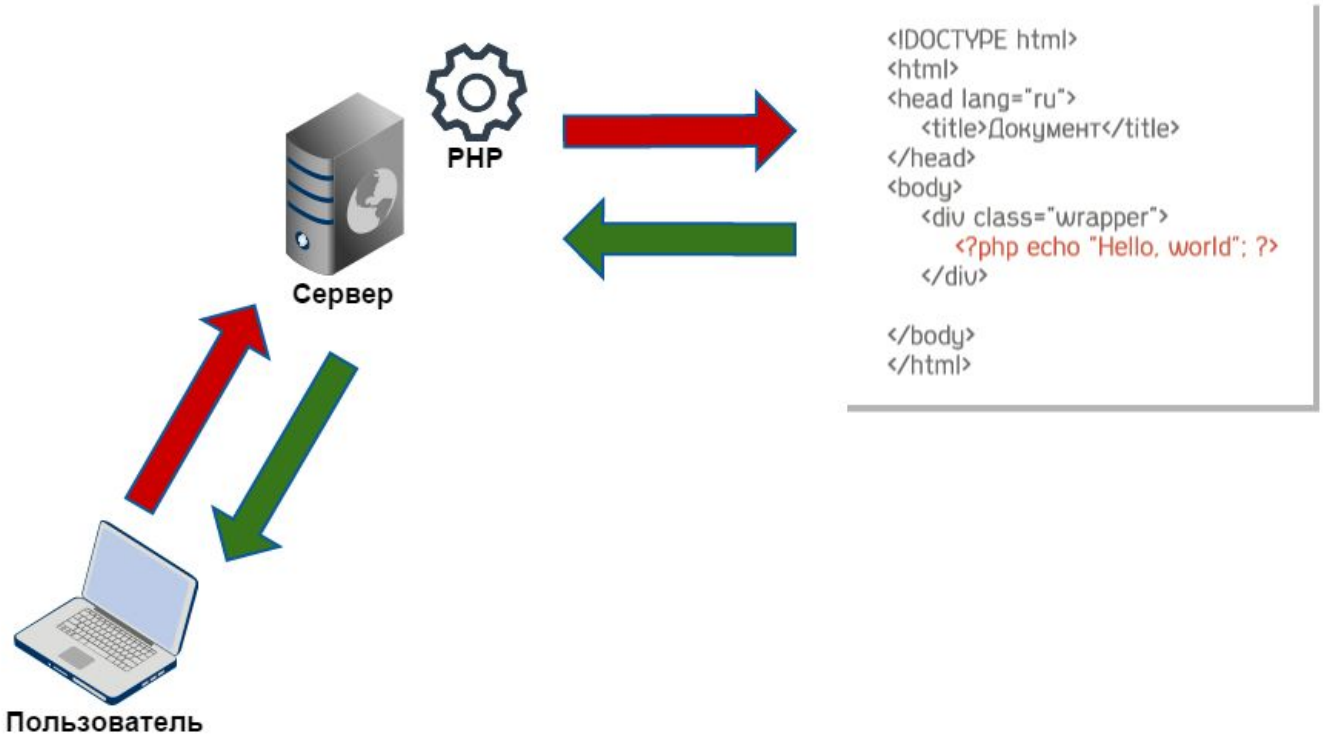
Методические указания по теме
“Основы PHP”

Введение

PHP (Hypertext Preprocessor) - скриптовый язык программирования общего назначения, интенсивно применяемый для разработки веб-приложений. В настоящее время поддерживается подавляющим большинством хостинг-провайдеров и является одним из лидеров среди языков программирования, применяющихся для создания динамических веб-сайтов. По полярности применения занимает 6-е место (на март 2015) среди всех языков программирования.

Принцип работа PHP:



1. Пользователь делает запрос на удаленный сервер
2. Если запрос был сделан на HTML файл, то сервер ищет его на у себя на жестком диске. Если находит, то сразу же его возвращает, если нет, то возвращает ошибку.
3. Если запрос был сделан на PHP страницу, то она передается интерпретатору PHP, который ищет в коде страницы PHP код и обрабатывает. После обработки он возвращает страницу обратно серверу, который, в свою очередь, отдает ее клиенту.

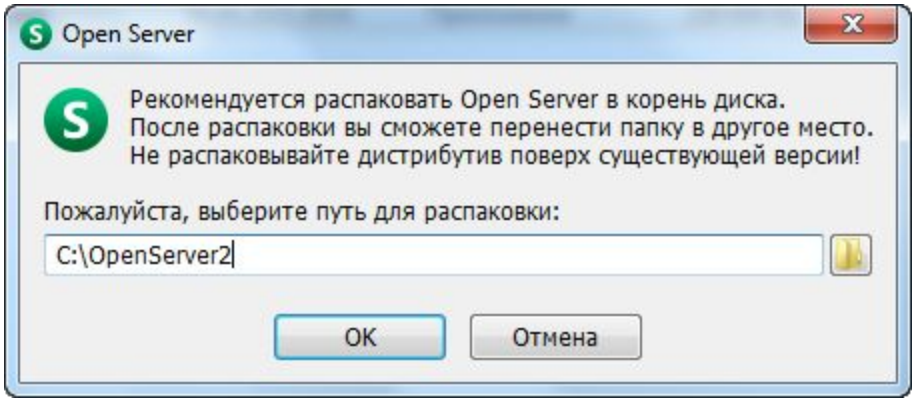
Локальный веб-сервер

На сегодняшний день существует достаточно широкий выбор продуктов, с помощью которых можно реализовать локальный веб-сервер на своем компьютере, например:

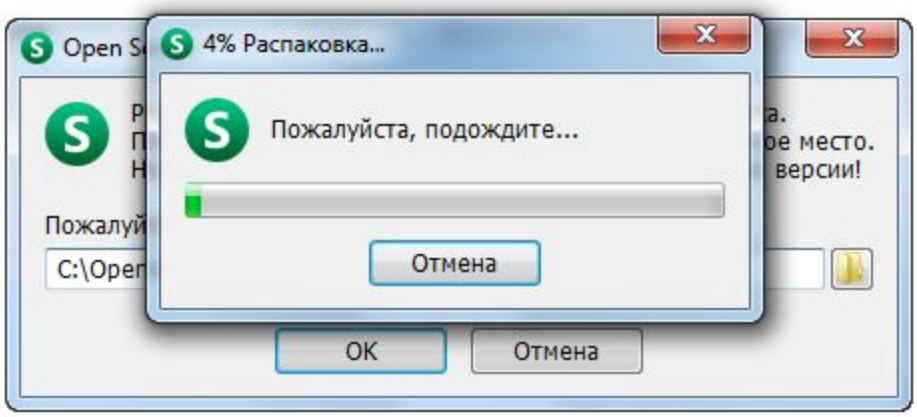
1. Open Server - http://open-server.ru/
2. Denwer - http://www.denwer.ru/
3. XAMPP - https://www.apachefriends.org/ru/index.html
4. Wamp Server - http://www.wampserver.com/ru/
5. Easy PHP - http://www.easyphp.org/
6. MAMP - https://www.mamp.info/en/
7. Winginx - http://winginx.com/ru/

Рассмотрим пример установки в качестве локального сервера пакет Open Server.

