Лекция 2. Синтаксис языка PHP

PHP и HTML-текст

Начав писать PHP-код, вы будете работать с самыми обычными текстовыми файлами, содержащими код PHP и HTML. **HTML** – это простой язык разметки, позволяющий определить, как должна выглядеть страница в окне браузера, но это всего лишь текст. Сервер никак не обрабатывает HTML-файлы перед их отправкой браузеру пользователя. В отличие от HTML, PHP-код должен быть как-то интерпретирован, прежде чем окончательный вариант страницы будет отправлен браузеру. Иначе такая страница на экране у пользователя превратится в смесь текста и программного кода.

Чтобы выделить PHP-код и тем самым проинформировать веб-сервер о необходимости его обработки, PHP-код размещают между формальными или неформальными тегами (**<? php ... ?>**), смешивая с HTML. В примере ниже демонстрируется это с помощью конструкций echo и print. Конструкции **echo** и**print** почти совпадают, за исключением того, что конструкция echo может принимать несколько аргументов и не возвращает никакого значения, тогда как конструкция print способна принимать только один аргумент.

Файл этого примера мы назвали index.php, а вы можете взять любое другое имя, главное чтобы оно имело расширение .php. Это расширение сообщает веб-серверу, что файл нужно обрабатывать как PHP-код:

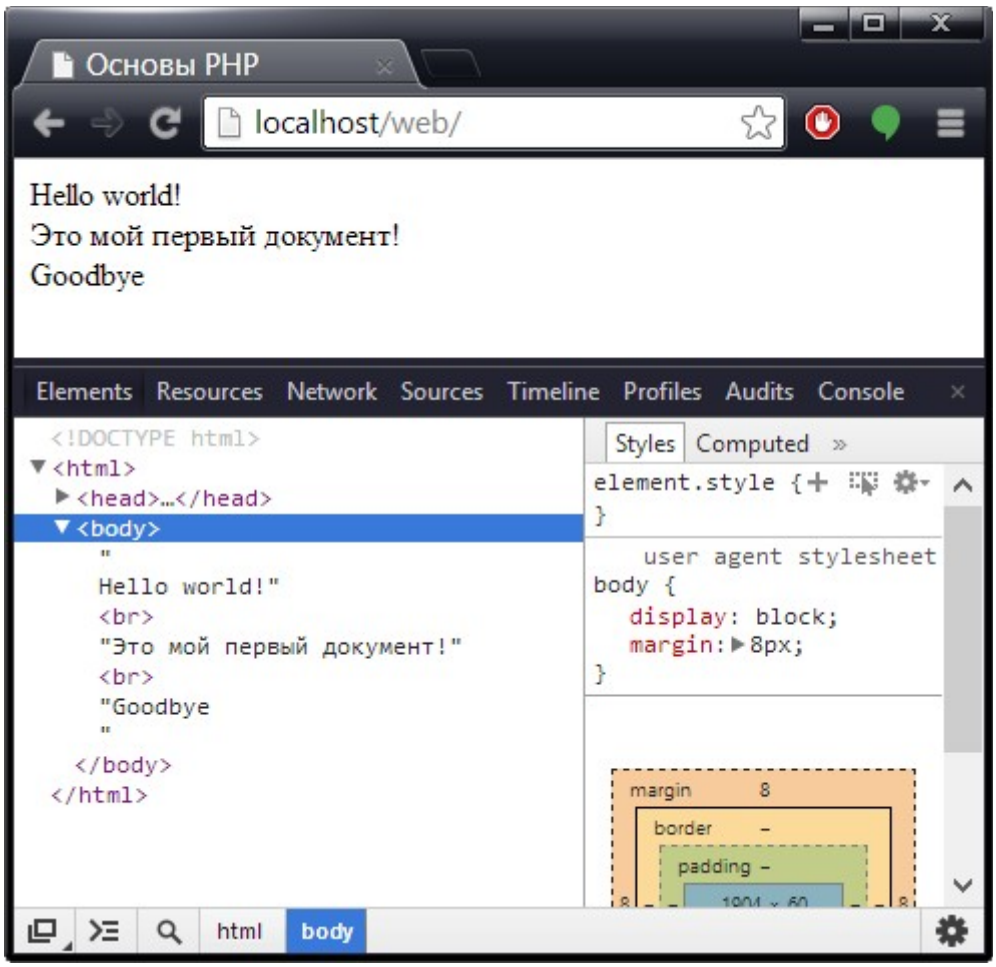
Конструкции echo и print

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>Основы PHP</title>
</head>

<body>
<?php
   echo 'Hello world!<br>';
   echo ("Это мой первый документ!<br>");
   print 'Goodbye';
?>
</body>
</html>
```

Когда браузер запросит этот файл, PHP интерпретирует его и воспроизведет текст в формате HTML:

Так выглядит результат вывода текста в окне браузера и исходная HTML-разметка



Включение файлов

Еще один способ, с помощью которого можно вводить код PHP в код HTML, состоит в том, что код PHP можно помещать в отдельный файл и вызывать его с использованием функций include и require языка PHP. Предусмотрено всего четыре функции типа include и require:

- **include('/filepath/filename');**
- **require('/filepath/filename');**
- **include_once('/filepath/filename');**
- **require_once('/filepath/filename').**

В предыдущих версиях PHP наблюдались значительные различия в функциональных возможностях и быстродействии между функциями типа include и require. Это больше не имеет места; оба набора функций отличаются только тем, что в случае неудачного завершения вырабатывают разные ошибки. Функции include() и include_once() при возникновении сбоя просто вырабатывают предупреждение об отказе, а функции require() и require_once() вырабатывают неисправимую ошибку и вызывают завершение сценария.

Инструкция **include** позволяет подключать и присоединять к вашему PHP-сценарию другие сценарии. Действие этой инструкции можно представить как вставку кода из подключаемого файла в ваш PHP-сценарий. В примере ниже приведен программный код подключаемого файла с именем add.php:

Файл add.php

```
<?php
    echo 'Hello world!<br>';
    echo ("Это мой первый документ!<br>");
    print 'Goodbye';
?>
```

Ниже показан пример, в котором этот файл добавляется в файл index.php с помощью инструкции include. В примере предполагается, что файл add.php находится в том же каталоге, что и index.php.

Использование инструкции include

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>Основы PHP</title>
</head>

<body>
<?php
     include 'add.php';
?>
</body>
</html>
```

Результат окажется аналогичным показанному на рисунке выше, только в этом сценарии мы вынесли работающий код в отдельный файл. Это позволяет не смешивать код HTML и PHP, облегчая разработку приложений.

При подключении PHP-сценариев с многоступенчатой вложенностью могут возникнуть проблемы из-за того, что инструкция include не проверяет, были ли ранее подключены данные сценарии. Например, изменим файл add.php, добавив в него функцию (с переменными и функциями мы познакомимся чуть позже, а пока просто вставьте этот код в файл add.php):

add.php - исходный код

```
<?php
    function sum($a, $b)
        {
                return $a + $b;
```

```
        }
?>
```

А теперь попытаемся выполнить такой сценарий в исходном файле index.php:

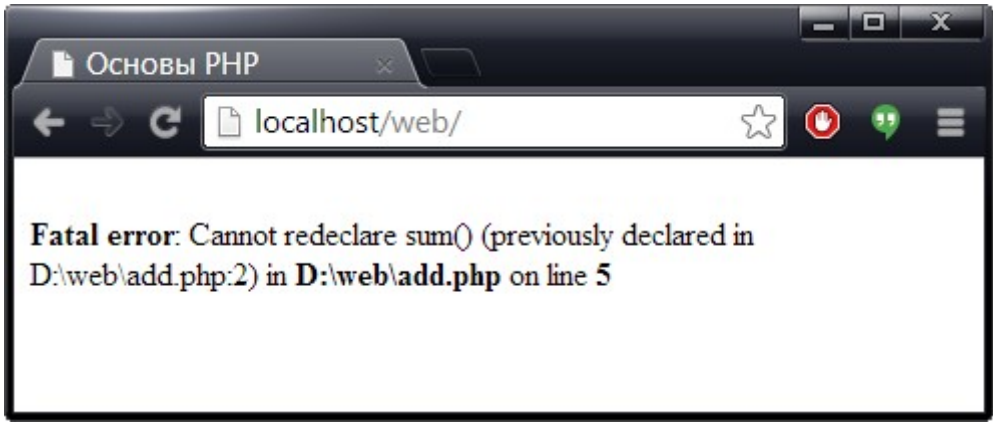
Исходный код

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>Основы PHP</title>
</head>

<body>
<?php
     include 'add.php';
         include 'add.php';

