Даниела Борисова

ОСНОВИ
НА УЕБ ПРОГРАМИРАНЕТО

София
2014

Европейски съюз

Европейски социален фонд

Учебникът е написан
по проект „Разработване на система за управление
на знанията във Факултета по информационни науки на УниБИТ“
Договор: BG051PO001-4.3.04-0066

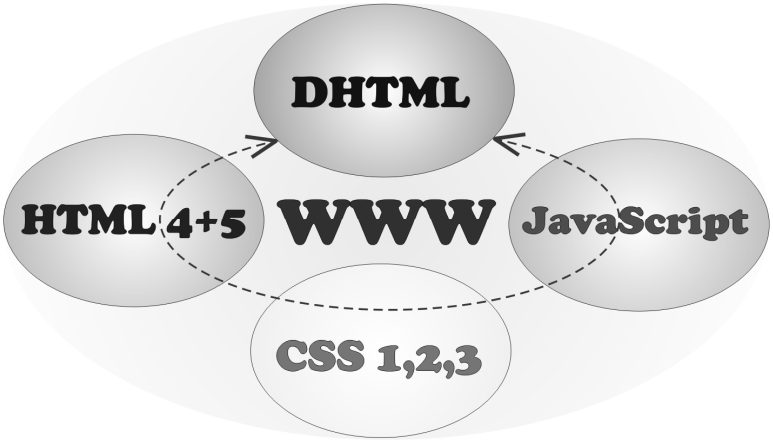
Проектът се осъществява с финансовата подкрепа
на Оперативна програма „Развитие на човешките ресурси“,
схема за безвъзмездна финансова помощ BG051PO0001-4.3.04
„Развитие на електронни форми на дистанционно обучение
в системата на висшето образование“,
съфинансирана от Европейския социален фонд на Европейския съюз.

Настоящият учебник е изготвен с финансовата помощ на Европейския социален фонд. УниБИТ носи цялата отговорност за съдържанието на настоящия учебник, и при никакви обстоятелства не може да се приеме като официална позиция на Европейския съюз.

ISBN 978-619-185-052-5

Даниела Борисова

ОСНОВИ НА УЕБ ПРОГРАМИРАНЕТО



За буквите
О писмена
София
2014

Даниела Борисова

ОСНОВИ НА УЕБ ПРОГРАМИРАНЕТО

София, 2014

Daniela Borissova

WEB PROGRAMMING BASICS

Sofia, 2014

Съдържание

Предговор

В този учебник са разгледани основите на уеб програмирането, като накратко са представени най-често използваните езици за програмиране в средата на Интернет. Основната част от изложението разглежда езика HTML – неговите тагове и атрибути. При описанието е направено съпоставяне между актуалната версия на езика HTML 4.01 и предстоящата версия HTML 5. Отделено е внимание на стилизиращите възможности на CSS версии 1, 2 и 3. Показани са част от CSS свойствата, използвани съвместно с HTML. Описани са възможностите за форматиране на текстове, списъци, таблици, цветове, фон, както и възможностите за форматиране в колони. Представени са свойствата за форматиране на контейнер, схемите за позициониране и възможностите за задаване на плаващи елементи. Показани са различни оформления на уеб страници, комбиниращи възможностите на CSS и HTML. В края, накратко е представен най-широко разпространеният език за програмиране в Интернет след HTML – JavaScript. Дадени са основни сведения за синтаксиса, типовете данни и променливи, операторите и функциите. Описани са някои от основните действия (събития), които се извършват върху уеб страницата и които могат да бъдат манипулирани със средствата на JavaScript. Показани са възможностите на обектите Date и Math, както и използването на т.нар. „регулярни изрази“ за проверка на информацията, която се изпраща от формулярите.

Показани са много фрагменти от програмния код, които непосредствено могат да бъдат използвани и тествани.

Учебникът е предназначен за студенти, изучаващи дисциплината „Основи на уеб програмирането“ във факултета „Информационни науки“ – УниБИТ, може да се използва и от студентите, изучаващи дисциплината „Уеб технологии“.

Глава 1. Въведение в уеб програмирането – езици за уеб програмиране, синтаксис на езика HTML

В тази глава са представени в резюме езиците, използвани за програмиране в средата на Интернет. Показана е основната структура на HTML документа и част от синтаксиса на езика HTML. Описани са основните тагове и атрибути, хипервръзки, коментари и символи, както и начините за форматиране на заглавия и параграфи. Дадени са кратки сведения за някои редактори за HTML. Представеният синтаксис е илюстриран с HTML код, а също така и резултата от изпълнението му в браузър. Представени са метаданните, използвани от браузърите, търсещите машини или други уеб услуги.

1.1. Въведение в World Wide Web

Идейната основа за създаването на Интернет е предложена от трима изследователи. Vannevar Bush написва първото въображаемо описание на потенциални приложения за информационните технологии с помощта „memex“ – автоматизирана библиотечна система. Norbert Wiener обособява област на кибернетиката, фокусираща се върху използването на технологиите, разширяващи човешките способности. През 1956 г. на конференция по изкуствен интелект изкристализира концепцията, че технологиите се подобряват експоненциално. Marshall McLuhan предлага идеята за глобалното село като свързана електронна система на част от нашата популярна култура. През 1957 г. Съветският съюз пуска Sputnik – първия направен от човека сателит на земята. САЩ формират Advanced Research Project Agency (ARPA) под покровителството на министерството на отбраната, като целта на създаването на ARPA е била САЩ да поддържа лидерството си в технологиите. Lawrence Roberts ръководи разработването на мрежа, базирана на новата идея за комутация на пакети, предложена от Paul Baran и няколко години по-късно от Donald Davies. Разработен е специален компютър, наречен Interface Message Processor, за реализиране на проекта ARPANET, който води началото си от 1969 г. Първите съобщения са били осъществени

между изследователския център Leonard Kleinrock в Университета на Калифорния в Лос Анджелис и центъра на Douglas Engelbart в Станфордския изследователски институт. На 29.10.1969 г. в Менло парк, Калифорния, ARPA прави демонстрация на комуникация между два компютъра, посредством обмен на пакети. Така се появява първата компютърна мрежа, която впоследствие се разраства и става известна под името ARPANET. Думата Интернет се използва за първи път за описанието на единната глобална компютърна мрежа, която използва протоколния стек TCP/IP. Това става в публикуваната през декември 1974 г. спецификация на протокола TCP, написана от Robert Kahn, Vinton Cerf и др. Впоследствие ARPANET е преустроена да работи на основата на TCP/IP и от 1 януари 1983 г. започва да функционира, използвайки TCP/IP. През следващите години мрежата става известна като Интернет.

Идеята за World Wide Web е предложена през 1989 г. от английския инженер Тим Бърнърс-Лий, който ръководи и нейната първа реализация в мрежата на Европейската организация за ядрени изследвания (CERN) в края на 1990 г. Към края на 1990 г. са разработени основните компоненти на World Wide Web:

- мрежови протокол HTTP
- маркиращ език HTML
- уеб браузър
- уеб сървър
- уеб страници

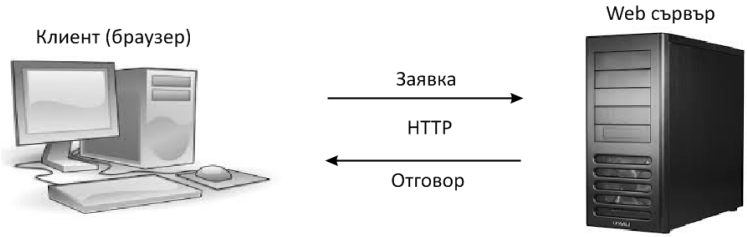
които описват самия проект.

През 1992 г. се появяват първите графични браузъри, които вече са предназначени за по-широко разпространената операционна система UNIX. Година по-късно се появява браузърът Mosaic, който играе важна роля в популяризирането на World Wide Web. През 1993 г. CERN обявява, че системата ще може да бъде използвана свободно и безплатно от всички. През 1994 г. е основана организацията World Wide Web Consortium (W3C), включваща различни заинтересовани организации и имаща за цел утвърждаването на технически стандарти, свързани с функционирането на World Wide Web.

През 1988 г. започва включването на комерсиални компании в световната мрежа. Създадени са първите компании, доставчици на Интернет услуги. Широкото разпространение на TCP/IP позволява бързото разрастване на Интернет. И други компютърни мрежи се

присъединяват към Интернет. На 6 август 1991 г. е публикуван проектът World Wide Web, с което започва и бързото нарастване на популярността на Интернет.

Структурата на Интернет е базирана на технологията клиент – сървър. Това са ключови понятия за разбирането на функционирането на световната мрежа. Принципът, на който действа тази технология, е показан на фиг. 1.1. Клиентът (браузър, FTP клиент или друга програма, работеща на локалния компютър) се свързва към отдалечения компютър (наречен сървър), който доставя услугата, заявена от клиента.



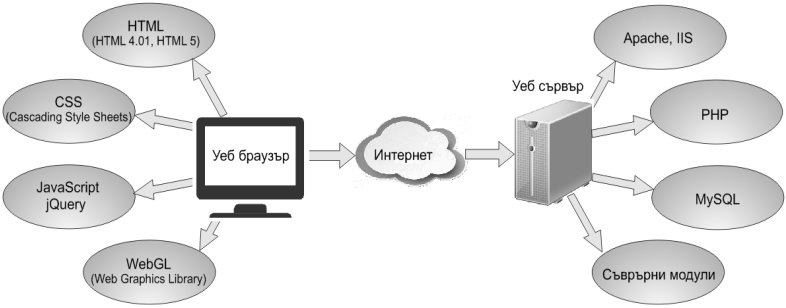
Фиг. 1.1. Клиент – сървър технология

World Wide Web (WWW или W3 или The Web) – наричана също „световна мрежа“ или само Web, е система от взаимно свързани хипертекстови документи, достъпни през компютърната мрежа Интернет. Документите в World Wide Web, наричани уеб страници, могат да съдържат текстове, изображения, видео и други мултимедийни компоненти, а връзките между тях се осъществяват с помощта на хипервръзки.

URL – Web-адрес. Всеки уеб ресурс има уникален уеб адрес или URL (uniform resource locator). Например за: http://www.unibit.bg/, *http* представлява протокола, а *unibit.bg* – домейна. Протоколът **http** (*hypertext transfer protocol*) е протокол за трансфер на хипертекст и за пренос на информация в компютърни мрежи. Създаден като средство за публикуване на HTML страници, протоколът води до формирането на световната уеб мрежа.

Предоставянето на информацията в Интернет пространството се реализира чрез комбиниране на технологии както от страна на сървъра, така и от страна на клиента. „Клиентската страна“ е потребителят, търсещ информация в Интернет посредством браузър,

който от своя страна препраща заявката към сървърната част (компютър), като се реализира взаимодействието им за разглеждане на уеб информация. Тази връзка между двете страни е показана на фиг. 1.2 (всяка елипса представя уеб технология):



Фиг. 1.2. Технологии клиент/сървър

Един от основните инструменти в уеб технологиите е уеб браузърът. Уеб браузърът (в една или друга форма) се използва в различните операционни системи. Изграждането на приложение, което се използва в средата на уеб браузър, дава възможност за лекота както при реализирането, така и при поддръжката. При това основната поддръжка се извършва на сървъра, вместо да се налага индивидуално инсталиране на клиентската страна. Най-общо технологиите (респективно езиците) за уеб програмиране могат да се разделят на:

- технологии/езици за програмиране от страна на клиента: HTML, CSS, JavaScript, VBScript, XHTML, DHTML, WML, AJAX, FLASH, и др.
- технологии/езици за програмиране от страна на сървъра: ASP, PHP, Perl, JSP, ASP.NET, JAVA, MySQL, SQL Server, и др.

1.2. Езици за уеб програмиране

HTML (*HyperText Markup Language*) е метаезик за описание на форматирани документи. Описанието на документите става чрез специални елементи, наречени етикети/маркери (tags). Чрез етикетите се оформят отделните елементи от текста – заглавия, цитати, раздели, хипертекстови препратки и т.н. Основното предимство на

HTML е, че документите, оформени по този начин, могат да се разглеждат на различни устройства – върху монитора на персонален компютър или мобилен телефон.

HTML 5 е последният голям проект на HTML стандарта, който към февруари 2012 г. все още е в процес на разработка. Както неговите предшественици HTML 4.01 и XHTML 1.1, HTML5 е стандарт за създаване и предоставяне на съдържание в уеб пространството. HTML5 добавя нови елементи и атрибути, много нови функции и по-голям набор от технологии, които позволяват по-разнообразни приложения. Семантиката му позволява да се опише по-точно съдържанието, позволява комуникиране със сървъра по иновативен начин, подобрява интегрирането на мултимедийно и графичното съдържание без необходимост от допълнителен софтуер.

DHTML (*Dynamic Hypertext Markup Language*) обединява съвкупност от технологии за създаване на интерактивни и анимирани уеб сайтове. DTTML = HTML+скриптове+CSS. DHTML позволява на скриптовите езици да сменят променливите на езика, описващ уеб страницата, което влияе на изгледа и функционалността на иначе „статичното“ HTML съдържание след зареждане на страницата и в процеса на преглеждането ѝ. Динамичната характеристика на езика DHTML се изразява в начина, по който той функционира по време на използването на страницата, а не в способността да се създава уникално съдържание при всяко зареждане на страницата.

XML (*eXtensible Markup Language*) е стандарт (метаезик), дефиниращ правилата за създаването на специализирани маркиращи езици. Той предоставя необходимата структура и правилата за описване на всякакъв вид информация на даден документ (чрез маркиране с етикети), като отделните маркери (етикети) не са определени предварително, а се въвеждат от програмиста.

XHTML (*Extensible HyperText Markup Language*) – по същество XHTML представлява преформулировка на HTML според правилата на XML. XHTML е HTML дефиниран като XML приложение и е по-строга и по-чиста версия на HTML. XHTML се поддържа от всички основни браузъри. http://www.w3.org/TR/xhtml1/

CSS (*Cascading Style Sheets*) е език за описание на стилове – използва се за описване на визуализирането на документи, написани на език за маркиране. Най-често се използва заедно с HTML, но може да се приложи и върху XML документ. Официално спецификацията на CSS се поддържа от W3C. Създаден първоначално като

средство за разделяне на съдържанието от представянето му, днес той се използва основно за визуално оформление на HTML страници. CSS определя как да изглеждат елементите на една HTML страница – шрифтове, размери, цветове, фонове, и др. CSS кодът се състои от последователност от стилови правила, всяко от които представлява селектор, следван от свойства и стойности. http://www.w3.org/

PHP (*Personal Home Page – Hypertext Preprocessor*) е скриптов език за програмиране, широко използван главно за сървърни приложения и за разработването на динамично съдържание. Автор на езика е датчанинът от канадски произход Размус Лердорф. PHP-кодът се вгражда в HTML страницата и може да се свърже с бази данни за генериране на динамично HTML съдържание. PHP-скриптовете могат да се направят така, че да работят на всяка операционна система с малки изменения или без промяна. http://www.php.net/

ASP (*Active Server Pages*) е технология, разработена от Microsoft. ASP използва някои специални тагове, които позволяват включване на HTML код за генериране на динамични уеб страници. ASP скриптове се изпълняват на сървъра. ASP страниците имат разширение ASP, което ги отличава от HTML страниците, и дава указания на уеб сървъра да премине към интерпретация на ASP страници. За написването на ASP страници може да се използва VBScript, JavaScript/Jscript или комбинация от двете. Голямо предимство при използването на ASP е лекотата на поддръжка на уеб сайта за сметка на зависимостта от Microsoft технологиите. http://www.asp.net/

JavaScript е интерпретиран език за програмиране – слабо типизиран, поддържа обектно ориентирани и функционални програмни стилове. Създаден е от Netscape. Може да бъде вграден в изходния код (HTML) на уеб страница, с цел да се добави допълнителна функционалност, да се ползва за писане на сървърни скриптове, както и за редица други приложения. JavaScript се изпълнява от браузъра на клиентския компютър и не изисква какъвто и да е софтуер от страна на сървъра. Тъй като всички команди на езика се изпълняват от браузъра, JavaScript е отговорен за интерактивността на дадена уеб страница. JavaScript не трябва да се бърка с Java. Скриптовите езици като JavaScript са по-лесни за кодиране в сравнение с езици като Java и C++. JavaScript е разработен първоначално от Брендан Ейч под името Mocha, като по-късно е преименуван на

LiveScript и накрая на JavaScript. LiveScript е официалното име на езика, когато за първи път бива пуснат в бета версиите на Netscape Navigator 2.0, през септември 1995 г., но е преименуван на JavaScript на 4 декември 1995 г.

JScript е скриптов език, базиран на стандарта ECMAScript, който се използва в Microsoft Internet Explorer. JScript се изпълнява от Windows Script Engine. JScript е реализация на Microsoft за JavaScript за Internet Explorer.

VBScript е клиент-базиран език, който работи само в средата на Internet Explorer и е разработена от Microsoft. За предпочитане е да се използва JavaScript или JScript тъй като те се изпълняват от всички популярни браузъри – Opera, Mozilla и Internet Explorer. Въпреки това, VBScript много често се използва за разработване на Active Server Pages.

Java е гъвкав и мощен обектноориентиран език. Много въпроси за зависимостта на софтуерните приложения от платформата, на която се изпълняват, са изгладени с навлизането на Java. По този начин Java-програми за Unix могат да работят на Windows или Mac операционни системи, с малка или никаква преработка. Голямо предизвикателство е наличието на Java Beans, Extended Java Beans и Java applications за различни бази данни и XML. С помощта на Java сървлетите могат да се разработват динамични Java Server Pages (JSP). Основното приложение на Java в Интернет средата е под формата на аплети, вградени в HTML страница, но могат да се създават и приложения, изпълнявани самостоятелно.

Perl (*Practical Extraction and Report Language*) е динамичен език за програмиране с общо предназначение. Perl е разработен от Larry Wall през 1987 г. като скриптов език за Unix с общо предназначение. Perl се използва за графично програмиране, системна администрация, мрежово програмиране и други приложения.

SQL (*Structured Query Language*) е стандартен език за достъп до бази данни, осигуряващ средства за манипулиране и създаване на бази данни като: MySQL, SQL Server, Access, Oracle, Sybase, DB2 и други системи от бази данни.

Python е мощен динамичен език за програмиране, който се използва в широк спектър от области на приложение.

VRML (*Virtual Reality Modeling Language*) е език за описание на триизмерни (3D) обекти и пространства и възможните взаимодействия на потребителя с тях. С негова помощ могат да се създават

виртуални светове, в които потребителят да се движи и да си взаимодейства с обектите по начин, наподобяващ този в реалния свят. http://www.w3.org/MarkUp/VRML/

CGI (*Common Gateway Interface*) е стандарт за интерфейс на външни програми (приложения) с информационните сървъри в Интернет. Тези приложения са известни като CGI скриптове и най-често са написани на скриптови езици, но мгат да се използват и други езици за програмиране (напр. Perl или „C“). CGI скриптовете обикновено се използват за обработка на информация, изпратена като заявка от посетителите чрез формуляр от дадена уеб страница. http://www.w3.org/CGI/

1.3. Синтаксис на езика HTML

HTML е основната технология, контролираща това, което уеб браузърът показва на екрана. Основните категории команди на езика HTML, включени във версиите 3.2 и 4.01, са предназначени за:

- специфициране на стилови формати и управление на текстовия поток (*Flow Control*);
- включване на графични изображения (*Images*);
- създаване на хипервръзки (*Links*);
- интегриране на аудио с външни графични обекти (*Sound and Maps*);
- създаване на интерактивни формуляри (*Forms*);
- разделяне на документа на отделни полета (*Frames*);
- включване на външни приложения (*Applet*), написани на езика Java;
- осъществяване на връзка с външни информационни структури (*CGI-script*).

Версии на езика HTML:

1991 – HTML – първа версия, създадена от Tim Berners-Lee
1993 – HTML 2 проект
1995 – HTML 2 – W3C
1995 – HTML 3 проект
1997 – HTML 3.2
1997 – HTML 4
1999 – HTML 4.01 (окончателен)

2000 – XHTML проект
2001 – XHTML (окончателен)
2008 – HTML5/XHTML5 проект
2011 – HTML5 уточнявяне на характеристики
2022 – HTML5 окончателни спецификации.

По своята същност един HTML документ е обикновен текстови файл, съдържащ специални кодове, наречени „тагове“ (етикети), които се поставят около блокове от текст. Описанието на документа става чрез специални елементи, наречени HTML елементи, които се състоят от тагове и ъглови скоби (като например елемента `<html>`). HTML елементите са основната градивна единица на уеб страниците. Чрез тях се оформят отделните части от текста на една уеб страница, като заглавия, цитати, раздели, хипертекстови препратки и т.н. Най-често HTML таговете на елементите са групирани по двойки `<h1>` и `</h1>` – отварящ и затварящ таг, като отварящият таг се поставя преди съдържанието, за което се отнася, а затварящия таг се поставя непосредствено след това съдържание.

HTML елемент е всичко от отварящия до затварящия таг. HTML елементите могат да имат атрибути. Атрибутите в HTML предоставят допълнителна информация за HTML елементите. Атрибутите се поставят само в отварящия таг. В Приложение 1 е даден списък на глобалните атрибути, които могат да бъдат използвани.

1.4. Структура на HTML документ

Основната структура на HTML документа включва три задължителни елемента `<html>`, `<head>` и `<body>`, както е показано на фиг. 1.3. Документът започва с деклариране на типа на документа `<!DOCTYPE html>.` Този елемент указва на браузъра какъв набор от стандарти е използван в документа. Чрез `<!DOCTYPE html>` (няма затварящ таг) може да се валидира софтуера, като предварително се зададе версията на използвания HTML код. След него се намира отварящият таг `<html>`, който формира целия HTML документ. Затварящият таг `</html>` е последното нещо, което съдържа всеки HTML документ.

Елементът `<html>` може да съдържа други елементи, например заглавна част на документа, формирана чрез двойката тагове `<head>` и `</head>`.

Фиг. 1.3. Структура на HTML документ

В нея се разполага информация за документа (метаданни), която не се визуализира в браузъра. Там се задава и заглавието на самия документ, посредством двойката тагове `<title>` и `</title>`. Елементът `<body>` формира цялото визуално съдържание на HTML документа.

В резюме може да обобщим, че основната структура на един HTML документ включва следните основни елементи за:

- деклариране на типа на документа `<!DOCTYPE html>`
- дефиниране на началото и края на уеб документа `<html>`
- заглавна част на документа `<head>`
- заглавие на документа `<title>`
- определяне на тялото на документа `<body>`.

Тези елементи често съдържат и други елементи. В тялото на документа неизменно участват много вложени елементи, като заглавия, параграфи, списъци, таблици и др.

