Общие сведения о PHP. История языка. Назначение. Примеры задач, решаемых с помощью PHP.

Что такое PHP?

PHP (официальное название "PHP: Hypertext Preprocessor (гипертекстовый процессор)" представляет собой встроенный в HTML язык сценариев.

Простой ответ, но что это значит? Вот пример:

Пример 1-1. Вводный пример

```
<html>
    <head>
        <title>Пример</title>
    </head>
    <body>

        <?php
        echo "Привет, это PHP-скрипт!";
        ?>

    </body>
</html>
```

Обратите внимание на отличие от CGI-скрипта, написанного на другом языке типа Perl или C: вместо написания программы со множеством команд для вывода HTML-кода Вы просто включаете в HTML специальный код, который производит некоторые действия (в данном случае выводит некоторый текст). Код PHP располагается между специальными начальным и конечным тегами, позволяющими входить в режим PHP и выходить из него.

В отличие от клиентского Javascript PHP-скрипты выполняются на сервере. В этом случае клиент просто получит результаты выполнения скрипта, совершенно не зная о том, каким образом эти результаты получены. Можно даже сконфигурировать web-сервер так, чтобы он обрабатывал все HTML-файлы с помощью PHP, тогда пользователи не будут знать, каким образом генерируются страницы.

Что может PHP?

Попросту говоря, PHP может все, что и любая программа CGI - обрабатывать данные форм, генерировать динамические страницы или отправлять и получать cookies.

Пожалуй, самая мощная и важная возможность PHP - поддержка множества различных баз данных. Написать web-страницу с поддержкой баз данных невероятно легко. В настоящее время поддерживаются следующие базы:

Adabas D	Ingres	Oracle (OCI7 и OCI8)
dBase	InterBase	Ovrimos
Empress	FrontBase	PostgreSQL
FilePro (только чтение)	mSQL	Solid

Hyperwave	Direct MS-SQL	Sybase
IBM DB2	MySQL	Velocis
Informix	ODBC	Unix dbm

Кроме того, в PHP имеется поддержка работы с другими службами с использованием протоколов IMAP, SNMP, NNTP, POP3, HTTP и многих других. Можно даже открывать сетевые сокеты и работать с использованием других протоколов.

История PHP

PHP был создан где-то осенью 1994 г. Расмусом Лердорфом. Самые первые версии не распространялись, а использовались у него на домашней странице - контролировали, кто просматривает его резюме. Первая версия, которая использовалась и другими людьми, появилась где-то в начале 1995 гю и называалась Personal Home Page Tools. Он включал простейший синтаксический анализатор, способный анализировать только небольшое количество специальных макросов, и ряд утилит, общепринятых в то время для персональных страниц. Гостевая книга, счетчик и пр. Синтаксический анализатор был переписан в середине 1995 г. и назван PHP/FI Version 2. "FI" пришло из другого пакета, написанного Расмусом для интерпретации данных форм html. Он объединил скрипты Personal Home Page с интерпретатором Form Interpreter и добавил поддержку mSQL - так появился PHP/FI. PHP/FI рос с достойной восхищения скоростью, и многие люди приложили свои усилия к дополнению и совершенствованию его кода.

Привести точную статистику непросто, но приблизительно в конце 1996 гю PHP/FI использовался не менее чем на 15,000 web-сайтов во всем мире. В середине 1997 гю это число превысило 50,000. В середине 1997 г. развитие PHP претерпело изменения. Он перестал быть личным проектом Расмуса, в который множество людей вносили свои дополнения и изменения, над ним начала работать команда. Синтаксический анализатор был переписан от начала до конца Зевом Сураски и Энди Гутмансом, после чего стал основой PHP версии 3. Большая часть кода утилит была перенесена из PHP/FI в PHP 3, и не меньшая часть была полностью переписана.

В последней версии (PHP 4) для повышения производительности используется ядро скриптов Zend, добавлена поддержка еще большего числа библиотек и расширений третьих разработчиков, а PHP выполняется как "родной" серверный модуль на всех популярных web-серверах.

На сегодняшний день (1/2001) PHP 3 или PHP 4 поставляется с рядом коммерческих продуктов, например, с web-сервером Red Hat's Stronghold. По самым скромным оценкам, основанным на экстраполяции данных Netcraft (см. также Netcraft Web Server Survey) PHP используется на 5,100,000 сайтов во всем мире. Это существенно превышает число сайтов, использующих сервер Microsoft IIS (5.03 миллиона).

Основной синтаксис языка PHP

Отделение от HTML

Имеется четыре способа отделения кода HTML от PHP:

Пример. Способы отделения от HTML

```
1.  <? echo ("это самый простой - инструкция обработки SGML\n"); ?>

2.  <?php echo("если Вы хотите работать с документами XHTML или XML, делайте
так\n"); ?>

3.  <script language="php">
        echo ("некоторые редакторы (типа FrontPage) не
              любят инструкции обработки");
    </script>

4.  <% echo ("Можно использовать и теги в стиле ASP"); %>
    <%= $variable; # Это сокращение для "<%echo .." %>
```

Первый способ можно использовать, только если активизированы сокращенные теги. Это достигается установкой параметра конфигурации short_open_tag в конфигурционном файле PHP или компиляцией PHP с параметром --enable-short-tags для **configure**.