         echo sum(4,10);
?>
</body>
</html>
```

Будет получено сообщение об ошибке, так как фактически мы дважды определили функцию с одним и тем же именем:



Ошибка объявления уже зарегистрированной функции из-за двойного включения одного и того же фала

Чтобы избежать ошибок подобного рода, следует использовать инструкцию **include_once**, как показано в примере ниже:

Исходный код

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
```

```
<title>Основы PHP</title>
</head>

<body>
<?php
     include_once 'add.php';
          include_once 'add.php';

          echo sum(4,10);
?>
</body>
</html>
```

Очевидно, что вы и не собираетесь помещать инструкции include для подключения одного и того же файла непосредственно друг за другом. Но показанная ситуация с двойным подключением является упрощенным вариантом распространенной ошибки, когда вы попытаетесь подключить файл, который присоединяется через другой подключаемый файл. Поэтому всегда следует использовать include_once, так как у этой инструкции нет недостатка инструкции include.

Чтобы гарантировать подключение файла или остановить работу программы, если выполнить подключение невозможно, используйте функцию **require** или парную ей **require_once**. Они делают то же самое, что инструкции include и include_once, но гарантируют подключение указанного файла либо прекращение исполнения сценария, что иногда может быть очень удобно!

Функцию require можно использовать вместо include, например, если подключается файл с определениями критически важных функций, которые должны выполняться в вашем сценарии, или переменных, описывающих параметры подключения к базе данных, без которых сценарий не сможет функционировать. Например, попытаемся с помощью функции require подключить несуществующий файл:

Исходный код

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>Основы PHP</title>
</head>

<body>
<?php
          require_once 'undefined.php';
?>
</body>
</html>
```

Будет получено следующее сообщение об ошибке:

Подключаемый файл undefined.php не найден

Наиболее широко распространенный способ использования возможностей функций включения языка PHP состоит в добавлении общих верхнего и нижнего колонтитулов ко всем веб-страницам сайта. При этом необязательно подключать файлы с расширением *.php, можно включать статичные файлы HTML:

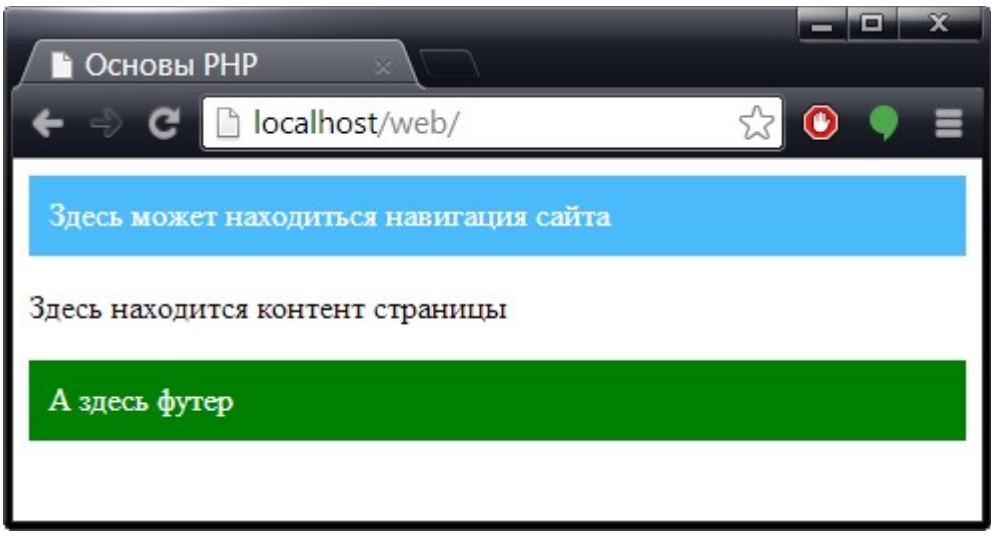
index.php

```
<?php
        require_once 'header.html';
?>

<p>Здесь находится контент страницы</p>

<?php
        require_once 'footer.html';
?>
header.html
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>Основы PHP</title>
</head>