Декларацията *doctype* за HTML5 има вида:

```
<!DOCTYPE html>
```

Декларацията *doctype* за HTML 4.01 има вида:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
```

Декларацията *doctype* за XHTML 1.0 има вида:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

Двойката тагове `<head>` и `</head>` определя границите на областта, където се съдържа информация, която не се показва директно в основния прозорец на браузъра. Елементът `<head>` може да се разглежда като контейнер за всички елементи в заглавната част на документа. Тук може да се добавя информация на скриптове, инструкции за браузъра, къде да намери стилове, предоставяне на мета информация и др. Тази допълнителна информация се добавя като се използват следните елементи: `<title>`, `<style>`, `<meta>`, `<link>`, `<script>`, `<noscript>` и `<base>`.

Тагът `<title>` определя заглавието на документа и е задължителен във всички HTML/XHTML документи. Това заглавие се появява, когато страницата се добави към *bookmarks*, а също така се показва и при резултат от търсене.

Тагът `<base>` определя основното местонахождение на всички относителни URL адреси в страницата:

```
<head>
   <base href="http://www.unibit.bg/images/" target="_blank">
</head>
```

Тагът `<link>` определя връзка към външен ресурс:

```
<head>
   <link rel="stylesheet" type="text/css" href="mystyle.css">
</head>
```

Тагът `<style>` се използва за дефиниране на стил за текущия HTML документ. В `<style>` тага може да се укаже как HTML елементите трябва да се визуализират в браузъра:

```
<head>
   <style type="text/css">
       body {background-color:yellow}
       p {color:blue}
   </style>
</head>
```

Тагът `<script>` се използва за определяне на скрипт от страна на клиента, като например JavaScript. Елементът `<script>` ще бъде обяснен на по-късен етап.

Тагът `<meta>` съдържа метаданни за HTML документа. Тази информация не се показва на страницата, но е анализируема от търсещите машини. Елементите `<meta>` обикновено се използват за описание на ключови думи, автор на документа, последна промяна и други метаданни. Най-общо, метатаговете могат да се разделят на две групи:

- метатагове, съдържащи атрибута *http-equiv* и атрибута *content*;
- метатагове с атрибута *name* и атрибута *content*.

Синтаксисът на тези два типа метатагове е следният:

```
<meta http-equiv="име" content="съдържание" />
<meta name="име" content="съдържание" />
```

Метатаговете *http-equiv* се използват, за да управляват определени действия на браузера, а метатаговете с атрибута *name* и атрибута *content* се използват, за да подават определена информация на търсещите машини. При писане на кирилица е по-добре да се използва не *iso-8859-5*, а кодовият стандарт за работа под Microsoft Windows – *windows-1251*:

```
<meta http-equiv="content-type" content="text/html;
charset=windows-1251" />
```

Ако се налага да се създаде документ на руски език, за стойност на *charset* е най-добре да се ползва кодът *KOI8-R*:

```
<meta http-equiv="content-type" content="text/html;
charset=KOI8-R" />
```

Някои често използвани кодови таблица са:

- *windows-1250* за централноевропейските езици – чешки, полски, унгарски, румънски, хърватски, словашки, словенски, албански и немски (подобен на iso-8859-2);
- *windows-1251* за езиците на кирилица – български, руски, белоруски, македонски, сръбски, украински (подобен на iso-8859-5);
- *windows-1252* за западноевропейските езици (подобен на iso-8859-1);
- *windows-1253* за гръцки (подобен на iso-8859-7);
- *windows-1254* за турски (подобен на iso-8859-9);
- *windows-1255* за еврейски (подобен на iso-8859-8);
- *windows-1256* за арабски (подобен на iso-8859-6);

- *windows-1257* за балтийските езици (подобен на iso-8859-4);
- *windows-1258* за виетнамски.

Пример за използване на метаданни – обновяване на документа на всеки 30 секунди:

```
<meta http-equiv="refresh" content="30">
```

За кратко описание на съдържанието на документа може да се използва метатагът с атрибутите `name` и `content`, като препоръчително е дължината на описанието да не надхвърля 150 символа.

```
<meta name="description" content="Кратко описание на страницата." />
```

Метатаг за ключови думи се използва за въвеждане на ключови думи, които имат отношение към съдържанието на документа и го описват най-добре. Дължината на ключовите думи не трябва да е повече от 500 символа.

```
<meta name="keywords" content="Ключови, думи, отделени, със, запетаи"
/>
```

Метатаг, указващ автора на документа:

```
<meta name= "author"content= "Име, Фамилия" />
```

Метатаг, който най-общо класифицира съдържанието на страницата.

```
<meta name="classification" content="business" />.
```

Други стойности на атрибута `content` са: *art, internet, education, entertainment, government, science, news, sport* и др.

Метатаг, който дава указание на търсачките каква част от сайта да се индексира. Например, индексиране на страницата като се следват връзките в нея:

```
<meta name="robots" content="index, follow" />
```

Възможни стойности на атрибута `content` са:

- `content="all"` – индексира страницата и всички нейни връзки;
- `content="none"` – без индексиране на страницата и връзките ѝ – при страници в процес на изграждане;
- `content="index, follow"` – указва индексирането на страницата и следването на връзките ѝ (отговаря на `content="all"`);
- `content="index, nofollow"` – указва индексирането на

страницата без да се следват връзките, намиращи се в нея;

- `content="noindex, follow"` – само проследява връзките от страницата без да я индексира;
- `content="noindex, nofollow"` – не се индексира страницата и не се следват връзките (отговаря на `content="none"`).

Атрибути на елемента *body*

Ако не се зададат атрибути на елемента `body`, се използват стойности по подразбиране – бял фон на страницата, черен цвят на текста, син цвят на връзките.

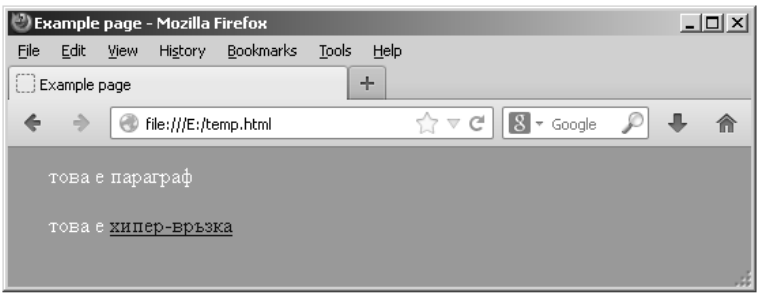
Атрибутът `bgcolor` задава фоновия цвят на страницата: `<body bgcolor="black">`. Освен цвят за фон, може да се използва фоново изображение чрез атрибута `background`: `<body background="picture.jpg">`. В този случай изображението ще изпълни целия прозорец на браузъра с многократно повтарящи се свои копия по хоризонтала и вертикала. Атрибутът `text` задава цвят на текста в страницата: `<body text="red">`.

Атрибутът `link` указва какъв да е цветът на текстовите хипервръзки в HTML документа: `<body link="green">`. Атрибутът `vlink` указва как да се оцветят вече посетените връзки: `<body vlink="silver">`.

С атрибутите `topmargin`, `leftmargin`, `rightmargin` и `bottommargin` се задават съответните разстояния между съдържанието на уеб страницата и рамката на браузъра в пиксели, както е показано:

```
<body bgcolor="#999999" text="white" link="blue" alink="yellow"
vlink="green" topmargin="10" leftmargin="30" rightmargin="10"
bottommargin="0">
<p>това е параграф <br>
това е <a href="#">хипер-връзка</a></p>
```

В браузър, кодът ще се визуализира по следния начин:

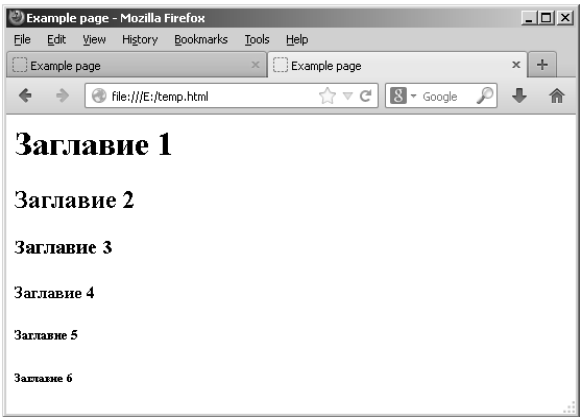


1.5. Заглавия, параграфи, хоризонтални линии и коментари

Заглавията в HTML се определят с таговете `<h1>` до `<h6>`, като `<h1>` определя най-голямото заглавие, а `<h6>` определя най-малкото заглавие. Например:

```
<h1>Заглавие 1</h1>
<h2>Заглавие 2</h2>
<h3>Заглавие 3</h3>
<h4>Заглавие 4</h4>
<h5>Заглавие 5</h5>
<h6>Заглавие 6</h6>
```

В браузър, кодът ще се визуализира по следния начин:



Браузърите автоматично добавят празно пространство преди и след всяко заглавие. Използваме таговете *h1, h2, ..., h6* само за означаване на заглавия, тъй като търсещите машини използват заглавията за индексиране на структурата и съдържанието на уеб страниците.

Параграфите представляват части от текста, отделени една от друга с празен ред и без отстъп – дефинират се чрез тага `<p>`:

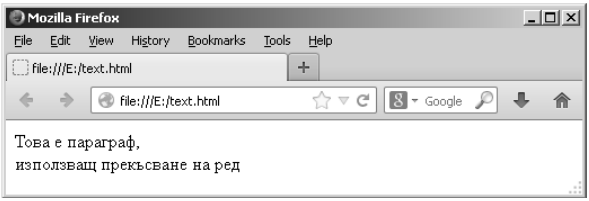
```
<p>Това е параграф</p>
<p>Това е следващ параграф</p>
```

Браузърите автоматично добавят празен ред преди и след параграфите. За прекъсването на текущия ред на текста и преминаването му на следващ ред, без започване на нов параграф се използва

таг <br>. Този таг
 няма затварящ таг. Може да се използва за оставяне на един, два и т.н. празни реда, като се изпишат толкова на брой тагове
 колкото празни редове трябва да се оставят.

```
<p>Това е параграф,<br>използващ прекъсване на ред</p>
```

В браузър, кодът ще се визуализира по следния начин:

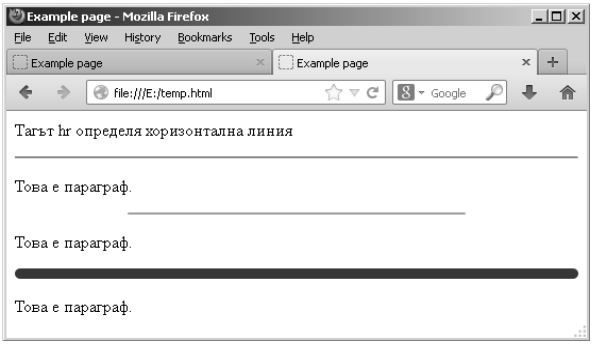


Тагът
 няма затварящ таг според стандарта HTML 4.01, но според стандарта XHTML този таг се изписва така <br />. Чрез таговете <nobr> </nobr> не се позволява разкъсването на текста и пренасянето на думи на по-долен ред, т.е. текстът се показва на един ред.

Тагът <hr> се използва за създаване на хоризонтална линия в HTML документ. Няма затварящ таг и може да използва атрибути като *size, width, align (left, right, center), color*. Елементът <hr> може да се използва за разделяне на съдържанието, както е показано:

```
<p> Тагът hr определя хоризонтална линия</p>
<hr> <p> Това е параграф.</p>
<hr align="center" width="60%" color="#33CC00">
<p> Това е параграф.</p>
<hr size="10" color="red"> <p> Това е параграф.</p>
```

В браузър, кодът ще се визуализира по следния начин:



Коментари се вмъкват в HTML документа, за да се направи по-разбираем кодът. Коментарите се игнорират от браузъра и не се показват. Коментарите се задават по следния начин:

```
<!-- този коментар не се показва  -->
<p>Това е параграф с <!-- коментар --> текст </p>
```

1.6. Хипервръзки

Една хипервръзка (или линк) може да бъде дума, група от думи или изображение, което може да се активира с бутон на мишката, за да се визуализира друг документ. Важно е да се отбележи, че хипервръзките винаги започват с протокола за предоставяне на услугата, например http://www.unibit.bg. За създаване на хипервръзки се използва тагът `<a>`. Самият таг `<a>` без атрибутите си не изпълнява почти никакви самостоятелни функции. Най-често използваният атрибут е *href*, съкращение от *hypertext reference* и указва на браузъра къде да намери информацията. Синтаксисът е:

```
<a href="URL">текст на хипервръзката</a>
```

По подразбиране всички хипервръзки се отварят в текущия прозорец на браузъра. Хипервръзките могат да се разделят на абсолютни и относителни. Абсолютните връзки включват цялото име на пътя до ресурса:

```
<a href="http://www.unibit.bg">Връзка към unibit.bg</a>
```

Относителните хипервръзки не включват цялото име на пътя към страницата, към която сочат. Вместо това, името на пътя, което се използва, се отнася към настоящата страница.

```
<a href="news.html">новини</a>
<a href="../students-b.html">студенти-бакалаври</a>
<a href="../students-m.html">студенти-магистри</a>
```

Хипервръзка към определено място вътре в самия документ може да бъде реализирана чрез използването на атрибута *name*. За да направим връзка към определено място в страницата, първо трябва да зададем неговото име. Създаването на такова име позволява да бъде направена хипервръзката към него по-късно. Името

или т.нар. „котва“ се създава с помощта на тага `<a>`, използващ атрибута *name*, както е показано:

```
<a name="paragraph-3"></a>.
```

Както се вижда, между отварящия и затварящия таг на котвата няма текст, който да се визуализира в средата на браузъра. Хипервръзката, която води към направената вече котва, има следния вид:

```
<a href="#paragraf-3">Връзка към параграф 3 на страницата</a>.
```

Знакът # указва на браузъра, че правим връзка към определено място на същата страница. Може да направим и хипервръзка, която да сочи към точно определено място на друга страница, например към петия параграф (*paragraph-5*) от другата страница:

```
<a href="students-b.html#paragraf-5">
Към параграф 5 на страницата students-b.html</a>.
```

В този случай браузърът първо ще търси страницата *students-b.html* и след това ще намери котвата, носеща името *paragraph-5*.

Чрез атрибута *target* указваме, къде хипервръзката да отвори свързания документ. Пример на хипервръзка, която се отваря в нов прозорец на браузъра:

```
<a href=http://www.unibit.bg target="_blank">
Връзка към unibit, която ще се отвори в нов прозорец</a>
```

Други възможни стойности на *target* са:

- `_self` – отваря хипервръзка в текущия прозорец;
- `_blank` – отваря хипервръзка в нов прозорец;
- `_parent` – отваря хипервръзка в родителския прозорец;
- `_top` – отваря хипервръзката в целия прозорец;
- `framename` – отваря хипервръзка в рамка със зададено име.

Протоколът `mailto` позволява да бъде създадена хипервръзка, която да стартира подразбиращият имейл софтуер. Активирането на такава хипервръзка ще отвори потребителския е-мейл за клиент със зададения имейл адрес. Синтаксисът е:

```
<a href="mailto:name@mail.com">Link text</a>
```

Възможните параметри и тяхното описание за активиране на имейл клиент за изпращане на съобщение са както следва:

параметър	описание
mailto:name@email.com	e-mail recipient address
cc=name@email.com	carbon copy e-mail address
bcc=name@email.com	blind carbon copy e-mail address
subject=subject text	subject of e-mail
body=body text	body of e-mail
?	first parameter delimiter
&	other parameters delimiter

Ето един пример за изпращане на e-mail до много получатели:

```
<a href="mailto:first@email.address,second@email.address,
third@email.address"> за контакти </a>
```

Пример за e-mail с добавен текст в полето *subject*:

```
<a href="mailto:your@email.address?subject=Comments about the color
blue"> за контакти </a>
```

Пример за e-mail, използващ полето *cc*:

```
<a href="mailto:first@email.address?cc=second@email.address,
third@email.address">за контакти</a>
```

В Приложение 2 е даден списък на използваните тагове в HTML 4.01 и HTML 5 и тяхното описание.

1.7. Символи

В HTML се използва специфичен начин за кодиране на определени символи като например © и ™. Този вид кодиране е удобен начин за въвеждане на символи, които се намират трудно или липсват на клавиатурата. За представяне на HTML документа се използват символите <, >, " и &, които са част от синтаксиса на комадните в HTML. Ако е необходимо да се визуализира на екрана знака „<“, в HTML документа трябва да се използват специални символи *<*. Всички символи в HTML започват със символа *&* като начало и завършват с точка и запетая (;) за край. Специалните символи и кодовете – числови и буквените им съответствия, са дадени в Приложение 3.

Кодирането на специалните символи намира приложение и при защитата от нежелани имейл съобщения (спам). Ако в уеб страницата публикуваме имейл адрес, можем да бъдем сигурни, че в най-скоро време ще завалят нежелани спам писма. Спамърите използват програми, които обикалят уеб страниците и събират имейл адреси. Тези програми обаче не „виждат“ това, което се показва на

екрана, а четат HTML кода на страниците. Така че ако замените някоя от буквите в имейл адреса си (нека адресът да е *ivan@ivanov* със съответния HTML код (напр. заменяме „i“ с неговия код i), то посетителите на страницата няма да забележат разликата с некодирания вариант, но за спамърските роботи адресът ще изглежда така: *ivan@ivanov*

Създаване и използване на *favicon* – малката икона, която се вижда до адреса в браузъра. Най-лесният начин да си направим такава икона е следния: 1) избираме изображение, което искаме да се показва до URL-адреса; 2) чрез подходаща програма намаляваме изображението до размери 16x16 или 20x20 пиксела и това го записваме във формат за икони (с разширение *.ico*). Така полученото изображение може да се добави към уеб страницата в секцията *head*, следвайки показания синтаксис:

```
<LINK REL="SHORTCUT ICON" HREF="http://www.domain.com/favicon.ico">
```

За писането и редактирането на HTML код могат да се използват 2 типа редактори:

- Просто редактиране – въвеждане на HTML кода ръчно чрез: Notepad, WorldPad, TextPad, и др.
- WYSIWYG (What You See Is What You Get) редактори. Те създават HTML код, имат възможност както за ръчно редактиране на кода, така и за визуализиране на написания код, като Macromedia Dreamweaver, Microsoft FrontPage, Adobe GoLive, AceHTML, CoffeeCup и др.

Могат да се използват и налични онлайн редактори (напр. http://www.tutorialspoint.com/html/html_editor.htm)

Глава 2. Форматиране на текст и списъци

В тази глава са описани възможностите за форматиране на текст в HTML документи – размер, тип и стил на шрифта, подравняване, разстояние между буквите, разстояние между редовете, цвят и др. По подразбиране текстовете в HTML документите се визуализират в шрифт Times New Roman с размер 12 pt, подравнени в ляво, с единично разстояние между редовете. В тази глава е показано как да персонализираме настройките на текста, вкл. шрифтовете, цветовете, отместването на първия ред от параграф, горните, долните индекси, форматирането на текст в списъци и др.

2.1. Форматиране на текст

За задаване на общ шрифт на текста в целия документ може да се използва тага `<basefont>`, който се поставя в началото след секцията `<body>` и има следния синтаксис:

```
<body>
<basefont face="arial, verdana, courier" size="4" color="green">
Съдържанието на документа е с предварително зададен шрифт – arial,
verdana или courier, размер 4 и в зелен цвят, както са зададени атри-
бутите на тага &lt;basefont&gt;.
</body>
```

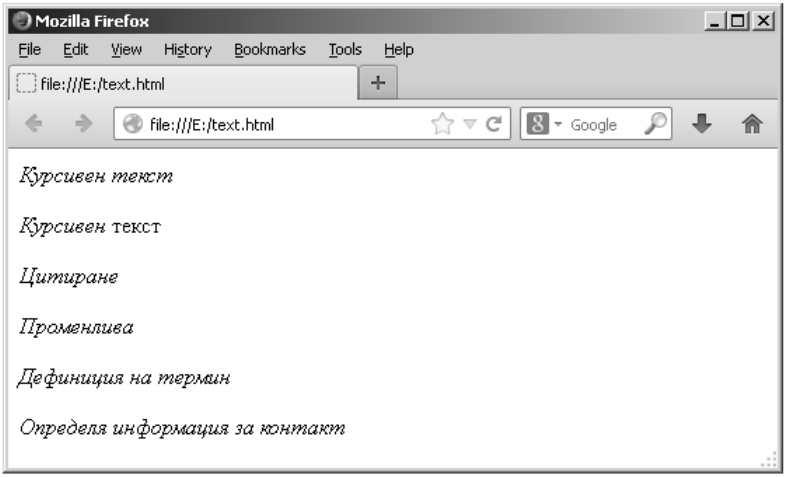
За съжаление, този таг не се поддържа от HTML5 и е отхвърлен в HTML 4.01.

По-долу ще бъдат разгледани възможностите на HTML по отношение на форматирането на части от текста в курсив (*italic*), получерен (*bold*) и подчертан (*underline*).

За постигане на курсивен текст се използва двойката тагове `<i>` и `</i>`. Същият ефект може да се постигне и чрез таговете `<em>` `<cite>`, `<var>`, `<dfn>`, `<address>`:

```
<p><i> Курсивен текст </i></p>
<p><em> Курсивен </em> текст </p>
<p><cite> Цитиране </cite></p>
<p><var> Променлива</var></p>
<p><dfn> Дефиниция на термин </dfn> </p>
<p><address> Определя информация за контакт </address> </p>
```

В браузър кодовете ще се визуализират по следния начин:



Задаването на получерен текст се постига чрез използване на двойката тагове и . Същият ефект се постига и чрез таговете и .

