Структура PHP

В этом уроке 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 всего остального.

Содержание урока

- Комментарии
- Основной синтаксис
- Переменные
- Операторы
- Присваивание значений переменным
- Многострочные команды

- Типы переменных
- Константы
- Предопределенные константы
- Различие между командами echo и print
- Функции
- Область видимости переменной

Комментарии

Существует два 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
```

Когда понадобится комментарий, состоящий из нескольких строк, нужно воспользоваться **вторым способом** комментирования, который показан ниже:

```
/* 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 [[язык C]] и [[Perl]], но все же больше похожий на [[Java]]. Он очень гибок, но существует несколько правил, относящихся к его синтаксису и структуре, которые следует изучить.

Точка с запятой

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

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

Возможно, чаще всего причиной ошибок, с которыми приходится сталкиваться при работе с PHP, становится 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 ниже:

```
<?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 по вашему усмотрению. В действительности же разумное использование того, что называется свободным пространством, обычно поощряется (наряду с всесторонним 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.

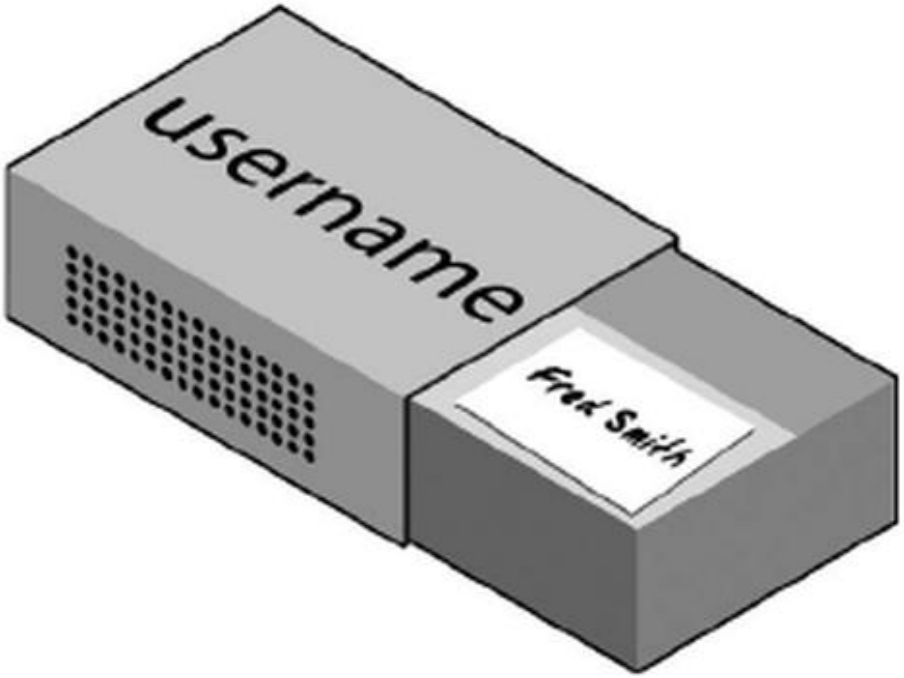
Переменные

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. Этот процесс похож на присваивание переменной строкового значения:

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



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 выглядит следующим образом:

```
echo $username;
```

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

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

Если вы стремитесь самостоятельно освоить работу с PHP, то можете попробовать вводить примеры, приводимые в этом уроке, в интегрированную среду разработки (согласно рекомендациям,

которые были даны ранее, чтобы тут же посмотреть на результаты, или же можете ввести код примера в текстовом редакторе и сохранить этот код в каталоге исходного источника документов вашего сервера под именем test1.php.

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

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

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

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 (см. рисунок ниже). В PHP эквивалентом этому действию будет следующий код:

```
$team = array('Bill', 'Mary', 'Mike', 'Chris', 'Anne');
```

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 кавычки. 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 в матрицу, состоящую из трех строк и трех столбцов.

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().
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 данной книге массивы с размерностью больше двух рассматриваться не будут. Подробнее о массивах мы поговорим позже.

Правила присваивания имен переменным

При создании PHP-переменных следует придерживаться четырех правил.

1. Имена переменных должны начинаться с буквы или с символа _ (подчеркивания).
2. Имена переменных могут содержать только символы: a–z, A–Z, 0–9 и _ (подчеркивание).
3. Имена переменных не должны включать в себя пробелы. Если имя переменной нужно составить более чем из одного слова, то в качестве разделителя следует использовать символ подчеркивания (например, $user_name).
4. 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 — это математические, строковые, логические команды и команды сравнения, такие как «плюс», «минус», «умножить» и «разделить». Код PHP во многом похож 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.

Арифметические операторы

Арифметические операторы проделывают вполне ожидаемую работу. Они применяются для выполнения математических операций. Их можно использовать для проведения четырех основных операций (сложения, вычитания, умножения и деления), а также для нахождения модуля (остатка от деления) и увеличения или уменьшения значения на единицу (см. таблицу).

Таблица. Арифметические операторы

Оператор
Описание
Пример
+
Сложение
$j + 1
–
Вычитание
$j – 6
*****
Умножение
$j * 11
/
Деление
$j / 4
%
Модуль (остаток от деления)
$j % 9
++
Инкремент (приращение)
++$j
——
Декремент (отрицательное приращение)
——$j

Операторы присваивания

Эти операторы используются для присваивания значений переменным. К ним относится самый простой оператор =, а также операторы +=, –= и т. д. (см. таблицу). Оператор += вместо полного замещения находящегося слева значения добавляет к нему значение, которое находится справа от него. Итак, если переменная $count имела начальное значение 5, то оператор:

```
$count += 1;
```

устанавливает значение $count равным 6 точно так же, как более привычный оператор присваивания:

```
$count = $count + 1;
```

Таблица. Операторы присваивания

Оператор
Пример

Эквивалент

=

```
$j = 15
```

$j = 15

+=

```
$j += 5
```

$j = $j + 5

–=

```
$j –= 3
```

$j = $j – 3

***=**

```
$j *= 8
```

$j = $j * 8

/=

```
$j /= 16
```

$j = $j / 16

.=

```
$j .= $k
```

$j = $j . $k

%=

```
$j %= 4
```

$j = $j % 4

У строк есть собственный оператор, точка (.), который более подробно будет рассмотрен в разделе Объединение строк.

Операторы сравнения

Как правило, операторы сравнения используются внутри таких конструкций, как инструкция if, в которых требуется сравнивать значения двух элементов. Например, если необходимо узнать, не достигло ли значение переменной, подвергающееся приращению, какого-то конкретного значения или не превышает ли значение другой переменной установленного значения и т. д..

Таблица. Операторы сравнения

Оператор
Описание
Пример
==
Равно
$j == 4
!=
Не равно
$j != 21
>
Больше

$j > 3
<
Меньше
$j < 100
>=
Больше или равно
$j >= 15
<=
Меньше или равно
$j <= 8
Учтите, что операторы = и == предназначены для разных действий. Если первый является оператором присваивания, то второй – оператором сравнения. Иногда в спешке даже более опытные программисты могут вместо одного из них поставить другой, поэтому будьте внимательны, используя эти операторы.

Логические операторы

Если логические операторы вам раньше не встречались, то поначалу они могут показаться чем-то необычным. Нужно представить, что вы делаете логические заключения на простом разговорном языке.

Например, можно сказать самому себе: «Если время уже больше 12, но меньше 14 часов, значит, нужно пообедать». В PHP код для такого высказывания может выглядеть следующим образом:

```
if ($hour > 12 && $hour < 14) dolunch();
```

Здесь набор инструкций для самого обеда помещен в функцию по имени dolunch, которую позже нужно будет создать. В этой инструкции отсутствует элемент then (тогда), поскольку его присутствие само собой разумеется. Как видно из предыдущего примера, логический оператор обычно используется для объединения результатов работы двух операторов сравнения, показанных в предыдущем разделе. Результат работы одного логического оператора может служить входным значением для другого логического оператора («Если время уже больше 12, но меньше 14 часов или же если в комнате пахнет жареным и тарелки уже стоят на

столе...»). Как правило, если какое-то действие имеет истинное или ложное значение — TRUE или FALSE, оно может служить входным значением для логического оператора, который берет два истинных или ложных входных значения и выдает в качестве результата истинное или ложное значение. Логические операторы показаны в таблице.

Таблица. Логические операторы

Оператор	Описание	Пример
&&	И	$j == 3 && $k == 2
and	Низкоприоритетное И	$j == 3 and $k == 2
||	ИЛИ	$j < 5 || $j > 10
or	Низкоприоритетное ИЛИ	$j < 5 or $j > 10

!
НЕ
! ($j == $k)
xor
Исключающее ИЛИ
$j xor $k
Заметьте, что оператор && обычно взаимозаменяем с оператором and; то же самое справедливо и для операторов || и or. Но у операторов and и or более низкий приоритет, поэтому в некоторых случаях, для того чтобы принудительно расставить приоритеты, могут понадобиться дополнительные круглые скобки. В то же время бывают случаи, когда применимы только операторы and или or, как в следующем предложении, использующем оператор or:

```
mysql_select_db($database)        or
die("Невозможно выбрать базу данных");
```

Наиболее непривычным из этих операторов является xor, предназначенный для операции исключающего ИЛИ, который

возвращает истинное значение TRUE, если любое из входных значений истинно, и возвращает ложное значение FALSE, если оба они имеют значение TRUE или FALSE. Чтобы понять его работу, представьте, что хотите изобрести чистящее средство для дома. Как аммиак (ammonia), так и хлорка (bleach) обладают хорошими чистящими свойствами, поэтому нужно, чтобы ваше средство содержало одно из этих веществ. Но оба они не могут в нем присутствовать, поскольку их сочетание опасно. В PHP это можно представить в следующем виде:

```
$ingredient = $ammonia xor $bleach;
```

В представленном фрагменте, если любая из двух переменных, $ammonia или $bleach, имеет значение TRUE, то значение переменной $ingredient также будет установлено в TRUE. Но если обе они имеют значение TRUE или значение FALSE, то значение переменной $ingredient будет установлено в FALSE.

Присваивание значений переменным

Синтаксис присваивания значения переменной всегда имеет вид переменная = значение. Для передачи значения другой переменной он имеет немного иной вид другая_переменная = переменная.
Есть еще несколько дополнительных операторов присваивания, которые могут оказаться полезными. Например, нам уже встречался оператор:

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

Он предписывает PHP-парсеру добавить значение, расположенное справа от него (в данном случае это значение равно 10), к значению переменной $x. Подобным образом можно вычесть значение:

```
$y -= 10;
```

Увеличение и уменьшение значения переменной на единицу Добавление или вычитание единицы — настолько часто встречающаяся операция, что PHP предоставляет для этого специальные операторы. Вместо операторов
+= и -= можно воспользоваться одним из следующих операторов:

```
++$x;
--$y;
```

В сочетании с проверкой (инструкцией if) можно воспользоваться таким кодом:

```
if (++$x == 10) echo $x;
```

Этот код предписывает PHP сначала увеличить значение переменной $x на единицу, а затем проверить, не имеет ли она значение 10; если переменная имеет такое значение, его следует вывести на экран. Можно также потребовать от PHP увеличить значение переменной на единицу

(или, как в следующем примере, уменьшить на единицу) после того, как ее значение будет проверено:

```
if ($y-- == 0) echo $y;
```

что дает несколько иной результат. Предположим, что первоначальное значение переменной $y до выполнения оператора было равно нулю. Операция сравнения вернет результат TRUE, но после того, как она будет проведена, переменной $y будет присвоено значение –1. Тогда что же отобразит инструкция echo: 0 или –1? Попробуйте догадаться, а потом, чтобы подтвердить свою догадку, испытайте работу инструкции в PHP-процессоре. Поскольку такая комбинация операторов может вас запутать, ее можно применять только в качестве обучающего примера, но ни в коем случае не рассматривать в качестве приемлемого стиля программирования.

Короче говоря, когда именно увеличено или

уменьшено на единицу значение переменной, до или после проверки, зависит от того, где помещен оператор инкремента или декремента — перед именем переменной или после него. Кстати, правильный ответ на предыдущий вопрос таков: инструкция echo отобразит результат –1, потому что значение переменной $y было уменьшено на единицу сразу же после того, как к ней получила доступ инструкция if, и до того, как к ней получила доступ инструкция echo.

Объединение строк

При объединении строк, когда к одной строке символов добавляется другая строка, используется символ точки (.). Самый простой способ объединения строк выглядит следующим образом:

```
echo "У вас " . $msgs . " сообщений.";
```

Если предположить, что переменной $msgs

присвоено значение 5, то эта строка кода выведет следующую информацию:

У вас 5 сообщений.

Так же как с помощью оператора += можно добавить значение к числовой переменной, с помощью оператора .= можно добавить одну строку к другой:

```
$bulletin .= $newsflash;
```

В данном случае, если в переменной $bulletin содержится сводка новостей, а в переменной $newsflash — экстренное сообщение, команда добавляет это сообщение к сводке новостей, и теперь переменная $bulletin включает в себя обе строки текста.

Типы строк

В PHP поддерживаются два типа строк, которые обозначаются типом используемых кавычек. Если требуется присвоить

переменной значение текстовой строки, сохраняя ее точное содержимое, нужно воспользоваться одинарными кавычками (апострофами):

```
$info = 'Предваряйте имена переменных символом $, как в данном примере: $variable';
```

В данном случае переменной $info присваивается каждый символ, находящийся внутри строки в одинарных кавычках. Если воспользоваться двойными кавычками, то PHP попытается вычислить $variable и получить значение переменной. В то же время, если требуется включить в состав строки значение переменной, используется строка, заключенная в двойные кавычки:

```
echo "На этой неделе ваш профиль просмотрело $count человек ";
```

Из этого следует, что данный синтаксис предлагает более простую форму объединения, в которой для добавления

одной строки к другой не нужно использовать символ точки или закрывать и снова открывать кавычки. Этот прием называется подстановкой переменной. Можно заметить, что в некоторых приложениях он используется довольно часто, а в других не применяется вообще.

Изменение предназначения символов

Иногда в строке должны содержаться символы, которые имеют специальное предназначение и могут быть неправильно интерпретированы. Например, следующая строка кода не будет работать, потому что вторая кавычка (апостроф), встреченная в слове spelling's, укажет PHP-парсеру на то, что достигнут конец строки. Следовательно, вся остальная часть строки будет отвергнута как ошибочная:

```
$text = 'My spelling's atroshus'; // Ошибочный синтаксис
```

Для исправления ошибки нужно непосредственно перед вызывающим неоднозначное толкование символом кавычки добавить обратный слеш (\), чтобы заставить PHP рассматривать этот символ буквально и не подвергать его интерпретации:

```
$text = 'My spelling\'s still atroshus';
```

Этот прием можно применить практически во всех ситуациях, где в противном случае PHP вернул бы ошибку, пытаясь интерпретировать символ. Например, следующая строка, заключенная в двойные кавычки, будет присвоена переменной без ошибок:

```
$text = "She wrote upon it, \"Return to sender\".";
```

Кроме того, для вставки в строку различных специальных символов, например табуляции, новой строки и возврата

каретки, могут применяться управляющие символы: \t, \n и \r соответственно. Вот пример, в котором символы табуляции используются для разметки заголовка (они включены в строку исключительно для иллюстрации применения символа обратного слеша, поскольку существуют более подходящие способы разметки веб-страниц):

```
$heading = "Дата\tИмя\tПлатеж";
```

Эти специальные символы, предваряемые символами обратного слеша, работают только в строках, заключенных в двойные кавычки. Если заключить предыдущую строку в одинарные кавычки, то вместо символов табуляции в ней будут отображены нелепые последовательности символов \t. Внутри строк, заключенных в одинарные кавычки, в качестве символов с измененным предназначением распознаются только измененный апостроф (\') и сам измененный обратный слеш (\\).

Многострочные команды

Иногда нужно вывести из PHP большой объем текста, а использование нескольких инструкций echo (или print) заняло бы много времени и было бы неразумным. PHP предлагает два удобных средства, предназначенных для того, чтобы справиться с подобной ситуацией. Первое из них состоит в заключении в кавычки нескольких строк. Переменным также можно присвоить значения способом, показанным в примерах ниже.

```
<?php
$author = "Steve Ballmer";
echo "Developers, Developers, developers,
developers,   developers,developers,
developers, developers, developers! -
$author.";
?>
```

```
<?php
$author = "Bill Gates";
$text = "Measuring programming progress by lines of code is like measuring aircraft building progress by weight. - $author.";
?>
```

В PHP можно также воспользоваться многострочной последовательностью, используя оператор <