Второй способ обычно предпочтителен, он обеспечивает поддержку PHP в следующем поколении XHTML.

Четвертый способ можно использовать, если с помощью параметра конфигурации asp_tags активизированы теги в стиле ASP.

Закрывающий тег для блока будет включать символ перехода на новую строку, если он есть.

Разделение инструкций

Инструкции отделяются друг от друга так же, как в языках C или perl - каждое выражение заканчивается точкой с запятой.

Закрывающий тег (?>), ероме того, подразумевает конец выржения, так что следующие строки эквивалентны:

```
<?php
    echo "Это тест";
?>

<?php echo "Это тест" ?>
```

Комментарии

В PHP поддерживаются комментарии в стиле C, C++ и Unix shell. Например:

```
<?php
    echo "Это тест"; // Комментарий в одну строку в стиле c++
    /* Многострочный комментарий
       вторая строка коментария */
    echo "Это еще один текст";
    echo "Последний тест"; # Комментарий в стиле shell
?>
```

Действие однострочного комментария распространяется до конца строки или до конца текущего блока кода PHP, в зависимости от того, что раньше встретится в тексте.

```
<h1>Вот <?php # echo "простой";?> пример.</h1>
<p>Описанный выше заголовок будет отображаться так: 'Вот пример'.
```

Избегайте вложения комментариев в стиле языка C при комментировании больших блоков.

```
<?php
 /*
    echo "Это тест"; /* Такой комментарий вызовет проблему */
 */
?>
```

Основные операторы PHP. Примеры использования операторов.

Список типов операторов:

- Арифметические операторы
- Операторы присваивания
- Побитовые операторы
- Операторы сравнения
- Операторы контроля ошибок
- Операторы выполнения
- Операторы инкремента/декремента
- Логические операторы
- Приоритет операторов
- Строковые операторы

Арифметические операторы

Помните основы арифметики? Так же работают арифметические операторы.

Пример	**Название**	**Результат**
$a + $b	Сложение	Сумма $a и $b.
$a - $b	Вычитание	Разность $a и $b.
$a * $b	Умножение	Произведение $a и $b.
$a / $b	Деление	Частное $a и $b.
$a % $b	Остаток	Остаток от деления $a на $b.

Оператор деления ("/") возвращает целое значение (результат деления целх чисел), если оба операланда являются целыми (или строками, преобразуемыми в целый числа), и частное также является целым числом. Если какой-либо из операндов представляет собой значение с плавающей точкой или результат операции не является целым число, возвращается значение с плавающей точкой.

Операторы присваивания

Основным оператором присваивания является оператор "=". Скорее всего, Вы воспримете его как знак равенства. Не нужно так делать. На самом деле он означает, что значение левого операнда устаналивается равным значению выражения справа (то есть "присваивается").

Значением выражения присвоения являетя присвоенное значение. То есть значением выражения "$a = 3" будет 3. Это позволяет проделывать некоторые интеренсые вещи:

```
$a = ($b = 4) + 5; // $a сейчас равно 9, а $b - 4.
```

Помимо основного оператора присваивания существуют так называемые "объединенные операторы" для всех операторов двоичной арфиметики и строковых операторов, позволяющие использовать значение в выражении, а затем присвоить значение результату выражения. Например:

```
$a = 3;
$a += 5; // значение $a устанавливается равным 8, как если бы мы записали: $a
= $a + 5;
$b = "Hello ";
$b .= "There!"; // устаналивает значение $b равным "Hello There!", точно так
же как оператор $b = $b . "There!";
```

Обратите внимание, что оператор присваивания копирует исходную переменную в новую (присвоение по значению), так что изменения одной переменной не повлияют на другую. Это уместно, если Вам нужна копия, скажем, большого массива в большом цикле. PHP 4 поддерживает присвоение по ссылке с использованием синтаксиса `$переменная = &$другая_переменная;`, но в PHP 3 это невозможно. 'Присвоение по ссылке' означает, что обе переменные будут указывать на одни и те же данные, и ничего никуда не копируется. Подробнее о ссылках см. в разделе о ссылках.

Побитовые операторы

Побитовые операторы позволяют устанавливать и сбрасывать отдельные биты в целом числе.

Пример	**Название**	**Результат**
$a & $b	И	Устанавливаются биты, установленные одновременно в $a и $b.
$a | $b	Или	Устанавливаются биты, установленные либо в $a, либо в $b.
$a ^ $b	Исключающее	Устанавливаются биты, установленные в $a или $b, но не в

Пример	Название	Результат
	или	обеих переменных одновременно.
~ $a	Не	Устанавливаются биты, не установленные в $a и наоборот.
$a << $b	Сдвиг влево	Сдвиг битов переменной $a на $b шагов влево (каждый шаг означает "умножение на два")
$a >> $b	Сдвиг вправо	Сдвиг битов переменной $a на $b шагов вправо (каждый шаг означает "деление на два")

Операторы сравнения

Операторы сравнения, как и следует из их названия, позволяют сравнить два значения.