Шаг 1. Необходимо скачать дистрибутив пакета с официального сайта.
Шаг 2. Запускаем инсталятор пакета.
Шаг 3. Выбираем местоположение, куда будем устанавливать пакет.



Шаг 4. Далее, ждем несколько минут, пока пакет распакуется в выбранную нами папку на нашем жестком диске

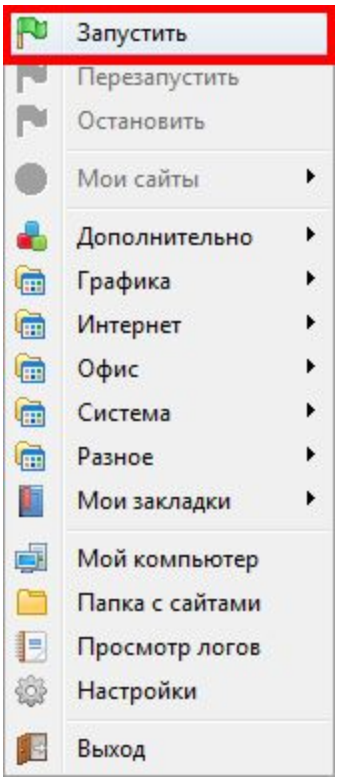


Шаг 5. Для создания проекта на Open Server, вам необходимо перейти в папку с установленным Open Server, в папку domains и создать в ней папку, например ***test-application.local*****, и создадим в ней файл index.php с текстом**

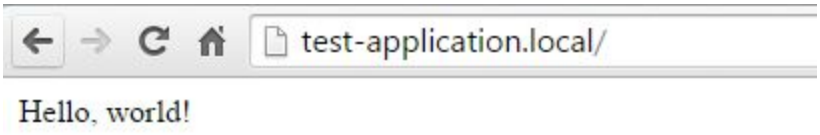
```
<?php
echo "Hello, world!";
```

шаг 6. Запускаем Open Server кликнув дважды по ярлыку программы в папке, где пакет был установлен. В трее должен появиться красный флажок программы

Кликаем по нему правой кнопкой мыши и выбираем пункт “Запустить”



После этого переходим в наш браузер, вводим адрес http://test-application.local/ и получаем вот такой результат

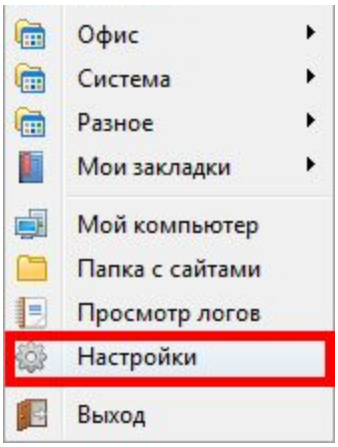


Настройка Open Server

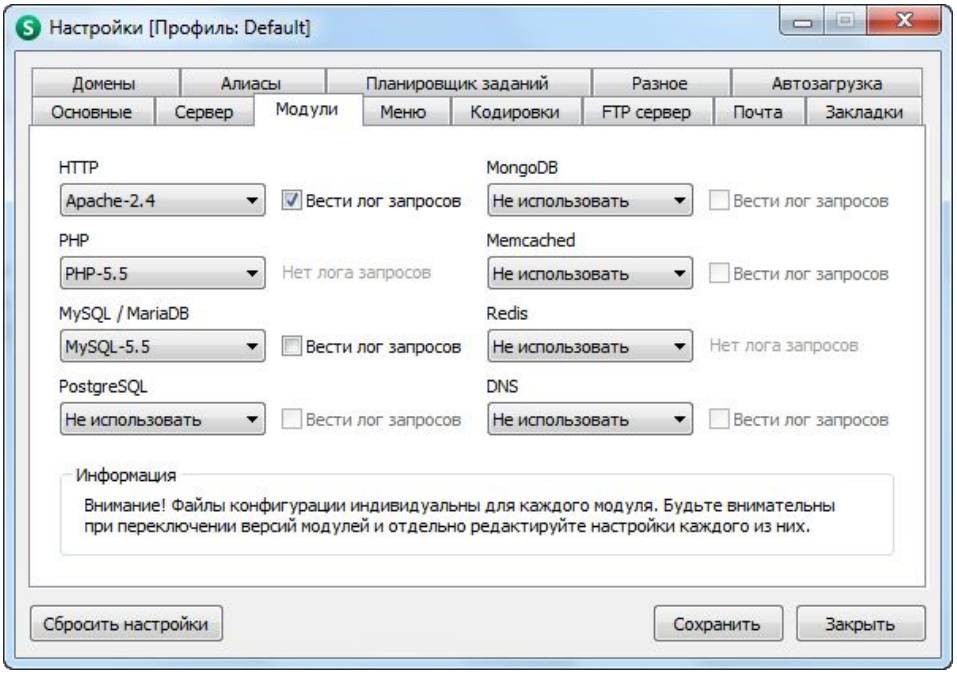
Настройки интерпретатора PHP в Open Server нахдится в папке **path/to/**modules/php/**version/**php.ini

где **path/to/** - папка с установленным Open Server
version - версия языка, установленная в настройках PHP

Для изменения настроек Open Server необходимо нажать на ярлык Open Server и выбрать пункт “Настройки”.



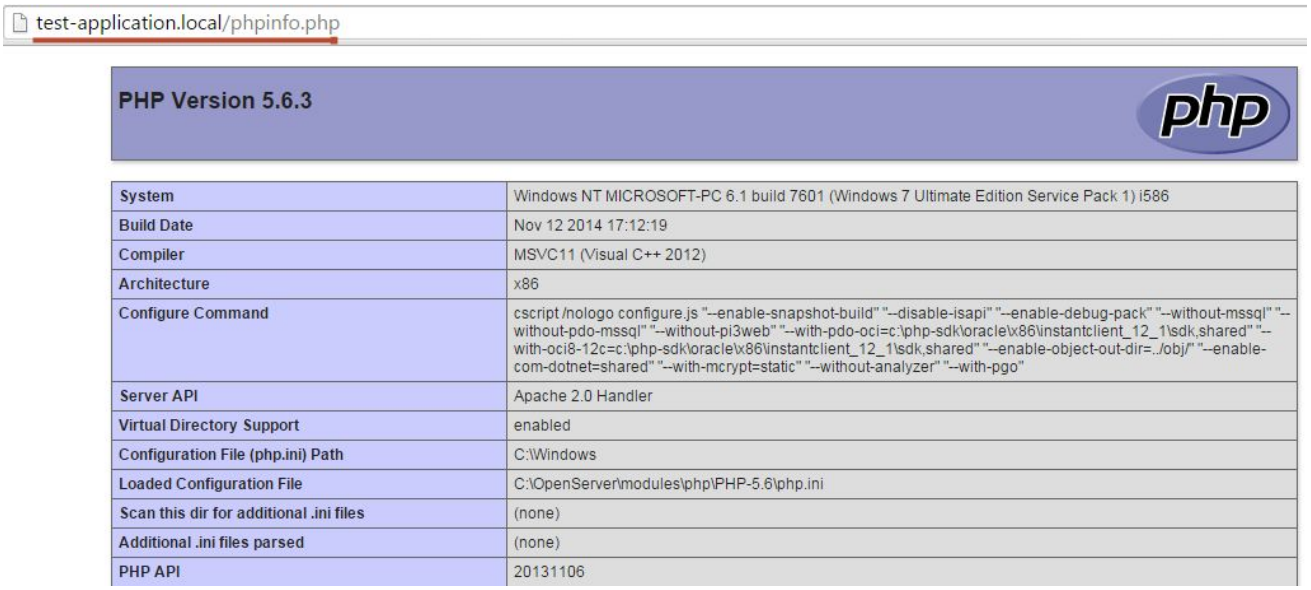
В появившемся окне выбрать вкладку “Модули” и из выпадающего списка выбрать версию PHP или любого другого модуля



