#### МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

**ОДЕСЬКА НАЦІОНАЛЬНА АКАДЕМІЯ**

**ХАРЧОВИХ ТЕХНОЛОГІЙ**



**Швець Н.В.**

## ПРОГРАМУВАННЯ ДЛЯ INTERNET

#### Посібник до виконання лабораторних робіт

**Одеса 2013**

**Швець Н.В.** Програмування для Internet: Посібник до виконання лабораторних робіт. Одеська національна академія харчових технологій, 2013. – ??? с.

Посібник розроблено згідно з робочою навчальною програмою дисципліни «Програмування для Internet» для студентів які навчаються за напрямом підготовки 6.050101«Комп’ютерні науки».

Призначено для виконання лабораторних робіт студентами по закріпленню окремих тем дисципліни. Наведено перелік теоретичних питань для поглибленого опрацювання з посиланнями на літературу, яка є в бібліотеці ОНАХТ.

Рецензент:

Розглянуто та рекомендовано до видання на засіданні кафедри інформаційних технологій та кібербезпеки .

Протокол № від 201 р.

Розглянуто та рекомендовано до видання на засіданні науково-методичної комісії з напряму підготовки 6.050101«Комп’ютерні науки»

Протокол № від 2013 р.

©ОНАХТ, 2013

**Зміст**

[Лабораторная работа №1 4](#_Toc367799665)

[Лабораторная работа №2 15](#_Toc367799666)

[Лабораторная работа №3 21](#_Toc367799667)

[Лабораторная работа №4 31](#_Toc367799668)

[Лабораторная работа №5 48](#_Toc367799669)

# Лабораторная работа №1

**Документы HTML5**

Цель работы: Практическое знакомство с возможностями HTML5 для создания структуры веб-сайта.

## Задание:

**1**. Определить тематику для разрабатываемого веб-сайта.

2. Выполнить обзор программных аналогов, функционирующих в Сети. На основе полученной информации создать макет главной страницы сайта.

3. Разработать html-документ, формирующий на экране страницу соответствующей структуры, к которому будет применяться файл таблицы стилей.

## Краткие теоретические сведения

Спецификация HTML5 представляет собой совместную эволюцию и описание трех технологий: HTML, CSS3 и JavaScript. HTML обеспечивает структурные элементы, CSS фокусируется на том, как превратить эту структуру в нечто визуально привлекательное и удобное в использовании, а JavaScript предлагает мощь, необходимую для обеспечения функциональности и построения полноценных веб-приложений.

### Структура документа

Основной частью документа является его структура. Она определяет элементы для размещения статического или динамического содержимого и служит основной платформой для приложений. Структура должна обеспечивать форму, порядок и гибкость, и она должна быть прочной как фундамент здания. К структуре документов HTML предъявляются строгие требования. Все части документы отделены друг от друга, каждая из них объявлена и заключена в определенные теги. Следующий листинг определяет структуру документа.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<meta name="description" content="This is an HTML5 example">

<meta name="keywords" content="HTML5, CSS3, JavaScript">

<title>This text is the title of the document</title>

<link rel="stylesheet" href="mystyles.css">

</head>

<body>

*Тело документа*

</body>

</html>

Элемент **<doctype>** указывает тип создаваемого нами документа. Эта строка должна идти самой первой в файле html, и перед ней не должно быть ни пробелов, ни пустых строк. Она активирует стандартный режим браузеров для интерпретации HTML5, когда это возможно (или игнорирует его в противном случае).

После объявления типа документа мы переходим к построению древовидной структуры HTML. Как всегда, в корне дерева находится элемент **<html>.** Внутри этого элемента помещается весь остальной код HTML. Единственный атрибут, указания которого требует HTML5, - это атрибут lang открывающего тега <html>. Он определяет язык содержимого в создаваемом нами документе. Для английского языка его значение равно “en”, для русского - “ru”. HTML5 чрезвычайно гибкий инструмент. Элемент <html> можно использовать без атрибутов или вообще не добавлять в код. Но в целях обеспечения совместимости, а также с учетом парсирования, рекомендуется следовать некоторым базовым правилам.

Элемент **<head>**. HTML-код, который добавляется между тегами <html>, делится на два основных раздела: первый представляет собой «голову», а второй – «тело» кода. Сам тег по сравнению с предыдущими версиями языка не изменился, т.е. его назначение остается таким же. Внутри тега <head> мы:

* опеределяем заголовок веб-страницы;
* объявляем кодировку символов;
* добавляем общую информацию о документе;
* приводим ссылки на внешние файлы, содержащие стили, сценарии.

За исключением заголовка, вся остальная информация, содержащаяся между тегами <head> и </head>, на экране обычно не отображается.

Содержимое видимой части документа описывается внутри тега <body>. Данный тег также не изменился по сравнению с предыдущими версиями HTML.

«Голова» документа. Здесь было добавлено несколько изменений и новшеств, среди которых тег **<meta>**. В документе <html> может быть несколько тегов <meta>, содержащих общие объявления, однако эта информация не отображается в окне браузера. Она используется только поисковыми системами и устройствами, которым требуются предварительные данные о нашем документе. Атрибут charset определяет кодировку символов, т.е. указывает на то, каким образом текст должен отображаться на экране. Атрибут name внутри тега <meta> определяет тип тега, а в атрибуте content объявляется его значение.

В HTML5 одинарные теги не обязательно должны иметь закрывающий слеш, но рекомендуется ставить его для совместимости, например:

<meta charset="utf-8" />

**<title>.** Как и раньше, используется для определения заголовка документа, отображаемого в заголовке окна.

**<link>**. Используется для подключения к документу стилей, сценариев из внешних файлов. В HTML5 избавились от необходимости указывать тип подключаемой таблицы стилей, таким образом, атрибут type больше не используется. Теперь для внедрения файла со стилями достаточно двух атрибутов: rel и href. Атрибут rel определяет отношение – чем для нашего документа является подключаемый файл, атрибут href – содержит адрес ресурса в сети.

### Структура тела документа

Структура тела документа определяет его видимую часть. Именно в этом коде находится содержимое веб-страницы. Язык HTML с самого начала предлагал разные способы построения и организации информации в теле документа. Одим из первых элементов, выполняющих эту функцию, был <table>. Таблицы позволяли организовывать данные, текст и изображения в строки и столбцы, хотя первоначально этот элемент создавался для других целей. Позднее эти функции взяли на себя другие элементы, позволявшие добиваться тех же результатов быстрее с использованием меньшего объема кода. Основным элементом стал <div>. Однако, ни <table>, ни <div>, не дают достаточно информации о той части тела документа, которую он представляет: между открывающим и закрывающим тегами может находиться все, что угодно – меню, текст, ссылки, сценарии, формы и т.д. Другими словами, например, <div> всего лишь указывает на некую составляющую тела документа, например, на ячейку в таблице, однако ничего не говорит о том, что это за составляющая, каково ее назначение и что находится внутри нее.

Для пользователей такие подсказки не слишком важны, но для браузеров правильная интерпретация содержимого обрабатываемого документа имеет критическое значение.

Именно поэтому в HTML5 появились новые элементы, помогающие определить каждую часть документаи упорядочить его тело. В HTML5 документ делится на несколько важных разделов, и основная структура больше не зависит от тегов <table> и <div>.

### Макет страницы

Обычный макет страницы, на основе которого построено большинство современных веб-сайтов можно представить так:

|  |  |
| --- | --- |
| **Заголовок** | |
| **Панель навигации** | |
| **Основная**  **информация** | **Боковая**  **панель** |
| **Служебная информация** | |

Рис. 1. Визуальное представление типичного макета веб-сайта

Несмотря на то, что все дизайнеры создают собственные шаблоны страниц, почти любой веб-сайт можно разбить на следующие основные разделы.

В верхней области, обозначенной «Загаловок», обычно находится логотип, название, подзаголовок и краткое описание веб-сайта или веб-страницы.

Под этой областью находится панель навигации, на которую почти все разработчики помещают меню или ссылки по перемещению по сайту. С помощью панели навигации пользователи переходят к различным страницам или документам, обычно в пределах одного веб-сайта.

Основное содержимое страницы размещается, как правило, в середине макета. В этой области представлена самая важная информация и ссылки. Чаще всего ее делят на несколько строк и столбцов. На рисунке отображены только два столбца, «Основная информация» и «Боковая панель», но данная область является очень гибкой, и дизайнеры настраивают ее в соответствии с поставленными требованиями, добавляя новые строки, разбивая столбцы на блоки меньшего размера и создавая разнообразные комбинации и варианты размещения информации. Содержимое этой части макета, как правило, имеет наибольший приоритет. Здесь могли бы содержаться список статей, описание продуктов, записи блога и любые другие сведения.

В область «Боковая панель» можно было бы поместить список ссылок на каждый из этих элементов.

В нижней части типичного макета можно увидеть одну или несколько панелей со служебной информацией и чаще всего содержит общие сведения о веб-сайте, его авторе и компании-владельце, здесь также можно найти ссылки на правила и условия, карты и любые дополнительные данные, которые разработчик посчитал необходимым указать. Панель со служебной информацией допоняет заголовок и считается неотъемлемой составляющей структуры страницы.

В HTML5 учитывается эта базовая структура и макет, и для каждого из разделов существуют новые элементы, позволяющие объявлять и различать их. Это позволяет указать браузеру, для чего предназначены разделы страницы.

|  |  |
| --- | --- |
| **<header> </header>** | |
| **<nav> </nav>** | |
| **<section>**  **</section>** | **<aside>**  **</aside>** |
| **<footer> </footer>** | |

Рис. 2. Визуальное представление структуры макета с помощью тегов HTML5

Рассмотрим пример:

<!DOCTYPE html>

<html lang="ru">

<head>

<meta charset="utf-8">

<meta name="description" content="Это пример HTML5">

<meta name="keywords" content="HTML5, CSS3, JavaScript">

<title>Этот текст – заголовок документа</title>

<link rel="stylesheet" href="mystyles.css">

</head>

<body>

<header>

<h1>Это главный заголовок веб-сайта</h1>

</header>

<nav>

<ul>

<li>Домой</li>

<li>Фото</li>

<li>Видео</li>

<li>Контакты</li>

</ul>

</nav>

<section>

</section>

<aside>

<blockquote>Статья 1</blockquote>

<blockquote>Статья 2</blockquote>

</aside>

<footer>

Copyright &copy; 2012-2013

</footer>

</body>

</html>

**<header>**. Один из новых элементов, появившихся в HTML5 - <header>. Аналогично <head>, тег <header> содержит вводную информацию (такую как заголовки, подзаголовки или логотоипы), однако области применения этих двух тегов различаются. Тег <head> предназначен для хранения информации обо всем документе, тогда как <header> используется только для тела документа или его разделов. Тег <header> указывает на начало основного содержимого документа, его видимой части. Начиная с этого тега мы будем видеть результаты нашего кода в окне браузера.

**<nav>**. Структура и порядок использования элементов в HTML5 определяются разработчиком. HTML5 – это очень гибкий язык, он всего лишь предоставляет базовые элементы и параметры, а как их применять – решает разработчик. Например, тег <nav> можно было бы поместить внутрь тега <header> или в любой другой раздел тела документа. Однако необходимо учитывать, что все эти новые теги создавались для того, чтобы предоставлять браузерам больше информации и помогать обрабатывающим программам распознавать наиболее важные части документа. Такой подход будет способствовать созданию понятного и переносимого кода. Этот элемент предназначен для определения навигационных элементов, таких как главное меню или основные панели навигации. Поэтому и следует использовать его только для этих целей.