<body>
    <nav style="padding:10px; background:#4cbbfb; color:white">
        Здесь может находиться навигация сайта
    </nav>
footer.html
    <footer style="padding:10px; background:green; color:white">
        А здесь футер
    </footer>
</body>
</html>
```



Динамически генерируемое содержимое за счет использования подключаемых файлов

Очевидно, что даже такой простой шаг существенно улучшает удобство сопровождения и масштабируемость всего сайта. После этого для перехода к использованию другого оформления или для обновления информации об авторских правах достаточно откорректировать содержимое одного файла, а не исправлять одни и те же строки на десятках или тысячах HTML-страниц.

Любые файлы с кодом PHP, предназначенные для включения из других файлов, должны обязательно проверяться на предмет того, чтобы в конце файла не было пустых новых строк. Следует помнить, что все, находящееся за пределами блока PHP, рассматривается как код HTML, даже любая пустая строка. Пустые строки или даже пробельные символы, стоящие вслед за закрывающим дескриптором PHP, интерпретируются как выходные данные. Если включение файла выполняется в том месте страницы, где не могут присутствовать выходные данные (скажем, в месте, предшествующем использованию заголовков HTTP), то выполнение сценария окончится неудачей и будет сформировано пространное сообщение об ошибке, говорящее о том, что выходной поток уже был начат во включаемом файле.

Поэтому если в подключаемых файлах используется только код PHP, разрешается и даже рекомендовано не использовать закрывающий тег. Пример с использованием фала add.php можно и нужно переписать следующим образом:

Файл add.php без закрывающего тега ?>

```
<?php
    function sum($a, $b)
        {
                return $a + $b;
        }
```

Комментарии

Комментарием называется та часть текста программы, которая предназначена только для чтения человеком. Комментарии предоставляют бесценную помощь тем, кому приходится разбираться в коде, написанном другими, и позволяют понять, в чем состоял замысел разработчиков, создавших этот код. Иногда по прошествии нескольких недель не удается понять без комментариев, как работает код, даже тому человеку, который его написал.

Разработчики языка PHP взяли за основу нескольких других языков программирования, в основном рассматривая в качестве прототипов синтаксические конструкции языков C и Perl, а также сценарии командного интерпретатора Unix. В результате оказалось, что язык PHP поддерживает стили комментариев из всех этих языков, а сами эти стили могут свободно сочетаться в коде PHP.

Многострочные комментарии в стиле языка C

В языке PHP многострочные комментарии имеют такую же синтаксическую конструкцию, как и в языке C - комментарий начинается с пары символов /* и заканчивается парой символов */, например, как показано ниже:

Исходный код

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>Основы PHP</title>
</head>

<body>
<?php
     /* Это - комментарий в сценарии на языке PHP */
?>
</body>
</html>
```

Наиболее важной характерной особенностью многострочных комментариев, о которой следует помнить, является то, что их нельзя вкладывать; это означает, что один многострочный комментарий нельзя вложить в другой. При попытке применения вложенного комментария внешний комментарий закроется после обнаружения первой пары символов */, относящейся к внутреннему комментарию, а остальная часть текста внешнего комментария, которая должна была рассматриваться как конец комментария, вместо этого будет интерпретироваться как код, что может привести к серьезному нарушению в работе. Следует учитывать, что такую ошибку можно легко допустить непреднамеренно; обычно она

возникает при попытке вывести из обработки фрагмент содержащего комментарий кода, закомментировав (т.е. обозначив как комментарий) этот блок.

Создание однострочных комментариев с помощью символов # и //

Кроме многострочных комментариев в формате /* ... */, язык PHP поддерживает два других способа ввода комментариев, размещаемых в конце какой-то конкретной строки. Один из этих способов унаследован от языков C++ или Java, а другой — из языка Perl и сценариев командного интерпретатора. Комментарий в стиле сценариев командного интерпретатора начинается со знака диеза (#), а комментарий в стиле языка C++ начинается с двух символов косой черты (//). Ввод в текст тех или других символов приводит к тому, что остальная часть текущей строки рассматривается как комментарий. Ниже приведены примеры комментариев в указанном стиле:

Исходный код

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>Основы PHP</title>
</head>

<body>
<?php
         #      Это - комментарий, а ниже
         #      приведена вторая часть комментария.