Пример	Название	Результат
$a == $b	Равно	Принимает занчение "истина", если $a равно $b.
$a === $b	Идентично	Принимает значение "истина", если $a равно $b, и обе переменные имеют один и тот же тип. (Только в PHP 4)
$a != $b	Не равно	Принимает занчение "истина", если $a не равно $b.
$a !== $b	Не идентично	Принимает значение "истина", если $a не равно $b или переменные имеют разные типы. (Только в PHP 4)
$a < $b	Меньше чем	Принимает занчение "истина", если $a строго меньше $b.
$a > $b	Больше чем	Принимает занчение "истина", если $a строго больше $b.
$a <= $b	Меньше или равно	Принимает занчение "истина", если $a меньше или равно $b.
$a >= $b	Больше или равно	Принимает занчение "истина", если $a больше или равно $b.

Еще одним уловным оператором является оператор "?:" (тернарный), который действует тк же, как в языке C и многих других языках.

```
(expr1) ? (expr2) : (expr3);
```

Значение выражения становится равным *expr2*, если *expr1* истинно и *expr3*, если *expr1* ложно.

Операторы контроля ошибок

В PHP поддерживается один оператор контроля ошибок: коммерческое at (@). Если этот оператор предшествует выражению PHP, все сообщения об ошибках, которые могли бы быть сгенерированы этим выражением, будут игнорироваться.

Если активизирован параметр track_errors, все сообщения об ошибках, генерируемые данным выражением, будут сохранены в глобальной переменной $php_errormsg. Эта переменная перезаписывается с каждой новой ошибкой, поэтому, если хотите ее использовать, проверяйтее ее сразу же.

Операторы выполнения

В PHP поддерживается один оператор контроля выполнения: обратные апострофы (``). Обратите внимание, что это не одинарыне кавычки! PHP пытается выполнить заключенные в обратные апострофы команды как шелловые; выходные данные будут возвращены (т.е. они не будут просто дампироваться в вывод; они могут быть присвоены какой-либо переменной).

Операторы инкремента/декремента

В PHP поддерживаются операторы пре- и пост-инкремента и декремента в стиле языка C.

Пример	**Название**	**Действие**
++$a	Пре-инкремент	Увеличивает $a на единицу и возвращает $a.
$a++	Пост-инкремент	Возвращает $a, затем увеличивает значение $a на единицу.
--$a	Пре-декремент	Уменьшает $a на единицу и возвращает $a.
$a--	Пост-декремент	Возвращает $a, затем уменьшает значение $a на единицу.

Логические операторы

Пример	**Название**	**Результат**
$a and $b	И	Истина, если и $a, и $b истинны.
$a or $b	Или	Истина, если либо $a, либо $b истинны.
$a xor $b	Исключающее или	Истина, если истинно либо $a, либо $b, но не оба сразу.
! $a	Не	Истина, если $a ложно.
$a && $b	И	Истина, если и $a, и $b истинны.
$a || $b	Или	Истина, если либо $a, либо $b истинны.

Строковые операторы

Имеется два строковых оператора. Первый - оператор конкатенации строк ('.'), возвращающий объединение своих левого и правого аргументов. Второй - оператор конкатенации с присвоением ('.='), добавляющий правый аргумент к левому. Подробнее см. в разделе Операторы присваивания.

```
$a = "Hello ";
$b = $a . "World!"; // сейчас переменная $b содержит "Hello World!"

$a = "Hello ";
$a .= "World!";     // сейчас переменная $a содержит "Hello World!"
```

Операторы Include () b require()

Оператор **require()** заменяется на указанный файл, аналогично директиве препроцессора `#include` в языке C.

Подробнее см. разделы Файлы на удаленном сервере и **fopen()**.

Следует отметить, что если файл указан в операторе **include()** или **require()**, синтаксический анализ в начале целевого файла выходит из режима PHP и переходит в режим HTML. Режим PHP восстанавливается в конце. По этой причине исполняемый код в целевом файле должен заключаться в допустимые начальные и конечные теги PHP.

require() на самом дле не является функцией PHP; это скорее языковая конструкция. Она подчиняется правилам, отличным от тех, которым подчиняются функции. Например, **require()** не зависит от управляющих структур. Еще пример: она н евозвращает значения; попытка прочесть значение из функции **require()** приведет к ошибке анализа.

В отличие от **include()**, **require()** *всегда* считывает целевой файл, *даже если строка, в которой он находится, никогда не выполняется*. Если Вам необходимо включать файл в зависимости от условия, используйте оператор **include()**. Условный оператор не влияет на оператор **require()**. Однако, если строка, в которой располагается оператор **require()**, не выполняется, не будет выполняться и код из целевого файла.

Аналогично не влияют на поведение оператора **require()** циклические структуры. Хотя код, содержащийся в целевом файле, по-прежнему зависит от цикла, сам оператор **require()** действует только один раз.

Это означает, что нельзя поместить оператор **require()** в цикл и ожидать, что на каждой итерации будет включаться содержимое нового файла. Для этого нужно использовать оператор **include()**.

```
require ('header.inc');
```

Если файл указан в поераторе **require()**, содержащийся в нем код наследует область действия переменных строки, в которой расположен оператор **require()**. Все переменные, доступные в этой строке вызывающего файла, будут доступны и в вызывемом файле. Если оператор **require()** в вызывающем файле располагается в функции, то есь код, содержащийся в вызываемом файле, будет действовать так, как если бы он был определен в функции.