**<section>**. Теперь в нашем типовом дизайне нужно определить разделы, обозначенные на макете как «Основная информация» и «Боковая панель». Как мы уже говорили, в области основной информации выводится главное содержимое документа. Эта область может принимать самые разные формы – ее можно поделить на несколько блоков или столбцов. Поскольку эти блоки и столбцы не имеют конкретного назначения, то элемент HTML5 для их определения имеет общее название <section>.

Теги, определяющие разделы тела документа, в коде следуют один за другим, но на веб-странице некоторые из них могут располагаться рядом, а не один под другим, как этого можно было ожидать. В HTML5 визуальное отображение элементов на экране определяется с помощью CSS. Каждому элементу назначается свой css-стиль. CSS мы рассмотрим далее.

**<aside>**. В типичном макете веб-сайта область под названием «Боковая панель» находится сбоку от основной информации. Данные в этой области обычно связаны с основной информацией страницы, но не так важны. Содержимое боковой панели связано с основной информацией, но само по себе не имеет никакого значения. Так, например, для стандартного макета блога главной информациейявляются записи блога, а ссылки и краткие анонсы этих записей представляют собой лишь средства навигации, а не то, что будет интересовать пользователя в первую очередь. В HTML5 такую вспомогательную информацию можно обозначить с помощью элемента <aside>. Элемент <aside> может располагаться на странице справа или слева, он не имеет конкретного местоположения. Этот тег описывает всего лишь заключенную в него информацию, а не ее место в структуре документа. Таким образом, элемент <aside> можно добавлять в любую область макета и использовать для любого содержимого, не относящегося к основной информации веб-сайта. Например, элемент <aside> можно поместить внутрь элемента <section> или даже внутрь основной информации документа (один из способов оформления цитат).

**<footer>**.Завершающий элемент должен визуально закончить дизайн и обозначить конец тела документа. В HTML5 для этого используется специальный тег <footer>. С его помощью мы определяем «подвал» нашего документа, в котором, как правило, содержатся общие сведения об авторе или компании, владеющей проектом, авторское право, условия использования и т.д. Этот элемент можно использовать в теле документа несколько раз – в конце разных разделов (точно так же в документе можно неоднократно использовать тег <header>).

### Строение тела документа

Базовая структура нашего документа определена. Теперь разберемся с его содержанием. Пока мы рассмотрели только элементы HTML5, помогающие определять разделы макета и указывать их назначение.

Информация в документе может включать в себя разные визуальные элементы, такие как заголовки. тексты, изображения, видео. Нам необходимо различать эти элементы и устанавливать взаимоотношения между ними.

Рассмотри пример кода:

<!DOCTYPE html>

<html lang="ru">

<head>

<meta charset="utf-8">

<meta name="description" content="Это пример HTML5">

<meta name="keywords" content="HTML5, CSS3, JavaScript">

<title>Этот текст - заголовок документа</title>

<link rel="stylesheet" href="mystyles.css">

</head>

<body>

<header>

<h1>Это главный заголовок веб-сайта</h1>

</header>

<nav>

<ul>

<li>Домой</li>

<li>Фотографии</li>

<li>Видео</li>

<li>Контактыt</li>

</ul>

</nav>

<section>

<article>

<header>

<hgroup>

<h1>Заголовок статьи 1</h1>

<h2>Подзаголовок статьи 1</h2>

</hgroup>

<p>Опубликовано 12-10-2012</p>

</header>

This is the text of my first post

<figure>

<img src="myimage.jpg">

<figcaption>

Это текст первой статьи

</figcaption>

</figure>

<footer>

<p>Комментарии (0)</p>

</footer>

</article>

<article>

<header>

<hgroup>

<h1>Заголовок статьи 2</h1>

<h2>Подзаголовок статьи 2</h2>

</hgroup>

<p>Опубликовано 12-15-2012</p>

</header>

Это текст второй статьи

<footer>

<p>Комментарии (0)</p>

</footer>

</article>

</section>

<aside>

<blockquote>Статья номер 1</blockquote>

<blockquote>Статья номер 2</blockquote>

</aside>

<footer>

Copyright &copy; 2012-2013

</footer>

</body>

</html>

**<article>**. Содержимое веб-сайтов чаще всего делится на фрагменты, обладающие схожими характеристиками. Определить каждый из этих фрагментов позволяет тег <article>. Название элемента <article> никак не ограничивает его применение, т. е. не обязательно описывать с его помощью только новостные статьи. Элементы <article> могут описывать любые независимые части содержимого документа: публикации на форуме, статьи в журнале, записи блога, комментарии пользователей и т.п. Данный элемент всего лишь группирует связанные друг с другом фрагменты информации, независимо от характера этой информации.

|  |  |
| --- | --- |
| **<header> </header>** | |
| **<nav> </nav>** | |
| **<section>**  **<article> </article>**  **<article> </article>**  **</section>** | **<aside>**  **</aside>** |
| **<footer> </footer>** | |

Как любая независимая часть документа, содержимое каждого элемента <article> обладает собственной структурой. При определении этой структуры мы можем пользоваться преимуществами, которые дает нам универсальность тегов <header> и <footer>. Это переносимые теги, и их можно использовать не только в теле документа, но и внутри любого его раздела.

В нашем листинге описаны две статьи с помощью элемента <article>, и у каждой статьи задана своя структура. Вначале определен тег <header> с заголовками <h1> и <h2>. Далее идет основное содержимое, т.е. текст статьи. Завершает каждую статью тег <footer> с указанием количества комментариев.

**<hgroup>**. Для создания заголовков мы можем использовать теги <h>: <h1>, <h2>. <h3>, <h4>, <h5> и <h6>. Однако для ускорения обработки документа и для того чтобы во время его интерпретации не создавались множественные разделы и подразделы, эти теги желательно группировать. Для этой цели в HTML5 используется тег <hgroup>. Необходимо соблюдать иерархию тегов <h>, т.е. сначала должен быть объявлен тег <h1>, затем для подзаголовка тег <h2> и т.д. Но в отличие от предыдущих версий HTML, в HTML5 тег <h> можно использовать в каждом разделе и заново строить такую иерархию.

Элемент <hgroup> следует добавлять только в том случае, если в одном теге <header> есть заголовки разных уровней. Если нужно использовать только тег <h1>, то в <hgroup> нет необходимости.

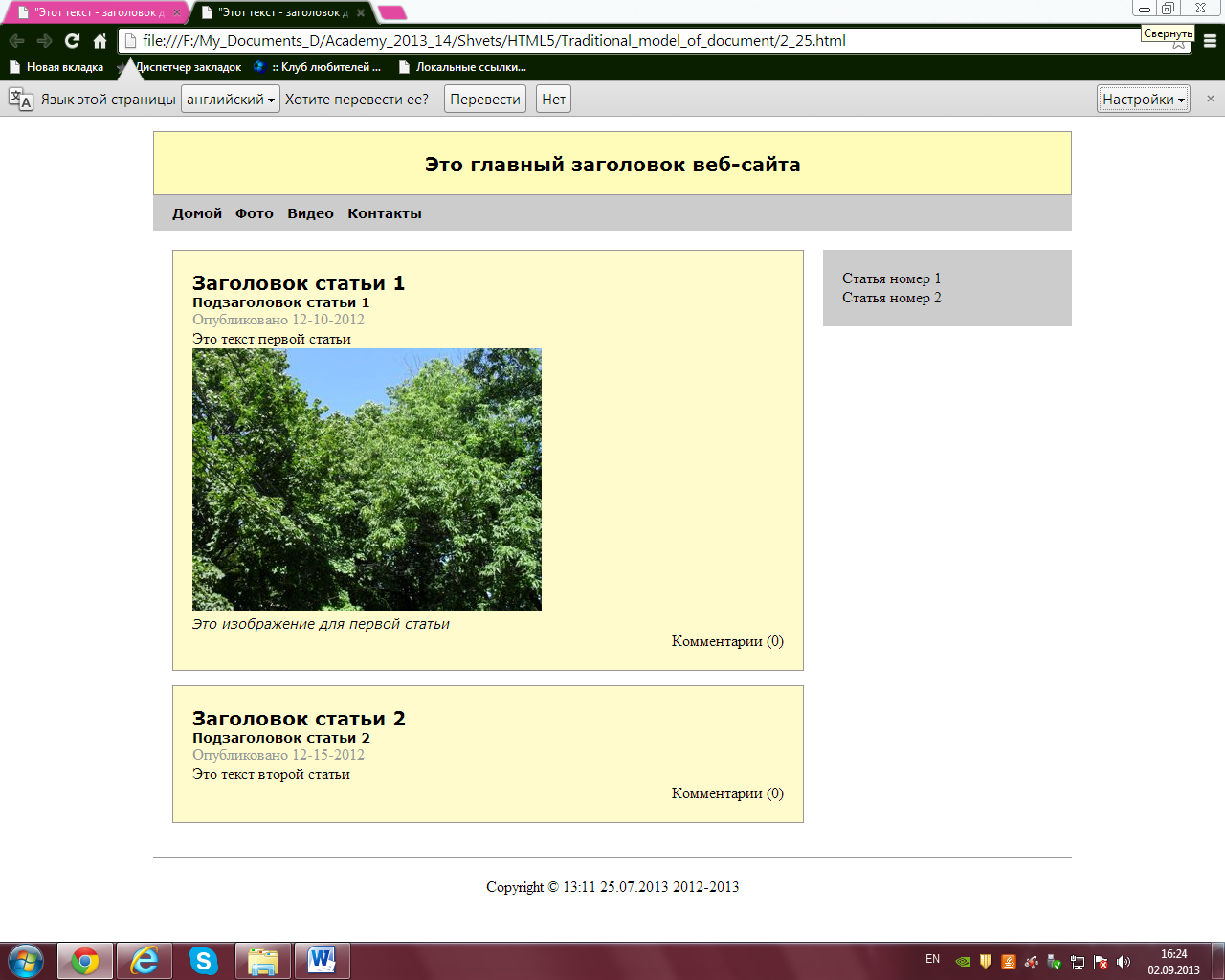
**<figure> и <figcaption>.** Элемент <figure> предназначен для более точного определения содержимого документа. До его появления невозможно было объявить содержимое, которое представляет собой изолированную часть документа: иллюстрации, рисунки, видео и т.п. Как правило. такие элементы являются частью основного раздела, но их можно спокойно удалять из документа, не нарушая его структуру. Если такая информация присутствует в документе, то она определяется тегом <figure>. В нашем листинге тег <figure> определяет визуальное дополнение текста и помогает отличить его от остальной информации в документе. Такие блоки информации, как изображения и видео, принято подписывать. В HTML5 предусмотрен специальный элемент для создания таких подписей. Тег <figcaption> определяет текст, относящийся к содержимому <figure>, и устанавливает отношение между этим элементами и их содержимым.