Для просмотра настроек интерпретатора создайте файл phpinfo.php и добавьте в него следующий код:

```
<?php
    phpinfo();
```

Открыв браузер для http://test-application.local/phpinfo.php



System	Windows NT MICROSOFT-PC 6.1 build 7601 (Windows 7 Ultimate Edition Service Pack 1) i586
Build Date	Nov 12 2014 17:12:19
Compiler	MSVC11 (Visual C++ 2012)
Architecture	x86
Configure Command	cscript /nologo configure.js "--enable-snapshot-build" "--disable-isapi" "--enable-debug-pack" "--without-mssql" "--without-pdo-mssql" "--without-pi3web" "--with-pdo-oci=c:\php-sdk\oracle\x86\instantclient_12_1\sdk,shared" "--with-oci8-12c=c:\php-sdk\oracle\x86\instantclient_12_1\sdk,shared" "--enable-object-out-dir=../obj/" "--enable-com-dotnet=shared" "--with-mcrypt=static" "--without-analyzer" "--with-pgo"
Server API	Apache 2.0 Handler
Virtual Directory Support	enabled
Configuration File (php.ini) Path	C:\Windows
Loaded Configuration File	C:\OpenServer\modules\php\PHP-5.6\php.ini
Scan this dir for additional .ini files	(none)
Additional .ini files parsed	(none)
PHP API	20131106

Основы синтаксиса PHP

Теги PHP

PHP код может быть вставлен в документ с помощью следующих тегов:

Пример №1
```
<script language="php">
    echo "Hello, world";
</script>
```

Пример №2
```
<?php
    echo "Hello, world";
?>
```

Пример №3
```
<?
    echo "Hello, world";
?>
```

Данный пример будет работать при условии включения настройки **short_open_tag** в конфигурации php (файле php.ini)

Пример №4
```
<%
    echo "Hello, world";
%>
```
Данный пример будет работать при условии включения настройки **asp_tags** в конфигурации php (файле php.ini)

Комментарии

В PHP поддерживает комментарии в стиле 'C', 'C++' и оболочки Unix (стиль Perl)

```
echo "Это тест"; // Это однострочный комментарий в стиле c++
/* Это многострочный комментарий
еще одна строка комментария */
echo "Это еще один тест";
echo "Последний тест"; # Это тоже однострочный комментарий
/*
echo "Это тест"; /* Этот комментарий вызовет проблему */
*/
```

Особенности языка

Любой скрипт на языке PHP состоит из последовательности инструкций. Каждая из них отделяется друг от друга символом точки запятой “;”. Например

```
<?php какая-то инструкция;?>

<?php инструкция 1; инструкция 2; ?>

<?php
какая-то
инструкция
в
несколько
строк;
?>
```

Вывод данных на экран

В языке существует несколько конструкций, для вывода информации на экран браузера:

1. echo - `echo "Привет мир!";`
2. print - `print("Привет мир!");`
3.
```
<?php $num = 2.12;
 printf("%.1f",$num); ?>
```

Переменные

Переменная - это область оперативной памяти, доступ к которой осуществляется по имени. Все данные, с которыми работает программа, хранятся в виде переменных (исключение — константа, которая, впрочем, может содержать только число или строку). Такого понятия, как указатель (как в Си), в PHP не существует — при присвоении переменная копируется один-в-один, какую бы сложную структуру она ни имела.

При создании и работе переменными в PHP необходимо учитывать следующие факторы:

- Переменные в PHP начинаются со знака доллара ($)
- Имя переменной должно начинаться с буквы или символа подчеркивания
- Последующие символы в имени переменной могут быть буквами, цифрами или символом

- подчеркивания в любом количестве
- Имя переменной чувствительно к регистру

Пример:
```
<?php
setlocale(LC_ALL, "ru_RU");
$day = strftime('%d');
$mon = strftime('%B');
$year = strftime('%Y');

echo 'Сегодня ', $day, ' число, ', $mon, ' месяц, ', $year, ' год.';
```

Копирование и ссылки на переменные

Следует различать процесс копирования переменной и установления ссылки на нее. При копировании переменной копируется ее значение переменной. Если меняется значение скопированной переменной, то это никак не отражается на второй переменной. Если мы устанавливаем ссылку на переменную, то при изменении значении одной переменной, будем меняться и другая переменная.

Например (копирование переменной):

```
<?php

$x = 10;
Создаем копию переменной $x
$y = $x;
$y = 20;
echo $x; // 10
echo $y; // 20
```

Например (ссылка на переменную):

```
$x = 10;
Создаем ссылку на переменную $x
$y =& $x;
$y = 20;
echo $x; // 20
echo $y; // 20
```

Переменные переменных

В PHP существует возможно вычислять имена переменных динамически. Такой прием называется переменные переменных.
```
<?php
```

```
$a = 'hello';
$$a = 'world';
// hello world
echo $a, ' ' , $hello;
$a = 'name';
$$a = 'Вася';
// Вас зовут: Вася
echo 'Вас зовут: ', $name;
```

Типы данных

PHP не требует (и не поддерживает) явного определения типа при объявлении переменной; тип переменной определяется согласно контексту, в котором она используется. То есть, если вы присвоите строковое значение переменной $var, $var станет строкой. Если вы затем присвоите $var целочисленное значение, она станет целым числом.

В PHP поддерживаются следующие типы данных:

Четыре скалярных типа:
- boolean
- integer
- float (число с плавающей точкой)
- string

Два смешанных типа:
- array
- object

Два специальных типа:
- resource
- NULL

Двоичные данные (Boolean)

Это простейший тип. Он выражает истинность значения - это может быть либо TRUE, либо FALSE. Чтобы определить булев тип, используйте ключевое слово TRUE или FALSE. Оба регистро-независимы.

Для преобразования значения в булев тип используйте приведение типа (**bool**) или (**boolean**). Однако в большинстве случаев вам нет необходимости использовать приведение типа, поскольку значение будет автоматически преобразовано, если оператор, функция или управляющая конструкция требует булев аргумент.