Если файл, указанный в операторе **require()**, вызывается по протоколу HTTP с использованием fopen wrappers и целевой сервер интерпретирует целевой файл как код PHP, в файл, указанный в операторе **require()**, с помощью строки запроса URL как в запросе HTTP GET могут передаваться переменные. Строго говоря, это не равнозначно помещению файла в оператор **require()** и установке наследования им области действия переменных родительского файла; скрипт на самом деле выполняется на удаленном сервере, а результат затем включается в локальный скрипт.

```
/* В данном примере предполагается, что someserver сконфигурирован для
анализа файлов .php
 * и не сконигурирован для анализа файлов .txt. Здесь 'работать' означает
доступность переменных
 * $varone и $vartwo в файле, указанном в операторе require(). */

/* Не будет работать; file.txt не обрабатывается someserver. */
require ("http://someserver/file.txt?varone=1&vartwo=2");

/* Не будет работать; поиск файла с именем 'file.php?varone=1&vartwo=2'
 * в локальной файловой системе. */
require ("file.php?varone=1&vartwo=2");
```

```
/* Работает. */
require ("http://someserver/file.php?varone=1&vartwo=2");

$varone = 1;
$vartwo = 2;
require ("file.txt");  /* Hf,jnftn. */
require ("file.php");  /* Работает. */
```

В PHP 3 файле, включенном в оператор **require()**, можно выполнять оператор `return`, если этот оператор находится в глобальной области действия файла. включенного в оператор **require()**. Он не должен заключаться в какой-либо блок (т.е. находиться в фигурных скобках ({}). В PHP 4, однако, эта возможность исключена. Если Вам она необходима, обратитесь к оператору **include()**.

Переменные в PHP. Объявление переменных, типы переменных.

Переменные в PHP представляются в виде строки, которая начинается знаком доллра, а за ним следует имя переменной. Имена переменные учитывают регистр.

Имена переменных подчиняются тем же правилам, что и любые метки в PHP. Допустимое имя переменной начинается с буквы или символа подчеркивания, за которым может следовать любой количество букв, цифр или символов подчеркивания. В виде регулярного выражения это можно представить так: '[a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]*'

Примечание: ЗДесь для удобства буквами считаются символы a-z, A-Z и символы набора ASCII 127 - 255 (0x7f-0xff).

```
$var = "Bob";
$Var = "Joe";
echo "$var, $Var";      // вывод: "Bob, Joe"

$4site = 'not yet';     // недопустимо; начинется с цифры
$_4site = 'not yet';    // допустимо; начинается с символа подчеркивания
$täyte = 'mansikka';    // допустимо; 'ä' имеет код ASCII 228.
```

В PHP 3 переменные всегда назначались по значению. То есть при назначении переменой выражения все его значение копировалось в переменную назначения. Это, в частности, означает, что после присвоения значения одной переменой другой изменение значения одной из переменных не влияло на другую. Подробнее см. в разделе Выражения.

В PHP 4 предлагается другой способ присвоения переменным выражений: *назначение по ссылке*. Это означает, что новая переменная просто ссылается (другими словами, "становится псевдонимом" или "указывает") на исходную переменную. Изменение новой переменной влияет на исходную и наоборот. Кроме того, это означает, что копирования не происходит; за счет этого присвоение происходит быстрее. Однако это ускорение можно будет заметить только в очень длинных циклах или при назначении очень больших массивов объектов.

Для присвоения по ссылке просто поставьте перед именем присваемой (исходной) переменной амперсанд (&). Например, следующий код выведет фразу: 'My name is Bob' дважды:

```
<?php
$foo = 'Bob';              // Значение 'Bob' присваивается переменной $foo
$bar = &$foo;          // Ссылка на $foo в переменой $bar.
$bar = "My name is $bar";  // Изменение переменной $bar...
echo $foo;                 // переменная $foo также изменилась.
echo $bar;
?>
```

Следует помнить одну важную вещь: присвоение по ссылке действует только для именованных переменных.

```
<?php
$foo = 25;
$bar = &$foo;      // Допустимое присвоение.
$bar = &(24 * 7);  // Недопустимо; ссылка на неименованное выражение.

function test() {
   return 25;
}

$bar = &test();    // недопустимо.
?>
```

Типы переменных

В PHP поддерживаются следующие типы данных:

- массив
- число с плавающей точкой
- целое число
- объект
- строка

Обычно проргаммист не устанавливает тип переменной; тип определяется PHP во время выполнения скрипта в зависимости от контекста, в котором используется перменная.

Если Вы хотите преобразовать переменную к конерктному типу, можно привести ее или использовать функцию **settype()**.

Обратите внимание. что переменная в различных ситуациях может вести себя по-разному, в зависимости от того, какой тип она сейчас имеет. Подробнее см. в разделе о преобразовании типов.