### Отображение документа в окне браузера

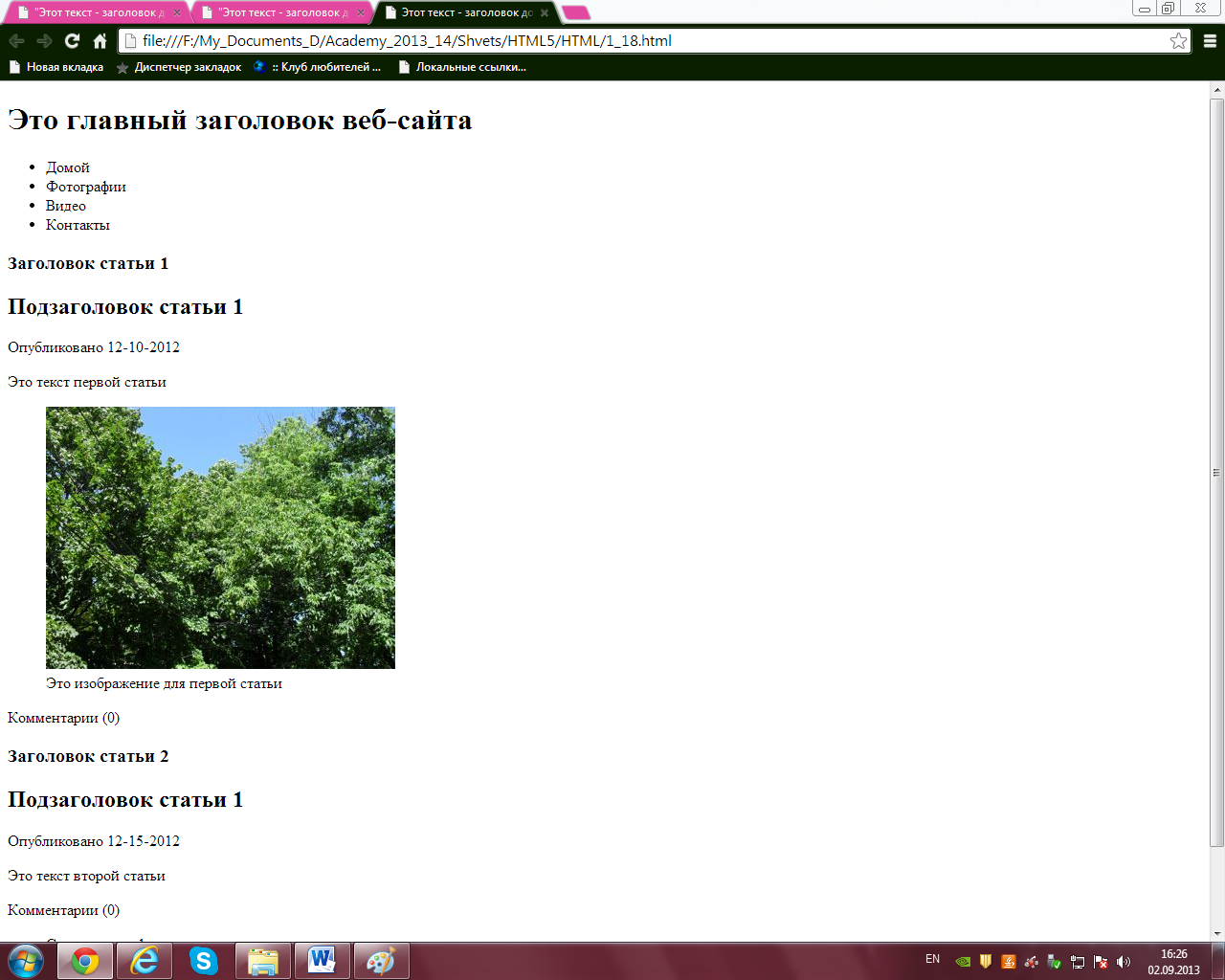
Теперь откроем наш документ в браузере. Сейчас не все браузеры поддерживают функциональность HTML5. Рекомендуется использовать новейшие версии Google Chrome и Firefox. Google Chrome основывается на WebKit – браузерном механизме с открытым кодом, который поддерживает почти все возможности, уже реализованные в HTML5.

Firefox – один из лучших браузеров для разработчиков, а его механизм Gecko также поддерживает функциональность HTML5.

Мы рассчитываем увидеть нечто похожее на следующее отображение.



Но получаем примерное такое:



Почему? Потому что, как мы уже говорили, в спецификации HTML5 HTML обеспечивает структуру документа,а CSS отвечает за его отображение. Теперь наша задача – разработать файл mystyles.css, содержащий стили, в соответствии с которыми должен отображаться наш документ. Этим мы займемся в следующей лабораторной работе.

### Контрольные вопросы и задания:

1. Что представляет собой спецификация HTML5?
2. Какова общая структура документа, созданного с использованием HTML5?
3. Опишите структуру тела документа.
4. Каково визуальное представление структуры макета веб-сайта с помощью тегов HTML5? Поясните их назначение.

### Задание для самостоятельной работы:

Ознакомиться с назначением и особенностями использования новых и старых элемнтов HTML в спецификации HTML5:

* <mark>
* <em>
* <strong>
* <b>
* <small>
* <cite>
* <address>
* <time>

# Лабораторная работа №2

**Стили CSS**

Цель работы: Базовые концепции применения стилей CSS. Новые способы определения ссылок на элементы HTML в CSS3.

## Задание:

**1**. Практически ознакомиться с базовыми и новыми способами определения ссылок на элементы в CSS3.

## Краткие теоретические сведения

Официально технология CSS никак не связана с HTML5. Она не является и никогда не была частью спецификации HTML5. В действительности это вспомогательная технология, которая разрабатывалась с целью преодоления ограничений и уменьшения сложности HTML. Первоначально некие базовые стили связывались с каждым элементом с помощью атрибутов в тегах HTML, однако, по мере того, как язык развивался, код становилось все сложнее разрабатывать и поддерживать, и вскоре оказалось, что одного HTML недостаточно для удовлетворения всех требований веб-дизайнеров. В результате на вооружение была взята технология CSS, позволяющая отделить структуру от представления. С тех пор CSS успешно развивалась, однако разработка данной технологии шла параллельно HTML и фокусировалась на нуждах дизайнеров, а не на необходимости поддерживать эволюцию HTML.

Третья версия CSS следует по аналогичному пути, однако ее разработчики принимают намного больше компромиссных решений. Спецификация HTML5 подразумевает, что за дизайн теперь отвечает CSS, из-за этого *интеграция между HTML и CSS3 стала критически важным элементом веб-разработки*.

Как известно, CSS – это язык, работающий совместно с HTML. Он связывает с элементами документа разнообразные визуальные стили, определяющие их размер, цвет, фон, рамки и т.п.

Сейчас возможности CSS3 уже встроены в последние версии большинства популярных браузеров, однако некоторые из них все еще находятся на стадии разработки. По этой причине для обеспечения их эффективной работы в названиях новых стилей необходимо использовать браузерные префиксы, такие как moz или webkit (в зависимости от механизма используемого браузера).

### Стили и стуктура

Каждый браузер по умолчанию связывает определенные стили с элементами HTML, однако эти стили не всегда соответствуют ожиданиям дизайнера. Дизайнерам и разработчикам зачастую приходится применять собственные стили, чтобы добиться желаемого оформления и организации данных на экране.

#### Блочные элементы

Что касается структуры, почти все браузеры по умолчанию располагают элементы, ориентируясь на их тип: блочный или строчный. Данная классификация определяет, как элементы выстраиваются на экране:

* блочные элементы располагаются на странице один за другим;
* строчные элементы располагаются бок о бок на одной строке, и разрыв строки не вставляется до тех пор, пока на экране хватает пространства по горизонтали.

Практически каждый струтурный элемент по умолчанию считается блочным. Это означает, что каждый тег HTML, предсталяющий собой некий компонент визуальной организации (например, <section>, <nav>, <header>, <footer>, <div>), будет помещаться ниже предыдущего.

Мы рассматривали HTML-документ, повторяющий макет типичного веб-сайта. Дизайн предусматривает горизонтальные полосы и два столбца в центре. Однако из-за использования правил отображения, по умолчанию принятых в браузерах, визуальный результат далек от ожидаемого.

Визуальное представление макета страницы, связанное с использованием стилей по умолчанию, будет выглядеть так:

|  |
| --- |
| **<header> </header>** |
| **<nav> </nav>** |
| **<section> </section>** |
| **<aside> </aside>** |
| **<footer> </footer>** |

Для того чтобы создавать собственные хорошие макеты, сначала необходимо разобраться, каким образом браузеры обрабатывают код HTML. Каждый элемент браузеры считают отдельным блоком. Веб-страница, фактически, представляет собой группу блоков, собранную воедино в соответствии с некоторыми правилами. Эти правила определяются стилями – либо встроенными в браузер, либо поставляемыми дизайнерами в форме кода CSS.

В CSS существует предустановленный набор свойств, позволяющий переопределять стили браузеров и создавать желаемый вариант дизайна. Это не какие-то узкоспециализированные свойства: их можно и нужно объединять для формирования правил, на основе которых блоки и будут группироваться, определяя правильный дизайн. Комбинацию таких правил обычно называют *моделью* или *системой макета*. Все вместе правила определяют *блочную модель*.

Сегодня только одна блочная модель считается стандартной, кроме нее существует еще несколько экспериментальных. Допустимая и повсеместноиспользуемая модель называется традиционной блочной моделью (Traditional Box Model), и она существует со времен первой версии CSS. Несмотра на доказанную эффективность данной модели, разрабатываются экспериментальные модели, нацеленные на преодоление ее недостатков. Наиболее значимой среди них является новая гибкая блочная модель (Flexible Box Model), появившаяся в CSS3 и полагаемая частью HTML5.

### Основы применения стилей

Вспомним базовые концепции применения стилей CSS.

#### Строчные стили

Одна из простейших техник определения стилей внутри элемента включает в себя использование атрибутов. Рассмотрим листинг:

<!DOCTYPE html>

<html lang="ru">

<head>

<title>Этот текст – заголовок документа</title>

</head>

<body>

<p style="font-size: 20px">Мой текст</p>

</body>

</html>

Здесь элемент <p> модифицируется атрибутом style, имеющим значение font-size: 20px. Атрибут style меняет размер по умолчанию для текста внутри элемента <p>, увеличивая его до 20 px. Однако описанную технику не рекомендуется применять для всего документа: она подразумевает, что стили нужно описать и повторить для каждого отдельного элемента, из-за чего документ раздувается до недопустимого размера, а поддерживать его и обновлять становится сложно.

#### Встроенные стили

Намного лучшая альтернатива – вставлять стили в «голову» документа, а затем ссылаться на них в соответствующих єлементах HTML.

Рассмотрим пример:

<!DOCTYPE html>

<html lang="ru">

<head>

<title>Этот текст – заголовок документа</title>

<style>

p { font-size: 20px }

</style>

</head>

<body>

<p>Мой текст</p>

</body>

</html>

Элемент <style> позволяет авторам вставлять стили CSS в код документа. В предыдущих версиях HTML необходимо было указывать, какой тип стилей будет использоваться. В HTML5 стилем по умолчанию считается CSS, следовательно, никакие другие атрибуты к открывающему тегу <style> добавлять не нужно.

Общее определение стиля в начале кода распространяется на все элементы <p> в документе. Благодаря этой технике можно сокращать объем кода и назначать желаемые стили определенным элементам с помощью ссылок.

