пишете программу, все в ней кажется ясным и понятным, но по прошествии нескольких месяцев бывает очень непросто что-то понять в написанном. Если вы не поленитесь создавать комментарии в своих

программах, то сэкономите много времени и сил себе, а особенно тому, кому придется разбираться в ваших программах после вас.

Отдельная строка комментария начинается двумя символами косой черты `//` или одним символом `#`. Любой текст от этих знаков до конца строки или до закрывающего тега PHP игнорируется.

```
// Это комментарий.

# Это тоже комментарий.
```

Несколько строк комментариев начинаются парой символов `/*` и заканчиваются парой `*/`.

```
/*

Это комментарии.
Все эти строки будут проигнорированы интерпретатором.
*/
```

Многострочные комментарии особенно удобны для записи сводной информации обо всей программе или ее части.

2. Переменные

Переменная — это средство для хранения данных определенного типа. Каждая переменная имеет имя, начинающееся со знака доллара, **`$`**. Имя переменной может состоять из букв, цифр и знака подчеркивания, при этом регистр символов учитывается. В имени не могут встречаться пробелы и какие-нибудь символы, отличные от букв и цифр. В следующем примере приведены правильные имена переменных.

```
$a;  $a_long_variable_name;  $_2453;
```

Помните о том, что точка с запятой ставится в конце команды, однако она не является частью имени переменной. В переменных могут храниться числа, строки символов, объекты, массивы или логические значения.

Как вы видите, существует широкий выбор для имени переменной, но не принято создавать имена из одних только цифр. Как правило, создание переменной и присваивание ей значения выполняется в одной и той же команде.

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

В приведенном примере созданы две переменные, и с помощью оператора присваивания в них записаны значения. Об операторе присваивания мы поговорим ниже. После того как переменной присвоено значение, ею можно пользоваться точно так же, как и самим значением. Другими словами,

выражение `print($num1)` эквивалентно выражению `print(8)`, если значение переменной `$num1` равно 8.

Строковое присваивание

Строкой называется последовательность символов, которая рассматривается как единое целое. Строки делятся на две категории в зависимости от типа ограничителя — пары кавычек (`" "`) или апострофов (`' '`). Между этими категориями существуют два принципиальных отличия.

Во-первых, имена переменных в строках, заключенных в кавычки, заменяются соответствующими значениями, а строки в апострофах интерпретируются буквально, даже если в них присутствуют имена переменных.

Два следующих присваивания дают одинаковый результат:

```
$user = "Ник";
```

```
$user = 'Ник';
```

Однако результаты следующих присваиваний сильно различаются:

```
$var1 = "Мой друг $user";
```

```
$var2 = 'Мой друг $user';
```

Переменной `$var1` присваивается строка

`Мой друг Ник.`

Обратите внимание: переменная `$user` автоматически интерпретируется. С другой стороны, переменной `$var2` присваивается строка

`Мой друг $user.`

В отличие от переменной `$var1`, в `$var2` переменная `$user` осталась просто как последовательность символов. Эти различия как раз и обусловлены использованием либо кавычек, либо апострофов.

Второе принципиальное различие между строками, заключенными в апострофы и в кавычки, связано с обработкой служебных символов. В PHP, как и в других языках программирования, строки могут содержать служебные символы (например, символы новой строки или табуляции), перечисленные в таблице 2.1.

Таблица 2.1 — Служебные символы в строках

Последовательность	Описание
\n	Новая строка
\r	Перевод строки
\t	Горизонтальная табуляция
\\	Обратная косая черта как символ
\$	Знак доллара как символ
\"	Кавычка как символ
\'	Апостроф как символ

Служебные символы \n, \r и \t используются лишь для создания удобочитаемого HTML-файла, на вывод текста в браузере они никак не влияют.

Итак, в строках, заключенных в кавычки, распознаются все существующие служебные символы, а в строках, заключенных в апострофы — только служебные символы \\ и \'. Следующий пример наглядно демонстрирует это различие:

```
print "первая строка \r вторая строка";

print 'первая строка \r вторая строка';
```

Если вывести обе строки в браузере, окажется, что в строке в кавычках, в HTML-файле («Просмотр в виде HTML»), будет выполнен перевод строки, хотя на экране все будет в одной строке: «первая строка вторая строка». А в строке в апострофах последовательность \r выведется на экран как обычные символы. Помните об этом, выбирая между кавычками и апострофами, и вам удастся избежать многих неожиданностей.