```
<p><b>получерен текст </b> чрез тага b</p>
<p><strong>получерен текст </strong> чрез тага strong </p>
```

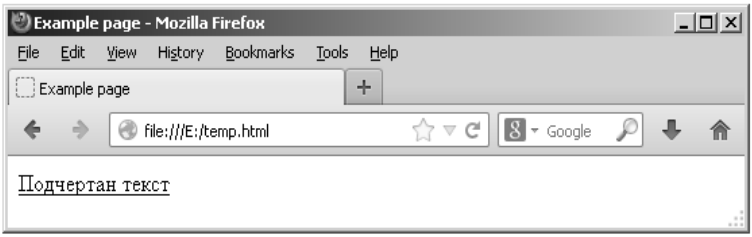
В браузър кодовете ще се визуализират по следния начин:

И двата тага и се поддържат от почти всички браузъри.

За подчертаване на текста се използва друга двойка тагове <u> и </u>:

```
<p><u> Подчертан текст </u></p>
```

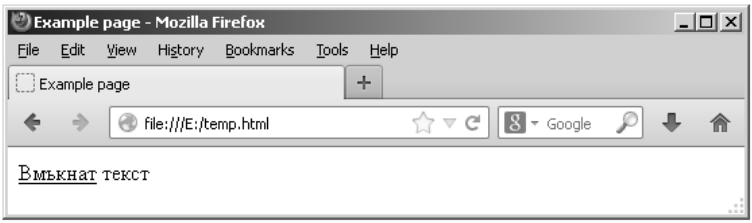
В браузър кодът ще се визуализира по следния начин:



Подобно на тага за подчертаване `<u>`, може да се използва двойката тагове `<ins>` и `</ins>` за маркиране на вмъкнат текст:

```
<p><ins>Вмъкнат</ins> текст</p>
```

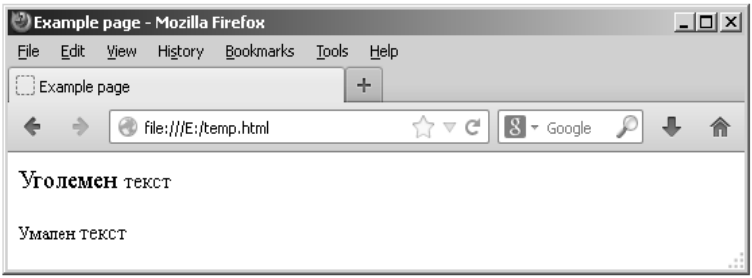
Браузърът ще визуализира вмъкнатия текст като подчертан, както е показано:



Може да увеличим част от даден текст чрез двойката тагове `<big>` и `</big>`. Тагът `<big>` не се поддържа от HTML5. Ако оградим дадена фраза с таговете `<small>` и `</small>`, тя ще бъде показана с по-малък размер от останалия текст:

```
<p><big> Уголемен </big> текст</p>
<p><small> Умален </small> текст</p>
```

В браузър кодовете ще се визуализират по следния начин:



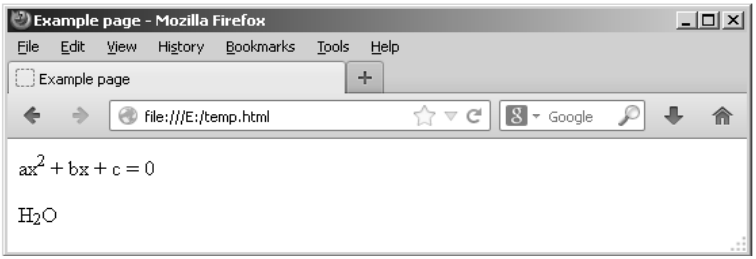
За форматирането на горните индекси се използва двойката тагове `<sup>` и `</sup>`. Например, за да покажем в браузъра едно квадратно уравнение, неговият запис трябва да бъде представен като:

```
<p>ax<sup>2</sup> + bx + c = 0</p>
```

Форматирането на долните индекси се постига посредством таговете `<sub>` и `</sub>`. Например, за да покажем химичната формула на водата, ще използваме следното представяне:

```
<p>H<sub>2</sub>O</p>
```

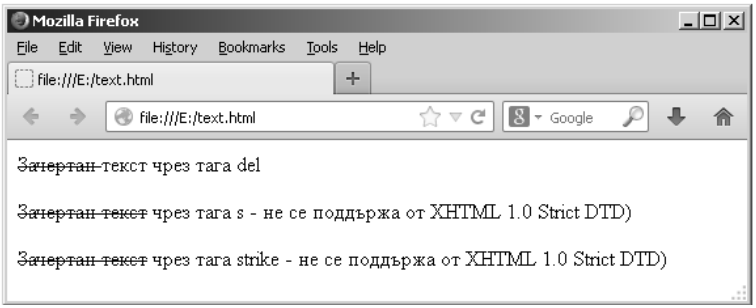
В браузър кодовете ще се визуализират по следния начин:



За да покажем текста като зачертан, т.е. с черта през средата му, могат да се използват няколко двойки тагове като `<del>`, `<s>` и `<strike>`:

```
<p><del>Зачертан </del> текст чрез тага del</p>
<p><s>Зачертан текст</s> чрез тага s – не се поддържа от XHTML 1.0
Strict DTD) </p>
<p><strike>Зачертан текст</strike> чрез тага strike – не се поддържа
от XHTML 1.0 Strict DTD) </p>
```

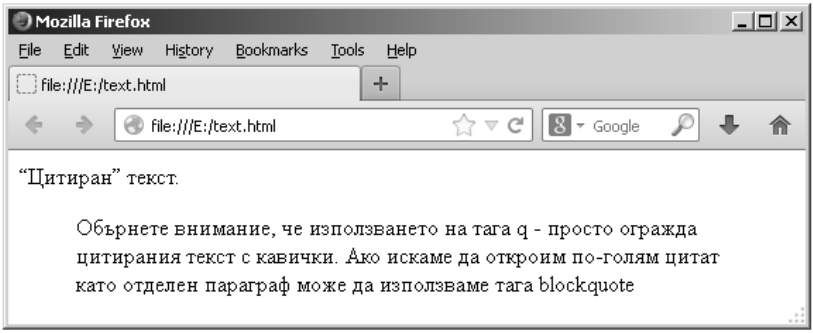
В браузър кодът ще се визуализира по следния начин:



За кратки цитати може да се използват таговете `<q>` и `</q>`, а при необходимост от форматиране на по-голям цитат се използва двойката тагове `<blockquote>` и `</blockquote>`:

```
<p><q>Цитиран</q> текст.</p>
<blockquote>Обърнете внимание, че използването на тага q – просто ог-
ражда цитирания текст с кавички. Ако искаме да откроим по-голям цитат
като отделен параграф може да използваме тага blockquote </blockquote>
```

В браузър кодът ще се визуализира по следния начин:

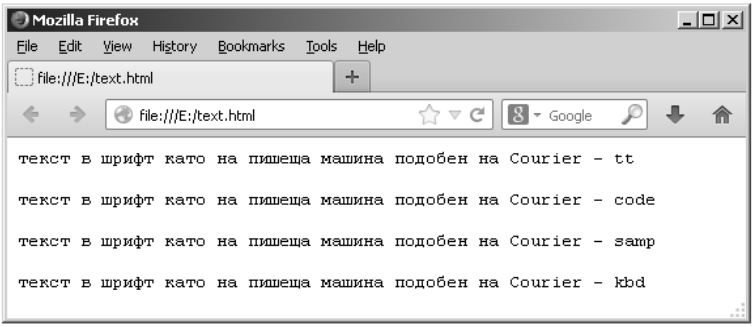


Трябва да обърнем внимание, че използването на тага `<q>` просто ограждa цитирания текст с кавички, докато `<blockquote>` отмества цитата от всички страни.

Други тагове използвани за форматиране на цитиране, при което текстът се показва в шрифт като на пишеща машина, подобен на `Courier`, са: `<tt>`, `<code>`, `<samp>`, `<kbd>`:

```
<p><tt>текст в шрифт като на пишеща машина подобен на Courier –
tt</tt></p>
<p><code>текст в шрифт като на пишеща машина подобен на Courier –
code</code></p>
<p><samp>текст в шрифт като на пишеща машина подобен на Courier –
samp</samp></p>
<p><kbd>текст в шрифт като на пишеща машина подобен на Courier –
kbd</kbd></p>
```

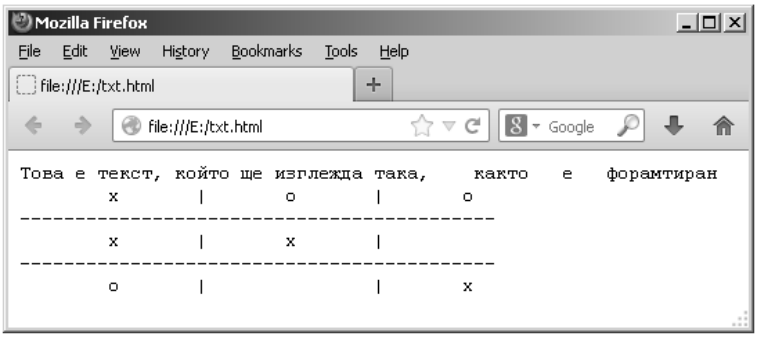
В браузър кодовете ще се визуализират по следния начин:



Тагът `<pre>` показва текста в браузъра по начина, по който е записан в кода. Защо тогава не използваме тага `<pre>` за всичко? Поради две причини: 1) тагът `<pre>` изобразява текста в шрифт с фиксирана широчина на символите, като напр. Courier, което изглежда като написано на пишеща машина и е подходящ за примери на програмен код; 2) няма гаранции, че страницата ще изглежда по начина, по който се вижда. Браузърите могат да интерпретират табулацията като повече или по-малко брой интервали, отколкото текстовия редактор.

```
<pre>Това е текст, който ще изглежда така,    както   е   форамтиран
        x   |   o   |   o
       -----------------------------------------
        x   |   x   |
       -----------------------------------------
        o   |       |   x
</pre>
```

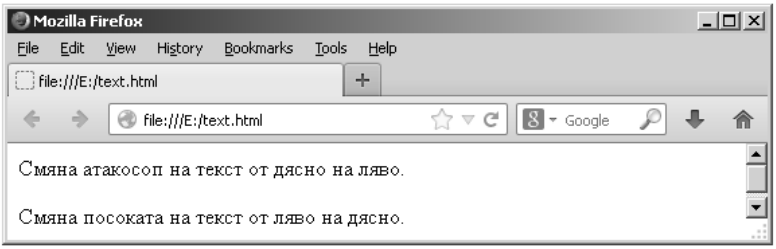
В браузър кодът ще се визуализира по следния начин:



Посоката на текста или на част от текста може да бъде сменена (думите да се изписват от ляво на дясно) като се използва атрибутът *dir* на тага <bdo> със стойност *rtl.* Текстът, ограден между отварящия и затварящия таг, ще се покаже огледално обърнат. Друга възможна стойност на атрибута *dir* е *ltr*, която показва текста от дясно на ляво:

```
<p>Смяна <bdo dir="rtl">посоката</bdo> на текст от дясно на ляво.</p>
<p>Смяна <bdo dir="ltr">посоката</bdo> на текст от ляво на дясно.</p>
```

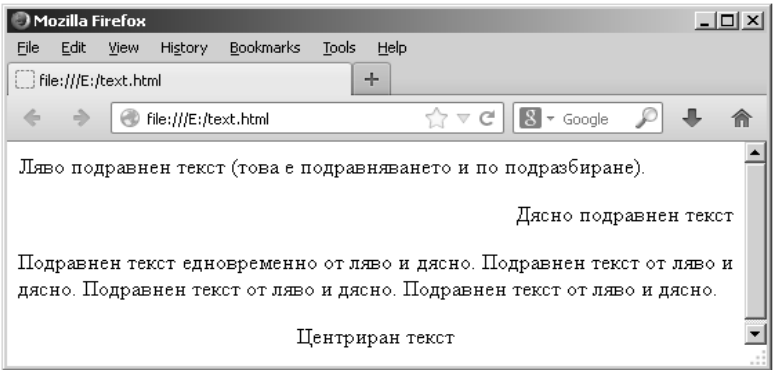
В браузър кодът ще се визуализира по следния начин:



Използването на атрибута *align* позволява да зададем желано подравняване на текста чрез стойностите *left*, *right, center* или *justify*:

```
<p align="left">Ляво подравнен текст (това е подравняването и по под-
разбиране).</p>
<p align="right">Дясно подравнен текст</p>
<p align="justify">Подравнен текст едновременно от ляво и дясно. Под-
равнен текст от ляво и дясно.  Подравнен текст от ляво и дясно.</p>
<p align="center">Центриран текст</p>
```

В браузър кодът ще се визуализира по следния начин:



За форматиране на текст с точно определен шрифт се използват таговете `<font>` и `</font>`, към които се добавят атрибути и стойности като:

- `face`="стойност" указва типа на шрифта. Стойността може да бъде както названието на един шрифт, така и на няколко шрифта (отделени със запетая). Използването на няколко шрифта е полезно, тъй като ако първият от тях отсъства на локалния компютър, следващият записан подред шрифт ще бъде използван за форматирането на текста.
- `size`="стойност" указва размера на текста. Стойността може да бъде зададена по 2 начина: чрез абсолютен размер, като се задава стойност от 1 (най-малкото) до 7 (най-голямото) или чрез относителен размер – задава се стойност -1, +1, -2, +2 и т.н. При стойност -1 размерът на шрифта ще бъде с единица по-малък от стандартния, а при +1 ще бъде с един размер по-голям от стандартния. Числата не съответстват на някакъв специфичен размер в точки или пиксели и при едни и същи настройки размерът на текста може да варира в зависимост от компютъра. (Каскадните набори от стилове (CSS) дават много по-прецизен контрол над атрибутите на шрифта).
- `color`="стойност" указва цвета на текста. Стойността може да бъде както името на цвета, така и RGB стойността.

```
<p>
<font color="#ff0000" face="arial" size="+2">
Това е текст с фонт arial, size 2 и червен.</font>
</p>
```

В браузър кодът ще се визуализира по следния начин:

Използването на атрибута *style* може да задава различни свойства на текста. Например, комбинацията на атрибута *style* с *text-transform* прави възможно текстът да бъде визуализиран само с малки, само с големи букви или всяка дума да започва с главна бук-

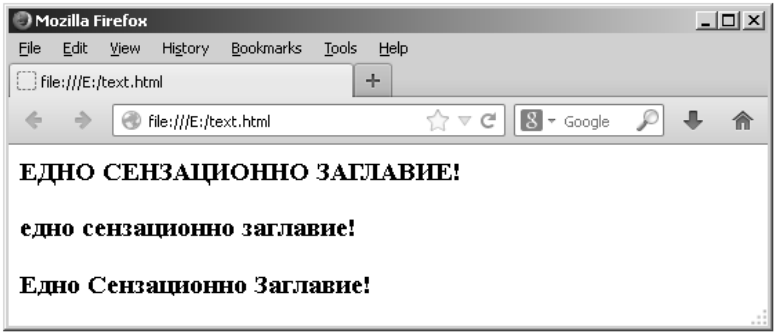
ва, независимо от това как е бил въведен текстът в неговия оригинален код:

- `text-transform: lowercase` форматира всички букви като малки;
- `text-transform: uppercase` форматира всички букви като главни;
- `text-transform: capitalize` форматира началната буква на всяка дума като главна.

Ето един пример за преобразуване от малки в главни букви и обратното, както и за преобразуване на всяка от думите с главна буква:

```
<h3 style="text-transform: uppercase">
едно сензационно заглавие!</h3>
<h3 style="text-transform: lowercase">
ЕДНО СЕНЗАЦИОННО ЗАГЛАВИЕ!</h3>
<h3 style="text-transform: capitalize">
едно сензационно заглавие!</h3>
```

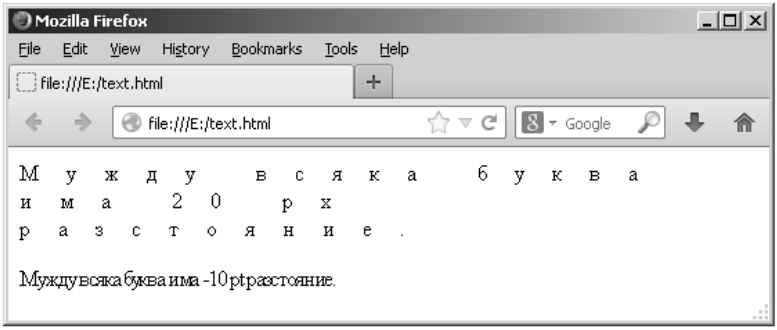
В браузър кодът ще се визуализира по следния начин:



За определяне на точно фиксирано разстояние между буквите използваме атрибута `style` и `letter-spacing`. Възможните стойности са *px, pt, cm*, като са допустими и отрицателни стойности:

```
<p style="letter-spacing:20px">
Мужду всяка буква има 20 px разстояние.</p>
<p style="letter-spacing:-1pt">
Мужду всяка буква има -10 pt разстояние.</p>
```

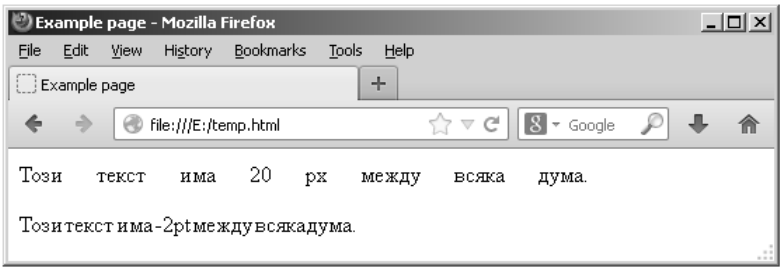
В браузър кодът ще се визуализира по следния начин:



За определяне на разстоянието между думите се използва атрибутът `style` със стойност `word-spasing`. Разстоянието може да бъде зададено в *px, pt, cm, em*, и др., като са позволени и отрицателни стойности:

```
<p style="word-spacing: 20px">
Този текст има 20 px между всяка дума. </p>
<p style="word-spacing: -1pt">
Този текст има -2pt между всяка дума. </p>
```

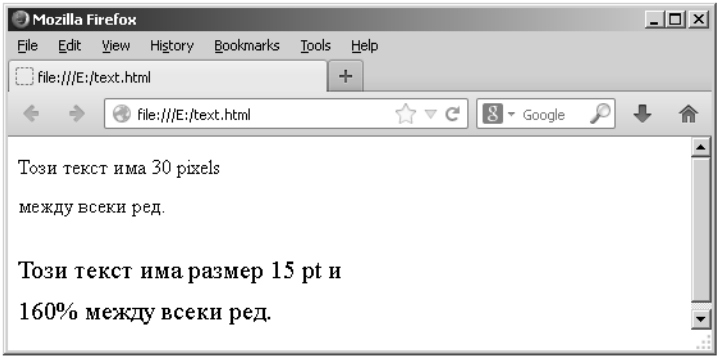
В браузър кодът ще се визуализира по следния начин:



За определяне на разстоянието между редовете се използва атрибутът `style` със стойност `line-height`. Възможните стойности са: *normal* (по подразбиране), *px, pt, cm* или проценти:

```
<p style="line-height:30px">
Този текст има 30 pixels<br> между всеки ред. </p>
<p style="line-height: 160%; font-size: 15pt">
Този текст има размер 15 pt и <br>160% между всеки ред.</p>
```

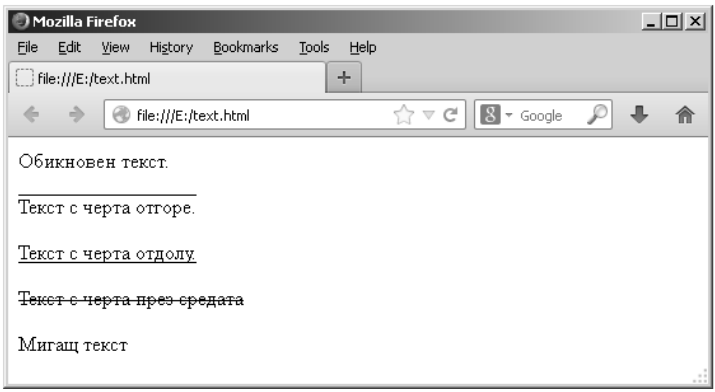
В браузър кодът ще се визуализира по следния начин:



За декориране на текста може да се използва атрибутът `style` със стойност `text-decoration`. Възможните стойности са: `none` (по подразбиране за обикновен текст), `overline`, `underline` (по подразбиране за всички хипервръзки), `line-through` или `blink`:

```
<p style="text-decoration:none">
Обикновен текст.</p>
<p style="text-decoration:overline">
Текст с черта отгоре. </p>
<p style="text-decoration:underline">
Текст с черта отдолу. </p>
<p style="text-decoration:line-through">
Текст с черта през средата</p>
<p style="text-decoration:blink">
Мигащ текст </p>
```

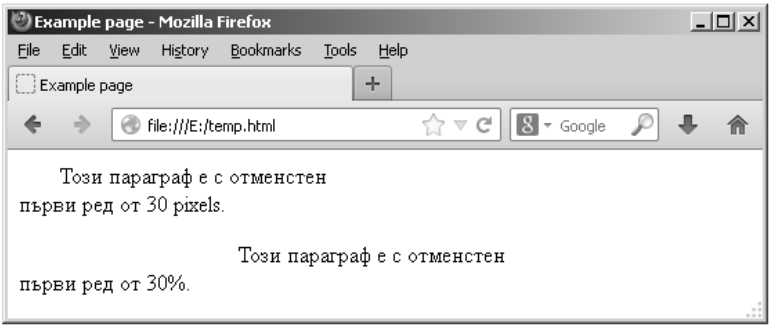
В браузър кодът ще се визуализира по следния начин:



За да отместим първия ред на параграф се използва атрибутът `style` и `text-indent`, а самото отместване може да бъде зададено в брой пиксели или проценти:

```
<p style="text-indent:30px">
Този параграф е с отменстен <br> първи ред от 30 pixels. </p>
<p style="text-indent:30%">
Този параграф е с отменстен <br> първи ред от 30%. </p>
```

В браузър кодът ще се визуализира по следния начин:



За вертикалното подравняване се използва атрибутът `style` и `vertical-align` за постигане на желаното подравняване, а възможните му стойности са:

- `vertical-align:baseline` – подравняване на изходното ниво на елемента с изходната линия на родителския елемент. Това подравняване е зададено по подразбиране;
- `vertical-align:sub` – подравнява елемента като долен индекс;
- `vertical-align:super` – подравнява елемента като горен индекс;
- `vertical-align:top` – горната част на елемента е подравнена с върха на най-високия елемент от линията;
- `vertical-align:text-top` – горната част на елемента е подравнена с върха на шрифта на основния елемент;
- `vertical-align:middle` – подравнява елемента спрямо центъра на основния елемент;
- `vertical-align:bottom` – долната част на елемента се подравнява с най-ниския елемент от линията;
- `vertical-align:text-bottom` – долната част на елемента се подравнява с долната част на шрифта на основния елемент;

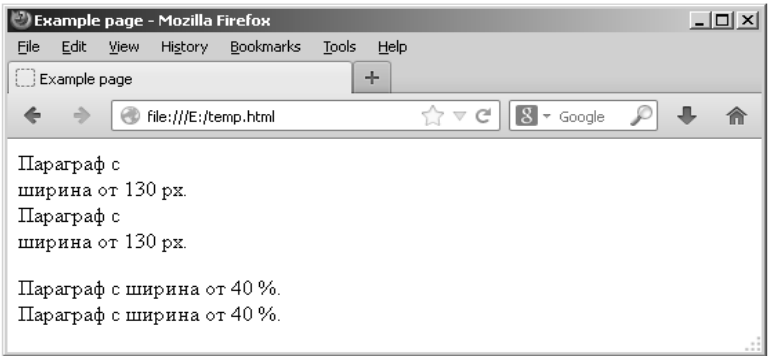
- `vertical-align: inherit` – указва, че стойността на вертикалното подравняване следва да се наследи от родителския елемент.

Вертикалното подравняване може да бъде зададено в *px*, *cm* или *%,* като са позволени и отрицателни стойности.

Задаването на определена ширина, в която да се разположи част от съдържанието на HTML документа, се реализира чрез атрибута `style` и `width`, като самата стойност се задава в *px*, *cm* или *%*. Например, нека зададем ширина на първия параграф *130 px*, а на втория параграф ширина *40 %*:

```
<p style="width: 130px">
Параграф с ширина от 130 px. Параграф с ширина от 130 px.</p>
<p style="width: 40%">
Параграф с ширина от 40 %. Параграф с ширина от 40 %. </p>
```

В браузър кодът ще се визуализира по следния начин:



2.2. Форматиране на списъци

Често в HTML документите се налага изброяването на някакви елементи и в тези случаи е подходящо да се използват специалните тагове на HTML за списъци, което придава подреден вид на страницата. В HTML се използват три вида списъци:

- неподредени: отделните елементи на списъка се открояват с един и същ графичен знак: кръгче, квадрат и т.н.;
- подредени: изброените елементи от списъка получават свой номер: римски, арабски цифри, латински малки или главни букви в азбучен ред и т.н.

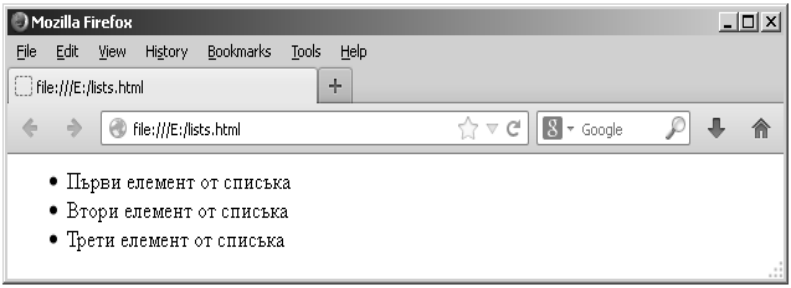
• дефиниращи: използват се за изброяване на понятия и техните дефиниции.

При използване на кой да е от списъците, браузърът визуализира списъка с автоматично отместване от ляво. Не е нужно да се изписва и тагът за нов ред `<br>`, тъй като това също става автоматично. Автоматично се добавя и по един празен ред над и под списъка.

Неподреден списък е списък, в който изброените елементи, не са номерирани с арабски цифри, римски цифри или букви. Браузърът изобразява списъците с автоматичен отстъп от ляво. Началото на неподредените списъци започва с тага `<ul>` и завършва с `</ul>`, като отделните елементи от списъка се отбелязват с таговете `<li>` и `</li>`. Синтаксисът на такъв списък е следният:

```
<ul>
   <li>Първи елемент от списъка</li>
   <li>Втори елемент от списъка</li>
   <li>Трети елемент от списъка</li>
</ul>
```

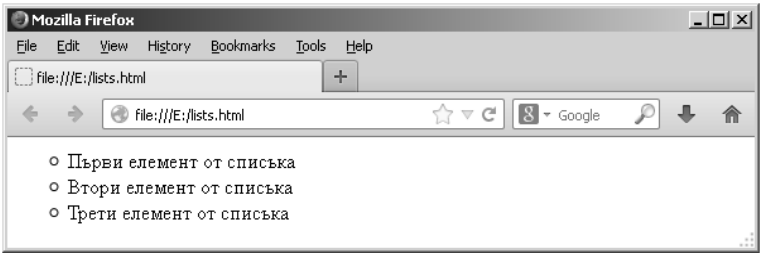
В браузър кодът ще се визуализира по следния начин:



Водещият символ (булет) в списъка, по подразбиране е черен запълнен кръг, но може да бъде променян чрез задаване на стойност на атрибута `type` от тага `<ul>`: `disc` (запълнен кръг, по подразбиране); `circle` (незапълнен кръг) или `square` (запълнен квадрат). Ето един пример за булет с незапълнен кръг:

```
<ul type="circle">
    <li>Първи елемент от списъка</li>
    <li>Втори елемент от списъка</li>
    <li>Трети елемент от списъка</li>
</ul>
```

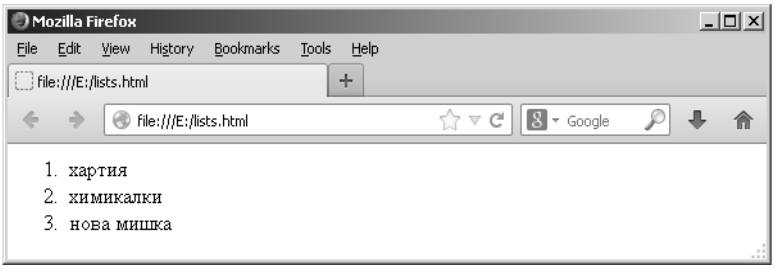
В браузър кодът ще се визуализира по следния начин:



Подредени списъци са тези списъци, в които изброяваните елементи получават номер. Синтаксисът е почти идентичен с този на неподредените списъци: единствената разлика е, че вместо `<ul>` и `</ul>` се използват `<ol>` и `</ol>`, съответно за начало и край на списъка. Ето и един пример:

```
<ol>
     <li> хартия </li>
     <li> химикалки </li>
     <li> нова мишка </li>
</ol>
```

В браузър кодът ще се визуализира по следния начин:



Подредените списъци автоматично добавят и показват номера 1, 2, 3, и т.н. в списъка. Възможно е списъкът да се показва с римски вместо с арабски цифри, като се използва атрибутът `type`, който определя какви символи да се визуализират пред елементите на списъка:

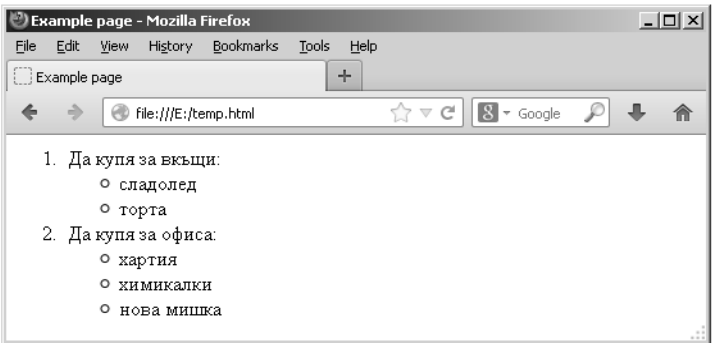
- `type` =1 – за списъци: 1, 2, 3,…
- `type` =a – за списъци: a, b, c, …
- `type` =A – за списъци: A, B, C, …

- `type` = i – за списъци: i, ii, iii, …
- `type` = I – за списъци: I, II, III, …

Списъците могат да бъдат влагани един в друг от един и същ вид или от различни видове, например:

```
<ol>
    <li> Да купя за вкъщи:</li>
         <ul TYPE=circle>
             <li> сладолед </li>
             <li> торта </li>
        </ul>
    <li> Да купя за офиса: </li>
         <ul TYPE=circle>
             <li> хартия </li>
             <li> химикалки </li>
             <li> нова мишка </li>
        </ul>
</ol>
```

В браузър кодът ще се визуализира по следния начин:



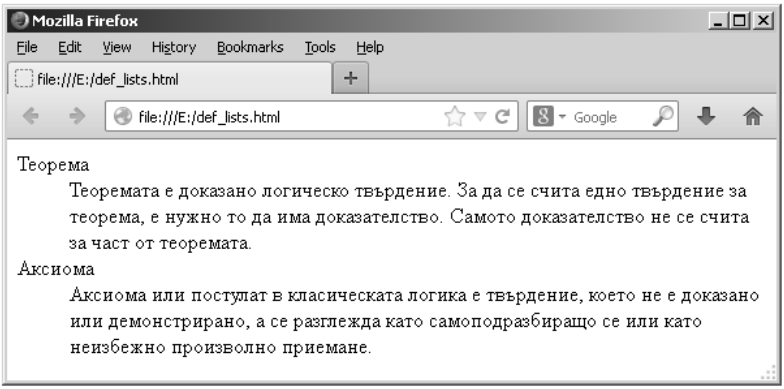
Дефиниращите списъци се задават с двойката тагове `<dl>` и `</dl>` за началото и за края на списъка. За изброяване на понятията се използват таговете `<dt>` и `</dt>`, а за техните дефиниции се използват таговете `<dd>` и `</dd>`.

Ето и един пример за дефиниращ списък:

```
<dl>
<dt> Теорема </dt>
<dd> Теоремата е доказано логическо твърдение. За да се счита едно
твърдение за теорема, е нужно то да има доказателство. Самото доказа-
телство не се счита за част от теоремата. </dd>
<dt> Аксиома </dt>
<dd> Аксиома или постулат в класическата логика е твърдение, което не
```

```
е доказано или демонстрирано, а се разглежда като самоподразбиращо се
или като неизбежно произволно приемане. </dd>
</dl>
```

Браузърът ще добавя автоматично нов ред след всички изброени дефиниращи понятия и също така автоматично ще направи отстъп от ляво за дефиниции към всяко дефиниращо понятие, както е показано:



Освен изброените до тук начини, за промяна стила на списъците можем да се възползваме от много по-богатите възможности, които дава CSS.

Глава 3. Използване на цветове и изображения

В тази глава е описано използването на цветове и изображения в HTML документи. Разгледани са предимствата и недостатъците на основните типове изображения, както и подробности за няхното използване. Специфична особеност на изображенията, за разлика от текста е, че те не се съхраняват на едно и също място. Текстът е част от самия HTML документ, докато изображението се съхранява във външен файл. Файлът на изображението трябва да бъде извикан от съответния таг, така че изображението да може да се покаже в HTML документа.

3.1. Използване на цветове

Цветовете в HTML могат да се задават чрез техните английски имена или чрез техния 16-ичен код. Чрез имена могат да се задават ограничен брой цветове и оттенъци, и освен това не всички браузъри възприемат имената. Препоръчително е цветовете да се задават в техния 16-ичен код. При 16-ична бройна система се използват цифрите от 0 до 9 плюс латинските букви от A до F. Следователно използваните символи при 16-сетичната бройна система са: 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, A, B, C, D, E, F.

Всеки цвят и оттенък може да се представи в 16-ична стойност. Точно тези стойности са разрешени за използване в HTML: 141 имена на цветове (17 стандартни цвята и 124 допълнителни цвята). Стандартните 17 цвята са: *aqua, black, blue, fuchsia, gray, green, lime, maroon, navy, olive, orange, purple, red, silver, teal, white* и *yellow*. Останалите 124 допълнителни цвята са дадени в Приложение 4.

3.2. Използвани формати на изображения

Форматите на изображения, използвани в HTML са:

- *Graphic Interchange Format (gif)* – този формат се използва предимно за изображения (илюстрации), които не са оцветени в богата цветова гама и не съдържат сложни оттенъци и светлосенки. Файловете с разширение *gif* имат две важни предимства: 1) могат да съдържат прозрачен елемент в себе си – това например може да бъде фонът на буквите от някакъв надпис, който ще прозира и под не-

го ще се вижда фонът на страницата. Тогава този файл ще бъде подходящ за страници, оцветени в различен фон, освен ако на изображението не е зададен ефектът сянка – тогава фонът трябва да е същият, с който е правен ефектът. 2) *gif*-файловете могат да бъдат анимирани, т.е. да съдържат движещо се изображение.

- *Joint Photographic Experts Group (jpeg)* или *jpg* форматите (и двете разширения са валидни и равностойни) се използват предимно за висококачествени фотографии. Възможностите на този формат за показване на цветове и оттенъци са по-богати от тези на *gif* формата, но за сметка на това *jpg* файла не може да бъде анимиран, нито да съдържа прозрачни елементи. *Jpeg* е стандарт на компресия „със загуби“, което означава, че от файла се отнема част от съдържащата се информация за даденото изображение.
- *Portable Network Graphics (png)* е графичен файлов формат, разработен специално за компресиране на изображения в Интернет среда. Възможните цветове във всеки пиксел са 16.7 милиона, като се използва начин за намаляване на размера на файла без загуба на качеството на съхраняваното изображение. Форматът *png* съдържа в себе си най-доброто от *gif* и *jpg* форматите.
- *Scalable Vector Graphics (SVG)* е файлов формат за векторна графика. Файловете са много малки, в сравнение с jpg и gif, и изображенията в тях да могат да бъдат преоразмерявани, без това да увеличи размера на файла.

Основните причини за създаването на файлови формати за графични изображения в Интернет пространството са изискванията за минимална големина и съответно бързо отваряне при зареждане на страницата.

3.3. Изображения – атрибути и стойности

Тагът `<img>` се използва за поставяне на изображение в документа и няма затварящ таг. По подразбиране едно изображение е вграден (in-line) елемент, което означава, че се премества заедно с текста, точно както става с една дума например. Вмъкването на изображение се отразява върху начина на пренасяне на редовете, а самото изображение се влияе от настройките на текста, като например зададените полета на страницата.

Тагът `<img>` има множество атрибути, които определят начина на показване на изображението. Най-важният от тях е атрибутът

`src`, който съдържа пътя до графичния файл – източника на изображението. Синтаксистът е:

```
<p><img src= "image.gif" /> </p>
```

Този пример използва относителен път, за да укаже файла на изображението, като файлът е търсен там, където се намира самият HTML документ. Ето подобен пример, в който е използван абсолютен път до файла с изображението:

```
<img src= "http://www.unibit.bg/imgs/image.gif" />
```

Когато изображението не се съхранява в същата директория, в която е HTML документът, винаги трябва да се използва абсолютен път.

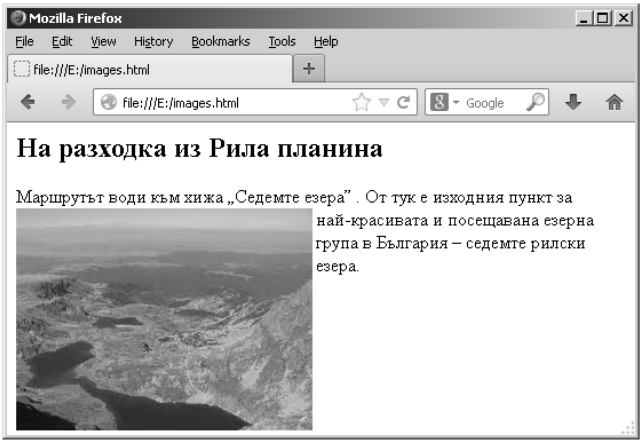
Освен задължителния атрибут `src`, тагът `<img>` притежава и друг важен атрибут, `alt`. Атрибутът `alt` се използва за присвояване на текст към изображенито – напр. кратък текст, поясняващ изображението.

```
<img src= "rila.jpg" alt="Това е част от Рила планина"/>
```

Следващият код илюстрира използването на изображение в HTML документ:

```
<h2>На разходка из Рила планина</h2>
<p>Маршрутът води към хижа "Седемте езера"
<img src= "rila.jpg" alt="Това е част от Рила планина" align="left" />.
От тук е изходният  пункт за най-красивата и посещавана езерна група в
България – Седемте рилски езера. </p>
```

В браузър кодът ще бъде визуализиран по следния начин:



Други два атрибута, които често се използват с тага `<img>`, са `width` и `height`. И двата атрибута се задават в пиксели. Ако не бъде зададена ширина и височина, браузърът показва подразбиращо се графично поле с определени размери, докато изображението се зареди в него. След това изображението се оразмерява точно, при което текстът на екрана се премества. Използването на атрибутите `width` и `height` позволява на браузъра да резервира точното място на екрана, преди изображението да е заредено. Този метод допринася за бързо показване на страницата при потребители с бавен достъп до Интернет. Ако на атрибутите `width` и `height` се зададат стойности, различни от тези на изображението, това ще принуди браузъра да премащабира изображението, като го разшири или свие до зададените размери. Ето един пример за използване на атрибутите `width` и `height`:

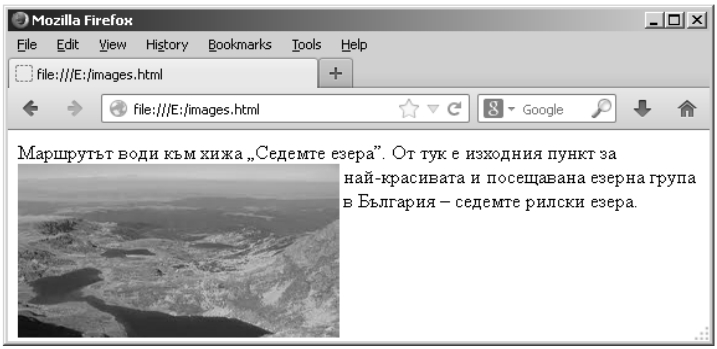
```
<img src= "image.jpg" alt="Това е част от Рила планина" width="260"
height="200" />
```

Тагът `<img>` чрез атрибута `align` може да управлява позиционирането на изображението и обтичането на текста около него. Подолу са представени възможните стойности на атрибута `align` и техните функции:

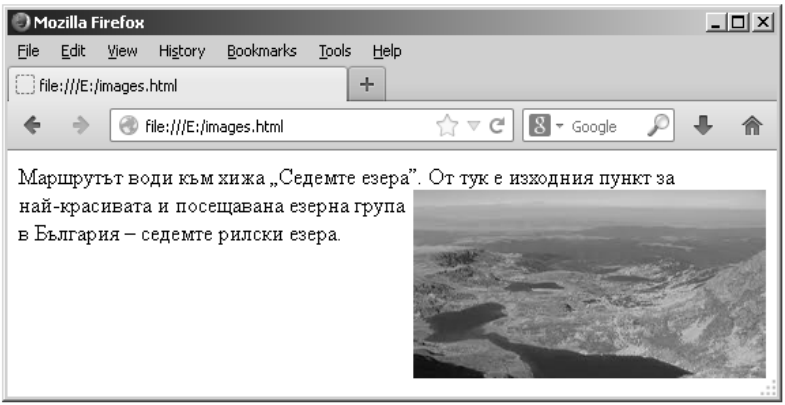
- `left` – подравнява изображението в ляво, като текстът обвива изображението от дясно.

```
<p>Маршрутът води към хижа "Седемте езера"
<img src= "rila.jpg" width="260" height="200" alt="Това е част от Рила
планина" align="left" />.
От тук е изходният пункт за най-красивата и посещавана езерна група в
България – Седемте рилски езера. </p>
```

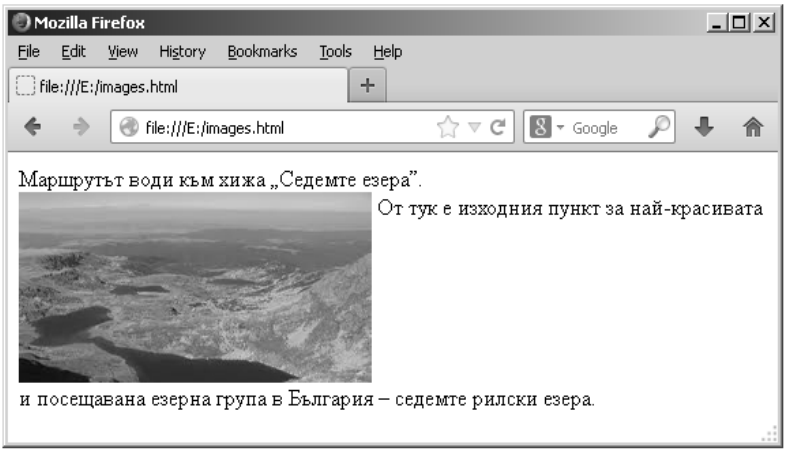
В браузър кодът ще се визуализира по следния начин:



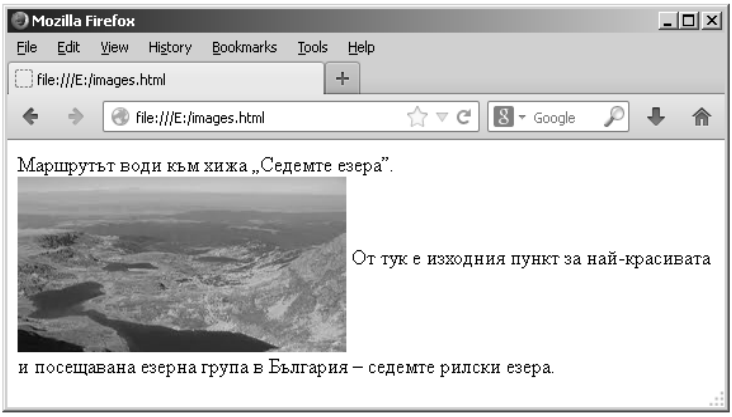
- `right` – подравнява изображението в дясно, като текстът обвива изображението отляво, както е показано:



- `top` – подравнява горната част на изображението по горния край на текущия ред от текста или по най-високото изображение в текущия ред, както е показано:



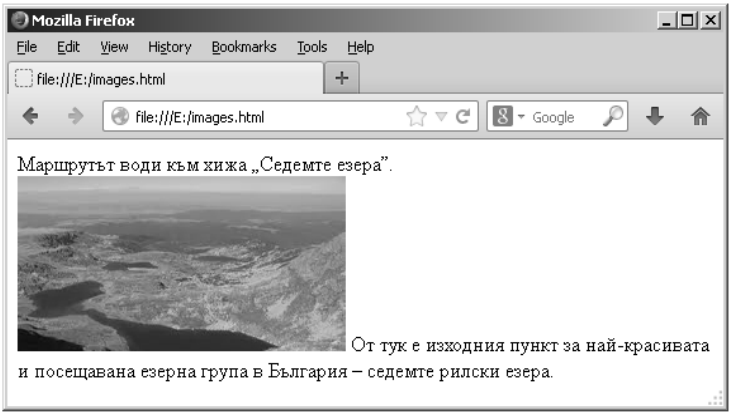
- `texttop` – подравнява горната част на изображението по горния край на текущия ред от текста.
- `middle` – подравнява средата на изображението по долния край на текущия ред от текста, често наричан „опорна линия" на текста:



• `absmiddle` – подравнява средата на изображението по средата на текущия ред от текста или по най-високото изображение в текущия ред.

• `center` – подравнява средата на изображението по средата на текущия ред от текста или по най-високото изображение в текущия ред, което дублира ефекта на *absmiddle*.

• `bottom` – подравнява долната част на изображението по долната част на текущия ред от текста (по опорната линия на текста):



• `baseline` – подравнява долната част на изображението по долната част на текущия ред от текста, по опорната линия на текста. Това дублира ефекта на *bottom*.

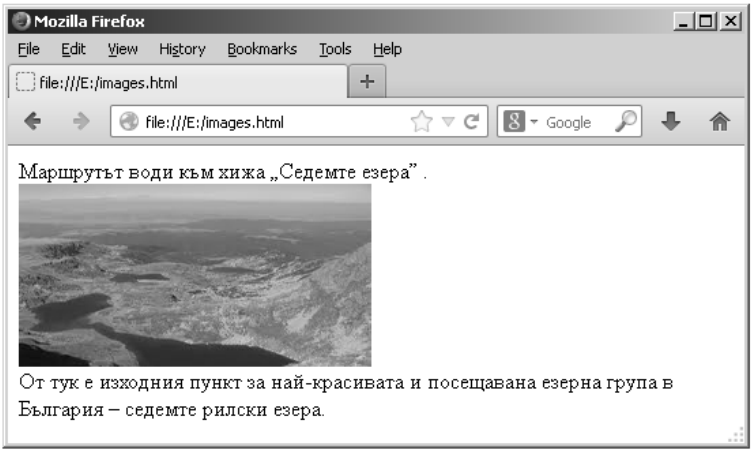
- `absbottom` – подравнява долната част на изображението по долната част на текущия ред от текста или по долната част на най-високото изображение в реда.

Прекъсване на подравняването на текста спрямо дясната или лявата страна на изображението можем да постигнем чрез атрибута `clear` на тага `br`. Три са възможните стойности:

- `left` – указва на текста да продължи, едва когато е свободно лявото пространство на страницата (когато отляво няма никакво изображение);
- `right` – указва на текста да продължи, едва когато е свободно дясното пространство на страницата (когато отдясно няма никакво изображение);
- `all` – указва на текста да продължи, едва когато е свободно пространството и отляво, и отдясно на страницата (когато и отляво, и отдясно няма изображение).