#### Внешние файлы

Объявление стилей в «голове» документа экономит пространство и делает код более единообразным, а его поддержку – удобной, однако требует создания копии стилей в каждом документе веб-сайта. Гораздо лучшее решение – переместить все стили во внешний файл. После этого с помощью элемента <link> данный файл можно будет вставить в любой документ, требующий применения стилей. Этот метод также позволяет быстро поменять весь набор стилей, всего лишь добавив ссылку на другой файл. Кроме того, становится проще модифицировать документы и адаптировать их к различным условиям и устройствам. Например:

<!DOCTYPE html>

<html lang="ru">

<head>

<title>This text is the title of the document</title>

<link rel="stylesheet" href="mystyles.css">

</head>

<body>

<p>Мой текст</p>

</body>

</html>

Файлы CSS представляют собой обычные текстовые файлы, их можно создавать в любых текстовых редакторах.

#### Ссылки

Собирать все стилив одном внешнем файле и вставлять этот файл во все нужные документы довольно удобно. Однако нам также нужен механизм установления определенного взаимоотношения между этими стилями и элементами документа, на которые они должны влиять.

Вот способы выбора элементов HTML, которые должны меняться под влиянием правила CSS:

* по ключевому слову элемента;
* атрибуту id;
* атрибуту class.

Спецификация CSS3 довольно гибкая и включает в себя новые, более точные способы определения ссылок на элементы HTML.

*Ссылка по ключевому слову*. Объявление правила CSS с упоминанием ключевого слова элемента позволяет изменить все соответствующие элементы в документе.

p{font-size: 20px}

Но что делать, если перед нами стоит задача сослаться только на один определенный тег? Метод строчных стилей устарел, и его следует избегать. Для выбора определенного элемента HTML в правилах файла CSS можно пользоваться двумя следующими атрибутами: id и class.

*Cсылка по атрибуту id*. Атрибут id – это что-то вроде имени, идентификатор элемента. Это означает, что значение данного атрибута не может повторяться в документе. Для того чтобы сослаться из файла CSS на определенный элемент по его атрибуту id, нужно объявить правило с символом # перед значением идентификатора.

#text1{font-size: 20px}

Это правило будет применяться только к элементу HTML. идентифицируемому по атрибуту id=”text1”.

Теперь любая ссылка в файле CSS, содержащая идентификатор text1, будет влиять только на элемент с этим значением идентификатора.

Это очень точный способ определения ссылки на элемнт, и он обычно используется для элементов общего назначения, таких как структурные теги. В действительности атрибут id в силу своей специфичности больше подходит для использования в коде JavaScript.

*Ссылка по атрибуту class*. Это более гибкий атрибут, который можно связать со всеми содержащимися в документе элементами HTML, требующими одинакового дизайна.

.text1{font-size: 20px}

Один и тот же класс можно связать с разными элементами одного документа. смысл добавления точки в начале имени класса заключается в том, что одно название класса можно связывать с разными элементами и назначать каждому из них собственный стиль.

p.text1{font-size: 20px}

Это правило, ссылающееся на класс text1, предназначено исключительно для элементов <p>. Даже если атрибут class любого другого элемента будет содержать то же значение, к этому элементу данный стиль применяться не будет.

В последних версиях CSS появились новые способы определения ссылок на элементы HTML.

#### Ссылка по любому атрибуту

Теперь сослаться можно нетолько по id или class, но и по любому другому атрибуту.

p[name]{font-size: 20px}

Это определение ссылки только на элементы <p> и только имеющие атрибут name.

p[name=”mytext”]{font-size: 20px}

Это определение ссылки на элементы <p>? имеющие атрибут name со значением mytext.

CSS3 позволяет комбинировать символ = с другими, определяя еще более детальные правила выбора.

Вот новые селекторы CSS3:

p[name^=”my”]{font-size: 20px}

p[name$=”my”]{font-size: 20px}

p[name\*=”my”]{font-size: 20px}

Правило с селектором ^= применяется ко всем элементам <p>, имеющим атрибут name, значение которого начинается с my.

Правило с селектором $= применяется ко всем элементам <p>, имеющим атрибут name, значение которого заканчивается на my.

Правило с селектором \*= применяется ко всем элементам <p>, имеющим атрибут name, значение которого содержит строку my.

#### Определение ссылок по псевдоклассам

В CSS3 появились также новые псевдоклассы, еще больше увеличивающие точность выбора элементов. Благодаря псевдоклассам мы можем воспользоваться тем фактом, что элементы документа являются потомками одних и братьями других элементов. Мы можем сослаться на конкретный элемент, ничего не зная о его атрибутах и значениях.

*Псевдокласс :nth-child()*

*p:nth-child(2)*

*{*

*background: #999999;*

*}*

Псевдокласс определяется двоеточием, которое нахолится между ссылклй и названием псевдокласса. Псевдокласс :nth-child() позволяет найти определенного потомка – означаетчто-то вроде «выбрать потомка под номером …». Число в круглых скобках представляет собой позицию потомка или его индекс (отсчет с 1).

Ключевое слово odd в определении псевдокласса nth-child() влияет на элементы <p>. являющиеся потомками другого элементами и находящиеся на нечетной позиции, а ключевое слово even – на элементы, занимающие четные позиции.

Существуют и другие важные псевдоклассы с аналогичной функциональностью. Например: *first-child, last-child и only-child*. Псевдокласс first-child ссылается только на первого потомка, last-child – только на последнего, а only-child относится к элементу, который для своего предка является единственным потомком. Эти псевдоклассы не требуют ключевых слов или дополнительных параметров.

Еще один важный псевдокласс позволяет описать отрицание – это псевдокласс not().

:not(p)

{margin: 0px;}

Правило относится ко всем элементам, за исключением <p>.

#### Новые селекторы

Рассмотрим несколько селекторов, добавленных в CSS3 или считающихся частью этой спецификации. В этих селекторах с помощью символов **> + ~** описываются отношения между двумя элементами.

div>p.mytext2

{

color: #990000;

}

Селектор ^ указывает, что правило применяется ко второму элементу при условии, что он является потомком первого.

p.mytext2+p

{

color: #990000;

}Этот селектор конструируется с использованием символа +. Он позволяет сослаться на второй элемент при условии, что прямо перед ним находится первый элемент. Предок обоих элементов должен быть одним и тем же.

Селектор ~ похож на предыдущий, однако элемент, на который распространяется правило, не обязательно должен следовать сразу за первым элементом. Кроме того, элементов может быть несколько.

p.mytext2~p

{

color: #990000;

}

Стиль применяется ко всем элементам <p>, являющимся братьями и расположенными после элемента <p> с классом mytext2. Если между ними встретятся другие элементы разных типов, это не будет играть роли.

### Контрольные вопросы и задания:

1. Назовите способы выбора элементов HTML, используемые в CSS?

2.

# Лабораторная работа №3

**Традиционная блочная модель**

Цель работы: Практическое знакомство с принципами построения традиционной блочной модели HTML-документа.

## Задание: Применить к шаблону документа, созданного в лабораторной работе №1, традиционную блочную модель.

## Краткие теоретические сведения

Как мы уже говорили, каждый структутрный элемент считается блоком и структура документа представляет собой группу блоков. Собранные вместе, блоки определяют то, что называется блочной моделью.

Мы рассмотрим две разные блочные модели: традиционную блочную модель и новую гибкую блочную модель. Традиционная блочная модель используется со времен появления первой версии CSS. В настоящее время ее поддерживают все браузеры, присутствующие на рынке, и она считается стандартом веб-дизайна. В отличие от нее гибкая блочная модель, включенная в состав CSS3, находится на этапе разработки, однако ее преимущества вполне могут позволить ей занять место отраслевого стандарта, сместив с него традиционную блочную модель.

Обе модели можно применять к одним и тем же структурам HTML, однако структура документа должна быть подготовлена для использования конкретной модели, и тогда стили будут накладываься правильно. Наша задача – адаптировать документы HTML к выбранной блочной модели.

Мы уже построили шаблон документа HTML5. Он включал в себя необходимые элементы для определения структуры нашего документа, однако для того чтобы подготовить его к стилизации с помощью CSS и применению традиционной блочной модели, требуются некоторые усовершенствования.

Традиционная блочная модель требует наличия оберток вокруг блоков, для того чтобы их можно было размещать горизонтально. Поскольку все содержимое тела документа создается из группы блоков, которые должны быть определенным образом выровнены и подогнаны по размеру, следует всегда добавлять элемент <div> в качестве обертки. Новый шаблон будет выглядеть так:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<meta name="description" content="Это пример HTML5">

<meta name="keywords" content="HTML5, CSS3, JavaScript">

<title>"Этот текст - заголовок документа"</title>

<link rel="stylesheet" href="mystyles.css">

</head>

<body>

<div id="wrapper">

<header id="main\_header">

<h1>Это главный заголовок веб-сайта</h1>

</header>

<nav id="main\_menu">

<ul>

<li>Домой</li>

<li>Фото</li>

<li>Видео</li>

<li>Контакты</li>

</ul>

</nav>

<section id="main\_section">

<article>

<header>

<hgroup>

<h1>Заголовок статьи 1</h1>

<h2>Подзаголовок статьи 1</h2>

</hgroup>

<time datetime="2012-12-10" pubdate>Опубликовано 12-10-2012</time>

</header>

Это текст первой статьи

<figure>

<img src="myimage.jpg">

<figcaption>

Это изображение для первой статьи

</figcaption>

</figure>

<footer>

<p>Комментарии (0)</p>

</footer>

</article>

<article>

<header>

<hgroup>

<h1>Заголовок статьи 2</h1>

<h2>Подзаголовок статьи 2</h2>

</hgroup>

<time datetime="2012-12-15" pubdate>Опубликовано 12-15-2012</time>

</header>

Это текст второй статьи

<footer>

<p>Комментарии (0)</p>

</footer>

</article>

</section>

<aside id="main\_aside">

<blockquote>Статья номер 1</blockquote>

<blockquote>Статья номер 2</blockquote>

</aside>

<footer id="main\_footer">

Copyright &copy; 13:11 25.07.2013 2012-2013

</footer>

</div>

</body>

</html>

В этом листинге содержится несколько элементов с атрибутом id. Это означает, что у нас теперь есть возможность сослаться на конкретный элемент по значению атрибута id.

Второе важное изменение по сравнению с предыдущим шаблоном заключается в добавлении элемента <div>. Этот элемент идентифицируется атрибутом id=”wrapper”. Данная обертка позволяет создать блочную модель с содержимым тела документа и определить горизонтальное расположение элементов.

Рассмотрим таблицу стилей mystyles.css, которая позволит обработать наш HTM-документ в соответствии с требованиями традиционной блочной модели.