Динамические переменные

В некоторых ситуациях бывает удобно использовать переменные, содержимое которых может динамически интерпретироваться как имя другой переменной. Таким образом, выражения присваивания

```
$client = "user"; $$client = "Nic";
```

эквивалентны следующей записи

```
$user = "Nic";
```

В переменной `$client` записана строка `"user"`, поэтому можно представить себе, что выражение `$$client` — это знак `$`, за которым следует значение переменной `$client`. Все вместе PHP интерпретирует это как `$user`.

На первый взгляд от этого способа создания переменных немного пользы, однако, воспользовавшись оператором конкатенации и циклом, можно динамически создать сразу много переменных.

При обращении к динамической переменной важную роль играет использование или не использование кавычек:

```
$user = "Nic"; print $user;
```

Это эквивалентно следующему:

```
$user = "Nic"; $client = "user"; print $$client;
```

Однако, для того чтобы вывести имя переменной, нужно обратиться к ней по-другому. Например, следующий фрагмент не выводит в окно браузера строку `Nic`, как можно было бы предположить, поскольку в операторе `print()` переменная стоит в кавычках:

```
$user = "Nic"; $client = "user"; print "$$client";
```

Вместо этого выводится знак `$`, а потом строка `user`, образуя строку `$user`. Как уже было сказано, когда вы обрамляете переменную кавычками, PHP подставляет вместо нее соответствующее значение. В данном случае PHP подставляет вместо переменной `$client` ее значение `user`.

В листинге 2.1 все сказанное объединено в программу, в которой с помощью строки, хранящейся в переменой, создается и инициализируется новая переменная `$user`.

Листинг 2.1. Создание динамической переменной и обращение к ней

```
<html> <head>
<title> Листинг 2-1. Создание динамической переменной
        и обращение к ней
</title>

</head>

<body>

<?php

$client = "user";

$$client = "Nic"; // <=> $user = "Nic";

print "1) $client<br>";       //выводится user
print '2) $client<br>';       //выводится $client
print "3) $user<br>";         //выводится Nic
print "4) $$client<br>";      //выводится $user
print "5) "; print $$client; //выводится Nic
?>
</body>

</html>
```

Ссылки на переменные

Обычно переменным присваиваются значения других переменных. Другими словами, если вы присвоите значение переменной `$var1` другой переменной, `$var2`, то копия значения первой переменной будет записана во вторую. В дальнейшем никакие изменения значения первой переменной никак не отразятся на значении второй.

Но в PHP можно сделать по-другому, заставив переменную `$var2` постоянно иметь то же самое значение, что и у переменной `$var1`. Это продемонстрировано в листинге 2.2.

Листинг 2.2. Создание ссылки на переменную

```
<html> <head>
<title> Листинг 2-2. Создание ссылки на переменную
</title>

</head>

<body>

<?php

$var1 = 1;
$var2 = &$var1;
$var1 = 10;
print $var2;  //выводится 10

?>
</body>

</html>
```

Символ `&` перед именем переменной `$var1` **говорит о том, что мы создаем ссылку на эту переменную**, и теперь все изменения ее значения отразятся на значении переменной `$var2`. Другими словами, обе эти переменные связаны с одним и тем же значением.

Поскольку такая техника позволяет избежать копирования значений из одной переменной в другую, это может привести к некоторому повышению производительности программы, но настолько незначительному, что вряд ли вы заметите это на глаз. Механизм ссылок появился только в PHP 4.0.

3. Типы данных

Различные типы данных занимают разный объем памяти и обработка разных типов выполняется по-разному. Поэтому некоторые языки программирования требуют от программиста, чтобы он заранее объявил, для какого типа данных предназначена та или иная переменная. PHP не имеет таких строгих требований по типам данных, т.е. он будет обрабатывать переменную в зависимости от того, какого типа значение в нее записано. Такой подход имеет как достоинства, так и недостатки. С одной стороны, программист может использовать переменные гибко, храня в них данные, необходимые в настоящий момент. Но, с другой стороны, такая свобода действий может привести к появлению ошибок, которые трудно обнаружить, особенно в больших программах, когда в переменной записано совсем не то, что ожидает там найти программист.

В таблице 3.1 перечислены шесть типов данных, поддерживаемые в языке PHP.