```
<p>Маршрутът води към хижа "Седемте езера"
<img src= "rila.jpg" alt="Това е част от Рила планина" align="left" />.
<br clear="all">От тук е изходният пункт за най-красивата и посещавана
езерна група в България – Седемте рилски езера. </p>
```

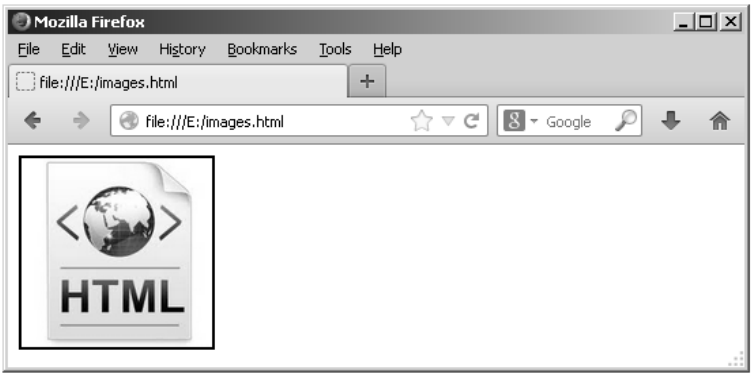
В браузър кодът ще се визуализира по следния начин:



Чрез атрибута `border` можем да зададем рамка около изображението в пиксели. Ако не използваме този атрибут, около изображението по подразбиране няма да се появи никаква рамка. Това се отнася само за изображения, които не са хипервръзки.

```
<img src="html.jpg" border="2" width="204" height="204"
alt="изображение">
```

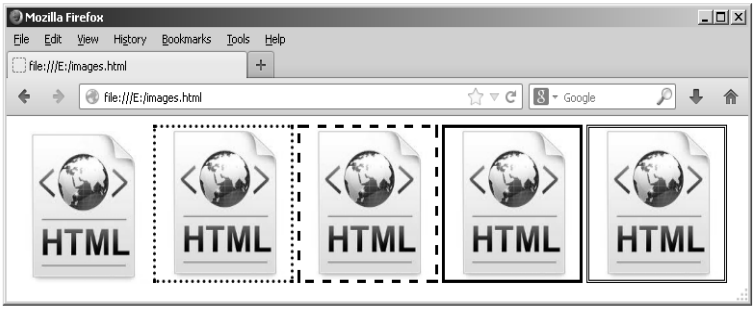
В браузър кодът ще се визуализира по следния начин:



Типът на рамката около изображението се определя чрез атрибута `border` като възможните стойности са: *none, dotted, dashed, solid, double, groove, ridge, inset, outset*.

```
<img src="html.jpg" border="2" width="204" height="204" style="border:
none">
<img src="html.jpg" border="2" width="204" height="204" style="border:
dotted">
<img src="html.jpg" border="2" width="204" height="204" style="border:
dashed">
<img src="html.jpg" border="2" width="204" height="204" style="border:
solid">
<img src="html.jpg" border="2" width="204" height="204" style="border:
double">
```

В браузър кодът ще се визуализира по следния начин:



Цветът на рамката около изображението се определя с атрибута *style* и съответен 16-ичен код или име:

```
<img src="html.jpg" border="2" width="204" height="204" style="border:
dotted blue">
<img src="html.jpg" border="2" width="204" height="204" style="border:
dashed red">
<img src="html.jpg" border="2" width="204" height="204" style="border:
solid #FFE477">
<img src="html.jpg" border="2" width="204" height="204" style="border:
double #6EFF75">
<img src="html.jpg" border="2" width="204" height="204" style="border:
solid #6600CC">
```

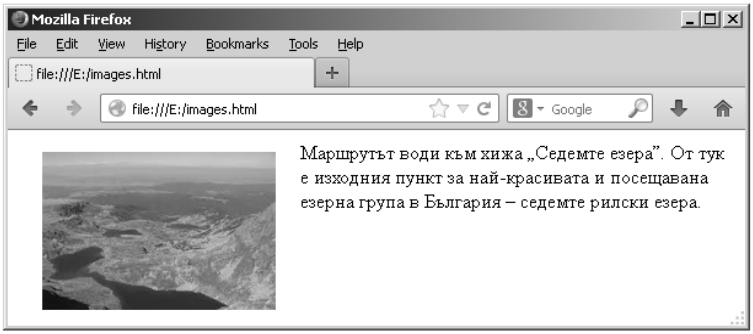
В браузър, кодът ще се визуализира по следния начин:

Тъй като изображенията често се показват наред с текста и другото съдържание, добре би било да ги отделим с малко пространство от останалото съдържание. Това осигурява по-голяма прегледност и позволява на потребителя да възприеме по-добре съдържанието на страницата. Задаването на такива празни пространства около изображенията се постига с атрибутите `hspace` и `vspace`.

Атрибутът `hspace` определя разстоянието (в пиксели) симетрично отляво и отдясно на изображението. Аналогично, `vspace` определя разстоянието (в пиксели) симетрично отгоре и отдолу на изображението. Ето и един пример:

```
<p>
<img src="rila-1.jpg" alt="Това е част от Рила планина" align="left"
hspace="20" vspace="10" width="260" height="200"> Маршрутът води към
хижа "Седемте езера". От тук е изходният  пункт за най-красивата и по-
сещавана езерна група в България – Седемте рилски езера.
</p>
```

В браузър кодът ще се визуализира по следния начин:



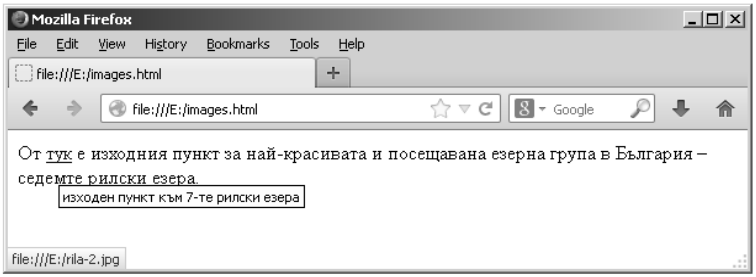
Изображенията могат да се използват и като хипервръзка: достатъчно е тагът за изображение да се вложи между таговете на хипервръзката `<a>` и `</a>`:

```
<a href="http://www.unibit.bg/">
<img src="www.jpg" border="0" width="110" height="110" alt="">
</a>
```

Какво обаче представлява препратката към външно изображение? Това е хипервръзка към файл на изображение, което се показва само, когато кликнем върху линка. Разликата при тези външни изображения е в това, че те не се показват директно в уеб страницата, заедно с линка.

```
<p>
От <a href="rila-2.jpg" title="изходен пункт към 7-те рилски езе-
ра">тук</a> е изходният пункт за най-красивата и посещавана езерна
група в България – Седемте рилски езера.
</p>
```

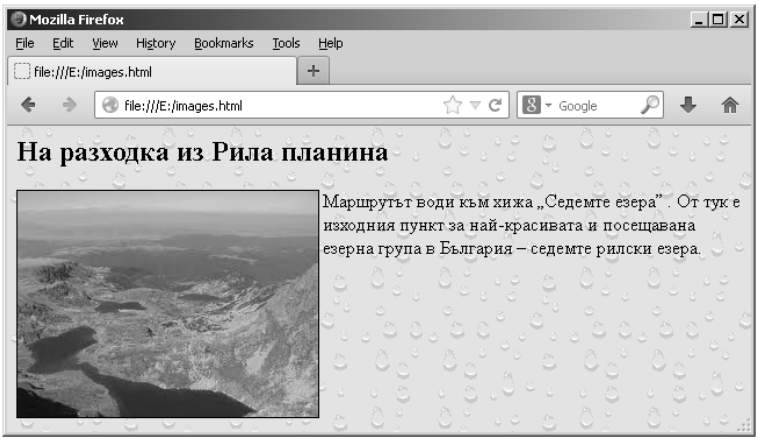
В браузър кодът ще се визуализира по следния начин:



За да добавим изображение, което да служи като фон на страницата, използваме атрибута `background` на елемента `body`:

```
<body background="water.gif">
<h2>На разходка из Рила планина</h2>
<p> Маршрутът води към хижа "Седемте езера" <img src="rila-1.jpg"
alt="Това е част от Рила планина" width="259" height="194"
align="left" border="1">. От тук е изходният пункт за най-красивата и
посещавана езерна група в България – Седемте рилски езера. </p>
</body>
```

В браузър кодът ще се визуализира по следния начин:



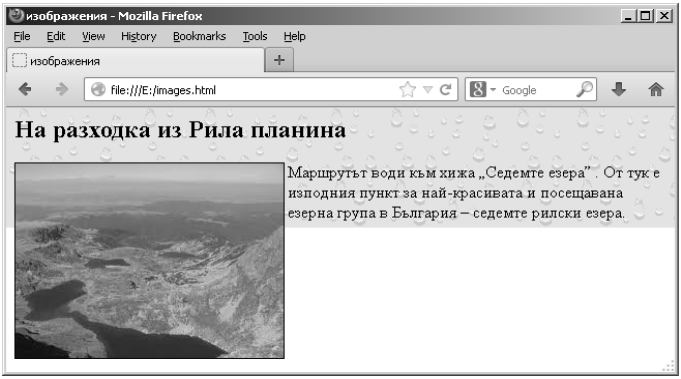
Когато използваме фоново изображение, трябва да се съобразим с няколко неща:

- като фон може да се ползва само едно изображение.
- фоновото изображение се наслагва по хоризонтала и вертикала толкова пъти, колкото е необходимо, за да запълни прозореца на браузъра.
- тескът на преден план трябва да е четлив, на фона на изображението.
- изображенията, използвани като фон, е желателно да са с малък размер на файла.

Чрез атрибута `style` може да зададем повтаряне на изображението само по хоризонтала (`background-repeat: repeat-x`), само по вертикала (`background-repeat: repeat-y`) или да не се повтаря (`background-repeat: no-repeat`). Например, за фоново изображение, повтарящо се по хоризонтала, кодът ще бъде:

```
<body background="water.gif" style="background-repeat: repeat-x">
```

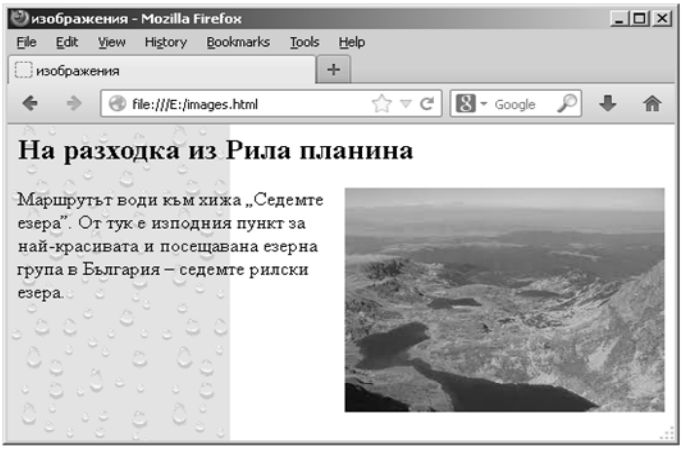
и ще се визуализира по следния начин:



За фоново изображение, повтарящо се по вертикала, кодът е:

```
<body background="water.gif" style="background-repeat: repeat-y">
```

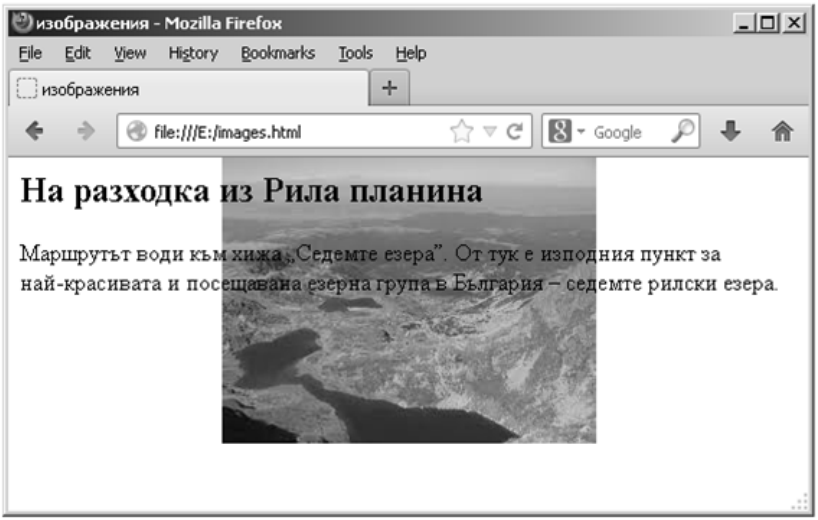
и в браузър ще се визуализира по следния начин:



Едно-единствено фоново изображение може да зададем чрез атрибута `style`, като определим пътя до изображението (`background-image`), неговото позициониране спрямо хоризонталата и вертикалата (`background-position`) и съответно неговото повторение `background-repeat`, както е показано:

```
<body style="background-image: url(rila-1.jpg); background-position:
center top; background-repeat: no-repeat">
```

В браузър кодът ще се визуализира по следния начин:



3.4. Изображение карта

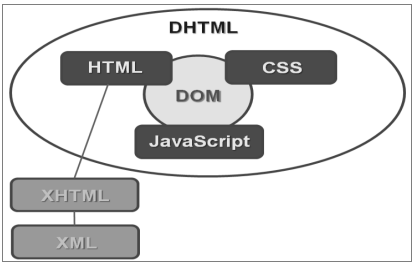
Дадено изображение може да бъде разделено на отделни части (области), като към тези части се присвоят определени хипервръзки. Такова изображение се нарича „изображение карта".

Дефинираните „горещи зони", към които се присвояват съответните връзки, могат да бъдат под формата на *правоъгълник, окръжност, многоъгълник* или *точка*. Координатите на изброените геометрични форми се задават в пиксели спрямо горния ляв ъгъл *(0,0)* на конкретното графично изображение.

- `rect` – правоъгълна зона, чийто координати се определят от стойностите в пиксели на горния ляв ъгъл и долния десен, спрямо нулевата координата на графиката. Активната зона е вътрешността на правоъгълника;
- `circle` – кръгова зона, чийто координати се определят от двойка точки. Първата определя центъра на окръжността, втората точка лежи на линията на окръжността. Активната зона е вътрешността на определената окръжност;

- `poly` – многоъгълна зона, зададена чрез списък от координати, представляващи върховете на многоъгълника. Многоъгълната зона може да бъде дефинирана с не повече от 100 двойки координати. Активната зона е вътрешността на многоъгълника;
- `point` – активната зона се дефинира от координатите на една точка, зададени спрямо горния ляв ъгъл на изобразената графика;
- `default` – е атрибут, с помощта на който се означават областите в графичното изображение, които не са били описани като активни зони от други дефиниции.

Нека разделим показаното изображение на 5 области (HTML, CSS, JavaScript, XHTML и XML), като присвоим съответна хипервръзка към определената област на съответния език.



Кодът за определените 5 правоъгълни области от изображението има вида:

```
<img src="dom.png" border="0" width="740" height="550" alt=""
usemap="#ime">
<map name="ime">
<area shape="rect" coords="108,103,305,171"
href="http://www.w3schools.com/html/default.asp">
<area shape="rect" coords="407,100,602,170"
href="http://www.w3schools.com/css/default.asp">
<area shape="rect" coords="243,259,469,328"
href="http://www.w3schools.com/js/default.asp">
<area shape="rect" coords="17,372,244,441"
href="http://www.w3schools.com/html/html_xhtml.asp">
<area shape="rect" coords="19,472,246,542"
href="http://www.w3schools.com/xml/default.asp">
</map>
```

Тагът `map` включва цялата необходима информация за определените „горещи зони“ от изображението. Тагът `map` и включената в него информация могат да бъдат разположени навсякъде в HTML страницата и не е задължително да са непосредствено след съответстващият им таг `img`. Атрибутът `name` се използва с тага `map`, за да се посочи за кое изображение се отнася направеното разпределение. Възможни атрибути на елемента `img` са:

Атрибут	Стойност	Описание
align	top, bottom, middle, left, right	Указва подравняването на изображението според околните елементи. Отхвърлен в HTML 4.01. Не се поддържа в HTML5.
alt	text	Задава алтернативен текст за изображението
border	pixels	Не се поддържа в HTML5. Отхвърлен в HTML 4.01. Определя ширината на рамката около изображението
crossorigin (new)	anonymous, use-credentials	Позволява използване на изображения от други сайтове като фон
height	pixels	Задава височината на изображението
hspace	pixels	Задава празно пространство на лявата и дясната страна на изображението. Отхвърлен в HTML 4.01. Не се поддържа в HTML5.
ismap	ismap	Задава изображение от страна на сървъра като изображение карта
src	URL	Задава URL на изображението
usemap	#mapname	Задава изображение от страна на клиента като изображение карта
vspace	pixels	Задава бялото пространство от горната и долната част на изображението. Отхвърлен в HTML 4.01. Не се поддържа в HTML5.
width	pixels	Задава ширината на изображението.

3.5. Мащабируема векторна графика

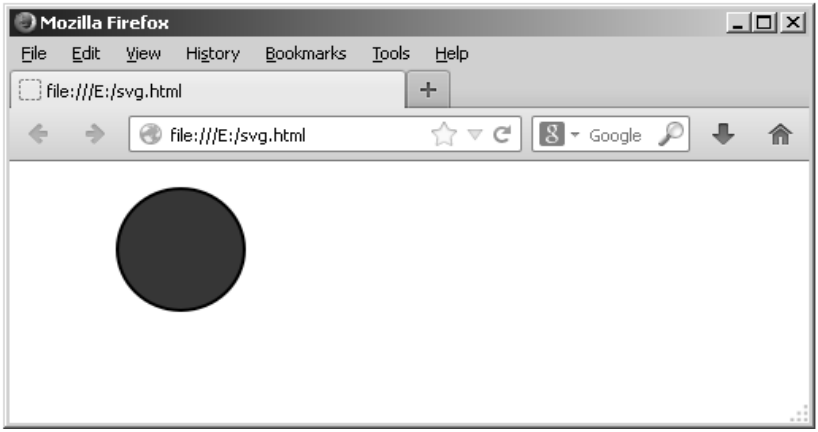
SVG (*Scalable Vector Graphics*) се базира на езика XML за описване на двумерна векторна графика, която поддържа интерактивност и анимация. Предимствата на използването на *SVG* изображенията, в сравнение с други графични формати (като *JPEG* и *GIF*), са:

- могат да се създават и редактират с всеки текстов редактор;
- могат да бъдат търсени, индексирани, компресирани;
- мащабируеми са;
- могат да бъдат отпечатани с високо качество и с всякаква разрешаваща способност;
- могат да бъдат увеличавани без загуба качеството;
- *SVG* е отворен стандарт;
- *SVG* файловете са чист XML код.

Ето един пример за използване на *SVG* формат – изображение на запълнена окръжност в червен цвят, с контурна линия в черен цвят:

```
<!DOCTYPE html>
<html>
<body>

<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
   <circle cx="100" cy="50" r="40" stroke="black" stroke-width="2"
fill="red" />
</svg>

</body>
</html>
```

Кодът на *SVG* формата започва с `<svg>`. Атрибутът *xmlns* дефинира пространство от имена на *SVG* и определя коя версия *SVG* да бъде използвана. Елементът `<circle>` се използва за да се нарисува кръг. Атрибутите *cx* и *cy* определят *x* и *y* координатата на центъра на кръга. Ако *cx* и *cy* са пропуснати, за център на кръга се подразбира точка с координати (0, 0). Атрибутът *r* определя радиуса на кръга. Атрибутите *stroke* и *stroke-width* управляват как да се визуализира очертанието на формата. В примера по-горе е зададено очертание на кръга в черен цвят и дебелина на линията от *2 px*. Атрибутът *fill* се отнася до цвета на запълване на кръга – в примера запълване с червен цвят. И накрая, затваряне на тага `</svg>`. В браузър кодът от примера ще се визуализира по следния начин:



Пример за изображение на правоъгълник в *SVG* формат със заоблени ъгли:

```
<!DOCTYPE html>
<html>
<body>

<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
  <rect x="50" y="20" rx="20" ry="20" width="150" height="100"
style="fill:red;stroke:black;stroke-width:5;" />
</svg>

</body>
</html>
```

В браузър кодът ще се визуализира по следния начин:

Пример за разполагане на текст по определена крива:

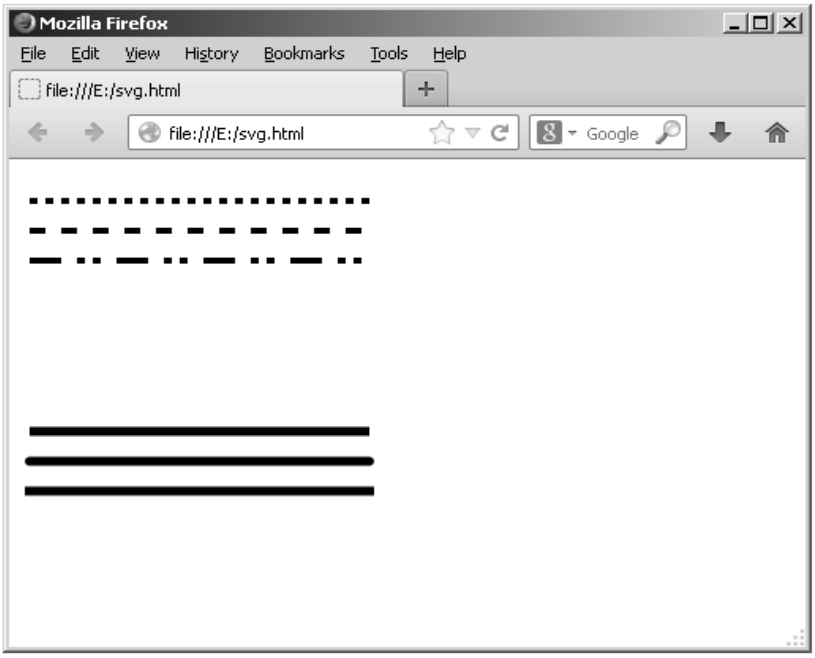
```
<!DOCTYPE html>
<html>
<body>