\* {

margin: 0px;

padding: 0px;

}

h1 {

font: bold 20px verdana, sans-serif;

}

h2 {

font: bold 14px verdana, sans-serif;

}

header, section, footer, aside, nav, article, figure, figcaption, hgroup{

display: block;

}

body {

text-align: center;

}

#wrapper {

width: 960px;

margin: 15px auto;

text-align: left;

}

#main\_header {

background: #FFFBB9;

border: 1px solid #999999;

padding: 20px;

text-align: center;

}

#main\_menu {

background: #CCCCCC;

padding: 5px 15px;

}

#main\_menu li {

display: inline-block;

list-style: none;

padding: 5px;

font: bold 14px verdana, sans-serif;

}

#main\_section {

float: left;

width: 660px;

margin: 20px;

}

#main\_aside {

float: left;

width: 220px;

margin: 20px 0px;

padding: 20px;

background: #CCCCCC;

}

#main\_footer {

clear: both;

text-align: center;

padding: 20px;

border-top: 2px solid #999999;

}

article {

background: #FFFBCC;

border: 1px solid #999999;

padding: 20px;

margin-bottom: 15px;

}

article footer {

text-align: right;

}

time {

color: #999999;

}

figcaption {

font: italic 14px verdana, sans-serif;

}

#### Универсальный селектор \*

Начнем с простейших правил, которые обеспечат единообразный дизайн.

\* {

margin: 0px;

padding: 0px;

}

Поля некоторых элементов по умолчанию ненулевой ширины и иногда оказываются даже слишком большими. Нулевая ширина полей лучше всего подходит для почти для любого документа. Дальше в коде мы будем модифицировать поля только в том случае, если для конкретного элемента их величина должна быть больше нуля.

Поле элемента – это пространство вокруг данного элемента, т.е. лежащее за пределами блока. (Забивка – это пространство вокруг содержимого элемента, но внутри блока, например, между заголовоком и границей виртуальной рамки, определяемой элементом <h1> для этого заголовка. Размер поля можно определять сразу для всех сторон элемента или для каждой стороны отдельно. Правило margin: 0px;в нашей таблице стилей определяет нулевую ширину полей со всех сторон вокруг блока. Если бы мы указали значение 5px, то блок был бы окружен пустым пространством шириной 5px, т.е. от соседних блоков его всегда отделяло бы пространство размером 5px.

#### Новая иерархия заголовков

В HTML5, как мы уже говорили, иерархию элементов H можно выстраивать заново в каждом разделе. Мы их используем и для определения главного заголовка, и во внутренних разделах. Определим подходящие для них стили:

h1 {

font: bold 20px verdana, sans-serif;

}

h2 {

font: bold 14px verdana, sans-serif;

}

#### Объявление новых элементов HTML5

Некоторые браузеры не распознают по умолчанию структурные элементы как блочные и могут обрабатывать их как строчные элементы.

header, section, footer, aside, nav, article, figure, figcaption, hgroup{

display: block;

}

Теперь элементы, на которые распространяется это правило, будут располагаться один над другим, если только позднее мы не объявим другое правило.

#### Выравнивание тела документа по центру

Первый элемент в блочной модели – это всегда <body>. Обычно содержимое данного элемента по тем или иным причинам необходимо размещать горизонтально. Кроме того, чтобы дизайн сохранял единообразие в разных конфигурациях, нужно указать размер или максимальный размер содержимого.

body {

text-align: center;

}

По умолчанию ширина тега <body> составляет 100%. Это означает, что тело документа растягивается на всю ширину окна браузера. Для того чтобы выровнять страницу по центру экрана, необходимо выровнять по центру содержимое тела документа. Благодаря нашему правилу выравниваться по центру экрана будет все, что мы поместили внутрь элемента <body>, т.е. вся страница.

#### Создание главного блока

Теперь нам нужно указать максимальный либо фиксированный размер содержимого «тела» документа. В листинге макета мы добавили к шаблону элемент <div>, обрамляющий все содержимое. Он и будет считаться главным блоком, внутри которого находятся все основные элементы. Таким образом, размер данного блока будет определять максимальный размер остальных элементов.

#wrapper {

width: 960px;

margin: 15px auto;

text-align: left;

}

Ширина современных веб-сайтов чаще всего колеблется между 960 и 980 px.

Для тега <body> мы указали, что содержимое тела документа будет выравниваться по центру, использовав для этого правило text-align: center. Однако это распространяется только на строчное содержимое, такое как текст и изображения. Для блочного содержимого, такого как элементы <div>, необходимо задавать конкретную ширину поля, обеспечивающую подгонку под размер родительского элемента. Для этого предназначено свойство margin, поддерживающее четыре значения: ширину верхнего, правого, нижнего и левого полей в указанном порядке. Однако. Если записать только первые два значения, то остальные примут такие же. В нашем правиле margin: 15px auto; стиль задает значения для верхнего и нижнего полей элемента <div> ширину 15px, а для левого и правого устанавливается автоматическое значение ширины. Таким образом, сверху и снизу тела документа всегда будет оставаться пустое пространство шириной 15 пикселов. В то же время размер пустого пространства слева и справа от этого блока будет рассчитываться автоматически в зависимости от размера тела документа и размера блока <div>. В результате содержимое блока будет выравниваться по центру окна браузера.

Теперь нужно сделать кое-что для предотвращения проблемы, которая возникает во многих браузерах. Свойство text-align наследуется иерархически. Это означает, что по центру будут выравниваться все элементы, составляющие тело документа, а не только главный блок. Действие данного стиля будет распространяться на всех потомков элементов <body>. Таким образом, необходимого для остального содержимого документа восстановить исходное значение стиля.Для этого мы добавили последнее правило.

В результате мы получаем следующую картину: содержимое тела документа выровнено по центру, однако содержимое главного блока (обертки<div>) выровнено по левому краю. Следовательно. остальной код наследует данный стиль и также по умолчанию выравнивается по левому краю.

#### Заголовок

По умолчанию ширина каждого блчного элемента, включая тело документа, составляет 100%. Это означает, что элемент растягивается по горизонтали, занимая все доступное пространство.

#main\_header {

background: #FFFBB9;

border: 1px solid #999999;

padding: 20px;

}

Для тела документа это вся видимая область экрана, однако для остальных элементов максимальная доступная ширина определяется размером родительского элемента, в нашем примере элементам внутри главного блока выделяется не более 960 пикселов.

Поскольку элемент <header>будет занимать в главном блоке все доступное по горизонтали пространство и поскольку он и так считается блочным элементом и располагается вверху страницы, нам остается связать с ним визуальные стили.

#### Навигационная полоса

Структурный элемент<nav> обеспечивает навигацию по нашему сайту. Ссылки, сгруппированные внутри этого элемента, представляют собой меню. Оно должно выглядеть как простая полоса под заголовком. <nav> относится к блочным элементам, следовательно, располагается сразу под предыдущим элементом, его ширина по умолчанию составляет 100%, поэтому он будет на экране таким же широким, как родительский элемент (обертка <div>).

Кроме того (и это уже стиль по умолчанию). его высота будет равна высоте содержимого плюс поля. Добавим серый фон и маленькие внутренние поля, отделяющие пункты меню от рамки.

#main\_menu {

background: #CCCCCC;

padding: 5px 15px;

}

#main\_menu li {

display: inline-block;

list-style: none;

padding: 5px;

font: bold 14px verdana, sans-serif;

}

Внутри навигационной полосы находится список на основе тегов <ul> и <li>. По умолчанию элементы списка располагаются в столбик. Чтобы поменять это поведение и вывести пункты меню на одной строке бок о бок и применяем правило display: inline-block, превращая их в строчные блоки. В отличие от блочных элементов, элементы, для которых определяется стандартизованный в CSS3 параметр inline-block, не создают разрывов строк. Тем не менее их также можно обрабатывать как блоки и объявлять для них значение ширины. если ширина не задается явно, то этот параметр устанавливает размер элементов равным размеру их содержимого. Кроме того, мы убрали небольшие рисунки, которые по умолчанию выводятся перед каждым пунктом списка.

#### Раздел и боковая врезка

Следующие структурные элементы в нашем коде – это два блока, находящиеся по соседству. Традиционная блочная модель строится на базе стилей CSS, позволяющих задать местоположение каждого блока. С помощью свойства float блоки можно поместить у правого или левого коая экрана в зависимости от того, какие перед нами стоят задачи.Для создания этих блоков в шаблоне HTML мы использовали теги<section> и <aside>.

#main\_section {

float: left;

width: 660px;

margin: 20px;

}

#main\_aside {

float: left;

width: 220px;

margin: 20px 0px;

padding: 20px;

background: #CCCCCC;

}

Свойство CSS float - одно из наиболее часто используемых в реализации традиционной блочной модели. Оно заставляет элемент прикрепляться к одному или другому краю доступного пространства. В зависимости от значения свойства float элементы переносятся к самому краю доступной для размещения области – как можно левее или правее. Свойство float перемещает блок к тому краю доступного пространства, на который указывает его значение.

Содержимое элемента <section> будет занимать 600 px по ширине плюс 40px на поля, итого 700px.

Для элемента <aside> будет доступным оставшееся оконное пространство, расположенное справа от <section>. Новый блок будет находиться в той же строке, что и первый, справа от него.Занимая оставшееся на этой строке место, он будет создавать второй столбец в нашем дизайне. Размер второго блока, как обычно, вычисляем по формуле: размер+поля+забивка+рамка, т.е. 220+20х2=260.

#### Нижний колонтитул

Для завершения применения традиционной блочной модели нужно создать и связать с элементом <footer> свойство, которое восстановит нормальную последовательность документа и позволит поместить <footer> ровно под последним элементом, а не в стороне.

#main\_footer {

clear: both;

text-align: center;

padding: 20px;

border-top: 2px solid #999999;

}

Мы восстанавливаем нормальную последовательность документа с помощью свойства clear. Это свойство очищает область, занимаемую элементом, и запрещает его отображение сбоку от плавающего блока. Чаще всего используется значение both, означающее, что очищаются обе стороны и теперь документ должен следовать нормальному потоку (т.е. не должен считаться плавающим). Для блочного элемента это означает, что он отображается ниже последнего элемента на новой строке.

Когда в традиционной блочной модели есть располагающиеся бок о бок блоки, всегда необходимо добавлять после них элемент со стилем clear: both; чтобы иметь возможность размещать новые блок под ними в естественном течении документа.

Следующий рисунок показывает визуальное прекдставление традиционной блочной модели и базовые стили CSS, позволяющие создать такой макет.

|  |  |
| --- | --- |
| **display: block** | |
| **display: block** | |
| **display: block**  **float: left** | **display: block**  **float: left** |
| **display: block**  **clear: both** | |

### Контрольные вопросы и задания:

1. Как строится традиционная блочная модель?
2. Каковы недостатки традиционной блочной модели?

# Лабораторная работа №4

**Принцип работы гибкой блочной модели**

Цель работы: Практическое знакомство с особенностями гибкой блочной модели и ее свойствами.

## Задание:

## 1. Продемонстрировать результат работы свойств гибкой блочной модели:

* display;
* box-orient;
* box-direction;
* box-ordinal-group;
* box-pack;
* box-flex;
* box-align.

Привести скриншоты работающих примеров.

2. Применить к макету веб-сайта, созданного в лабораторной работе №1, гибкую блочную модель

## Краткие теоретические сведения

### Принцип работы гибкой блочной модели

Разберемся как работает гибкая блочная модель и изучим ее свойства.

Одной из главных причин создания гибкой блочной модели была необходимость иметь возможность эффективно распределять экранное пространство между элементами. Приходилось уменьшать или увеличивать элементы в зависимости от того, сколько свободного места было в соответствующем контейнере. Чтобы понять, какое количество пространства доступно для распределения, нужно было точно определять размеры контейнера, и это привело к возникновению понятия родительских блоков.