При преобразовании в логический тип, следующие значения рассматриваются как FALSE:

- Сам булев FALSE
- целое 0 (ноль)
- число с плавающей точкой 0.0 (ноль)
- пустая строка и строка "0"
- массив с нулевыми элементами
- объект с нулевыми переменными-членами
- специальный тип NULL (включая неустановленные переменные)

Все остальные значения рассматриваются как TRUE.
Внимание! -1 считается TRUE, как и любое ненулевое (отрицательное или положительное) число!

Целые числа (Integer)

Целое - это число из множества Z = {..., -2, -1, 0, 1, 2, ...}, обычно длиной 32 бита (от –2 147 483 648 до 2 147 483 647).
Целые могут быть указаны в десятичной, шестнадцатеричной или восьмеричной системе счисления, по желанию, с предшествующим знаком (- или +).
Если вы используете восьмеричную систему счисления, вы должны предварить число 0 (нулем), для использования шестнадцатеричной системы нужно поставить перед числом **0x**.

Если вы определите число, превышающее пределы целого типа, оно будет интерпретировано как число с плавающей точкой. Также, если вы используете оператор, результатом работы которого будет число, превышающее пределы целого, вместо него будет возвращено число с плавающей точкой.

Для несомненного преобразования значения в целое используйте приведение типа (int) или (integer). Однако в большинстве случаев вам нет необходимости использовать приведение типа, поскольку значение будет автоматически преобразовано, если оператор, функция или управляющая конструкция требует целый аргумент. Вы также можете преобразовать значение в целое при помощи функции intval().

Пример использования:

```
<?php
        $a = 1234; // десятичное число
        $a = -123; // отрицательное число
        $a = 0123; // восьмеричное число (эквивалентно 83 в десятичной системе)
        $a = 0x1A; // шестнадцатеричное число (эквивалентно 26 в десятичной системе)
```

Числа с плавающей точкой

Double - вещественное число довольно большой точности (ее должно хватить для подавляющего большинства математических вычислений).

Числа с плавающей точкой (они же числа двойной точности или действительные числа) могут быть определены при помощи любого из следующих синтаксисов:

Пример использования:

```
<?php
$a = 1.234;
$b = 1.2e3;
$c = 7E-10;
```

Строки

Строка в PHP - это набор символов любой длины. В отличие от Си, строки могут содержать в себе также и нулевые символы, что никак не повлияет на программу. Иными словами, строки можно использовать для хранения бинарных данных. Длина строки ограничена только размером свободой оперативной памяти.

Простейший способ определить строку - это заключить ее в одинарные кавычки (символ ').

Чтобы использовать одинарную кавычку внутри строки, как и во многих других языках, ее необходимо предварить символом обратной косой черты (\), т. е. экранировать ее. Если обратная косая черта должна идти перед одинарной кавычкой либо быть в конце строки, вам необходимо продублировать ее.

Пример использования:

```
<?
$a = "Это просто текст, записанный в строковую переменную";
echo $a; //Выводит 'Это просто текст, записанный в строковую переменную'
```

Массивы

Массивы (arrays) - это упорядоченные наборы данных, представляющие собой список однотипных элементов.

Существует два типа массивов, различающиеся по способу идентификации элементов.

1. В массивах первого типа элемент определяется индексом в последовательности. Такие массивы называются простыми массивами.
2. Массивы второго типа имеют ассоциативную природу, и для обращения к элементам используются ключи, логически связанные со значениями. Такие массивы называют ассоциативными массивами.

Важной особенностью PHP является то, что PHP, в отличие от других языков, позволяет создавать массивы любой сложности непосредственно в теле программы (скрипта).

Массивы могут быть как одномерными, так и многомерными.

Простые массивы и списки в PHP

При обращении к элементам простых индексируемых массивов используется целочисленный индекс, определяющий позицию заданного элемента.

Простые одномерные массивы:

Обобщенный синтаксис элементов простого одномерного массива:

$имя[индекс];
Массивы, индексами которых являются числа, начинающиеся с нуля - это списки:

```
<?php
// Простой способ инициализации массива
$names[0]="Апельсин";
$names[1]="Банан";
$names[2]="Груша";
$names[3]="Помидор";
// Здесь: names - имя массива, а 0, 1, 2, 3 - индексы массива
?>
```

Доступ к элементам простых массивов (списков) осуществляется следующим образом:

```
<?php
// Простой способ инициализации массива
$names[0]="Апельсин";
$names[1]="Банан";
$names[2]="Груша";
$names[3]="Помидор";
// Здесь: names - имя массива, а 0, 1, 2, 3 - индексы массива

// Выводим элементы массивов в браузер:
echo $names[0]; // Вывод элемента массива names с индексом 0
echo "<br>";
echo $names[3]; // Вывод элемента массива names с индексом 3
// Выводит:
// Апельсин
// Помидор
?>
```

С технической точки зрения разницы между простыми массивами и списками нет.

Простые массивы можно создавать, не указывая индекс нового элемента массива, это за вас сделает PHP. Вот пример:

```
<?php
```

```
// Простой способ инициализации массива, без указания индексов
$names[]="Апельсин";
$names[]="Банан";
$names[]="Груша";
$names[]="Помидор";
// PHP автоматически присвоит индексы элементам массива, начиная с 0

// Выводим элементы массивов в браузер:
echo $names[0]; // Вывод элемента массива names с индексом 0
echo "<br>";
echo $names[3]; // Вывод элемента массива names с индексом 3
// Выводит:
// Апельсин
// Помидор
?>
```

В рассмотренном примере вы можете добавлять элементы массива names простым способом, то есть не указывая индекс элемента массива:

```
$names[]="Яблоко";
```

Новый элемент простого массива (списка) будет добавлен в конец массива. В дальнейшем, с каждым новым элементом массива, индекс будет увеличиваться на единицу.

Простые многомерные массивы:

Обобщенный синтаксис элементов многомерного простого массива:

```
$имя[индекс1][индекс2]..[индексN];
```

Пример простого многомерного массива:

```
<?php
// Многомерный простой массив:
$arr[0][0]="Овощи";
$arr[0][1]="Фрукты";
$arr[1][0]="Абрикос";
$arr[1][1]="Апельсин";
$arr[1][2]="Банан";
$arr[2][0]="Огурец";
$arr[2][1]="Помидор";
$arr[2][2]="Тыква";

// Выводим элементы массива:
echo "<h3>".$arr[0][0].":</h3>";
for ($q=0; $q<=2; $q++) {
echo $arr[2][$q]."<br>";
}
echo "<h3>".$arr[0][1].":</h3>";
```

```
for ($w=0; $w<=2; $w++) {
echo $arr[1][$w]."<br>";
}
?>
```

Ассоциативные массивы в PHP

В PHP индексом массива может быть не только число, но и строка. Причем на такую строку не накладываются никакие ограничения: она может содержать пробелы, длина такой строки может быть любой.

Ассоциативные массивы особенно удобны в ситуациях, когда элементы массива удобнее связывать со словами, а не с числами.

Итак, массивы, индексами которых являются строки, называются ассоциативными массивами.