Целые

Целые числа задаются с использованием следующего синтаксиса:

```
$a = 1234; # десятичное число
$a = -123; # отрицательное число
```

```
$a = 0123; # восьмеричное число (эквивалнетно десятичному 83)
$a = 0123; # шестнадцатеричное число (эквивалнетно десятичному 18)
```

Размер целого цисла зависит от платформы, обычно максимлаьным может быть знаечение около 2 миллиардов (32 бита со знаком).

Числа с плавающей точкой

Числа с плавающей точкой ("числа двойной точности") задаются с использованием следующего синтаксиса:

```
$a = 1.234; $a = 1.2e3;
```

Размер числа с плавающей точкой зависит от платформы, обычно максимальным может быть число ~1.8e308 с точностью около 14 десятичных цифр (64-битный формат IEEE).

Строки

Строки могут задаваться с использованием одного или двух наборов разделителей.

Если строка заключена в двойные кавычки ("), переменные в строке будут раскрываться (в соответствии с некоторыми ограничениями синтаксического разбора). Как и в C и Perl, при указании специальных символов может использоваться обратный слеш ("\"):

последовательность	**значение**
`\n`	перевод строки (LF или 0x0A в кодировке ASCII)
`\r`	возврат каретки (CR или 0x0D в кодировке ASCII)
`\t`	горизонтальная табуляция (HT или 0x09 в кодировке ASCII)
`\\`	обратный слеш
`\$`	знак доллара
`\"`	двойная кавычка
`\[0-7]{1,3}`	последовательность символов, соответствующая регулярному выражению, является символом в восьмеричной нотации
`\x[0-9A-Fa-f]{1,2}`	последовательность символов, соответствующая регулярному выражению, является символом в шестнадцатеричной нотации

Если Вы попытаетесь представить таким образом другой символ, выведены будут и обратный слеш, и сам символ. В PHP 3 при этом будет выдано предупреждение на уровне `E_NOTICE`. В PHP 4 предупреждающее сообщение не генерируется.

Второй способ отделения строк - использование одиночных кавычек ("'"). Если строка заключается в одинарные кавычки, в ней можно использовать только символы "\\" и "\'". Таикм образом Вы смоежет указывать одинарные кавычки и обратные слеши в строка, отделенной одинарными кавычками. Переменные в строке, отделенной одинарными кавычкамИ, разворачиваться *не* будут.

Еще один способ разделения строк - с помощью синтаксиса документов ("<<<"). После <<< укажите идентификатор, затем строку, а затем тот же самый идентификатор, чтобы закрыть "кавычки".

Закрывающий идентификатор *должен* начинатья в первом столбце строки. Кроме того, идентификтаор должен удовлетворять правилам именвоания, как и любая метка в PHP: он должен содержать только алфавитно-цифровые символы и подчеркивания и н едолжен начинаться с цифры или подчеркивания.

Такой текст ведет себя как строка в двойных кавычках, но без двойных кавычек. Это означает, что Вам не придется кодировать кавычки с помощью слеша, но использовать коды со слешем все равно можно. Переменные раскрываются, но следует проявлять осторожность при выржении сложных переменных в таикх строках.

Массивы

Массивы обычно действуют как хэш-таблицы (ассоциативные массивы) и индексированные массивы (векторы).

Одномерные массивы

PHP поддерживает скалярные и ассоциативные массивы. Между ними фактичесик нет никакой разницы. Создать массив можно с помощью функций **list()** или **array()** или путем явного определения значений элементов массива.

```
$a[0] = "abc";
$a[1] = "def";
$b["foo"] = 13;
```

Можно создать массив и путем простого добавления в него элементов. При присвоении значения переменной массива с пустыми скобками это значение просто добавляется в конец массива.

```
$a[] = "hello"; // $a[2] == "hello"
$a[] = "world"; // $a[3] == "world"
```

Сортировтаь массивы можно с помощью функций **asort()**, **arsort()**, **ksort()**, **rsort()**, **sort()**, **uasort()**, **usort()** и **uksort()** в зависимости от необходимого типа сортировки.

Подсчитать число элементов в массиве можно с помощью функции **count()**.

Перемещаться по массиву можно с помощью функций **next()** и **prev()**. Еще один способ перемещения по массиву - использование функции **each()**.

Многомерные массивы

Многомерные массивы на самом деле очень просты. Для каждого измерения масива в конце добавляется другой [ключ]:

```
$a[1]      = $f;               # примеры одномерных массивов
$a["foo"]  = $f;
```

```
$a[1][0]     = $f;             # двумерный массив
$a["foo"][2] = $f;             # (числовые и ассоциативные индексы можно
смешивать)
$a[3]["bar"] = $f;             # (числовые и ассоциативные индексы можно
смешивать)

$a["foo"][4]["bar"][0] = $f;   # четырехмерный массив
```

Объявление функций в PHP. Параметры функций, возвращаемые значения.

Функции, определяемые пользователем

Функция может определяться с использованием следующего синтаксиса:

```
function foo ($arg_1, $arg_2, ..., $arg_n) {
    echo "Пример функции.\n";
    return $retval;
}
```

Функция может заключать в себе любой допустимый код PHP, даже другие функции и определения классов.

В PHP 3 функции должны определяться до ссылки на них. В PHP 4 такого ограничения нет.