Таблица 3.1 — Типы данных

Тип	Пример	Описание
`Integer`	`5`	Целое число
`Double`	`3.234`	Число с плавающей точкой
`String`	`"hello"`	Строка символов
`Boolean`	`true`	Логический, принимающий значения `true` или `false`
`Array`	`$a[10]`	Массив
`Object`		Объект (элемент ООП)

Проверка и изменение типа переменной

В языке PHP есть функция, с помощью которой можно проверить тип переменной. Если вы вызовете функцию `gettype()`, передав ей в качестве аргумента имя переменной, то получите строку, описывающую тип этой переменной. Кроме того, существует функция `settype()`, служащая для изменения типа переменной. Для того чтобы изменить тип переменной, вы должны при вызове этой функции указать переменную, тип которой вы хотите изменить, и новый тип данной переменной. В листинге 3.1 приведен пример того, как с помощью функций `gettype()` и `settype()` проверяется и изменяется тип переменной `$var` поочередно для нескольких рассматриваемых типов.

Листинг 3.1. Проверка и изменение типа переменной

```
<html> <head>
<title> Листинг 3-1. Проверка и изменение типа переменной

</title> </head> <body>

<?php

$var = 3.14;

print gettype($var);    // double
print " - $var<br>";    // 3.14

settype($var, "string");

print gettype($var);    // string
print " - $var<br>";    // 3.14

settype($var, "integer");

print gettype($var);    // integer
print " - $var<br>";    // 3

settype($var, "double");

print gettype($var);   // double
print " - $var<br>";   // 3

settype($var, "boolean");

print gettype($var);   // boolean
print " - $var<br>";   // 1

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

Каждый раз после изменения типа переменной мы проверяем ее новый тип для того, чтобы убедиться, что все сработало правильно, и выводим значение переменной в окно браузера. Когда мы преобразовали строку `"3.14"` в целое число, дробная часть числа оказалась отброшенной навсегда, поэтому значение переменной `$var` осталось равным `3`, после того как мы преобразовали ее в действительное число. В конце концов мы преобразуем переменную в тип `boolean`, и ее значение становится равным `1`, так как все значения, отличные от нуля, в таком случае преобразуются в `1`. При выводе на печать логической переменной значение `true` выводится как `1`, а значение `false` — как пустая строка.

Преобразование типа переменной

Тип переменной можно преобразовать временно. Для этого нужно указать новый тип в скобках перед именем переменной. При этом создается копия данной переменной, значение которой преобразуется к новому типу. Принципиальная разница между таким преобразованием и использованием функции `settype()` состоит в том, что `settype()` изменяет тип переменной навсегда, а преобразование типа только создает временную копию нового типа, оставляя саму переменную без изменений. Это продемонстрировано в листинге 3.2.

Листинг 3.2. Преобразование типа переменной

```
<html> <head>
<title> Листинг 3-2. Преобразование типа переменной
</title> </head> <body>

<?php

$var = 3.14;

$var2 = (double) $var;
print gettype($var2);   // double
print " - $var2<br>";   // 3.14

$var2 = (string) $var;
print gettype($var2);   // string
print " - $var2<br> ";  // 3.14

$var2 = (integer) $var;
print gettype($var2);   // integer
print " - $var2<br>";   // 3

$var2 = (double) $var;
print gettype($var2);   // double
print " - $var2<br>";   // 3.14

$var2 = (boolean) $var;
print gettype($var2);    // boolean
print " - $var2<br>";    //   1

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

В этом примере мы не изменяем тип самой переменной `$var` — он всегда остается `double`. На самом деле мы каждый раз создаем временную копию нового типа и ее значение присваиваем переменной `$var2`. Поскольку мы работаем только с копией переменной `$var`, ее значение не изменяется, как это происходило в предыдущем примере в листинге 3.1.

4. Операторы и выражения

Оператором называют символ или последовательность символов, с помощью которых можно из нескольких переменных получить новое значение. Те значения, к которым применяются операторы для получения новых значений, называются *операндами*.

Комбинация операндов и операторов, производящая некоторое значение, называется *выражением*. Однако не обязательно для образования выражения использовать операторы. Выражением в PHP считается все, что имеет некоторое значение. Например, константа 654, или переменная `$user`, или функция `gettype()` — все это выражения. Таким образом, выражение `(4+5)` состоит из двух выражений и одного оператора.