         //     Это - также комментарий. Комментарий, оформленный в этом
стиле и
         //     в стиле командного интерпретатора, распространяется только
на одну
         //     строку, поэтому последнее слово данного комментария поступит
на
         //     обработку к интерпретатору и вызовет серьезное нарушение в
         работе
?>
</body>
</html>
```

Лексические особенности языка PHP

Невосприимчивость языка PHP к пробельным символам

Пробельными называются символы, применяемые при вводе текста, которые обычно остаются невидимыми на экране, включая пробелы, знаки табуляции и символы возврата каретки (символы обозначения конца строки). То, что язык PHP невосприимчив к пробельным символам, не означает, что пробелы и прочие подобные символы не играют в языке PHP никакой роли. (В действительности такие символы выполняют крайне важные функции, хотя бы в том, что разделяют ключевые слова в тексте на языке PHP.) Вместо этого

под данным утверждением подразумевается, что почти никогда не имеет значения количество расположенных подряд пробельных символов, т.е. один пробельный символ равнозначен нескольким подобным символам.

Например, все приведенные ниже операторы PHP, в которых значение суммы 2 + 2 присваивается переменной $four, являются эквивалентными:

Исходный код

```
<?php
        $four = 2 + 2;    // Одинарные пробелы
        $four          =       2       +       2;    // Пробелы и знаки
табуляции
        $four =
2
+

2; // Размещение на нескольких строках
?>
```

Удобно и то, что пробельными считаются символы обозначения конца строки, поскольку это означает, что при разработке программ не приходится заботиться о том, чтобы операторы помещались на одной строке.

Разные требования к регистру символов в языке PHP в разных контекстах

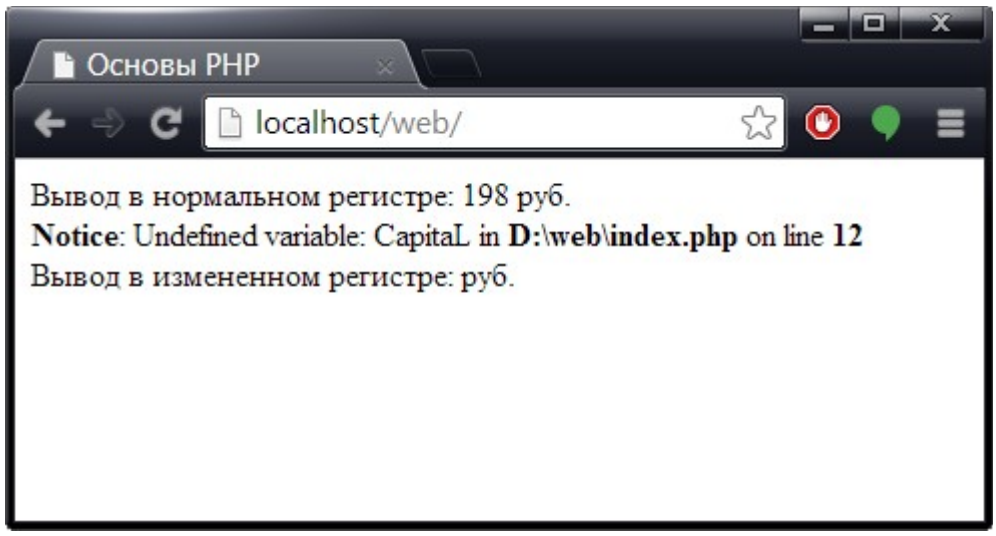
Язык PHP не предъявляет слишком жесткие требования к синтаксису, но PHP иногда требует соблюдения правильного регистра символов (т.е. учитывает различия между строчными и прописными буквами). В частности, все имена переменных чувствительны к регистру. Например, если в HTML-страницу будет включен код:

Исходный код

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>Основы PHP</title>
</head>

<body>
<?php
        $capital = 198;
        print "Вывод в нормальном регистре: $capital руб.";
        print "Вывод в измененном регистре: $CapitaL руб.";
?>
</body>
</html>
```

то вывод будет выглядеть таким образом:



Переменная CapitaL не найдена

Из-за того, что в разных именах применялись символы разного регистра, переменные стали разными. С другой стороны, имена функций в языке PHP не чувствительны к регистру, и таковыми являются также основные языковые конструкции (if, then, else, while и т.д.).