PHP не поддерживает перегрузку функций, отмена определения или переопределение ранее объявленных функций также невозможны.

Аргументы функций

Передавать в функции информацию можно с помощью списка аргументов, представляющего собой разделенный запятыми список переменных и/или констант.

В PHP поддерживается передача аргументов по значению (по умолчанию), передача по ссылке и значения аргументов по умолчанию. Списки аргументов переменной длины поддерживаются только в PHP 4 и выше, см. раздел Списки аргументов переменной длины и поисания функций **func_num_args()**, **func_get_arg()** и **func_get_args()**. Аналогичного эффекта можно достичь в PHP 3, передавая в функцию массив аргументов:

```
function takes_array($input) {
    echo "$input[0] + $input[1] = ", $input[0]+$input[1];
}
```

Значения аргументов по умолчанию

Для скалярных аргументов в функции могут определяться значения по умолчанию, аналогично тому, как это делается в C++:

```
function makecoffee ($type = "капуччино") {
    return "Сделаем чашечку $type.\n";
}
echo makecoffee ();
echo makecoffee ("эспрессо");
```

Этот фрагмент кода даст такой вывод:

```
Сделаем чашечку капуччино.
Сделаем чашечку эспрессо.
```

Значение по умолчанию должно содержать постоянное выражение и не может содержать переменную или член класса.

Обратите внимание, что при использовании аргументов по умолчанию все значения по умолчанию должны находиться правее аргументов, не использующих значения по умолчанию; в противном случае все будет работать не так, как нужно. Рассмотрим следующий пример:

```
function makeyogurt ($type = "ацидофильного", $flavour) {
    return "Сделаем тарелочку $type $flavour.\n";
}

echo makeyogurt ("клубничного");   // не будет работать, как ожидается
```

Этот фрагмент кода даст такой вывод:

```
Warning: Missing argument 2 in call to makeyogurt() in
/usr/local/etc/httpd/htdocs/php3test/functest.html on line 41
Сделаем тарелочку клубничного .
```

Сравните:

```
function makeyogurt ($flavour, $type = "ацидофильного") {
    return "Сделаем тарелочку $type $flavour.\n";
}

echo makeyogurt ("клубничного");   // работает, как ожидалось
```

Этот фрагмент кода даст такой вывод:

```
Сделаем тарелочку ацидофильного клубничного.
```

Возвращаемые значения

Значения возвращаются с помощью необязательного оператора return. Могут возвращаться значения всех типов. включая списки и объекты.

```
function square ($num) {
    return $num * $num;
}
echo square (4);   // вывод '16'.
```

Функция не может возвращать несколько значений, но этого результата можно достичь путем возвращения списка.

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

Чтобы функция возвращала ссылку, следует использовать оператор ссылки & в объявлении функции и при присвоении возвращаемого значения переменной:

```
function &returns_reference() {
    return $someref;
}

$newref =&returns_reference();
```

Операторы условий в PHP. Примеры использования.

if

Конструкция `if` является одной из наиболее важных функций многих языков, включая PHP. Она обеспечивает выполнение фрагментов кода в соответствии с условием. В PHP структура `if` сходна с аналогичной структурой языка C:

```
if (expr)
    утверждение
```

Как описано в разделе о выражениях, оценивается истинность значения выражения expr. Если значением выражения *`expr`* является `TRUE`, PHP выполнит утверждение, в случае `FALSE` устверждение будет проигнорировано.

В этом примере на экран будет выведено `a больше b`, если *`$a`* больше *`$b`*:

```
if ($a > $b)
    print "a больше b";
```

Зачастую нужно, чтобы при определенном условии выполнялось несколько утверждений. Разумеется, нет необходимости заключать каждое из таких утверждений в выражение `if`. Вместо этого можно сгруппировтаь несколько утверждений. Например, следующий фрагмент кода выведет `a больше b`, если *`$a`* больше *`$b`*, а затем присвоит значение *`$a`* переменной *`$b`*:

```
if ($a > $b) {
    print "a больше b";
    $b = $a;
}
```

Утверждения if могут до бесконечности вкладываться в другие утверждения `if`, что обеспечивает гибкость выполнения различных частей программы в зависимости от условий.

else

Часто бывает нужно выполнить одно утверждение, если определенное условие выполнено, и другое утверждение, если условие не выполнено. Для этого используется конструкция `else`. `else` дополняет утверждение `if` утверждением, выполняемым в случае, если выражение в утверждении `if` имеет значение `FALSE`. Например, следующий фрагмент кода выведет `a больше b`, если *`$a`* больше *`$b`*, а в противном случае - `a НЕ больше b`:

```
if ($a > $b) {
    print "a больше b";
} else {
    print "a НЕ больше b";
}
```

Утверждение `else` выполняется только если выражение `if` имеет значение `FALSE`, и, если имеются выражения `elseif` - только если они также имеют значение `FALSE` (см. elseif).

elseif

`elseif`, как следует из названия, представляет собой комбинацию `if` и `else`. Как и `else`, это выражение позволяет в утверждении `if` выполнить другие утверждения, если значением первого выражения `if` является `FALSE`. Однако в отличие от `else` авлтернативное выражение будет выполнено, только если значение условного выражения `elseif` равно `TRUE`. Например, следующий код выведет `a больше b`, `a равно b` или `a is меньше b`:

```
if ($a > $b) {
    print "a больше b";
} elseif ($a == $b) {
    print "a равно b";
} else {
    print "a меньше b";
}
```

В одном утверждении `if` может быть несколько `elseif`. Выполняется первое выражение `elseif` (если оно имеется), значением которого является `true`. В PHP можно писать и 'else if' (в два слова) - действие такого утверждения идентично действию 'elseif' (в одно слово). Синтаксическое значение слегка отличается (если Вы знакомы с языком C, там все происходит так же), но в результате оба утверждения дадут одинаковый эффект.