Выражение — это любая комбинация чисел, переменных и вызовов функций, объединенных операторами. Выражение можно использовать как и любое другое значение.

Теперь рассмотрим некоторые операторы PHP.

Оператор присваивания

Вы уже встречали оператор присваивания каждый раз, когда мы говорили о создании переменной. Этот оператор состоит из знака «равно» (=). Оператор присваивания записывает значение своего правого операнда в левый операнд:

```
$name = "Nic";
```

Теперь в переменной `$name` записана строка `"Nic"`. Обратите внимание на то, что эта конструкция представляет собой выражение. На первый взгляд может показаться, что оператор присваивания просто записывает значение в переменную, но это не совсем так. На самом деле при выполнении оператора присваивания создается временная копия его правого операнда, и все выражение получает значение этой копии. Таким образом, следующая конструкция не только присваивает значение переменной, но и выводит в окно браузера строку `"Nic"`.

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

Арифметические операторы

Арифметические операторы языка PHP перечислены в таблице 4.1. Оператор «%» вычисляет остаток от целочисленного деления левого операнда на правый.

Таблица. 4.1 — Арифметические операторы

Оператор	Название	Пример	Результат
+	Сложение	10+3	13
-	Вычитание	10-3	7
*	Умножение	10*3	30
/	Деление	10/3	3.333333333
%	Остаток от деления	10%3	1

Оператор конкатенации

Символ оператора конкатенации — **простая точка**. Этот оператор объединяет две строки, точнее, присоединяет правую строку к левой. Таким образом, выражение

```
"hello"." world"
```

имеет значение

```
"hello world"
```

Независимо от типа своих операндов, оператор конкатенации всегда обрабатывает их как строки и результат его выполнения всегда является строкой.

Дополнительные операторы присваивания

Вообще, существует только один оператор присваивания, однако есть несколько операторов, которые представляют собой как бы комбинацию арифметического оператора и оператора присваивания.

Например, пусть требуется переменной `$x` присвоить значение 7 следующим образом:

```
$x = 3;
$x = $x+4;
```

Данную запись можно немного сократить, если воспользоваться дополнительными операторами присваивания. Тогда это будет выглядеть так:

```
$x = 3;
$x +=4;
```

Для каждого арифметического оператора существует соответствующий дополнительный оператор присваивания, который выглядит как пара знаков арифметического оператора и оператора присваивания. Также имеется и дополнительный оператор присваивания для конкатенации.

Эти операторы перечислены в таблице 4.2.

Таблица 4.2 — Дополнительные операторы присваивания

Оператор	Пример	Эквивалентная запись
+=	`$x += 5`	`$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"`

Операторы сравнения

Операторы сравнения возвращают значение `true`, если сравнение своих операндов успешно, и `false` — в противном случае. Условные выражения, образуемые с помощью таких операторов, часто применяются в таких командах, как `if` или `while`.

Операторы сравнения перечислены в таблице 4.3.

Таблица. 4.3 — Операторы сравнения

Оператор	Название	Условие выполнения	Пример	Результат при $x, равном 3
==	Равенство	Левый операнд равен правому	`$x == 5`	`false`
!=	Неравенство	Левый операнд не равен правому	`$x != 5`	`true`
===	Идентичность	Операнды равны и их типы совпадают	`$x === 5`	`false`
>	Больше чем	Левый операнд больше правого	`$x > 3`	`false`

>=	Больше или равно	Левый операнд больше правого или равен ему	`$x >= 3`	`true`
<	Меньше чем	Левый операнд меньше правого	`$x < 3`	`false`
<=	Меньше или равно	Левый операнд меньше правого или равен ему	`$x <= 3`	`true`

Чаще всего эти операторы применяются для сравнения чисел, как целых, так и действительных, но иногда их используют и для сравнения строк.

Логические операторы

Логические операторы предназначены для построения логических выражений. Например, логическое `ИЛИ` возвращает значение `true`, если хоть один из его операндов имеет значение `true`. Таким образом, выражение

```
true || false
```

имеет значение `true`.

Логическое `И` возвращает `true`, если оба операнда имеют значение `true`. Таким образом, выражение

```
true && false
```

имеет значение `false`. Конечно, маловероятно, что вам понадобится сравнивать логические константы. Гораздо больше смысла в сравнении логических выражений, например,

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

имеет значение `true`, если значение `$x` находится в интервале между 2 и 15. Скобки в этом выражении расставлены просто для того, чтобы выражение легче читалось. Логические операторы перечислены в таблице. 4.4.