Одномерные ассоциативные массивы:

Одномерные ассоциативные массивы содержат только один ключ (элемент), соответствующий конкретному индексу ассоциативного массива. Приведем пример:

```
<?php
// Ассоциативный массив
$names["Иванов"]="Иван";
$names["Сидоров"]="Николай";
$names["Петров"]="Петр";
// В данном примере: фамилии - ключи ассоциативного массива
// , а имена - элементы массива names
?>
```

Доступ к элементам одномерных ассоциативных массивов осуществляется так же, как и к элементам обыкновенных массивов, и называется доступом по ключу:

```
echo $names["Иванов"];
```

Многомерные ассоциативные массивы:

Многомерные ассоциативные массивы могут содержать несколько ключей, соответствующих конкретному индексу ассоциативного массива. Рассмотрим пример многомерного ассоциативного массива:

```
<?php
// Многомерный массив
$A["Ivanov"] = array("name"=>"Иванов И.И.", "age"=>"25", "email"=>"ivanov@mail.ru");
$A["Petrov"] = array("name"=>"Петров П.П.", "age"=>"34", "email"=>"petrov@mail.ru");
$A["Sidorov"] = array("name"=>"Сидоров С.С.", "age"=>"47", "email"=>"sidorov@mail.ru");
```

```
?>
```

Доступ к элементам многомерного ассоциативного массива осуществляется следующим образом:

```
echo $A["Ivanov"]["name"]; // Выводит Иванов И.И.
echo $A["Petrov"]["email"]; // Выводит petrov@mail.ru
```

Как вы уже заметили, для создания многомерного ассоциативного массива мы использовали специальную функцию array, мы ее рассмотрим позже, когда будем рассматривать операции над массивами.

Ассоциативные многомерные массивы можно создавать и классическим способом, хотя это не так удобно:

```
<?php
// Многомерный ассоциативный массив
$A["Ivanov"]["name"]="Иванов И.И.";
$A["Ivanov"]["age"]="25";
$A["Ivanov"]["email"]="ivanov@mail.ru";

$A["Petrov"]["name"]="Петров П.П.";
$A["Petrov"]["age"]="34";
$A["Petrov"]["email"]="petrov@mail.ru";

$A["Sidorov"]["name"]="Сидоров С.С.";
$A["Sidorov"]["age"]="47";
$A["Sidorov"]["email"]="sidorov@mail.ru";

// Получаем доступ к ключам многомерного ассоциативного массива
echo $A["Ivanov"]["name"]."<br>"; // Выводит Иванов И.И.
echo $A["Sidorov"]["age"]."<br>"; // Выводит 47
echo $A["Petrov"]["email"]."<br>"; // Выводит petrov@mail.ru
?>
```

Специальный тип Resource

Ресурс - это специальная переменная, содержащая ссылку на внешний ресурс. Ресурсы создаются и используются специальными функциями. Поскольку тип ресурс содержит специальные указатели на открытые файлы, соединения с базой данных, область изображения и тому подобное, вы не можете преобразовать какое-либо значение в ресурс.

Специальный тип Null (пустой тип)

Специальное значение NULL говорит о том, что эта переменная не имеет значения. NULL - это единственно возможное значение типа NULL.

- Переменная считается NULL если
- ей была присвоена константа NULL.
- ей еще не было присвоено какое-либо значение.
- она была удалена с помощью unset().

Константы

Константой называется именованная величина, которая не изменяется в процессе выполнения программы (скрипта).

В отличие от переменных, вы не можете изменять значения констант, которые были им присвоены при их объявлении. Константы удобно использовать для хранения значений, которые не должны изменяться во время работы программы. Константы могут содержать только скалярные данные (логического, целого, плавающего и строкового типов).

В PHP константы определяются функцией define(). Эта функция имеет следующий формат:

define ($name, $value, $case_sen), где:

$name - имя константы;
$value - значение константы;
$case_sen - необязательный параметр логического типа,
указывающий, следует ли учитывать регистр букв (true) или нет (false).

Если параметр $case_sen равен true, то интерпретатор будет учитывать регистр символов при работе с константой. Обратите внимание, что константы используются без предваряющего знака $.

Различия между константами и переменными:

- У констант нет приставки в виде знака доллара ($);
- Константы можно определить только с помощью функции define(), а не присваиванием значения;
- Константы могут быть определены и доступны в любом месте без учета области видимости;
- Константы не могут быть определены или аннулированы после первоначального объявления; и
- Константы могут иметь только скалярные значения.

Проверка существования констант

Для проверки существования константы можно использовать функцию defined(). Данная функция возвращает true, если константа объявлена.

Пример:

```
<?php
```

```
// Объявляем константу pi
define("pi",3.14,true);
if (defined("pi")==true) echo "Константа pi объявлена!";
// Скрипт выведет 'Константа pi объявлена!'
?>
```

Операторы PHP. Общие сведения

Для осуществления операций с переменными существуют различные группы операторов.

Оператором называется нечто, состоящее из одного или более значений (выражений, если говорить на жаргоне программирования), которое можно вычислить как новое значение (таким образом, вся конструкция может рассматриваться как выражение). Отсюда следует, что функции или любые другие конструкции, которые возвращают значение (например, print()) являются операторами, в отличие от всех остальных языковых конструкций (например, echo()), которые ничего не возвращают.

Арифметические операторы

Пример	Название	Результат
-$a	Отрицание	Смена знака $a
$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 от деления $a на $b
$a % $b	Деление по модулю	Целочисленный остаток от деления $a на $b

Строковый оператор

Оператор конкатенации ('.') возвращает объединение левого и правого аргумента.

Пример:

```
<?php
$a = "Hello ";
$b = $a . "World!";
// $b теперь содержит строку "Hello World!"
$a = "Hello"; $b = "World!";
$c = $a . " " . $b;
// $c теперь содержит строку "Hello World!"
$c = "$a $b"; // $c тоже "Hello World!"
```

Комбинированные операторы

В дополнение к базовому оператору присваивания имеются "комбинированные операторы" для всех бинарных арифметических и строковых операций, которые позволяют использовать некоторое значение в выражении, а затем установить его как результат данного выражения.

```
<?php
$a = 3;
$a += 5; // устанавливает $a в 8, как если бы
мы написали: $a = $a + 5;
$b = "Hello ";
$b .= "World!"; // устанавливает $b в "Hello
World!", как и $b = $b . "World!";
```

Побитовые операторы

Пример	Название	Результат
$a & $b	И	Устанавливаются только те биты, которые установлены и в $a, и в $b
$a | $b	Или	Устанавливаются только те биты, которые установлены и в $a, или в $b
$a ^ $b	Исключающее или	Устанавливаются только те биты, которые установлены либо в $a, либо в $b
~ $a	Отрицание	Устанавливаются те биты, которые не установлены в $a, и наоборот
$a << $b	Сдвиг влево	Все биты переменно $a сдвигаются на $b позиций влево
$a >> $b	Сдвиг вправо	Все биты переменно $a сдвигаются на $b позиций вправо

Примеры:

```
$result = 1 & 5; // $result = 1
$result = 1 | 5; // $result = 5
$result = 1 ^ 5; // $result = 4
$result = 4 >> 1; // $result = 2
$result = 4 << 1; // $result = 8
```

Условные операторы

Условные операторы являются, пожалуй, наиболее распространенными конструкциями во всех алгоритмических языках программирования. Рассмотрим основные условные операторы языка PHP.