Утверждение `elseif` выполняется, только если предшествующее выражение `if` и все предшествующие выражения `elseif` имели значение `FALSE`, а текущее выражение `elseif` имеет значение `TRUE`.

Альтернативный синтаксис управляющих структур

Для некоторых управляющих структур PHP, а именно для `if`, `while`, `for`, `foreach` и `switch` имеется альтернативный синтаксис. В каждом случае основной формой альтернативного синтаксиса является замена открывающей скобки на двоеточие (:) и закрывающей скобки на `endif;`, `endwhile;`, `endfor;`, `endforeach;` или `endswitch;` соответственно.

```
<?php if ($a == 5): ?>
A равно 5
<?php endif; ?>
```

В приведенном выше примере блок HTML "A = 5" вложен в утверждение `if`, записанное с использоваинем альтернативного синтаксиса. Блок HTML будет выводиться на экран, только если значение переменной $a равно 5.

Альтернативный синтаксис также применяется к `else` и `elseif`. Далее показана структура `if` с `elseif` и `else` в альтернативном формате:

```
if ($a == 5):
    print "a равно 5";
    print "...";
elseif ($a == 6):
    print "a равно 6";
    print "!!!";
else:
    print "a не ранво ни 5, ни b";
endif;
```

while

Цикл `while` - простейший тип цикла в PHP. Они работают точно так же. как и в языке C. Основная форма выражения `while` такая:

```
while (выражение) утверждение
```

Значение утверждения `while` просто. Оно дает PHP указание выполнять вложенные утверждения до тех пор, пока выражение `while` не примет значения `TRUE`. Значение выражения проверяется каждый раз в начале цикла, так что, даже если оно изменяется во вермя выполнения вложенных выражений, выполнение не прекратится до тех пор, пока не будет выполнена вся итерация целиком (итеарцией называется каждое выполнение утверждений цикла). Иногда если значением выражения `while` является `FALSE` с первой же итерации, вложенные утверждения ни буду выполняться ни разу.

Как и в утверждении `if`, в цикле `while` можно группировать несколько утверждений, объединяя их фигурными скобками или используя альтернативный синтаксис:

```
while (выражение): утверждение ... endwhile;
```

Следующие примеры идентичны и оба выводят на печать цифры от 1 до 10:

```
/* пример 1 */

$i = 1;
while ($i <= 10) {
    print $i++;  /* будет печататься значение
                    $i до увеличения
                    (пост-инкремент) */
}

/* пример 2 */

$i = 1;
while ($i <= 10):
    print $i;
    $i++;
endwhile;
```

switch

Оператор `switch` напоминает ряд операторов IF в одном выражении. Часто бывает необходимо сравнить одну и ту же переменную (или выражение) с несоклкьими различными значениями и в зависимости то результата выполнить разные фрагменты кода. Именно для этого и предназначен оператор `switch`.

Следующие два примера демонстрируют два различных способа записи одного и того же, в одном исопльзуется ряд операторов `if`, а в дургом - оператор `switch`:

```
if ($i == 0) {
    print "i равно 0";
}
if ($i == 1) {
    print "i равно 1";
}
if ($i == 2) {
    print "i равно 2";
}

switch ($i) {
    case 0:
        print "i равно 0";
        break;
    case 1:
        print "i равно 1";
        break;
    case 2:
        print "i равно 2";
        break;
}
```

Во избежание ошибок важно понимать, как выполняется оператор `switch`. Оператор `switch` выполняется построчно (то есть оператор за оператором). Сначала не выполняется

никакой код. PHP начинает выполнять операторы только после обнаружения оператора `case` со значением, соответствующим значению выражения `switch`. PHP продолжает выполнение операторов до конца блока оператора `switch` или до первого выражения `break`. Если в конце списка выражений для одного случая отсутствует оператор `break`, PHP продолжит выполнения выражений следующего оператора case.

Операторы циклов в PHP. Примеры использования.

do..while

Циклы `do..while` очень похожи на циклы `while` с те лишь исключением, что истинность выражения првоеряется не в начале каждой итерации, а в конце. Основное отличие от обычных циклов `while` состоит в том. что первая итерация цикла `do..while` выполнится обязательно (истинность выражения првоеряется только в конце итерации), в то время как обычный цикл `while` может не выполниться ни разу (истинность выражения проверяется в начале каждой итерации, и если с самого начала значением выражения является `FALSE`, выполнение цикла сразу же прекращается).