Таблица 4.4 — Логические операторы

Оператор	Название	Условие истинности	Пример	Результат
`||`	ИЛИ	Хотя бы один из операндов истинен	`true || false`	`true`
`OR`	ИЛИ	Хотя бы один из операндов истинен	`true OR false`	`true`
`XOR`	Исключающее ИЛИ	Только один из операндов истинен	`true XOR true`	`false`
`&&`	И	Оба операнда истинны	`true && false`	`false`
`AND`	И	Оба операнда истинны	`true AND false`	`false`
`!`	НЕ	Операнд не истинен	`!true`	`false`

Вас может удивить то, что существует две версии операторов как `И`, так и `ИЛИ`. Тут все дело в разном порядке вычисления операторов, но об этом мы поговорим ниже.

Логические операторы часто используются для проверки результата вызова функций:

```
file_exists("ris.gif") OR print "Файл не найден!";
```

Увеличение и уменьшение целой переменной

При написании программ на PHP вам часто может понадобиться увеличить или уменьшить на единицу значение целой переменной. Особенно часто это приходится делать со счетчиками циклов. Вы уже знакомы по крайней мере с двумя способами осуществлять это. Значение переменной `$x` можно увеличить на единицу с помощью оператора сложения, например так:

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

То же самое можно сделать с помощью дополнительного оператора присваивания:

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

В обоих случаях переменной `$x` присваивается новое значение. Выражения такого типа встречаются настолько часто, что в языке PHP предусмотрены специальные операторы для увеличения или уменьшения значения целой переменной на 1. Эти операторы бывают двух типов — префиксные (*pre-decrement* и *pre-increment*) и постфиксные (*post-decrement* и *post-increment*).

Постфиксный оператор выглядит как два знака «−» или «+» после имени переменной.

```
$x++;  // увеличение $x на 1
$x--;  // уменьшение $x на 1
```

Если такой постфиксный оператор использовать в условном выражении, то значение переменной будет изменено только после вычисления выражения. Рассмотрим пример.

```
$x = 3;
$x++ < 4;  // выражение истинно
```

В этом примере значение переменной `$x` при ее сравнении с константой 4 равно 3, поэтому условие выполняется, и значение условного выражения равно `true`. После вычисления выражения значения переменной увеличивается.

Но в других обстоятельствах вам может понадобиться сделать так, чтобы значение переменной изменялось до того, как будет вычислено все выражение. Для этого вам придется воспользоваться префиксной формой оператора. Такая форма отличается от рассмотренной выше только тем, что знаки операции стоят перед именем переменной.

```
++$x;  // увеличение $x на 1
--$x;  // уменьшение $x на 1
```

Если такую форму оператора использовать в условном выражении, то значение переменной будет изменено до того, как вычислять все выражение.

```
$x = 3;
++$x < 4;  // выражение ложно
```

В этом случае при сравнении переменной `$x` с константой 4 значение переменной уже равно 4, т.е. оно не меньше 4, а следовательно, выражение ложно.

Порядок вычисления операторов

Несмотря на то, что существует определенный порядок вычисления выражений, в сложных выражениях желательно применять скобки для того, чтобы не ошибиться самому и облегчить чтение своих программ другим. В таблице 4.5 перечислены операторы по порядку старшинства, начиная с самого высокого, т.е. с того, который вычисляется раньше всех.

Таблица 4.5 — Порядок вычисления операторов

`++ --`
`/ * %`
`+ -`
`< <= => >`
`== === !=`
`&&`
`||`
`= += -= /= *= %= .=`
`AND`
`XOR`
`OR`

Как видно из этой таблицы, оператор `OR` имеет более низкий, уровень старшинства, чем `||`, а оператор `AND` — более низкий, чем `&&`. Таким образом, вы можете пользоваться этим для изменения порядка вычисления своих выражений. Однако здесь легко запутаться и сделать ошибку. Например, следующие два выражения эквивалентны, но второе гораздо проще для понимания.

```
$x && $y || $z
($x AND $y) OR $z
```

5. Константы

Константой называется именованная величина, которая не изменятся во время выполнения программы. Для создания константы используется функция `define()`. Константа, созданная с помощью этой функции, не может быть изменена. Для того чтобы создать такую константу, необходимо передать функции `define()` в качестве параметров имя константы и ее значение.

```
define("PI", "3.141592");
```

Значение, которое вы связываете с именем константы, может быть только числом или строкой. Имя константы должно состоять только из заглавных букв. Для обращения к константе указывается лишь ее имя, знак доллара перед именем не ставится. В листинге 5.1 приведен пример создания константы и обращения к ней.