Конструкция if

Синтаксис конструкции if аналогичен конструкции if в языке Си:

```
<?php
if (логическое выражение) оператор;
?>
```

Согласно выражениям PHP, конструкция if содержит логическое выражение. Если логическое выражение истинно (true), то оператор, следующий за конструкцией if будет исполнен, а если логическое выражение ложно (false), то следующий за if оператор исполнен не будет.

Пример:

```
<?php
if ($a > $b) echo "значение a больше, чем b";
?>
```

Конструкция else

Часто возникает потребность исполнения операторов не только в теле конструкции if, если выполнено какое-либо условие конструкции if, но и в случае, если условие конструкции if не выполнено. В данной ситуации нельзя обойтись без конструкции else. В целом, такая конструкция будет называться конструкцией if-else.

Синтаксис конструкции if-else такой:

```
if (логическое_выражение)
инструкция_1;
else
инструкция_2;
```

Действие конструкции if-else следующее: если логическое_выражение истинно, то выполняется инструкция_1, а иначе — инструкция_2. Как и в любом другом языке, конструкция else может опускаться, в этом случае при получении должного значения просто ничего не делается.

Если инструкция_1 или инструкция_2 должны состоять из нескольких команд, то они, как всегда, заключаются в фигурные скобки. Например:

```
<?php
if ($a > $b) {
    echo "a больше, чем b";
} else {
    echo "a НЕ больше, чем b";
}
?>
```

Конструкция if-else имеет еще один альтернативный синтаксис:

```
if (логическое_выражение):
        команды;
elseif(другое_логическое_выражение):
        другие_команды;
else:
        иначе_команды;
endif
```

Конструкция elseif

elseif - это комбинация конструкций if и else. Эта конструкция расширяет условную конструкцию if-else.
Приведем синтаксис конструкции elseif:

```
if (логическое_выражение_1)
        оператор_1;
elseif (логическое_выражение_2)
        оператор_2;
else
        оператор_3;
```

Практический пример использования конструкции elseif:

```
<?php
if ($a > $b) {
    echo "a больше, чем b";
} elseif ($a == $b) {
    echo "a равен b";
} else {
    echo "a меньше, чем b";
}
?>
```

Полезные функции: isset и empty

if (isset($var))

- Устанавливает, определена ли переменная
- Возвращает TRUE, если переменная определена; FALSE в противном случае

if (empty($var))

- Определяет, считается ли переменная пустой
- Возвращает FALSE, если переменная является непустой и ненулевым значением

Таблица сравнения типов

Выражение	gettype()	empty()	isset()	boolean : *if($x)*
`$x = "";`	string	TRUE	TRUE	FALSE
`$x = null;`	NULL	TRUE	FALSE	FALSE
`$x` неопределена	NULL	TRUE	FALSE	FALSE
`$x = array();`	array	TRUE	TRUE	FALSE
`$x = false;`	boolean	TRUE	TRUE	FALSE
`$x = true;`	boolean	FALSE	TRUE	TRUE
`$x = 1;`	integer	FALSE	TRUE	TRUE
`$x = 42;`	integer	FALSE	TRUE	TRUE
`$x = 0;`	integer	TRUE	TRUE	FALSE
`$x = -1;`	integer	FALSE	TRUE	TRUE
`$x = "1";`	string	FALSE	TRUE	TRUE
`$x = "0";`	string	TRUE	TRUE	FALSE
`$x = "-1";`	string	FALSE	TRUE	TRUE
`$x = "php";`	string	FALSE	TRUE	TRUE
`$x = "true";`	string	FALSE	TRUE	TRUE
`$x = "false";`	string	FALSE	TRUE	TRUE

Тип операнда 1	Тип операнда 2	Результат
null или string	string	**NULL** преобразуется в "", числовое или лексическое сравнение
bool или null	что угодно	Преобразуется в bool, **FALSE** < **TRUE**
string, resource или number	string, resource или number	Строки и ресурсы переводятся в числа, обычная математика
array	array	Массивы с меньшим числом элементов считаются меньше, если ключ из первого операнда не найден во втором операнде - массивы не могут сравниваться, иначе идет сравнение соответствующих значений
array	что угодно	array всегда больше
object	что угодно	object всегда больше

Таблица операторов сравнения

Пример	Название	Результат
`$a == $b`	Равно	**TRUE** если *$a* равно *$b* после преобразования типов.
`$a === $b`	Тождественно равно	**TRUE** если *$a* равно *$b* и имеет тот же тип.
`$a != $b`	Не равно	**TRUE** если *$a* не равно *$b* после преобразования типов.
`$a !== $b`	Тождественно не равно	**TRUE** если *$a* не равно *$b* или в случае, если они разных типов
`$a < $b`	Меньше	**TRUE** если *$a* строго меньше *$b*.
`$a > $b`	Больше	**TRUE** если *$a* строго больше *$b*.
`$a <= $b`	Меньше или равно	**TRUE** если *$a* is меньше или равно *$b*.
`$a >= $b`	Больше или равно	**TRUE** если *$a* больше или равно *$b*.

Таблица логических операторы

Пример	Название	Результат
`$a and $b`	И	**TRUE** если и *$a*, и *$b* **TRUE**.
`$a or $b`	Или	**TRUE** если или *$a*, или *$b* **TRUE**.
`$a xor $b`	Исключающее или	**TRUE** если *$a*, или *$b* **TRUE**, но не оба.
`!$a`	Отрицание	**TRUE** если *$a* не **TRUE**.
`$a && $b`	И	**TRUE** если и *$a*, и *$b* **TRUE**.
`$a || $b`	Или	**TRUE** если или *$a*, или *$b* **TRUE**.

Конструкция Switch-Case

Часто вместо нескольких расположенных подряд инструкций if-else целесообразно воспользоваться специальной конструкцией выбора switch-case. Данная конструкция предназначена для выбора действий, в зависимости от значения указанного выражения. Конструкция switch-case чем-то напоминает конструкцию if-else, который, по сути, является ее аналогом. Конструкцию выбора можно использовать, если предполагаемых вариантов много, например, более 5, и для каждого варианта нужно выполнить специфические действия. В таком случае, использование конструкции if-else становится действительно неудобным.

Синтаксис конструкции switch-case такой:

```
switch(выражение) {
case значение1: команды1; [break;]
case значение2: команды2; [break;]
. . .
case значениеN: командыN; [break;]
[default: команды_по_умолчанию; [break]]
}
```

Принцип работы конструкции switch-case такой:

1. Вычисляется значение выражения;

2. Просматривается набор значений. Пусть значение1 равно значению выражения, вычисленного на первом шаге. Если не указана конструкция (оператор) break, то будут выполнены команды i, i+1, i+2, ... , N. В противном случае (есть break) будет выполнена только команда с номером i.
3. Если ни одно значение из набора не совпало со значением выражения, тогда выполняется блок default, если он указан.