Родительские блоки определяются свойством display, и их можно описывать как блочные элементы, используя значение box, либо как строчные элементы, используя значение inline-box.

Базовый HTML-код, который мы будем использовать для изучения свойств гибкой блочной модели, такой:

<!DOCTYPE html>

<html lang="en">

<head>

<title>Flexible Box Model</title>

<link rel="stylesheet" href="test.css">

</head>

<body>

<section id="parentbox">

<div id="box-1">Box 1</div>

<div id="box-2">Box 2</div>

<div id="box-3">Box 3</div>

<div id="box-4">Box 4</div>

</section>

</body>

</html>

Далее мы будем создавать правила CSS и применять их к этому документу и оценивать их работу. Правила будем добавлять в файл test.css.

#### Свойство display

Определим родительский блок как блочный элемент, используя значение box.

#parentbox {

display: -moz-box;

display: -webkit-box;

}

#### Свойство box-orient

По умолчанию родительский блок определяет горизонтальную ориентацию для всех своих потомков. С помощью свойства box-orient можно задать определенную ориентацию дочерних элементов.

#parentbox

{

display: -moz-box;

display: -webkit-box;

-moz-box-orient: vertical;

-webkit-box-orient: vertical;

}

Свойство box-orient может принимать одно из четырех значений:

* horizontal;
* vertical;
* inline-axis;
* block-axis.

#### Свойство box-direction

Как видно из вышеуказанных примеров, блоки соответствуют нормальному потоку документа и выводятся слева направо по гризонтали и сверху вниз по вертикали. Однако нормальный поток можно поменять на обратный, применив свойство box-direction, которое может принимать следующие значения:

* normal;
* reverse;
* inherit.

Разумеется, значением по умолчанию является normal.

#parentbox

{

display: -moz-box;

display: -webkit-box;

-moz-box-orient: vertical;

-webkit-box-orient: verical;

-moz-box-direction: reverse;

-webkit-box-direction: reverse;

}

#### Свойство box-ordinal-group

Порядок дочерних блоков можно настраивать совершенно произвольно. Свойство box-ordinal-group позволяет определить место каждого отдельного блока.

#parentbox

{

display: -moz-box;

display: -webkit-box;

-moz-box-orient: horizontal;

-webkit-box-orient: horizontal;

}

#box-1{

-moz-box-ordinal-group:2;

-webkit-box-ordinal-group:2;

box-ordinal-group: 2;

}

#box-2{

-moz-box-ordinal-group:4;

-webkit-box-ordinal-group:4;

box-ordinal-group: 4;

}

#box-3{

-moz-box-ordinal-group:3;

-webkit-box-ordinal-group:3;

box-ordinal-group:3;

}

#box-4{

-moz-box-ordinal-group:1;

-webkit-box-ordinal-group:1;

box-ordinal-group:1;

}

Как мы уже говорили, самая важная особенность гибкой блочной модели заключается в возможности эффетивно распределять пространство между экранными элементами. Возможны самые разные ситуации, когда доступное пространство приходится делить между дочерними блоками, например: они принадлежат гибкому родительскому блоку, они сами гибкие и для них не определены конкретные размеры или же ширина родительского блока больше суммы ширины всех его дочерних блоков.

Рассмотрим пример:

* #parentbox {
* display: -moz-box;
* display: -webkit-box;
* -moz-box-orient: horizontal;
* -webkit-box-orient: horizontal;
* width: 600px;
* }
* #box-1{
* width: 100px;
* }
* #box-2{
* width: 100px;
* }
* #box-3{
* width: 100px;
* }
* #box-4{
* width: 100px;

}

В нашем случае размер родительского блока фиксирован и равен 600 px, а размер каждого дочернего блока - 100 px. Таким образом у нас остается 200 px свободного пространства, которое нужно разделить между дочерними блоками.

Распределить свободное пространство можно разными способами. По умолчанию дочерние блоки выводятся в последовательности, как указано в документе, слева направо и вплотную друг к другу. Таким образом, в конце остантся свободное место. Однако есть и альтернативные варианты, которые мы должны рассмотреть.

#### Сойство box-pack

Свойство box-pack указывает, каким образом в родительском окне размещаются дочерние блоки и учитывается свободное пространство. Данное свойство может принимать четыре значения:

* start;
* end;
* center;
* justify.

#parentbox {

display: -moz-box;

display: -webkit-box;

-moz-box-orient: horizontal;

-webkit-box-orient: horizontal;

width: 600px;

-moz-box-pack: center;

-webkit-box-pack: center;

}

#box-1{

width: 100px;

}

#box-2{

width: 100px;

}

#box-3{

width: 100px;

}

#box-4{

width: 100px;

}

Свойство box-pack по-разному влияет на блоки в зависимости от их ориентации. Если блоки ориентированы горизонтально, то box-pack меняет вариант распределения только горизонатального свободного пространства. Точно так же для вертикальных блоков оно определяет, как будет использоваться свободное вертикальное пространство.

#### Свойство box-flex

До сих пор мы делали то, что в определенном смысле противоречит принципам рассматриваемой модели. Мы не пользовались преимуществами, предлагаемыми гибкими элементами. Свойство box-flex позволяет объявить блок как гибкий или негибкий, что помогает эффективно распределять пространство. По умолчанию блоки считаются негибкими, а значение данного свойства равно нулю. Присваивая ему значение 1 или больше, вы объявляете гибкие блоки. Такие блоки растягиваются или сжимаются для заполнения дополнительного пространства. Их размер может меняться как по горизонтали, так и по вертикали – в зависимости от того, какая ориентация была объявлена в свойствах родительского блока.

Распределение пространства также зависит от свойств остальных блоков. Если все дочерние блоки объявлены как гибкие, то размер каждого из них определяется размером родительского блока и значением свойства box-flex.

Рассмотрим пример:

#parentbox {

display: -moz-box;

display: -webkit-box;

-moz-box-orient: horizontal;

-webkit-box-orient: horizontal;

width: 600px;

}

#box-1{

-moz-box-flex: 1 ;

-webkit-box-flex: 1 ;

}

#box-2{

-moz-box-flex: 1 ;

-webkit-box-flex: 1 ;

}

#box-3{

-moz-box-flex: 1 ;

-webkit-box-flex: 1 ;

}

#box-4{

-moz-box-flex: 1 ;

-webkit-box-flex: 1 ;

}

Размер каждого из блоков рассчитывается так: размер родительского блока умножается на значение свойства box-flex данного дочернего блока, а затем делится на сумму значений свойств box-flex всех дочерних элементов. В нашем примере формула для блока 1 будет выглядеть так: 600Х1/4.

Потенциал данного свойства становится очевиден, когда для разных блоков определяются разные значения box-flex, когда гибкие блоки сочетаются с негибкими и когда мы задаем точные размеры гибких блоков.

#parentbox {

display: -moz-box;

display: -webkit-box;

-moz-box-orient: horizontal;

-webkit-box-orient: horizontal;

width: 600px;

}

#box-1{

-moz-box-flex: 2 ;

-webkit-box-flex: 2 ;

}

#box-2{

-moz-box-flex: 1 ;

-webkit-box-flex: 1 ;

}

#box-3{

-moz-box-flex: 1 ;

-webkit-box-flex: 1 ;

}

#box-4{

-moz-box-flex: 1 ;

-webkit-box-flex: 1 ;

}

Теперь формула для вычисления размера блока box-1 выглядит так: 600х2/5=240. Применив ту же формулу ждя остальных дочерних блоков, определяем их размер:

600х1/5=120.

Сравнивая результаты, мы видим, что теперь пространство распределено по-другому. Доступное пространство было разделено на количество частей, равное сумме значений свойства box-flex всех дочерних блоков (в нашем случае 5). После этого получившиеся части распределены между блоками. Блок номер 1 получил две части, а остальные получили по одной, т.к. их box-flex равен 1.

Возможны и другие сценарии. Например, когда один из дочерних блоков по умолчанию негибкий и для него явно не задан размер, остальные дочерние блоки делят между собой оставшееся свободное пространство.

#parentbox {

display: -moz-box;

display: -webkit-box;

-moz-box-orient: horizontal;

-webkit-box-orient: horizontal;

width: 600px;

}

#box-1{

width: 300px;

}

#box-2{

-moz-box-flex: 1 ;

-webkit-box-flex: 1 ;

}

#box-3{

-moz-box-flex: 1 ;

-webkit-box-flex: 1 ;

}

#box-4{

-moz-box-flex: 1 ;

-webkit-box-flex: 1 ;

}

Здесь размер первого блока равен 300 px, поэтому оставшееся пространство, которое будет делиться между остальными потомками, равно 600-300=300px. Браузер вычислит размер каждого гибкого блока по нашей формуле: 300х1.3=100.

Размер может быть задан явно как для одного блока, так и для нескольких. Тем не менее принцип не меняется: между остальными блоками распределяется только свободное пространство.

Кроме того, можно объявить гибкими блоки, обладающие конкретными размерами.

#parentbox {

display: -moz-box;

display: -webkit-box;

-moz-box-orient: horizontal;

-webkit-box-orient: horizontal;

width: 600px;

}

#box-1{

width: 200px;

-moz-box-flex: 1 ;

-webkit-box-flex: 1 ;

}

#box-2{

width: 100px;

-moz-box-flex: 1 ;

-webkit-box-flex: 1 ;

}

#box-3{

width: 100px;

-moz-box-flex: 1 ;

-webkit-box-flex: 1 ;

}

#box-4{

width: 100px;

-moz-box-flex: 1 ;

-webkit-box-flex: 1 ;

}

В данном случае для каждого блока указана предпочтительная ширина, однако для размещения всех блоков у нас остается еще 100px свободного пространства. Это свободное место делится между гибкими блоками. Для вычисления доли, которая достанется каждому из блоков, используется уже знакомая нам формула: 100х1/4=25. Это означает, что к предпочтительной ширине каждого блока будет добавлено еще 25px.

#### Свойство box-align

Еще одно свойство, помогающее распределять оконное пространство, называется box-align. Оно работает аналогично box-pack, однако выравнивает блоки в направлении, не совпадающем с ориентацией блоков. Для блоков с вертикальной ориентацией оно определяет позиционирование по горизонтали и наоборот. Данное свойство удобно применять для выравнивания блоков по вертикали – этой возможности очень не хватало разработчикам после того, как повсеместно отказались от таблиц.

#parentbox {

display: -moz-box;

display: -webkit-box;

-moz-box-orient: horizontal;

-webkit-box-orient: horizontal;

width: 600px;

height: 200px;

-moz-box-align: center;

-webkit-box-align: center;

}

#box-1{

height: 100px;

-moz-box-flex: 1 ;

-webkit-box-flex: 1 ;

}

#box-2{

height: 100px;

-moz-box-flex: 1 ;

-webkit-box-flex: 1 ;

}

#box-3{

height: 100px;

-moz-box-flex: 1 ;

-webkit-box-flex: 1 ;

}

#box-4{

height: 100px;

-moz-box-flex: 1 ;

-webkit-box-flex: 1 ;

}

В листинге мы указали точное значение высоты для всех блоков, включая родительский. Оставшееся свободное пространство высотой 100 px будет распределяться в соответствии со значением свойства box-align.

Свойство box-align может принимать следующие значения:

* start;
* end;
* center;
* baseline;
* stretch.