Листинг 5.1. Создание константы

```
<html> <head>
<title> Листинг 5-1. Создание константы </title>
</head>

<body>

<?php

define ("PI", "3.141592");

print "Число Пи равно ".PI."<br>";

$pi2 = PI*PI;

print "Пи в квадрате равно $pi2";
?>
</body>

</html>
```

Обратите внимание на то, что для присоединения константы к строке `"Число Пи равно"` мы применили оператор конкатенации, а для того, чтобы вычислить значение константы в квадрате пришлось использовать дополнительную переменную.

6. Условные операторы

Большинство программ изменяют свое поведение в зависимости от изменяющихся условий, и для этого им приходится вычислять значения некоторых выражений. Как и многие другие языки программирования, PHP осуществляет это с помощью оператора `if`.

Оператор `if` с блоком `else`

При выполнении оператора `if` вычисляется выражение в скобках. Если значение этого выражения равно `true`, то блок программы выполняется.

Для того, чтобы указать альтернативный блок команд, который следует выполнить в том случае, если условие не выполняется нужно после блока `if` поместить блок `else`.

```
if (выражение) {
   // этот фрагмент выполняется, если выражение истинно
}
else {
   // этот фрагмент выполняется, если выражение ложно
}
```

В листинге 6.1 приведен пример использования условного оператора.

Листинг 6.1. Оператор `if` с блоком `else`

```
<html> <head>
<title> Листинг 6-1. Оператор if с блоком else </title>
</head> <body>
<?php
$var = "плохо";
if ($var == "хорошо")
     {print "Я в хорошем настроении!";}
else {print "Мне $var";}
?>
</body> </html>
```

В переменной `$var` записана строка `"плохо"` — она не совпадает со строкой `"хорошо"`, поэтому условие не выполняется. Это означает, что первый блок команд пропускается и выполняется блок команд, следующий за словом `else`. Соответствующее сообщение выводится на экран браузера.

Блок `elseif` оператора `if`

С помощью конструкции `if-elseif-else` можно проверить несколько условий перед тем, как выполнить фрагмент программы.

```
if (выражение_1) {
// этот фрагмент выполняется, если выражение истинно
}
elseif (выражение_2) {
// этот фрагмент выполняется, если выражение_1 ложно,
// а выражение_2 истинно
}
else {
// этот фрагмент выполняется во всех остальных случаях
}
```

Если первое выражение не истинно, то первый блок игнорируется. В блоке `elseif` вычисляется второе выражение, и если оно истинно, то выполняется соответствующий фрагмент программы. В противном случае выполняется блок команд `else`. Блоков `elseif` может быть сколько угодно, а блок `else` может отсутствовать, если в нем нет необходимости.

Пример приведен в листинге 6.2.

Листинг 6.2. Использование блоков `else` и `elseif`

```
<html> <head>
<title> Листинг 6-2. Использование блоков else и elseif
</title> </head> <body>
<?php
$var = "плохо";
if ($var == "хорошо") {print "Я в хорошем настроении!";}
elseif ($var == "плохо") {print "Не отчаиваться!";}
else {print "Непонятно, просто $var";}
?>
</body> </html>
```

Как и раньше, в переменной `$var` записана строка `"плохо"`. Она не совпадает со строкой `"хорошо"`, поэтому первый блок игнорируется. В блоке `elseif` переменная `$var` сравнивается со строкой `"плохо"` и, поскольку это выражение истинно, выполняется блок `elseif`.

Одно важное замечание. Если в скрипте 6.2 убрать строку с присваиванием

```
$var = "плохо";
```

то тогда значение этой переменной должно задаваться при вызове скрипта в виде:

```
ls6-1.php?var=так+себе
```

т.е. после имени скрипта идет знак вопроса, затем имя переменной, знак «равно» и значение переменной. Причем, если значением переменной является строка, содержащая пробелы, то они должны заменяться на знак «+». Такая замена в скрипте может выполняться с помощью функции **urlencode(),** которая возвращает строку, в которой все не алфавитно-числовые символы (кроме минуса, подчерка и точки) заменены на знак процентов (%) с последующими двумя 16-ричными цифрами, а пробелы заменены на знаки плюс (+).