Приведем примеры использования конструкции **switch-case**:

```
<?php
$x=1;
// Используем if-else
if ($x == 0) {
    echo "x=0<br>";
} elseif ($x == 1) {
    echo "x=1<br>";
} elseif ($x == 2) {
    echo "x=2<br>";
}
// Используем switch-case
switch ($x) {
case 0:
    echo "x=0<br>";
    break;
case 1:
    echo "x=1<br>";
    break;
case 2:
    echo "x=2<br>";
    break;
}
?>
```

Циклы - Конструкция While

На втором месте по частоте использования, после конструкций условий (условных операторов), находятся циклы.

Циклы позволяют повторять определенное (и даже неопределенное - когда работа цикла зависит от условия) колличество раз различные операторы. Данные операторы называются телом цикла. Проход цикла называется итерацией.

Цикл с предусловием while работает по следующим принципам:

Вычисляется значение логического выражения.
Если значение истинно, выполняется тело цикла, в противном случае - переходим на следующий за циклом оператор.

Синтаксис цикла с предусловием:

```
while (логическое_выражение)
инструкция;
```

В данном случае телом цикла является инструкция. Обычно тело цикла состоит из большого числа операторов. Приведем пример цикла с предусловием while:

```
<?php
$x=0;
while ($x++<10) echo $x;
// Выводит 12345678910
?>
```

Обратите внимание на последовательность выполнения операций условия $x++<10. Сначала проверяется условие, а только потом увеличивается значение переменной. Если мы поставим операцию инкремента перед переменной (++$x<10), то сначала будет выполнено увеличение переменной, а только затем - сравнение. В результате мы получим строку 123456789. Этот же цикл можно было бы записать по-другому:

```
<?php
$x=0;
while ($x<10)
{
$x++; // Увеличение счетчика
echo $x;
}
// Выводит 12345678910
?>
```

Цикл с постусловием do while

В отличие от цикла while, этот цикл проверяет значение выражения не до, а после каждого прохода (итерации). Таким образом, тело цикла выполняется хотя бы один раз. Синтаксис цикла с постусловием такой:

```
do
{
тело_цикла;
}
while (логическое_выражение);
```

После очередной итерации проверяется, истинно ли логическое_выражение, и, если это так, управление передается вновь на начало цикла, в противном случае цикл обрывается.
Альтернативного синтаксиса для do-while разработчики PHP не предусмотрели (видимо, из-за того, что, в отличие от прикладного программирования, этот цикл довольно редко используется при программировании web-приложений).

Пример скрипта, показывающего работу цикла с постусловием do-while:

```
<?php
$x = 1;
do {
    echo $x;
} while ($x++<10);
?>
```

Рассмотренный сценарий выводит: 12345678910

Цикл со счетчиком for

Цикл со счетчиком используется для выполнения тела цикла определенное число раз. С помощью цикла for можно (и нужно) создавать конструкции, которые будут выполнять действия совсем не такие тривиальные, как простая переборка значения счетчика.

Синтаксис цикла for такой:

for (инициализирующие_команды; условие_цикла; команды_после_итерации) { тело_цикла; }

Цикл for начинает свою работу с выполнения инициализирующих_команд. Данные команды выполняются только один раз. После этого проверяется условие_цикла, если оно истинно (true), то выполняется тело_цикла. После того, как будет выполнен последний оператор тела, выполняются команды_после_итерации. Затем снова проверяется условие_цикла. Если оно истинно (true), выполняется тело_цикла и команды_после_итерации, и.т.д.

```
<?php
for ($x=0; $x<10; $x++) echo $x;
?>
```

Данный сценарий выводит: 0123456789

Есть вариант вывода строки 12345678910:

```
<?php
for ($x=0; $x++<10;) echo $x;
// Выводит 12345678910
?>
```

В данном примере мы обеспечили увеличение счетчика при проверке логического выражения. В таком случае нам не нужны были команды, выполняющиеся после итерации.

Если необходимо указать несколько команд, их можно разделить запятыми, пример:

```
<?php
for ($x=0, $y=0; $x<10; $x++, $y++) echo $x;
// Выводит 0123456789
?>
```

Приведем еще один, более практичный пример использования нескольких команд в цикле for:

```
<?php
for($i=0,$j=0,$k="Точки"; $i<10; $j++,$i+=$j) { $k=$k."."; echo $k; }
// Выводит Точки.Точки..Точки...Точки....
?>
```

Рассмотренный пример (да и вообще любой цикл for) можно реализовать и через while, только это будет выглядеть не так изящно и лаконично.

Цикл перебора массивов foreach

Синтаксис цикла foreach выглядит следующим образом:

```
foreach (массив as $ключ=>$значение)
команды;
```

Здесь команды циклически выполняются для каждого элемента массива, при этом очередная пара ключ=>значение оказывается в переменных $ключ и $значение. Приведем пример работы цикла foreach:

```
<?php
        $names["Иванов"] = "Андрей";
        $names["Петров"] = "Борис";
        $names["Волков"] = "Сергей";
        $names["Макаров"] = "Федор";
        foreach ($names as $key => $value) {
        echo "<b>$value $key</b><br>";
}
?>
```

Рассмотренный сценарий выводит:

Андрей Иванов
Борис Петров
Сергей Волков
Федор Макаров

У цикла foreach имеется и другая форма записи, которую следует применять, когда нас не интересует значение ключа очередного элемента. Выглядит она так:

```
foreach (массив as $значение)
команды;
```

В этом случае доступно лишь значение очередного элемента массива, но не его ключ. Это может быть полезно, например, для работы с массивами-списками:

```
<?php
$names[] = "Андрей";
$names[] = "Борис";
$names[] = "Сергей";
$names[] = "Федор";
foreach ($names as $value) {
echo "<b>$value</b><br>";
}
?>
```

Внимание: Цикл foreach оперирует не исходным массивом, а его копией. Это означает, что любые изменения, которые вносятся в массив, не могут быть "видны" из тела цикла. Что позволяет, например, в качестве массива использовать не только переменную, но и результат работы какой-нибудь функции, возвращающей массив (в этом случае функция будет вызвана всего один раз - до начала цикла, а затем работа будет производиться с копией возвращенного значения).