<svg xmlns="http://www.w3.org/2000/svg" version="1.1"
xmlns:xlink="http://www.w3.org/1999/xlink">
  <defs>
    <path id="path1" d="M75,20 a1,1 0 0,0 100,0" />
  </defs>
  <text x="10" y="100" style="fill:red;">
    <textPath xlink:href="#path1">I love SVG I love SVG</textPath>
  </text>
</svg>

</body>
</html>
```

В браузър кодът ще се визуализира по следния начин:

Свойството *stroke-dasharray* се използва за създаването на пунктирани линии, а свойството *stroke-linecap* дефинира различен тип за крайщата на линията:

```
<!DOCTYPE html>
<html>
<body>

<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
  <g fill="none" stroke="black" stroke-width="4">
    <path stroke-dasharray="5,5" d="M5 20 l215 0" />
    <path stroke-dasharray="10,10" d="M5 40 l215 0" />
    <path stroke-dasharray="20,10,5,5,5,10" d="M5 60 l215 0" />
  </g>
</svg>

<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
  <g fill="none" stroke="black" stroke-width="6">
    <path stroke-linecap="butt" d="M5 20 l215 0" />
    <path stroke-linecap="round" d="M5 40 l215 0" />
    <path stroke-linecap="square" d="M5 60 l215 0" />
  </g>
</svg>

</body>
</html>
```

В браузър кодът ще се визуализира по следния начин:



Глава 4. Таблици – тагове и атрибути

В тази глава е описано използването на таблици, техните тагове и атрибути. Таблиците са изключително гъвкави и предоставят значителен контрол върху позиционирането на информацията в уеб страниците. Една таблица може да бъде влагана в друга таблица, което дава още по-голяма гъвкавост при структурирането на уеб страниците. Ето защо таблиците са важен инструмент при създаването на една уеб страница. Илюстрирани са различни начини за форматиране както на цялата таблица, така и на отделните редове и на отделните клетки в нея. Показани са възможности за форматиране на стила и цвета на рамката на таблицата, както и позиционирането на таблицата в уеб документа. Описани са начините, чрез които може се реализира обединяване на клетки от една таблица по редове и/или обединяване на клетки по колони.

4.1. Тагове и атрибути на таблици

Таблиците в HTML се създават посредством 3 основни тага

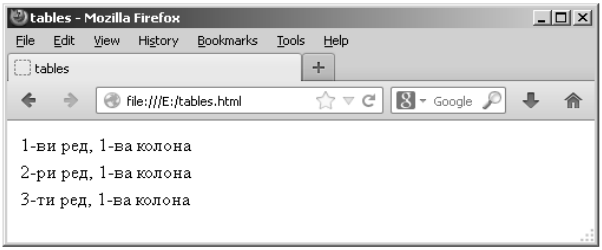
- `<table>` и `</table>` за начало и край на таблицата;
- `<tr>` и `</tr>` за начало и край на табличен ред;
- `<td>` и `</td>` за начало и край на клетка в таблицата.

Таблицата се разделя на редове с тага `<tr>`, а всеки ред се разделя на клетки чрез тага `<td>`. Тагът `<td>` може да съдържа текст, линкове, изображения, списъци, формуляри, други таблици и т.н.

Ето как би изглеждал кодът за една таблица, състояща се от 3 реда и 1 колона:

```
<table>
 <tr>
    <td> 1-ви ред, 1-ва колона</td>
 </tr>
 <tr>
    <td> 2-ри ред, 1-ва колона</td>
 </tr>
 <tr>
    <td> 3-ти ред, 1-ва колона</td>
 </tr>
</table>
```

Браузърът би интерпретирал този код по следния начин:

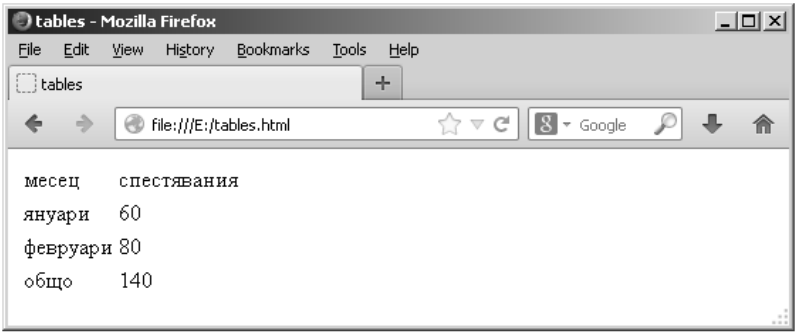


Тагът `<tbody>` се използва за групиране съдържанието на тялото в HTML таблица. Тагът `<tbody>` се използва във връзка с таговете `<thead>` и `<tfoot>` за специфициране на всяка част от таблицата (тяло, заглавна, долна). Браузърите използват тези елементи и позволяват превъртане на тялото на таблицата, независимо от заглавната и долната част. Също така, когато се отпечатва голяма таблица, която обхваща няколко страници, тези елементи дават възможност за отпечатване както на горната, така и на долната част на всяка страница.

Ето и един пример за таблица, използваща таговете `<thead>`, `<tfoot>` и `<tbody>`:

```
<table>
  <thead>
    <tr>
      <td>месец</td>
      <td>спестявания</td>
    </tr>
  </thead>
  <tfoot>
    <tr>
      <td>общо</td>
      <td>140</td>
    </tr>
  </tfoot>
  <tbody>
    <tr>
      <td>януари</td>
      <td>60</td>
    </tr>
    <tr>
      <td>февруари</td>
      <td>80</td>
    </tr>
  </tbody>
</table>
```

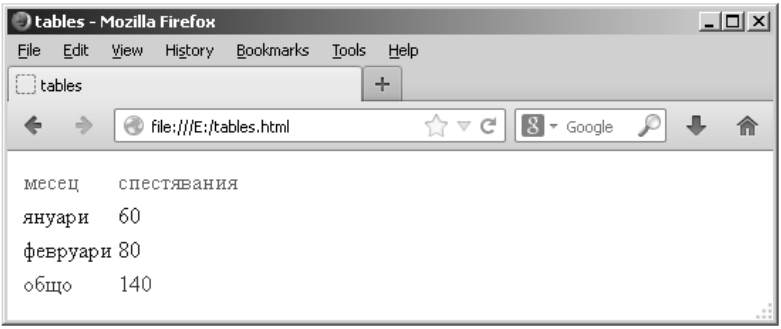
В браузър кодът ще се визуализира по следния начин:



Както се вижда, елементите `<thead>`, `<tbody>`, и `<tfoot>` не влияят върху структурата на таблицата и на нейното визуализиране. За да покажем как се манипулират тези елементи, ще дефинираме определени стилове чрез средствата на CSS. Например, нека да зададем различни цветове за съдържанието на елементите `<thead>`, `<tbody>` и `<tfoot>`:

```
<style type="text/css">
     thead {color:green;}
     tbody {color:blue;}
     tfoot {color:red;}
</style>
```

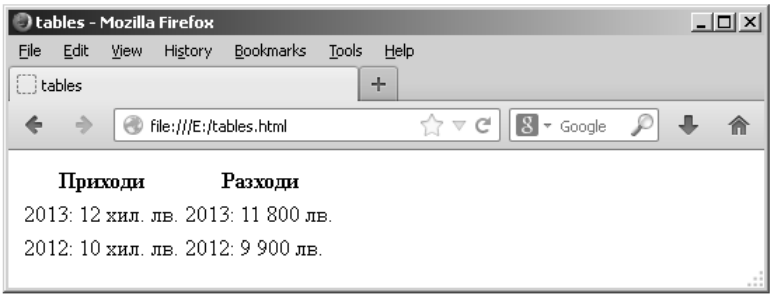
Тогава браузърът ще визуализира таблицата по следния начин:



Чрез двойката тагове `<th>` и `</th>` може да задаваме заглавие в таблицата. Всеки текст, разположен между тези тагове, се показва получерен и центриран в клетката на таблицата:

```
<table>
  <tr>
      <th> Приходи </th>
      <th> Разходи </th>
  </tr>
  <tr>
      <td> 2013: 12 хил. лв.</td>
      <td> 2013: 11 800 лв.</td>
  </tr>
  <tr>
      <td> 2012: 10 хил. лв.</td>
      <td> 2012: 9 900 лв.</td>
  </tr>
</table>
```

Браузърът би интерпретирал този код по следния начин:



Чрез атрибута `summary` може да бъде зададено кратко резюме на таблицата, което не се визуализира в браузъра:

```
<table summary="Приходи и разходи на семейство Иванови">
```

Ако не се използва атрибута `border`, таблицата ще се показва без рамка. Понякога това може да бъде полезно, но в по-голяма част от случайте бихме искали да покажем и рамката. Аналогично на всички атрибути, атрибутът `border` се поставя в отварящия таг за таблицата.

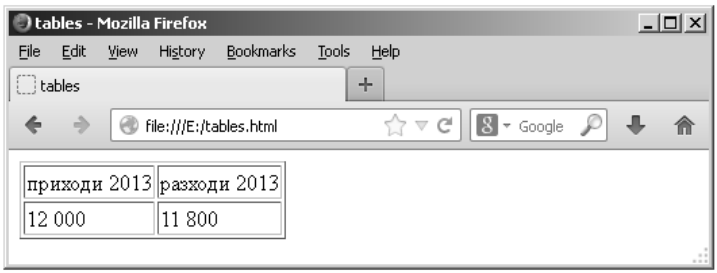
Пример за таблица с рамка, състояща се от 1 ред и 3 колони:

```
<table border="1">
  <tr>
      <td> 1-ви ред, 1-ва колона </td>
      <td> 1-ви ред, 2-ра колона </td>
      <td> 1-ви ред, 3-та колона </td>
</tr>
</table>
```

Пример за таблица с рамка, състояща се от 2 реда и 2 колони:

```
<table border="1">
  <tr>
      <td> приходи 2013 </td>
      <td> разходи 2013 </td>
  </tr>
  <tr>
      <td> 12 000 </td>
      <td> 11 800 </td>
  </tr>
</table>
```

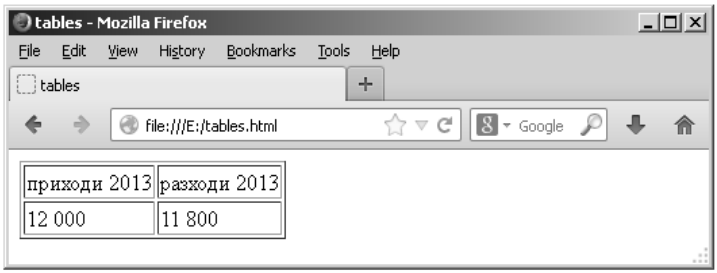
Браузърът би интерпретирал този код по следния начин:



Използването на атрибута `bordercolor` прави възможно задаването на определен цвят на рамката на таблицата:

```
<table border="1" bordercolor=#FF0066>
```

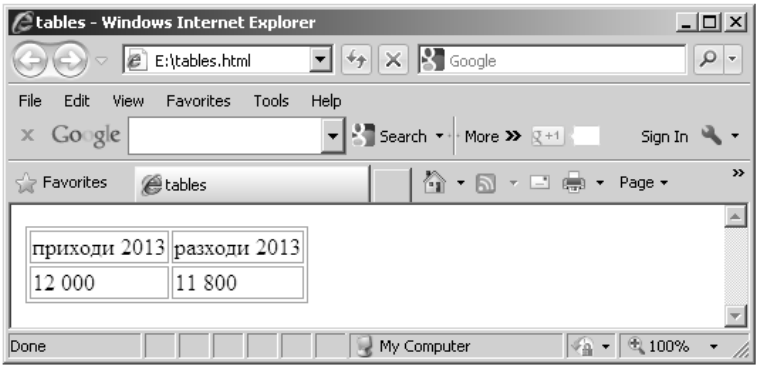
Браузърът би интерпретирал този код по следния начин:



За Internet Explorer можем да добавим още два атрибута: `bordercolordark` – за цвета отляво и отгоре за външната рамка, и `bordercolorlight` – за определяне на цвета на външната рамката отдясно и отдолу:

```
<table border="1" bordercolordark="#66CC00" bordercolorlight="#FF9900">
```

Браузърът би интерпретирал този код по следния начин:



Въпреки че не е широко използван, атрибутът `frame` е официално част от HTML 4.01 спецификацията и служи за определяне на начина за визуализиране на рамката около таблицата. Този атрибут може да има следните стойности:

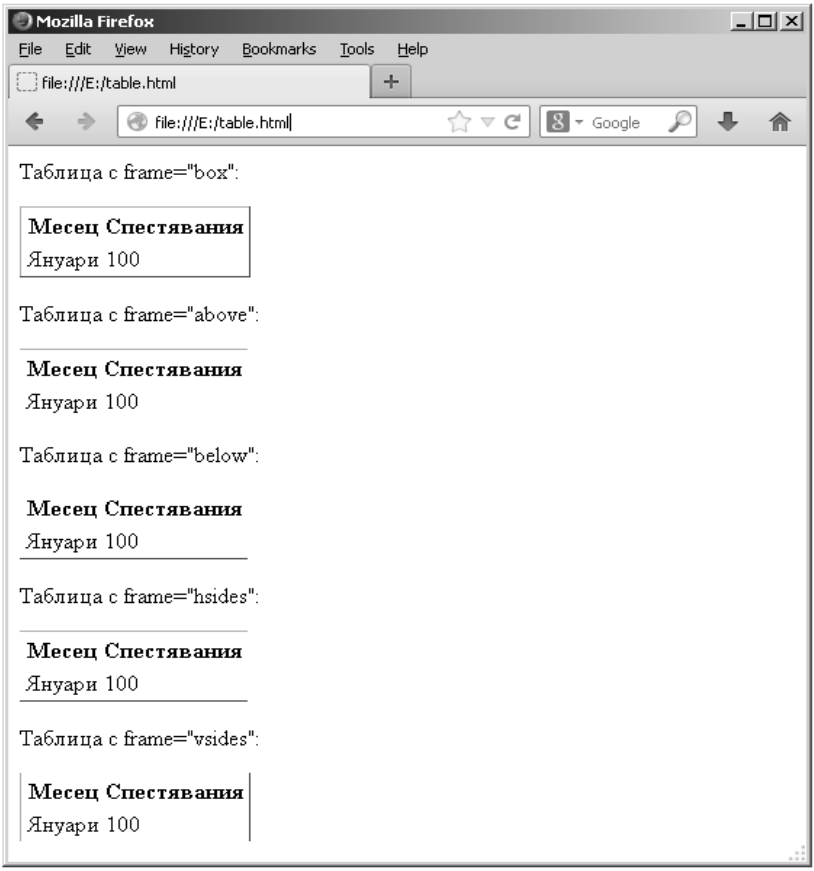
- `above` – показва само горната рамка;
- `below` – показва само долната рамка;
- `hsides` – показва само хоризонталните рамки (горните и долните рамки);
- `lhs` – показва лявата външна рамка;
- `rhs` – показва дясната външна рамка;
- `vsides` – показва лявата и десните външни рамки;
- `box` – показва външните рамки на всичките четири страни;
- `border` – показва външните рамки на всичките четири страни;
- `void` – не показва външните рамки.

В следващият пример са показани няколко таблици, илюстриращи възможностите на атрибута `frame`.

```
<p>Таблица с frame="box":</p>
<table frame="box">
  <tr>
    <th>Месец</th>
    <th>Спестявания</th>
  </tr>
  <tr>
    <td>Януари</td>
    <td> 100</td>
```

```
  </tr>
</table>

<p>Таблица с frame="above":</p>
<table frame="above">
  <tr>
    <th>Месец</th>
    <th>Спестявания</th>
  </tr>
  <tr>
    <td>Януари</td>
    <td> 100</td>
  </tr>
</table>

<p>Таблица с frame="below":</p>
<table frame="below">
  <tr>
    <th>Месец</th>
    <th>Спестявания</th>
  </tr>
  <tr>
    <td>Януари</td>
    <td> 100</td>
  </tr>
</table>

<p>Таблица с frame="hsides":</p>
<table frame="hsides">
  <tr>
    <th>Месец</th>
    <th>Спестявания</th>
  </tr>
  <tr>
    <td>Януари</td>
    <td> 100</td>
  </tr>
</table>

<p>Таблица с frame="vsides":</p>
<table frame="vsides">
  <tr>
    <th>Месец</th>
    <th>Спестявания</th>
  </tr>
  <tr>
    <td>Януари</td>
    <td> 100</td>
  </tr>
</table>
```

В средата на браузъра, таблиците с различните стойности на атрибута `frame` ще се визуализират по следния начин:



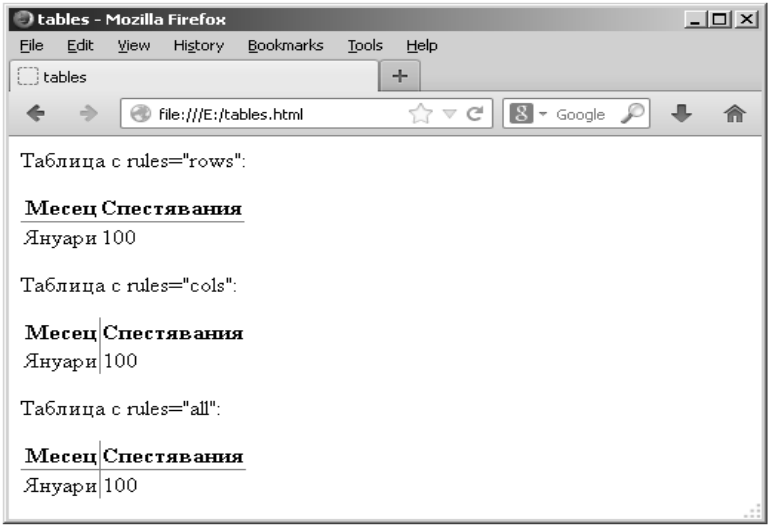
Атрибутът `rules` определя кои части на вътрешните граници ще се визуализират. Този атрибут използва следните стойности:

- `none` – без линии;
- `groups` – линии между групи от редове и групи от колони;
- `rows` – линии между редовете;
- `cols` – линии между колоните;
- `all` – линии между редовете и колоните.

Ето пример за използване на атрибута `rules` с различни стойности за визуализиране на части от рамката на таблиците:

```
<p>Таблица с rules="rows":</p>
<table rules="rows">
  <tr>
     <th>Месец</th>
     <th>Спестявания</th>
  </tr>
  <tr>
     <td>Януари</td>
     <td> 100</td>
  </tr>
</table>

<p>Таблица с rules="cols":</p>
<table rules="cols">
  <tr>
     <th>Месец</th>
     <th>Спестявания</th>
  </tr>
  <tr>
     <td>Януари</td>
     <td> 100</td>
  </tr>
</table>

<p>Таблица с rules="all":</p>
<table rules="all">
  <tr>
     <th>Месец</th>
     <th>Спестявания</th>
  </tr>
  <tr>
     <td>Януари</td>
     <td> 100</td>
  </tr>
</table>
```

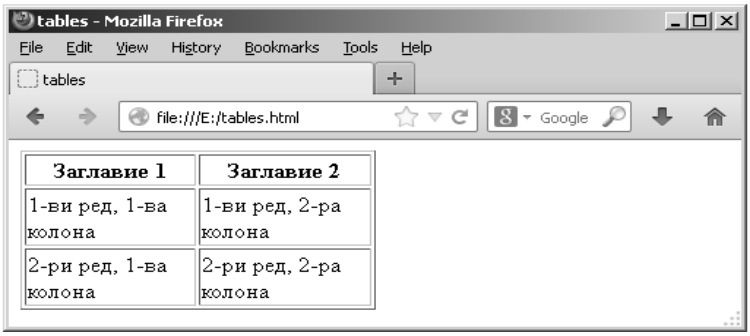
В средата на браузъра таблиците ще се визуализират по следния начин:



Ширината на таблица може да зададем в проценти или в брой пиксели чрез атрибута `width`:

```
<table width=50% border=1>
```

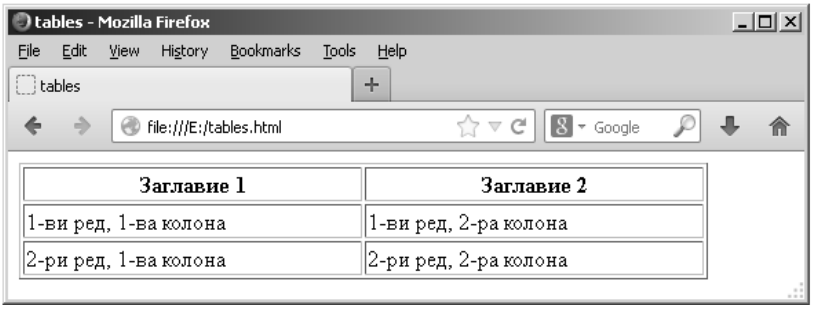
Браузърът би интерпретирал този код по следния начин:



Задавайки ширината на таблицата в проценти се осигурява процентното съотношение в средата на браузъра, докато използването на пиксели осигурява една и съща ширина на таблицата, независимо от разделителната способност на екрана. Пример за таблица с точно 500 пиксела ширина:

```
<table width=500 border=1>
```

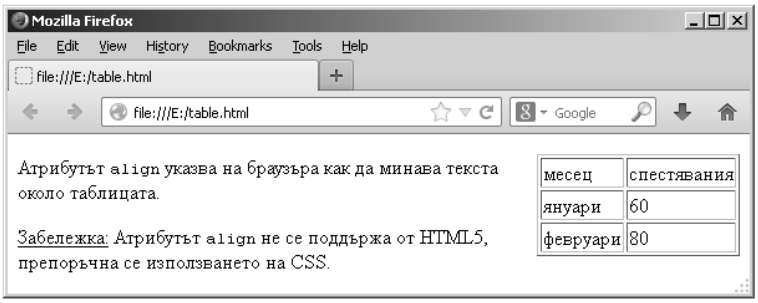
Браузърът би интерпретирал този код по следния начин:



За подравняването на таблиците се използва атрибутът `align`, със стойности: *left, center* или *right*. Това определя подравняването на таблицата в прозореца на браузъра вляво (по подразбиране), центрирана или вдясно, спрямо заобикалящия я текст. Показани са два примера за центрирана и дясно подравнена таблица.