Последнее значение растягивает блоки по вертикали, подгоняя размер дочерних блоков под размер свободного пространства.

Эта характеристика настолько важна,что именно значение stretch выбрано значением по умолчанию. В результате применения данного значения все дочерние блоки автоматически подгоняются под высоту их родительского блока, независимо от того, какая высота задана для них самих.

#### Гибкая блочная модель

Гибкая блочная модель элегантно решает проблемы традиционной блочной модели. Благодаря новой реализации у нас появились блоки, представляющие виртуальные строки и столбцы. Теперь мы получили полный контроль над макетом, позициями и размерами блоков, распределением блоков в общем оконном пространстве. Гибкая блочная модель все еще находится на экспериментальной стадии разработки.

Одна из главных особенностей данной модели заключается в том, что некоторые возможности (например, ориентация – вертикальная или горизонтальная) объявляются в родительских блоках. В новой модели у каждого набора блоков обязательно должен быть родительский блок.

В шаблоне нашего веб-сайта несколько родительских блоков уже определены. Элемент <body> и <div> можно с успехом превратить в родительские блоки. Одной родительский блок требуется и для еще одной составляющей структуры. Мы добавим новый элемент <div> для обрамления поднабора блоков, представляющего два столбца в середине страницы (в коде документа они определяются элементами<section> и <aside>. Вот как наш шаблон будет выглядеть сейчас:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<meta name="description" content="Это пример HTML5">

<meta name="keywords" content="HTML5, CSS3, JavaScript">

<title>"Этот текст - заголовок документа"</title>

<link rel="stylesheet" href="mystyles.css">

</head>

<body>

**<div id="wrapper">**

<header id="main\_header">

<h1>Это главный заголовок веб-сайта</h1>

</header>

<nav id="main\_menu">

<ul>

<li>Домой</li>

<li>Фото</li>

<li>Видео</li>

<li>Контакты</li>

</ul>

</nav>

**<div id=”container”>**

<section id="main\_section">

<article>

<header>

<hgroup>

<h1>Заголовок статьи 1</h1>

<h2>Подзаголовок статьи 1</h2>

</hgroup>

<time datetime="2012-12-10" pubdate>Опубликовано 12-10-2012</time>

</header>

Это текст первой статьи

<figure>

<img src="myimage.jpg">

<figcaption>

Это изображение для первой статьи

</figcaption>

</figure>

<footer>

<p>Комментарии (0)</p>

</footer>

</article>

<article>

<header>

<hgroup>

<h1>Заголовок статьи 2</h1>

<h2>Подзаголовок статьи 2</h2>

</hgroup>

<time datetime="2012-12-15" pubdate>Опубликовано 12-15-2012</time>

</header>

Это текст второй статьи

<footer>

<p>Комментарии (0)</p>

</footer>

</article>

</section>

<aside id="main\_aside">

<blockquote>Статья номер 1</blockquote>

<blockquote>Статья номер 2</blockquote>

</aside>

**</div>**

<footer id="main\_footer">

Copyright &copy; 13:11 25.07.2013 2012-2013

</footer>

**</div>**

</body>

</html>

Итак, шаблон готов. Создадим новый файл mystyles.css. Сначала добавим в новую таблицу стилей базовые правила, общие для обеих моделей.

\* {

margin: 0px;

padding: 0px;

}

h1 {

font: bold 20px verdana, sans-serif;

}

h2 {

font: bold 14px verdana, sans-serif;

}

header, section, footer, aside, nav, article, figure, figcaption, hgroup{

display: block;

}

Теперь, когда базовые правила готовы, можно связать с нашим шаблоном гибкую блочнуб модель.

В этой модели каждый элемент, внутри которого находятся структурные элементы, должен быть объявлен как родительский блок. Первый родительский блок в нашем документе – это само тело документа.

body {

width: 100%;

display: -moz-box;

display: -webkit-box;

-moz-box-pack: center;

-webkit-box-pack:center;

}

Для того чтобы сделать элемент родительским блоком, нужно связать с ним свойство display и присвоить ему значение box. Мы также определили стиль, выравнивающий содержимое элемента <body> по центру. Значение center свойства box-pack выравнивает по центру дочерние элементы родительского блока. В нашем случае у <body> только один дочерний элемент, а именно <div id=”wrapper”>, поэтому настройка выравнивания распространяется на всю веб-страницу.

Еще одна важная характеристика данной модели заключается в возможности с легкостью увеличивать и уменьшать оазмер любого элемента веб-страницы в зависимости от доступного свободного пространства. Однако для того чтобы блок обладал подобной гибкостью, он должен быть потомком такого же гибкого родительского элемента. Размер элементов невозможно корректировать. если неизестен размер их родительских элементов, поэтому необходимо указать, что тело документа займет все простанство окна браузера. Для этого мы объявили ширину 100%.

Одно из новых свойств CSS3 называется box-orient. Оно определяет вертикальную или горизонтальную ориентацию дочерних элементов. Значение по умолчанию равно horizontal, поэтому для элемента <body> его указывать не нужно, однако для обертки оно потребуется.

#wrapper {

max-width: 960px;

margin: 15px 0px;

display: -moz-box;

display: -webkit-box;

-moz-box-orient: vertical;

-webkit-box-orient: vertical;

-moz-box-flex; 1;

-webkit-box-flex; 1;

}

Блок, для которого объявляется свойство display: box обладает свойствами блочного элемента и занимает все свободное пространство в своем контейнере. В предыдущей модели мы использовали фиксированное значение ширины, равное 960px. Это значение фиксировало размер не только данного блока, но и всей веб-страницы. Для того чтобы воспользоваться преимуществами гибких свойств новой блочной модели, необходимо снизить уровень точности. Если <body> уже занимает 100% окнато обертка будет вести себя очно также и в некоторых случаях это приведет к искажению пропорций страницы. чтобы избежать этого, но все же иметь на странице гибкое содержимое, мы в вышеуказанном правиле использовали свойство max-width со значением 960px. Таким образом мы определили переменную ширину для обертки (и, следовательно, для всей веб-страницы), которая никогда не будет превышать 960px. Теперь размер веб-страницы будет адаптироваться под каждое устройство и любые условия, а затем, чтобы единообразие дизайна не нарушалось, будет следить свойство, определяющее максимальную ширину.

Так как блок <div> обрамляет все содержимое веб-страницы, его также необходимо объявить родительским блоком, применив для этого display: box. На этот раз его дочерние элементы будут располагаться один под другим, поэтому мы присвоилисвойству bo-orient значение vertical.

Для того чтобы сделать родительский блок гибким мы добавили свойство box-flex. Без этого свойства блок <div> невозможно было бы увеличить или уменьшить: его ширина всегда совпадала бы с шириной его содержимого. Значение 1 определяет гибкий блок, 0 – фиксированный.

Далее в нашем HTML-документе встречаемся с блоками, создаваемыми элементами <header> и <nav>. Это первые потомки родительского блока <div id=”wrapper”>. В следующем листинге мы лишь объявим некоторые визуальные стили; что касается позиционирования, эти элементы уже обладают свойствами вертикальных блоков, унаследованных от родительского блока.

Под блоком, созданным тегом <nav>, находится еще один блок, являющийся предком для двух потомков. Это новый блок <div>, который мы добавили для обрамления столбцов в центре веб-страницы.

#main\_header {

background: #FFFBB9;

border: 1px solid #999999;

padding: 20px;

}

#main\_menu {

background: #CCCCCC;

padding: 5px 15px;

}

#main\_menu li {

display: inline-block;

list-style: none;

padding: 5px;

font: bold 14px verdana, sans-serif;

}

Следующее правило определяет родительский блок для двух столбцов в середине страницы.

#container {

display: -moz-box;

display: -webkit-box;

-moz-box-orient: horizontal;

-webkit-box-orient: horizontal;

}

Первый стиль создает блок, а второй определяет горизонтальную ориентацю его дочерних элементов. Этот контейнер не нужно делать гибким, так же как мы не определяли гибкие блоки для заголовка и строки меню. Его размер по умолчанию равен 100%, поэтому он занимает все доступное пространство, предоставляемое контейнером. внутрикоторого он находится. Кроме того, поскольку родительский элемнт данного блока гибкий, то и сам блок тоже гибкий.

Подготовив стили для родительских элементов, переходим к работе над столбцами.

#main\_section {

-moz-box-flex; 1;

-webkit-box-flex: 1;

margin: 20px;

}

#main\_aside {

width: 220px;

margin: 20px 0px;

padding: 20px;

background: #CCCCCC;

}

После применения этих правил первый столбец, определяемый элементом <section>, станет гибким, т.е. потряет фиксированный размер.

Комбинация гибких и фиксированных столбцов – не уникальное свойство гибкой блочной модели, однако ее возможности весьма широки, и некоторые эффекты, для разработки которых когда-то требовалась не одна строка кода, теперь можно реализовать с помощью нескольких простых свойств.

Есть еще одно важное свойство, которое нам объявлять не требуется, т.к. оно автоматически связывается с обоими столбцами. Свойство box-align со значением по умолчанию stretch растягивает столбцы по вертикали так, чтобы оно занимали все доступное пространство. Благодаря ему правый столбец в нашем шаблоне занимает столько же места по вертикали, сколько и левый.

Результат применения этих свойств – явным образом или по умолчанию – заключается втом, что ширина и высота определенного содержимого веб-страницы могут меняться в зависимости от того, сколько свободного места осталось в блоке. столбец, создаваемый элементом <section>, сможет растягиваться и сжиматься по горизонтали, как и его содержимое. Кроме того, если мы уменьшим или увеличим окно браузера, то автоматически поменяется размер заголовка, навигационной полосы ти нижним колонтитулом. Всех этих эффектов было сложно достичь в старой блочной модели.

Теперь у нас есть все необходимые стили для организации блоков и применения гибкой блочной модели к нашему шаблону. Осталось только добавить немного визуальных стилей к оставшимся элементам, чтобы восстановить предыдущий вариант офрмления.

#main\_footer {

text-align: center;

padding: 20px;

border-top: 2px solid #999999;

}

article {

background: #FFFBCC;

border: 1px solid #999999;

padding: 20px;

margin-bottom: 15px;

}

article footer {

text-align: right;

}

time {

color: #999999;

}

figcaption {

font: italic 14px verdana, sans-serif;

}

Теперь соберем все CSS-стили в файл mystyles.css и проверим результат отображения нашего шаблона.

### Контрольные вопросы и задания:

1. Каково назначение гибкой блочной модели?
2. Поясниет назначение и особенности работы свойств гибкой блочной модели:

* display;
* box-orient;
* box-direction;
* box-ordinal-group;
* box-pack;
* box-flex;
* box-align.

# Лабораторная работа №5

**Свойства CSS3**

Цель работы:

## Задание:

## Краткие теоретические сведения

Рассмотрим, какой вклад внесла спецификация CSS3 в развитие HTML5, а также ее новые свойства.

Спецификация CSS3 всегда относилась только к внешнему виду и форматированию, но теперь все поменялось. В попытке сократить использование JavaScript и в целях стандартизациипопулярных возможностей разработчики CSS3 описали не только дизайн и веб-стили, но также форму и движение.