Для циклов `do..while` существует только один синтаксис:

```
$i = 0;
do {
   print $i;
} while ($i>0);
```

Приведенный выше цикл будет выполнен ровно один раз, поскольку после первой итерации значением выражения будет `FALSE` ($i не больше 0), и выполнение цикла прекратится.

Опытные пользователи языка C, наверное, знакомы с другим способом использования цикла `do..while`, который позволяет прекратить выполнение в середине блоков кода, путем заключения их в цикл `do..while`(0) и использования оператора `break`. Это демонстрирует следующий фрагмент кода:

```
do {
    if ($i < 5) {
        print "i недостаточно велико";
        break;
    }
    $i *= $factor;
    if ($i < $minimum_limit) {
        break;
    }
    print "i уже ничего так";

     ...обработка i...

} while(0);
```

Не беспокойтесь, если Вы сейчас не понимаете этого и даже если не понимаете вообще. Писать достаточно мощные скрипты можно и без этой возможности.

for

Циклы `for` - самые сложные в PHP. Они работают так же, как и в языке C. Синтаксис циклов `for` таков:

```
for (expr1; expr2; expr3) statement
```

Знчение первого выражения (*expr1*) определяется один раз без каких бы то ни было условий в начале цикла.

В начале каждой итерации определяется значение *expr2*. Если оно равно `TRUE`, выполнение цикла продолжается, и выполняются все вложенные операторы. Если оно равно `FALSE`, выполнение цикла прекращается.

В конце каждой итерации определяется значение *expr3*.

Все выражения могут быть пустыми. Пустое выражение *expr2* означает бесконечное выполнение цикла (PHP неявно считет его значением `TRUE`, как и в C). Это не настлько бесполезно, как Вы могли бы подумать, поскольку часто закончить выоплнение цикла можно с помощью условного оператора `break`, а не с помощью выражения `for`.

Рассмотрим следующие примеры: Все они выводят цифры от 1 до 10:

```
/* пример 1 */

for ($i = 1; $i <= 10; $i++) {
    print $i;
}

/* пример 2 */

for ($i = 1;;$i++) {
    if ($i > 10) {
        break;
    }
    print $i;
}

/* пример 3 */

$i = 1;
for (;;) {
    if ($i > 10) {
        break;
    }
    print $i;
    $i++;
}

/* пример 4 */

for ($i = 1; $i <= 10; print $i, $i++) ;
```

Конечно, первый прмер кажется самым лучшим (или, по крайней мере. четвертый), но возможность использования пустых выражений в циклах `for` часто бывает очень удобна.

В PHP для циклов `for` поддерживается и альтернативный синтаксис.

```
for (expr1; expr2; expr3): statement; ...; endfor;
```

В других языках для прохождения по массиву или хэшу имеется оператор `foreach`. В PHP 3 такой конструкции нет, но в PHP 4 есть (см. foreach). В PHP 3 для достижения того же эффекта можно объединять while с функциями **list()** и **each()**. Примеры см. в документации по этим инструкциям.

foreach

В PHP 4 (в PHP 3 нет) имеется конструкция `foreach`, аналогичная конструкции perl и некоторых других языков. Она предоставляет удобный способ работы с массивами. Имеется два синтаксиса; второй является немного расширенной версией первого:

```
foreach(array_expression as $value) statement
foreach(array_expression as $key => $value) statement
```

Первая форма обеспечивает перемещение по по массиву, задаваемому `array_expression`. На каждой итерации значение текущего элемента присваивается переменной `$value`, а внутренний указатель массива смешается на один элемент (так что на следующей итерации будет использоваться следующий элемент).

break

Оператор `break` прерывает выполнение текущей структуры `for`, `while`, or `switch`.

Оператор `break` может имтеь необязательный цифровой аргумент, сообщающий о количестве прерываемых вложенных структур.

```
$arr = array ('one', 'two', 'three', 'four', 'stop', 'five');
while (list (, $val) = each ($arr)) {
    if ($val == 'stop') {
        break;    /* Здесь можно записать: 'break 1;'. */
    }
    echo "$val<br>\n";
}

/* Использование необязательного аргумента. */

$i = 0;
while (++$i) {
    switch ($i) {
    case 5:
        echo "At 5<br>\n";
        break 1;  /* Выход только из оператора switch. */
    case 10:
        echo "At 10; quitting<br>\n";
        break 2;  /* Выход из операторов switch и while. */
    default:
        break;
    }
}
```

continue

Оператор `continue` используется в циклах для пропуска последующих операторов текущей итерации цикла и продолжения выполнения с начал следующей итерации.

Оператор `continue` может имтеь необязательный цифровой аргумент, сообщающий о количестве уровней вложенного цикла.

```
while (list ($key, $value) = each ($arr)) {
    if (!($key % 2)) { // пропустить четсные числа
        continue;
    }
    do_something_odd ($value);
}

$i = 0;
while ($i++ < 5) {
    echo "Outer<br>\n";
    while (1) {
        echo "  Middle<br>\n";
        while (1) {
            echo "  Inner<br>\n";
            continue 3;
        }
        echo "Этот вариант не дает вывода\n";
    }
    echo "И этот тоже.<br>\n";
}
```