В скрипте использование значения, переданного из строки вызова,

```
ls6-1.php?var=так+себе
```

осуществляется с помощью суперглобального массива **$_GET[]** следующим образом:

```
$var2=$_GET['var'];
```

В качестве индекса массива **$_GET[]** стоит имя аргумента из строки вызова скрипта, находящееся после знака «вопрос». Подробнее о суперглобальных массивах будет рассказано позднее.

Оператор switch

Оператор switch — это еще один способ изменить поток выполнения программы в зависимости от значения некоторого выражения. Между операторами switch и if есть существенная разница. С помощью оператора if и блока elseif вы можете для принятия решения вычислить несколько выражений, а оператор switch вычисляет только одно выражение, но в зависимости от его значения выполняет один из нескольких блоков программы. Выражение в операторе if может быть только истинным или ложным, а выражение оператора switch может принимать много значений, но они должны быть простого типа, т.е. целое, или строка, или булево значение.

```
switch (выражение)        # case [выражение] of (в Паскале)
{
case значение_1:          # значение: [оператор]; (в Паскале)
// выполняется, если выражение равно значение_1
break;

case значение_2:
// выполняется, если выражение равно значение_2
break;
```

```
default:                    # else [оператор] (в Паскале)
// выполняется, если выражение не приняло
// ни одного из перечисленных значений
}
```

Выражение в операторе `switch` чаще всего бывает простой переменной. В каждом из блоков `case` значение переменной сравнивается с указанным значением, и если они совпадают, то выполняется соответствующий блок. Команда `break` прерывает выполнение всей конструкции `switch`. Если слово `break` опустить, то будет проверяться следующий блок `case`. Если ни в одном из блоков `case` значение переменной не совпало с указанным значением, то выполняется блок `default`.

Не забывайте ставить команду `break` в конце каждого блока `case`. Если этого не сделать, то программа будет продолжать сравнивать значение выражения со всеми значениями в последующих блоках `case` и в конце концов доберется до блока `default`. Почти всегда это будет не то, что вам нужно.

В листинге 6.3 приведен пример использования оператора `switch`.

Листинг 6.3. Оператор `switch`

```
<html> <head>
<title> Листинг 6-3. Оператор switch </title>
</head> <body>
<?php
$var = "плохо";
switch ($var) {
case "хорошо":
      print "Я в хорошем настроении!";
      break;
case "плохо":
      print "Не отчаиваться!";
      break;
default:
      print "Непонятно, просто $var";
}
?></body> </html>
```

Как и раньше, в переменной `$var` записана строка `"плохо"`. Эта переменная используется в качестве выражения в операторе `switch`.

В первом блоке case проверяется совпадение этой переменной со строкой "хорошо". Совпадения нет, поэтому выполнение программы переходит ко второму блоку case. Здесь значение переменной $var совпадает со строкой "плохо" и, следовательно, выполняются соответствующие команды. Заканчивается все командой break.

Оператор ?

Оператор ? возвращает значение одного из двух выражений, разделенных знаком двоеточия. Какое из двух выражений сформирует возвращаемое значение — зависит от истинности тестового выражения.

```
(тестовое_выражение) ? выражение_1: выражение_2;
```

Если тестовое выражение принимает значение true, то возвращается значение первого выражения; в противном случае возвращается значение второго выражения. В листинге 6.4 приведен пример использования оператора ?, возвращаемое значение которого зависит от значения переменной $var.

Листинг 6.4. Оператор ?

```
<html> <head>
<title> Листинг 6-4. Оператор ? </title>
</head> <body>
<?php
$var = "плохо";
$text = ($var == "хорошо") ? "Хорошо!" : "Мне $var";
print "$text";
?>
</body> </html>
```

В переменной $var записана строка "плохо". Значение этой переменной сравнивается со строкой "хорошо". Поскольку совпадения не происходит, возвращается второе из значений.

Оператор ? тяжеловат для чтения, но он может пригодиться, если у вас есть всего два выбора и вы хотите сделать текст программы компактным.