Спецификация CSS3 разбита на модули, обеспечивающие стандартизацию каждого аспекта, участвующего в визуальном представлении документа. От скругленных углов и теней до трансформации и реорганизации элементов, уже визуализированных на экране – разработчикам предоставляется возможность реализовать любые эффекты, при создании которых раньше было не обойтись без JavaScript. Благодаря таким обширным изменениям спецификайия CSS3 превратилась в совершенно новую технологию по сравнению с предыдущими версиями.

#### Шаблон

Для изучения новых свойств CSS3 создадим шаблон:

<!DOCTYPE html>

<html lang="en">

<head>

<title>New CSS3 styles</title>

<link rel="stylesheet" href="newcss3.css">

</head>

<body>

<header id="mainbox">

<span id="title">CSS Styles Web 2.0</span>

</header>

</body>

</html>

В нашем документе содержится всего один раздел с коротким текстом. Элемент <header>, который мы использовали в данном шаблоне, можно было бы заменить на <div>, <span>, <section>или любой другой структурный элемент в зависимости от его местоположения и выполняемой функции. После того как мы определим стили, блок из нашего примера будет выглядеть как заголовок – вот почему в код добавлен именно элемент <header>.

так как от элемента <font> в HTML5 отказались, для отображения текста обычно используют тег <span>, если речь идет о короткой строке, или <p>, если это целый абзац. Можно оформлять текст и с помощью других тегов.

Создадим файл newcss3.css, в котором будем сохранять стили.

Вот базовые правила CSS, с которых все начинается.

body {

text-align: center;

}

#mainbox {

display: block;

width: 500px;

margin: 50px auto;

padding: 15px;

border: 1px solid #999999;

background: #FFFFFF;

}

#title {

font: bold 36px verdana, sans-serif;

}

Как видно на экране, у блока прямые углы. Изменим углы этого поля.

#### Свойство border-radius

В течение многих лет скругленные углы представляли собой невероятную сложность для разработчиков. Посмотрим, как легко это решается сейчас:

body {

text-align: center;

}

#mainbox {

display: block;

width: 500px;

margin: 50px auto;

padding: 15px;

border: 1px solid #999999;

background: #FFFFFF;

-moz-border-radius: 20px;

-webkit-border-radius: 20px;

border-radius; 20px;

}

#title {

font: bold 36px verdana, sans-serif;

}

Если все углы должны быть одинаковыми. то можно свойству передать только одно значение. Однако аналогично свойствам margin и padding, данное свойство позволяет указывать индивидуальные значения для каждого угла.

body {

text-align: center;

}

#mainbox {

display: block;

width: 500px;

margin: 50px auto;

padding: 15px;

border: 1px solid #999999;

background: #FFFFFF;

-moz-border-radius: 20px 10px 30px 50px;

-webkit-border-radius: 20px 10px 30px 50px;

border-radius: 20px 10px 30px 50px;

}

#title {

font: bold 36px verdana, sans-serif;

}

Четыре значения, связанные со свойством border=radius, представляют четыре местоположения в следующем порядке: верхний левый угол, верхний правый угол, нижний правый угол и нижний левый угол.

Так же как свойства margin и padding, свойство border-radius способно принимать набор всего из двух значений. В таком случае первое значение связывается с первым и третьим углами,а второе – со вторым и четвертым (углы отсчитываются по часовой стрелке, начиная с левого верхнего).

Для определения формы углов можно указывать двойные значения, разделенные косой чертой. в таком случае значение слева от косой черты представляет горизональный радиус, а значение справа – вертикальный радиус. Сочетание таких значений определяет форму эллипса.

body {

text-align: center;

}

#mainbox {

display: block;

width: 500px;

margin: 50px auto;

padding: 15px;

border: 1px solid #999999;

background: #DDDDDD;

-moz-border-radius: 20px / 10px;

-webkit-border-radius: 20px / 10px;

border-radius: 20px / 10px;

}

#title {

font: bold 36px verdana, sans-serif;

}

#### Свойство box-shadow

Благодаря CSS3 и свойству box-shadow можно добавить тень к нашему полю:

body {

text-align: center;

}

#mainbox {

display: block;

width: 500px;

margin: 50px auto;

padding: 15px;

border: 1px solid #999999;

background: #DDDDDD;

-moz-border-radius: 20px;

-webkit-border-radius: 20px;

border-radius: 20px;

-moz-box-shadow: rgb(150, 150, 150) 5px 5px;

-webkit-box-shadow: rgb(150, 150, 150) 5px 5px;

box-shadow: rgb(150, 150, 150) 5px 5px;

}

#title {

font: bold 36px verdana, sans-serif;

}

Свойству box-shadow необходимо передать, как минимум, три значения. Первое - это цвет. Для формирования этого значения мы использовали функцию rgb(), передав ей необходимые аргументы в десятичной системе счисления, однако можно записать цвет в шестнадцатеричном представлении.

следующие два значения, указываемые в пикселах, задают смещение для тени. Оно может быть положительным или отрицательным. Эти значения определяют расстояние от тени до элемента по горизонтали и по вертикали соответственно. Отрицательные значения сдвигают тень влево и вверх, и наоборот, положительные значения создают тень справа и ниже элемента. Нулевое значение позволяет поместить иень прямо за элемент, например, для создания эффекта размытости вокруг него.

Пока что тень получается сплошной, без градиентов или эффекта прозрачности, и, следовательно, не очень похожей на настоящую. Мы можем добавить еще несколько параметров и улучшить ее внешний вид.четвертое значение, которое принимает данное свойство, задает радиус размытия.

-moz-box-shadow: rgb(150, 150, 150) 5px 5px 10px;

-webkit-box-shadow: rgb(150, 150, 150) 5px 5px 10px;

box-shadow: rgb(150, 150, 150) 5px 5px 10px;

Последнее значение в пикселах, добавляемое в конце свойства, определяет рассеяние тени. Этот эффект немного меняет саму природу тени, расширяя охватывающую область. Как правило, использовать его не рекомендуется, однако в некоторых случаях оно может оказаться уместным.

#### box-shadow: rgb(150, 150, 150) 5px 5px 10px 20px;

Последнее возможное значение свойства box-shadow – это не число, а ключевое слово inset. Оно превращает внешнюю тень во внутреннюю, создавая эффект вдавленного поля.

-moz-box-shadow: rgb(150, 150, 150) 5px 5px 10px inset;

-webkit-box-shadow: rgb(150, 150, 150) 5px 5px 10px inset;

box-shadow: rgb(150, 150, 150) 5px 5px 10px inset;

Этот стиль определяет внутреннюю тень на расстоянии 5px от рамки поля, которая дополнительно размывается на 10 px.

Тени не растягивают элемент и не уваличивают его размер, поэтому необходимо тщательно проверять доступное пространство.

#### Свойство text-shadow

Свойство box-shadow специально предназначено для добавления тени к полям. Для добавления тени к фигурам неправильной формы, т.е. текстовым символам, используется свойство text-shadow.

Значения, которые принимает text-shadow аналогичны значениям box-shadow. Можно указать цвет тени, расстояние от объекта по горизонтали и по вертикали, а также радиус размытия. В нашем примере к заголовку добавлена синяя тень. Она находится на расстоянии 3 px от букв, а радиус ее размытия равен 5px.

#### Свойство @font-face

Свойство @font-face позволяет дизайнерам подключать определенный шрифт и применять его для отобоажения текста на веб-странице. Теперь любой желаемый шрифт можно встроить в веб-сайт, просто бобавив файл этого шрифта.

Например, новый шрифт для заголовка:

body {

text-align: center;

}

#mainbox {

display: block;

width: 500px;

margin: 50px auto;

padding: 15px;

border: 1px solid #999999;

background: #FFFFFF;

}

#title {

font: bold 36px MyNewFont, verdana, sans-serif;

text-shadow: rgb(0,0,150) 3px 3px 5px;

}

@font-face {

font-family: 'MyNewFont';

src: url('font.ttf');

}

Много бесплатных шрифтов можно загрузить со страницы http://www.moorstation.org/typoasis/designers/steffmann/.

Свойство @font-face требует указания еще по крайней мере двух свойств для объявления шрифта и загрузки файла. Свойство font-family задает название, с помощью которого мы будем ссылаться на данный шрифт, а свойствоsrc – URL-адрес файла с кодами для визуализации шрифта.

Файл шрифта должен находиться в том же домене, что и веб-страница (или в нашем случае на том же ПК). такое ограничение действует в нескольких браузерах, например, Firefox.

После того как шрифт загружен, его можно использовать с любыми элементами документа, всего лишь указав имя шрифта (MyNewFont).

#### Линейный градиент

Градиенты – одна из самых привлекательных новинок в CSS3.

body {

text-align: center;

}

#mainbox {

display: block;

width: 500px;

margin: 50px auto;

padding: 15px;

border: 1px solid #999999;

background: #DDDDDD;

-moz-border-radius: 20px;

-webkit-border-radius: 20px;

border-radius: 20px;

-moz-box-shadow: rgb(150, 150, 150) 5px 5px 10px;

-webkit-box-shadow: rgb(150, 150, 150) 5px 5px 10px;

box-shadow: rgb(150, 150, 150) 5px 5px 10px;

background: -webkit-linear-gradient(top , #FFFFFF, #666666);

background: -moz-linear-gradient(top , #FFFFFF, #666666);

}

#title {

font: bold 36px MyNewFont, verdana, sans-serif;

text-shadow: rgb(0,0,150) 3px 3px 5px;

}

@font-face {

font-family: 'MyNewFont';

src: url('font.ttf');

}

Градиенты определяются как фоны, поэтому для их добавления на страницу можно использовать свойства background и background-image. Синтаксис значений, присваивакмых этим свойствам таков:

linear-gradient(начальная\_позиция, начальный\_цвет, конечный\_цвет)

Первым может идти значение в пикселах или процентах, или одно из ключевых слов: top, bottom, left или right. В качестве начальной позиции можно также указать угол, для того чтобы задать точное направление градиента.

background: -webkit-linear-gradient(30deg, #FFFFFF, #666666);

background: -moz-linear-gradient(30deg, #FFFFFF, #666666);

Помимо этого для каждого цвета можно определить конечную точку.

background: -webkit-linear-gradient(top, #FFFFFF 50%, #666666 90%);

background: -moz-linear-gradient(top, #FFFFFF 50%, #666666 90%);

#### Радиальный градиент

Стандартный синтаксис для радиального градиента отличается от рассмотренного синтаксиса для линейного градиента всего в нескольких аспектах. необходимо использовать функцию radial-gradient() и новый атрибут, определяющий форму.

background: -webkit-radial-gradient(center, circle, #FFFFFF, #000000);

background: -moz-radial-gradient(center, circle, #FFFFFF, #000000);

Начальное положение – это источник градиента, и его можно задать как значение в пикселах или процентах, или как комбинацию ключевых слов center, top, bottom, left и right. Для определения фигуры доступны два ключевых слова – circle и ellipse, а описание промежуточных точек включает в себя значение цвета и позиции, в которой начинается переход.

RGBA

**Швець Н.В.**

## ПРОГРАМУВАННЯ ДЛЯ INTERNET

#### Посібник до виконання лабораторних робіт

Підписано до друку ?????201Х р. Формат 60×84 1/16.

Умовн. друк. арк. **х**,**х**. Наклад **х** прим.

Надруковано видавницький центром ОНАХТ «Технолог».

65039, Одеса, вул. Канатна, 112