```
<table border="1" align="right">
  <tr>
      <td>месец</td>
      <td>спестявания</td>
  </tr>
  <tr>
      <td>януари</td>
      <td>60</td>
  </tr>
  <tr>
      <td>февруари</td>
      <td>80</td>
  </tr>
</table>
<p>Атрибутът <code>align</code> указва на браузъра как да минава текс-
та около таблицата.</p>
<p><u>Забележка:</u> Атрибутът <code>align</code> не се поддържа от
HTML5, препоръчна се използването на CSS. </p>
```

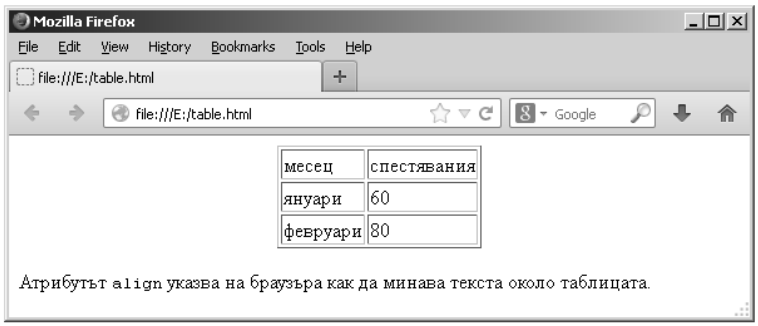
Браузърът би интерпретирал този код по следния начин:



Подравняване на таблицата в центъра на документа:

```
<table border="1" align="center">
  <tr>
      <td>месец</td>
      <td>спестявания</td>
  </tr>
  <tr>
      <td>януари</td>
      <td>60</td>
  </tr>
  <tr>
      <td>февруари</td>
      <td>80</td>
  </tr>
</table>
<p>Атрибутът <code>align</code> указва на браузъра как да минава текс-
та около таблицата.</p>
```

Браузърът би интерпретирал този код по следния начин:

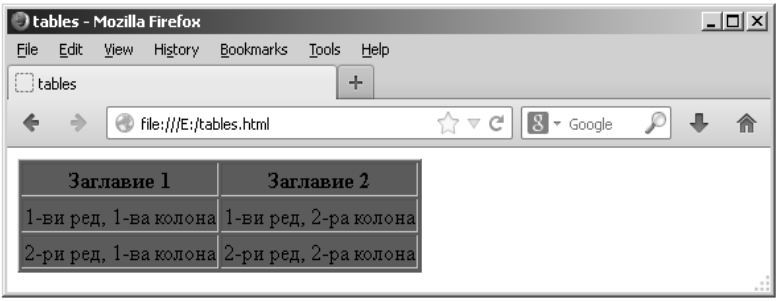


Атрибутът `align` не се поддържа от HTML5, затова се препоръчва използването на CSS за подравняване на таблиците.

С атрибута `bgcolor` може да зададем цвят на таблицата чрез съответно име на цвета, чрез 16-ичен код, име или чрез RGB стойност. Ето един пример за таблица с червен фон, зададен в 16-ичен код:

```
<table border="1" bgcolor="#FF3300">
```

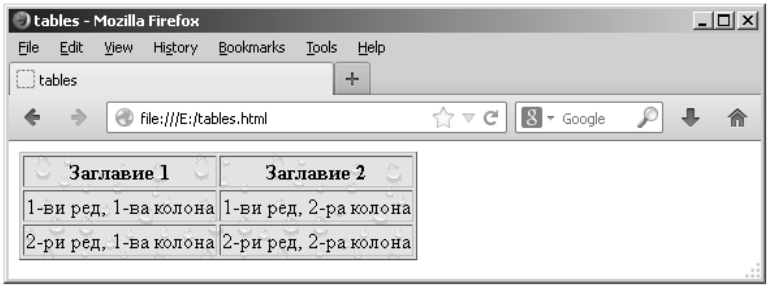
Браузърът би интерпретирал този код по следния начин:



Освен цвят, като фон на таблицата може да се използва изображение, което ще се повтори толкова пъти, колкото е необходимо, така че да запълни цялата таблица. Това се постига чрез използване на атрибута `background`:

```
<table border="1" background="water.gif">
```

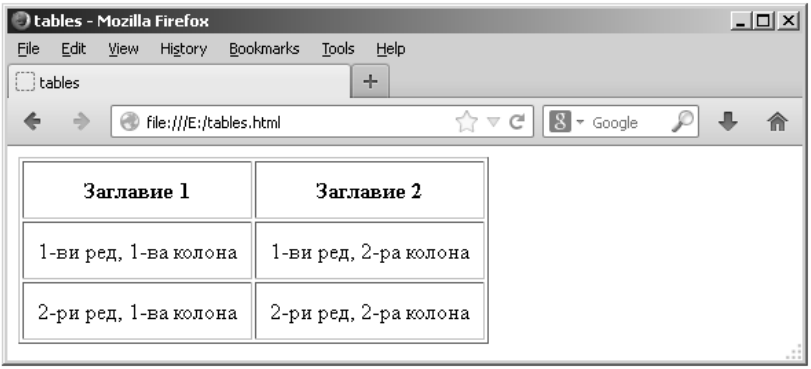
Браузърът би интерпретирал този код по следния начин:



За отместване на съдържанието на клетката от стената на самата клетка може да се използва атрибутът `cellpadding`. Той определя отместването в брой пиксели:

```
<table border="1" cellpadding="10">
```

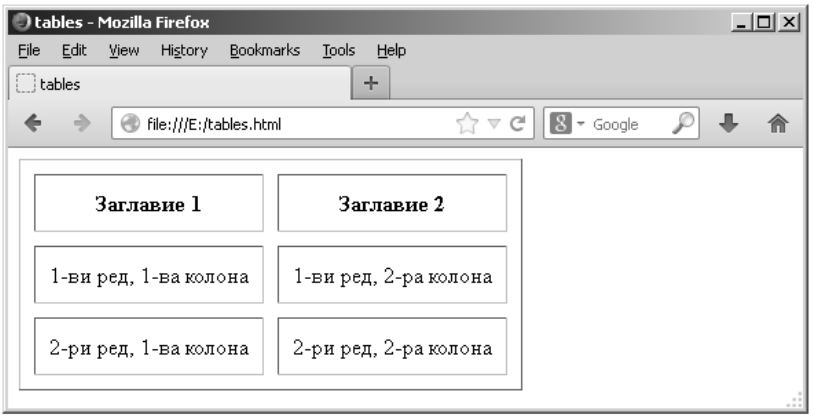
Браузърът би интерпретирал този код по следния начин:



За определяне на повече пространството между отделните клетки на таблицата се използва атрибутът `cellspacing`. Неговата стойност също се задава в брой пиксели. Нека зададем по 10 пиксела разстояние от ръба на клетката до нейното съдържание и също 10 пиксела разстояние между отделните клетки на таблицата, както е показано:

```
<table border="1" cellpadding="10" cellspacing="10">
```

Браузърът би интерпретирал този код по следния начин:



4.2. Форматиране на съдържанието на клетките

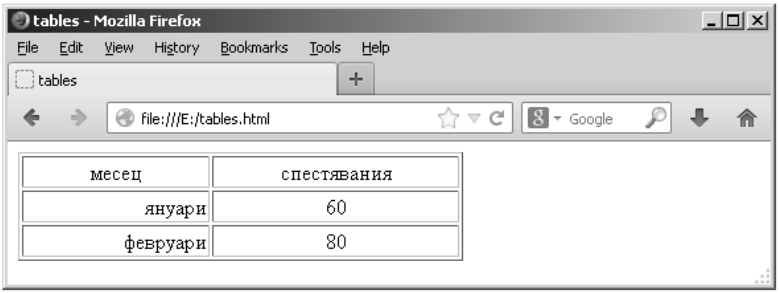
По подразбиране, съдържанието на клетките в таблицата се подравнява вляво (по хоризонтала) и в средата (по вертикала). Това форматиране може да бъде променяно като се използват различни стойности на атрибута `align` и `valign`, съответно:

- стойности за хоризонтално подравняване: *left* (по подразбиране), *middle*, *right*, *justified* и *char*;
- стойности за вертикално подравняване: *middle*, *top*, *bottom*, *based on line*.

Ето един пример за хоризантално подравняване на съдържание на клетките в таблица:

```
<table width=60% border=1 summary="">
  <tr>
      <td align=center>месец</td>
      <td align=center>спестявания</td>
  </tr>
  <tr>
      <td align="right">януари</td>
      <td align="center">60</td>
  </tr>
  <tr>
      <td align="right">февруари</td>
      <td align="center">80</td>
  </tr>
</table>
```

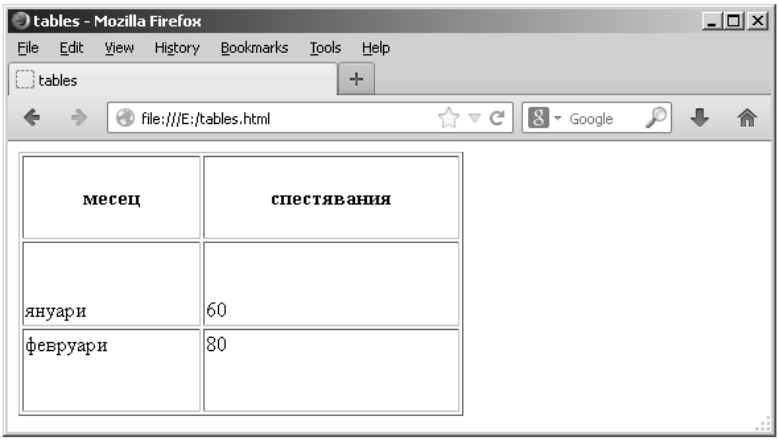
Браузърът би интерпретирал този код по следния начин:



Нека зададем различни стойности за вертикалното подравняване и да зададем ширина на таблицата, която да бъде 60 % от ширината на прозореца на браузъра, а височината да е 200 пиксела:

```
<table width=60% height=200 border=1 summary="">
 <tr>
     <th>месец</th>
     <th>спестявания</th>
 </tr>
 <tr>
     <td valign=bottom>януари</td>
     <td valign=bottom>60</td>
 </tr>
 <tr>
     <td valign=baseLine>февруари</td>
     <td valign=baseLine>80</td>
 </tr>
</table>
```

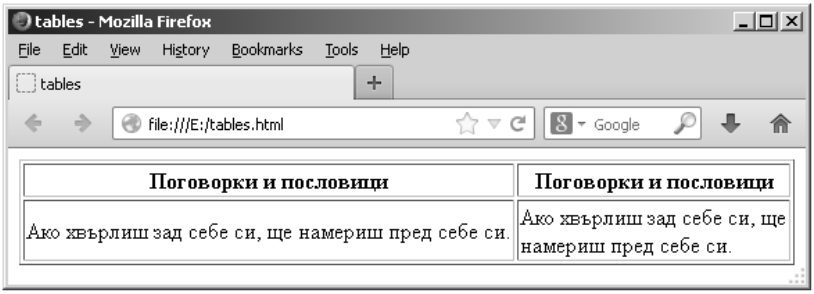
Браузърът би интерпретирал кода по следния начин:



Атрибутът `nowrap` се използва, когато е необходимо съдържанието в клетката да не се пренася на нов ред:

```
<table border="1">
  <tr>
     <th>Поговорки и пословици</th>
     <th>Поговорки и пословици</th>
  </tr>
  <tr>
     <td nowrap>Ако хвърлиш зад себе си, ще намериш пред себе си.</td>
     <td>Ако хвърлиш зад себе си, ще намериш пред себе си.</td>
  </tr>
</table>
```

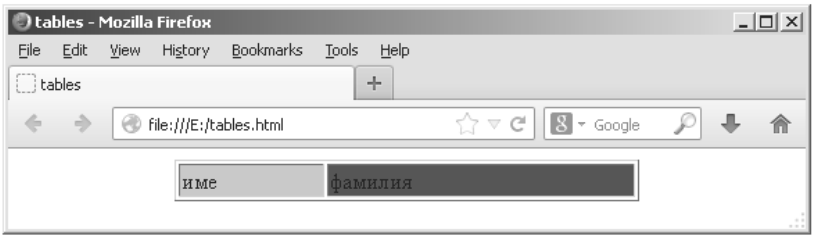
Браузърът би интерпретирал този код по следния начин:



Може да се променя цвета на фона на отделните клетки от таблицата:

```
<table align=center border=1 width="60%">
   <tr>
       <td bgcolor="#00FF00">име</td>
       <td bgcolor="#FF0000">фамилия</td>
   </tr>
</table>
```

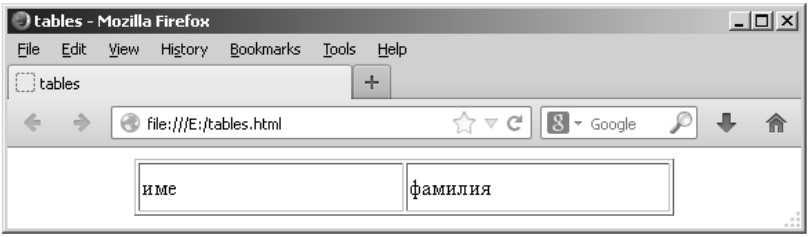
Браузърът би интерпретирал кода по следния начин:



За задаването на ширината и височината на клетките от таблицата се използват атрибутите `width` и `height`:

```
<table align=center border=1 width="70%">
   <tr>
       <td width="50%">име</td>
       <td height="35">фамилия</td>
   </tr>
</table>
```

Браузърът би интерпретирал кода по следния начин:



Тагът `<col>` задава определени свойства за всяка колона в елемента `<colgroup>`:

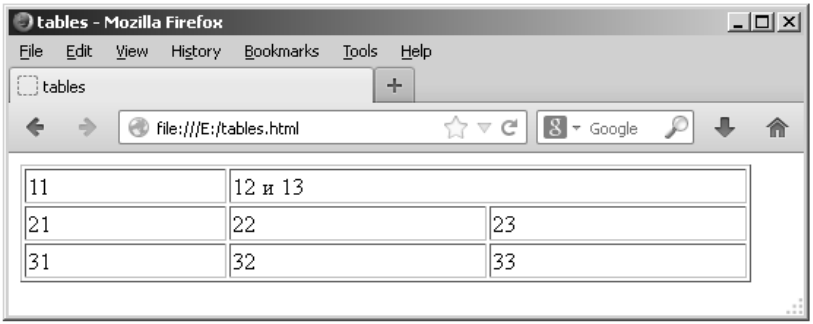
```
<table border="1">
  <colgroup>
     <col span="2" style="background-color:red">
     <col style="background-color:yellow">
  </colgroup>
  <tr>
     <th>ISBN</th>
     <th>Заглавие</th>
     <th>Цена</th>
  </tr>
  <tr>
     <td>3476896</td>
     <td>учебник по HTML</td>
     <td>53</td>
  </tr>
  <tr>
     <td>5869207</td>
     <td>учебник по CSS</td>
     <td>49</td>
  </tr>
</table>
```

Браузърът би интерпретирал кода по следния начин:

Тагът `<colgroup>` определя група от една или повече колони в таблицата за форматиране. Чрез тага `<colgroup>` могат да се прилагат стилове на цели колони, вместо стиловете да се повтарят за всяка клетка, за всеки ред. Тагът `<colgroup>` трябва да бъде част от елемента `<table>`, след всички тогове `<caption>` и преди `<thead>`, `<tbody>`, `<tfoot>` и `<tr>`. Тагът `<col>` се използва за прилагане на стилове на цели колони, вместо повтаряне на стилове за всяка клетка или за всеки ред. Атрибутът `span` определя броя на колоните, които трябва да обхваща. Атрибутът `colspan` определя броя на колоните, които трябва да бъдат обединени.

```
<table width=500 border=1>
  <tr>
     <td>11</td>
     <td colspan=2>12 и 13</td>
  </tr>
  <tr>
     <td>21</td>
     <td>22</td>
     <td>23</td>
  </tr>
  <tr>
     <td>31</td>
     <td>32</td>
     <td>33</td>
  </tr>
</table>
```

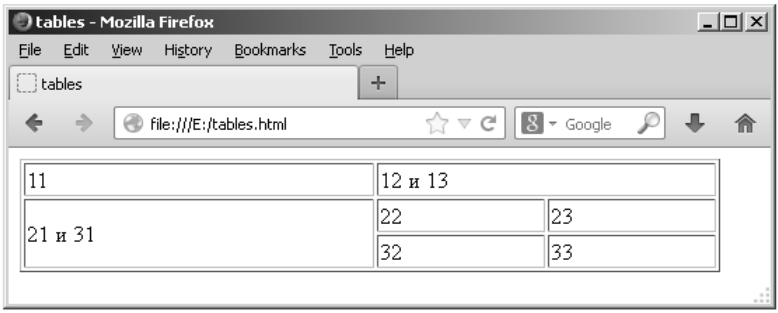
Браузърът би интерпретирал кода по следния начин:



Атрибутът `rowspan` определя броя на редовете, които трябва да бъдат обединени.

```
<table width=500 border=1>
  <tr>
     <td>11</td>
     <td colspan=2>12 и 13</td>
  </tr>
  <tr>
     <td rowspan=2>21 и 31</td>
     <td>22</td>
     <td>23</td>
  </tr>
  <tr>
     <td>32</td>
     <td>33</td>
  </tr>
</table>
```

Браузърът би интерпретирал кода по следния начин:



Атрибутът `headers` определя една или повече клетки със заглавия от таблицата. Например:

```
<table border="1">
  <tr>
     <th id="name">Име</th>
     <th id="email">Email</th>
     <th id="phone">телефон</th>
     <th id="addr">адрес</th>
  </tr>
  <tr>
     <td headers="name">Даниела</td>
     <td headers="email">mail@mail.com</td>
     <td headers="phone">+3592 XX XX XX</td>
     <td headers="addr">София – 1000, ул. Сердика</td>
  </tr>
</table>
```

Атрибутът `abbr` определя кратка версия на съдържанието на клетката. Например:

```
<table border="1">
  <tr>
     <th>Производител</th>
     <th>Модел</th>
  </tr>
  <tr>
     <td>Състезателни коли</td>
     <td abbr="volkswagen">Volkswagen Beetle GRC</td>
  </tr>
  <tr>
     <td>Състезателни коли</td>
     <td abbr="chevrolet">Chevrolet Corvette C7R</td>
  </tr>
</table>
```

И двата атрибута не се отразяват визуално върху клетките на таблицата. Атрибутите на елемента `table` са:

Атрибут	**Стойност**	**Описание**
align	left, center, right	Указва подравняването на таблицата според околния текст. Не се поддържа от HTML5.
bgcolor	rgb(x,x,x), #xxxxxx, colorname	Определя цвета на фона на таблицата. Не се поддържа от HTML5.
border	1, ""	Указва дали клетките на таблицата трябва да имат рамки или не.
cellpadding	pixels	Определя пространството между стената на клетката и съдържанието на клетката. Не се поддържа от HTML5.
cellspacing	pixels	Определя пространството между клетките. Не се поддържа от HTML5.
frame	void, above, below, hsides, lhs, rhs, vsides, box, border	Определя кои части на външните рамки трябва да се виждат. Не се поддържа от HTML5.
rules	None, groups, rows, cols, all	Определя кои части на вътрешните рамки трябва да се виждат. Не се поддържа от HTML5.
summary	text	Задава резюме на съдържанието на таблицата. Не се поддържа от HTML5.
width	pixels, %	Задава ширина на таблицата. Не се поддържа от HTML5.

От всички описани атрибути HTML5 поддържа само атрибута `border`, със стойност 1. Поради това се препоръчва използването на CSS.

Атрибути на елемента `<td>`:

Атрибут	Стойност	Описание
abbr	text	Задава съкратен вариант на съдържанието в клетката. Не се поддържа от HTML5.
align	left, right, center, justify, char	Подравнява съдържанието в клетка. Не се поддържа от HTML5.
bgcolor	rgb(x,x,x), #xxxxxx, colorname	Определя цвета на фона на клетката. Не се поддържа от HTML5.
colspan	number	Задава броя на колоните, които се обединяват в клетка.
headers	header_id	Няма визуален ефект в уеб браузърите, но може да се използва за определяне на заглавия на клетки.
height	pixels, %	Задава височината на клетката. Не се поддържа от HTML5.
nowrap	nowrap	Указва, че съдържанието вътре в клетката не трябва да се пренася. Не се поддържа от HTML5.
rowspan	number	Задава броя редове, които трябва да се обединят в клетка.
valign	top, middle, bottom, baseline	Вертикално подравнява съдържанието в клетката. Не се поддържа от HTML5.
width	pixels, %	Задава ширина на клетката. Не се поддържа от HTML5.

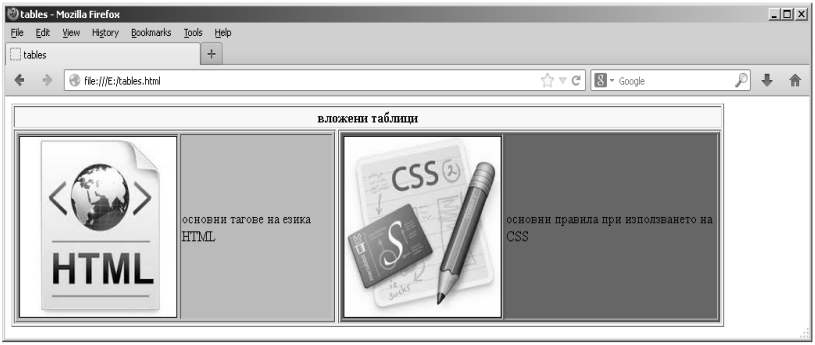
4.3. Вложени таблици

Таблиците могат да бъдат влагани една в друга. В следващия пример са показани 3 таблици – основната таблица в жълт цвят и вложени в нея 2 други таблици, съответно в син и червен цвят с текстове и изображения:

```
<table width=70% border=1 bgcolor=#FFFF99 summary="">
  <tr>
     <th colspan=2>вложени таблици</th>
  </tr>
  <tr>
     <td>
<table width=100% border=1 bgcolor=#66CCFF summary="">
       <tr>
         <td><img  src="html.jpg"  border="1"  width="200"  height="200"
alt=""></td>
```

```
          <td>основни тагове на езика HTML</td>
       </tr>
</table>
    </td>
    <td>
<table width=100% border=1 bgcolor=#FF3399 summary="">
       <tr>
          <td><img  src="css.jpg"  border="1"  width="200"  height="200"
alt=""></td>
          <td>основни правила при използването на CSS</td>
       </tr>
</table>
    </td>
  </tr>
</table>
```

Браузърът би интерпретирал този код по следния начин:



Глава 5. Формуляри – елементи и методи за изпращане на информация от формуляри

В тази глава се разглежда използването на различни елементи на формулярите, както и методите за изпращането им. Основната задача на всеки формуляр е да събира данни, които да бъдат препращани към определен сървър, с цел последваща обработка. Формулярите се състоят от полета за въвеждане (текстови полета), полета за избор и бутони. Елементите на формулярите са част от потребителския интерфейс. Съществуват различни елементи, които могат да се използват при създаването на формуляри. Всеки от тях е проектиран да събира определен тип данни от потребителя, като елементи за име и парола при регистрация и др. Попълненият формуляр се изпраща на сървър, където се обработват постъпилите данни от попълнения формуляр.

5.1. Създаване на формуляр и методи за изпращане на информация

Структурата на всеки формуляр започва с отварящ таг `<form>` и завършва със затварящ таг `</form>`. Цялото съдържание на формуляра е разположено между тези два тага.