Конструкция break

Очень часто для того, чтобы упростить логику какого-нибудь сложного цикла, удобно иметь возможность его прервать в ходе очередной итерации (к примеру, при выполнении какого-нибудь особенного условия). Для этого и существует конструкция break, которая осуществляет немедленный выход из цикла. Она может задаваться с одним необязательным параметром - числом, которое указывает, из какого вложенного цикла должен быть произведен выход. По умолчанию используется 1, т. е. выход из текущего цикла, но иногда применяются и другие значения. Синтаксис конструкции break:

```
break; // По умолчанию
break(номер_цикла); // Для вложенных циклов (указывается номер прерываемого цикла)
```

Приведем примеры:

```
<?php
$x=0;
while ($x++<10) {
if ($x==3) break;
echo "<b>Итерация $x</b><br>";
}
// Когда $x равен 3, цикл прерывается
?>
```

Рассмотренный сценарий выводит:

Итерация 1
Итерация 2

Если нам нужно прервать работу определенного (вложенного) цикла, то нужно передать конструкции break параметр - номер_цикла, например, break(1). Нумерация циклов выглядит следующим образом:

```
for (...) // Третий цикл
{
  for (...) // Второй цикл
  {
    for (...) // Первый цикл
    {
    }
  }
}
```

Конструкция continue

Конструкция continue так же, как и break, работает только "в паре" с циклическими конструкциями. Она немедленно завершает текущую итерацию цикла и переходит к новой (конечно, если выполняется условие цикла для цикла с предусловием). Точно так же, как и для break, для continue можно указать уровень вложенности цикла, который будет продолжен по возврату управления.
В основном continue позволяет вам сэкономить количество фигурных скобок в коде и увеличить его удобочитаемость. Это чаще всего бывает нужно в циклах-фильтрах, когда требуется перебрать некоторое количество объектов и выбрать из них только те, которые удовлетворяют определенным условиям. Приведем пример использования конструкции continue:

```
<?php
$x=0;
while ($x++<5) {
if ($x==3) continue;
echo "<b>Итерация $x</b><br>";
}
// Цикл прервется только на третьей итерации
?>
```

Рассмотренный скрипт выводит:

Итерация 1
Итерация 2
Итерация 4
Итерация 5

Конструкции включений в PHP

Конструкции включений позволяют собирать PHP программу (скрипт) из нескольких отдельных файлов.

В PHP существуют две основные конструкции включений: **require** и **include**.

Конструкция включений require

Конструкция **require** позволяет включать файлы в сценарий PHP до исполнения сценария PHP. Общий синтаксис require такой:

require имя_файла;

При запуске (именно при запуске, а не при исполнении!) программы интерпретатор просто заменит инструкцию на содержимое файла имя_файла (этот файл может также содержать сценарий на PHP, обрамленный, как обычно, тэгами <? и ?>). Причем сделает он это непосредственно перед запуском программы (в отличие от include, который рассматривается ниже). Это бывает довольно удобно для включения в вывод сценария различных шаблонных страниц HTML-кодом. Приведем пример:

Файл header.html:
<html>
<head><title>It is a title</title></head>
<body bgcolor=green>
Файл footer.html:
© Home Company, 2005.
</body></html>
Файл script.php
<?php
require "header.htm";
// Сценарий выводит само тело документа
require "footer.htm";
?>

Таким образом, конструкция require позволяет собирать сценарии PHP из нескольких отдельных файлов, которые могут быть как html-страницами, так и php-скриптами.

Конструкция require поддерживает включения удаленных файлов.

Например:

<?php
// Следующий пример на работает, поскольку пытается включить локальный файл
require 'file.php?foo=1&bar=2';
// Следующий пример работает
require 'http://www.example.com/file.php?foo=1&bar=2';

```
?>
```

Конструкция require позволяет включать удаленные файлы, если такая возможность включена в конфигурационном файле PHP. Подробная информация далее.

Конструкция включений include

Конструкция include также предназначена для включения файлов в код сценария PHP.

В отличие от конструкции require конструкция include позволяет включать файлы в код PHP скрипта во время выполнения сценария. Синтаксис конструкции include выглядит следующим образом:

```
include имя_файла;
```

Поясним принципиальную разницу между конструкциями require и include на конкретном практическом примере. Создадим 10 файлов с именами 1.txt, 2.txt и так далее до 10.txt, содержимое этих файлов - просто десятичные цифры 1, 2 ..... 10 (по одной цифре в каждом файле). Создадим такой сценарий PHP:

```
<?php
// Создаем цикл, в теле которого конструкция include
for($i=1; $i<=10; $i++) {
include "$i.txt";
}
// Включили десять файлов: 1.txt, 2.txt, 3.txt ... 10.txt
// Результат - вывод 12345678910
?>
```

В результате мы получим вывод, состоящий из 10 цифр: "12345678910". Из этого мы можем слелать вывод, что каждый из файлов был включен по одному разу прямо во время выполнения цикла! Если мы поставим теперь вместо include require, то сценарий сгенерирует критическую ошибку (fatal error). Сравните результат.

PHP преобразует сценарий во внутреннее представление, анализируя строки сценария по очереди, пока не доходит до конструкции include. Дойдя до include, PHP прекращает транслировать сценарий и переключается на указанный в include файл. Таким образом из-за подобного поведения транслятора, быстродействие сценария снижается, особенно при большом колличестве включаемых с помощью include файлов. С require таких проблем нет, поскольку файлы с помощью require включаются до выполнения сценария, то есть на момент трансляции файл уже включен в сценарий.

Таким образом, целесообразнее использовать конструкцию require там, где не требуется динамическое включение файлов в сценарий, а конструкцию include использовать только с целью динамического включения файлов в код PHP скрипта.

Конструкция include поддерживает включения удаленных файлов.

Например:

```
<?php
// Следующий пример на работает, поскольку пытается включить локальный файл
include 'file.php?foo=1&bar=2';
// Следующий пример работает
include 'http://www.example.com/file.php?foo=1&bar=2';
?>
```

Конструкция include позволяет включать удаленные файлы, если такая возможность включена в конфигурационном файле PHP.

Конструкции однократного включения require_once и include_once

В больших PHP сценариях инструкции include и require применяются очень часто. Поэтому становится довольно сложно контролировать, как бы случайно не включить один и тот же файл несколько раз, что чаще всего приводит к ошибке, которую сложно обнаружить.

В PHP предусмотрено решение данной проблемы. Используя конструкции однократного включения require_once и include_once, можно быть уверенным, что один файл не будет включен дважды. Работают конструкции однократного включения require_once и include_once так же, как и require и include соответственно. Разница в их работе лишь в том, что перед включением файла интерпрететор проверяет, включен ли указанный файл ранее или нет. Если да, то файл не будет включен вновь.

Конструкции однократных включений также require_once и include_ince также позволяют включать удаленные файлы, если такая возможность включена в конфигурационном файле PHP.

Включения удаленных файлов

PHP позволяет работать с объектами URL, как с обычными файлами. Упаковщики, доступные по умолчанию, служат для работы с удаленными файлами с использованием протокола ftp или http.

Если "URL fopen-оболочки" включены в PHP (как в конфигурации по умолчанию), вы можете специфицировать файл, подключаемый с использованием URL (через HTTP), вместо локального пути. Если целевой сервер интерпретирует целевой файл как PHP-код, переменные могут передаваться в подключаемый файл с использованием URL-строки запроса, как в HTTP GET. Строго говоря, это не то же самое, что подключение файла и наследование им области видимости переменных родительского файла; ведь скрипт работает на удалённом сервере, а результат затем подключается в локальный скрипт.

Для того, чтобы удаленное включение файлов было доступно, необходимо в конфигурационном файле (php.ini) установить allow_url_fopen=1.