```
<form>
     елементи на формуляра
</form>
```

Някои от задължителните атрибути на тага `<form>` са `name`, `action` и `method`. Атрибутът `name` задава името на формуляра. Атрибутът `action` определя къде да бъде изпратен формулярът с данни, когато се подава. Има два възможни начина за изпращане на данни от формуляр:

- чрез задаване на абсолютен URL – препращане към друг сайт (като action="http://www.example.com/example.htm");
- чрез задаване на относителен URL – указване на файла (като action="example.htm").

Още една подробност около изпращането на формулярите е указването на файла, който ще обработи информацията, и начинът

на изпращане на информацията – използват се съответно `action="име-на-обработващ-файл"` и подходящ метод за изпращане `method="get"` или `method="post"`.

- `get` метод – чрез него клиентът прави заявка за ресурс зададен чрез URL. Могат да се изпращат и ограничено количество данни, закодирани директно в самия URL (отделени чрез въпросителен знак).
- `post` метод – позволява на клиента да изпрати данни на сървъра. Тази заявка обикновено се генерира при изпращането на уеб формуляр, а данните могат да бъдат: текст, написан от потребителя във формуляра, файл на клиентския компютър и др.

Атрибутът `accept-charset` определя кодовата таблица, която трябва да се използва при изпращането на формуляра. Първите 128 символа на ISO-8859-1 са оригиналните ASCII символи (числата 0-9, главните и малките букви на английската азбука, както и някои специални знаци). Останалата част на ISO-8859-1 (кодове 160-255) съдържа символите, използвани в западноевропейските страни и някои често използвани специални символи. ISO-8859-5, Windows-1251 и UTF-8 са често използвани кодови таблици за латиница и кирилица. Ето един пример с явно задаване на кодировката на формуляра:

```
<form name="ff1" action="demo_form.asp" method="get" accept-
charset="ISO-8859-1">
     ……….
</form>
```

Елементите на формуляра, чрез които потребителят въвежда съответната информация, са следните:

Елемент	Описание
<input>	Определя елемент на формуляра
<textarea>	Определя многоредово текстово поле
<label>	Определя етикет на <input> елемент
<fieldset>	Групира свързани елеманти във формуляр
<legend>	Определя заглавие за <fieldset> елемент
<select>	Определя падащо списъчно меню
<optgroup>	Определя група от свързани елементи от падащото списъчно меню
<option>	Определя елемент от списъчно падащо меню
<button>	Определя бутон за кликане
<datalist> (new)	Задава списък с предварително дефинирани опции за input елемент
<output> (new)	Определя резултат от изчислението

5.2. Елемент <input>

Почти всички елементи на формулярите се въвеждат чрез тага `<input>`, който може да съдържа няколко атрибута, най-важните от които са атрибута `type` – определящ неговия вид, и атрибута `name`, задаващ името му.

Елементът `<input>` може да варира в зависимост от вида на използвания атрибут, например елементът `<input>` може да бъде от тип *text field, checkbox, password, radio button* или *submit button*.

Възможните стойности на атрибута `type` от тага `<input>`, задаващ типа на елемента, са:

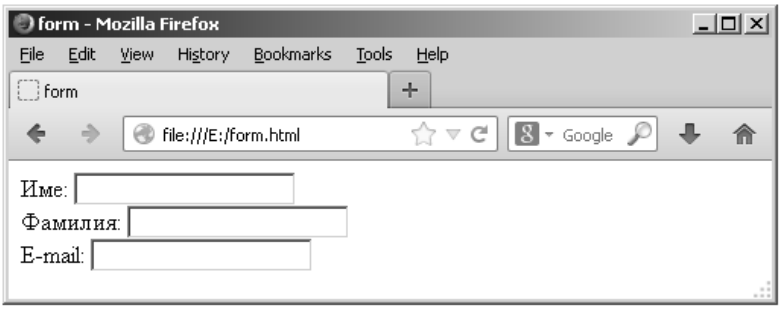
- `button` – резултатът е бутон с име, дефинирано от атрибута `name`="..." и стойност, дефинирана от атрибута `value` ="...";
- `checkbox` – резултатът е отметка, която може да има две стойности: маркирана и немаркирана. Изпраща се стойността на атрибута `value`;
- `image` – създава бутон от външно изображение;
- `file` – резултатът е бутон, чрез който потребителя може да избере файл от локалния компютър;
- `hidden` – невидимо поле, което се използва най-често за прехвърляне на информация между две сесии. Името на полето се задава с атрибута `name`, а стойността – с `value`;
- `password` – създава текстово поле, в което се въвежда парола. Разликата е, че въведената информация се визуализира като звездички;
- `radio` – резултатът е т.нар. радио-бутон. Могат да се групират чрез атрибута `name`. Това става, когато този атрибут е еднакъв за групата. В една група може да има само един маркиран радио-бутон;
- `reset` – резултатът е бутон, при натискането на който се изчиства цялата информация, въведена във формуляра. Ако не е определена стойност на атрибута `value`, стойността на бутона е `reset`;
- `submit` – резултатът е бутон, при натискането на който се изпраща формулярът. Ако не се дефинира атрибутът `value`, стойността на бутона е `submit`;
- `text` – резултатът е текстово поле от един ред, в което може да се въвежда произволна информация.

Дефинирането на едноредово поле за въвеждане на текст от потребителя се реализира чрез използването на тага `input`. Обикновено едноредовите текстови полета са предхождани от описате-

лен текст, който указва на потребителя какво да въведе в полето, както е показано:

```
<form>
   Име: <input type="text" name="name"> <br>
   Фамилия: <input type="text" name="family"> <br>
   E-mail: <input type="text" name="mail">
</form>
```

Браузърът би интерпретирал кода по следния начин:



Дължината на текстовото поле може да се определи с атрибута `size` (например `size`="30") на тага `input`. Ако не се зададе дължина на текстовото поле, по подразбиране дължината е 20 знака. Атрибутът `maxlenght` се използва за ограничаване на максималния брой символи, които потребителят може да въведе за дадено поле. Например `maxlenght`="30" ще ограничи броя на символите, които потребителят може да въведе в дадено поле до 30.

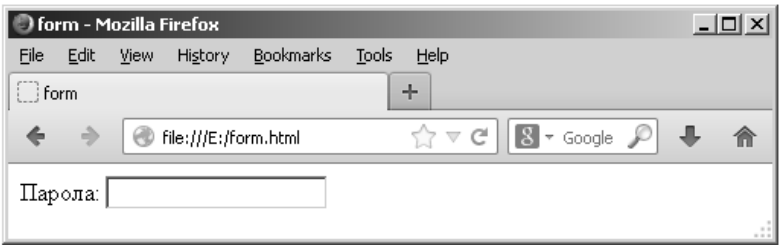
```
<form action="">
Лична информация:<br>
Име:
<input type="text" size="36" maxlength="30"><br>
Фамилия:
<input type="text" size="30"><br>
E-mail:
<input type="text" size="35">
</form>
```

HTML позволява да бъдат създавани два вида текстови полета – за обикновен текст (който разгледахме) и текстово поле за пароли. Основната разлика между двата вида е, че текстовите полета за пароли не показват директно въвежданите букви и цифри, а звездички или други знаци вместо тях. За създаването

на едноредово поле за въвеждане на парола използваме `password`, като стойност на атрибута `type`:

```
<form>
   Парола: <input type="password" name="pwd">
</form>
```

Браузърът би интерпретирал кода по следния начин:



Действителната парола не се криптира по никакъв начин при обработването на формуляра.

Радио бутоните дават възможност на потребителя да избере само един от определения брой възможности за избор. Радио бутоните се реализират с тага `input` и стойността `radio` на атрибута `type`:

```
<p> Обичам да карам: </p>
<form>
<input type="radio" name="vehicle" value="bike"> Велосипед <br>
<input type="radio" name="vehicle" value= "car"> Кола  <br>
<input type="radio" name="vehicle" value="airplane"> Самолет
</form>
```

Браузърът би интерпретирал този код по следния начин:

Следва да обърнем внимание, че атрибутът `name` включва една и съща стойност за всички опции, което прави свързани и трите избора. При обработването на формуляра се разпознава коя опция е избрана и стойността се предава посредством атрибута `value`.

Ако е необходимо радио бутоните да са предварително маркирани при първоначалното отваряне на страницата, може да се използва атрибутът `checked` в тага `input`.

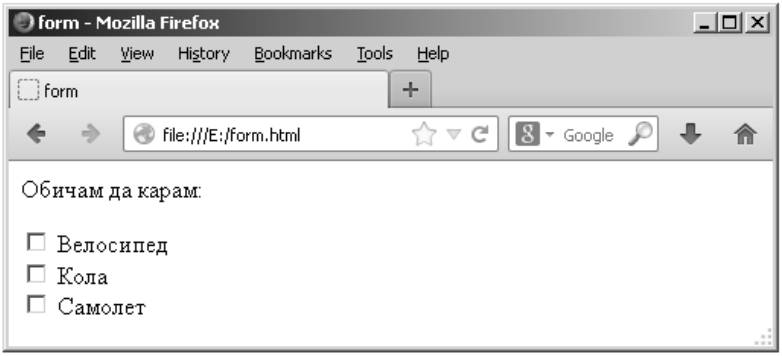
```
<form>
<input type="radio" name="vehicle" value="bike"> Велосипед <br>
<input type="radio" name="vehicle" value="car" checked> Кола  <br>
<input type="radio" name="vehicle" value="airplane"> Самолет
</form>
```

Браузърът ще интерпретирал кода по следния начин:

Полетата за отметка са аналогични на радио бутоните, тъй като и при тях потребителят не въвежда информация, а отбелязва избраната от него възможност. Използването на `checkbox` позволява на потребителя да избере нула или няколко възможни варианта от определен брой за избор.

```
<p> Обичам да карам: </p>
<form>
<input type="checkbox" name="vehicle" value="bike"> Велосипед <br>
<input type="checkbox" name="vehicle" value="car"> Кола <br>
<input type="checkbox" name="vehicle" value="airplane"> Самолет
</form>
```

Браузърът би интерпретирал кода по следния начин:



Както и при радио бутоните, стойността на атрибута `name` за всички опции трябва да бъде еднаква. Единствената информация, показвана в полето за отметка, дори при използване на атрибута `value`, е самото поле за отметка. Стойността на атрибута `value` се изпраща към уеб сървъра при подаването на формуляра, ако елементът е бил избран. Използваме атрибута `checked`, когато искаме определено поле да е маркирано при отваряне на страницата.

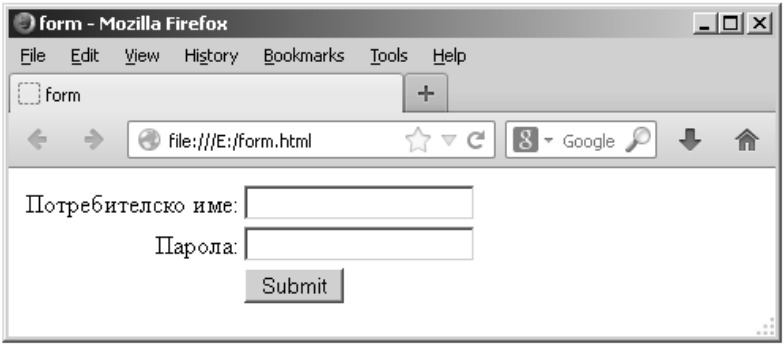
```
<p> Мога да карам: </p>
<form>
<input type="checkbox" name="vehicle" value="bike" checked="yes"> Ве-
лосипед <br>
<input type="checkbox" name="vehicle" value="car"> Кола <br>
<input type="checkbox" name="vehicle" value="airplane"> Самолет
</form>
```

Браузърът би интерпретирал кода по следния начин:

Бутонът `submit` се използва за изпращане на данни от формуляри към сървъра. Данните се изпращат към файла, зададен в атрибута `action` на формуляра:

```
<form name="input" action="html_form_action.asp" method="get">
<table summary="" border="0">
  <tr>
     <td align="right">Потребителско име:</td>
     <td><input type="text" name="user"></td>
  </tr>
  <tr>
     <td align="right">Парола: </td>
     <td><input type="password" name="pwd"></td>
  </tr>
  <tr>
     <td align="right"></td>
     <td><input type="submit" value="Submit"></td>
  </tr>
</table>
</form>
```

Браузърът би интерпретирал кода по следния начин:



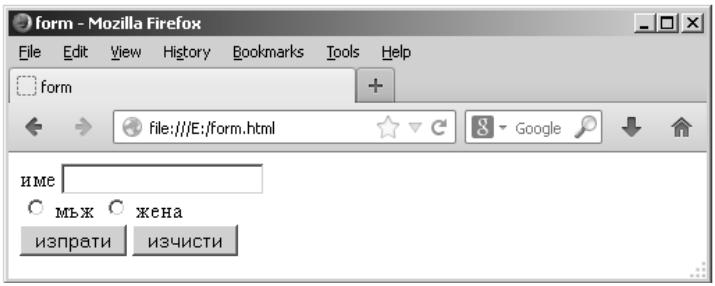
Бутоните от типа `reset` се използват за изчистване на цялата въведена информация във формуляра. Синтаксисът на този вид бутони е следният:

```
<input type ="reset" value ="reset message">
```

Чрез този запис се създава бутон за изчистване на всички полета във формата, където `value ="reset message"` е текстът, който ще се покаже върху бутона. Ето пример за бутон от типа `reset`:

```
<form>
име <input type ="text" name="Text1" size="20"><br>
<input type="radio" name="sex" value="bike"> мъж
<input type="radio" name="sex" value="car"> жена  <br>
<input type ="submit" value="изпрати" name="Button1">
<input type ="reset" value="изчисти" name="Button2">
</form>
```

Браузърът би интерпретирал кода по следния начин:

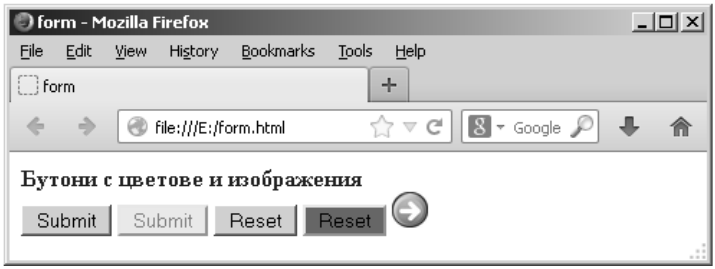


Бутони – цветове, изображения

Няколко различни начина за форматиране на бутони от типа `submit` и `reset` са показани в следния код:

```
<form>
<h3 style="color: #990000"> Бутони с цветове и изображения</h3>
<input type="submit" value="Submit" name="B1">
<input type="submit" value="Submit" name="B1a" style="color: #FF6600;
background: #CCFF66">
<input type="reset" value="Reset" name="B2">
<input   type="reset"   value="Reset"   name="B2a"   style="background:
#FF3399">
<input type="image" src="go.gif" alt="Submit" width="25" height="25">
</form>
```

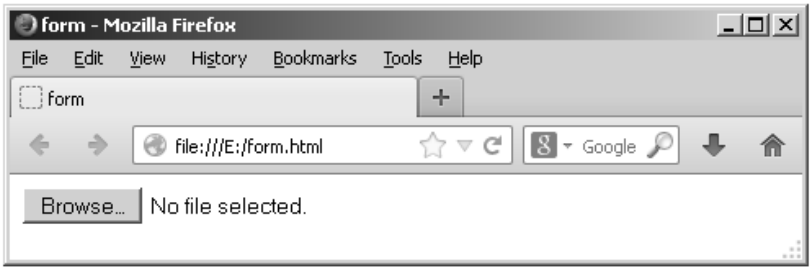
Браузърът би интерпретирал кода по следния начин:



За прехвърлянето на файлове може да се използва интерфейс, който позволява на потребителя да избере локален файл и да го качи на уеб сървър. Областта за избор на файл има две части – едната представлява празно текстово поле, а друга е бутон *Browse*, който отваря *Window explorer* на компютъра на потребителя. Това позволява бързо намиране на локалните файлове и автоматично попълване на пътя до файла в текстовото поле.

```
<form name="myWebForm" action="mailto:youremail@email.com"
method="post">
    <input type="file" name="uploadField" />
</form>
```

Браузърът би интерпретирал кода по следния начин:



Прехвърлянето на файлове в Интернет пространството е сложен процес и трябва да включва много нива на сигурност. HTML сам не може да гарантира безопасен и сигурен файлов трансфер, но може да предложи първа стъпка на защита. С помощта на `MAX_FILE_SIZE` скритото поле може да ограничи размера на файловете, които се прехвърлят. Това се постига като атрибута `name` се използва със стойност `MAX_FILE_SIZE`, а в атрибута `value` се задава максимален размер на файла:

```
<form name="myWebForm" action="mailto:youremail@email.com"
method="post">
    <input type="hidden" name="MAX_FILE_SIZE" value="500" />
    <input type="file" name="uploadField" />
</form>
```

Атрибути на елемента `input`:

Атрибут	Стойност	Описание
accept	audio/*, video/*, image/*, *MIME_type*	Определя типовете файлове, които сървърът приема (само за type="file")
align	Left, right, top, middle, bottom	Указва подравняването на изображението (само за type="image"). Не се поддържа в HTML5.
alt	*text*	Задава алтернативен текст на изображене (само за type="image")
autocomplete (new)	on, off	Указва дали <input> елемент трябва да има функция за автоматично довършване
autofocus (new)	autofocus	Указва дали <input> елемент трябва автоматично да вземе фокуса, след като се зареди страницата
checked	checked	Указва, че <input> елемент трябва да бъде предварително избран, когато страница се зареди (за type="checkbox" или type="radio")
disabled	disabled	Указва, че <input> елемент трябва да бъде забранен
form (new)	*form_id*	Задава един или повече формуляри за принадлежност на елемента <input>
formaction (new)	*URL*	Указва URL адреса на файла, който ще обработва формата, когато се изпраща (type="submit" и type="image")
formenctype (new)	application/x-www-form-urlencoded, multipart/form-data, text/plain	Уточнява как данните от формуляра трябва да бъдат кодирани, когато се изпратят на сървър (за type="submit" и type="image")
formmethod (new)	get, post	Определя метод (HTTP) за изпращане на данни към URL (за type="submit" и type="image")
formnovalidate (new)	formnovalidate	Определя, че елементът на формуляра не трябва да бъде валидиран, когато се изпраща
formtarget (new)	_blank, _self, _parent, _top, *framename*	Указва къде да се покаже информацията, която се получава след подаване на формуляра (за type="submit" и type="image")

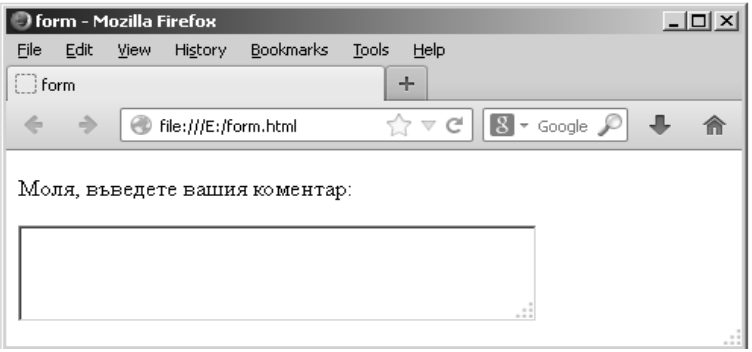
Атрибут	Стойност	Описание
height (new)	*pixels*	Задава височина на <input> елемент (само за type="image")
list (new)	*datalist_id*	Отнася се за елемент <datalist>, който съдържа предварително дефинирани опции за <input> елемент
max (new)	*number, date*	Указва максималната стойност за един елемент <input>
maxlength	*number*	Определя максималния брой знаци, разрешени в <input> елемент
min (new)	*number, date*	Указва минималната стойност за <input> елемент
multiple (new)	multiple	Показва, че потребителят може да въведе повече от една стойност в <input> елемент
name	*text*	Задава името на <input> елемент
pattern (new)	*regexp*	Задава регулярен израз, за който се проверява даден <input> елемент
placeholder (new)	*text*	Задава кратък запис, който описва очакваната стойност на <input> елемент
readonly	readonly	Указва, че полето е само за четене
required (new)	required	Указва, че полето трябва да бъде попълнено преди изпращането на формуляра
size	*number*	Задава ширина, в брой символи на <input> елемент
src	*URL*	Задава URL на изображение, използвано като бутон *Submit* (само за type="image")
step (new)	*number*	Определя допустимия брой интервали на поле за въвеждане
type	button, checkbox, color, date, datetime, datetime-local, email, file, hidden, image, month, number, password, radio, range, reset, search, submit, tel, text, time, url, week	Определя типа на <input> елемент
value	text	Определя стойността на даден <input> елемент

5.3. Елемент <textarea>

Елементът <textarea> е подобен на елемента за текстово поле с тази разлика, че тук редовете са повече от един. Многоредовите полета дават възможност за въвеждане на по-дълъг текст, напр. коментар или съобщение. Ако дължината на въведения текст е по-голяма от зададените параметри на визуалния елемент, то автоматично се добавя лента за превъртане. Многоредовото текстово поле се дефинира от двойката тагове <textarea> и </textarea>. Ширината на текстовото поле се определя в брой символи чрез атрибута cols, височината се определя в брой редове чрез атрибута rows, а атрибутът name задава името на полето:

```
<form>
    <p>Моля, въведете вашия коментар:</p>
    <textarea name="coment" rows="5" cols="40"> </textarea>
</form>
```

Браузърът би интерпретирал кода по следния начин:



В примера, текстовата област е настроена така, че да побира 5 реда текст, като всеки ред е с ширина 40 символа. Един пример за използване на различни размери на текстовото поле на формуляр е показан в следния код:

```
<form>
    <textarea rows="4" cols="20" name="usrtxt" wrap="hard">
Американски биолози за пръв път в света са създали изкуствено сърце от
препрограмирани стволови клетки на човек, съобщи сп. Nature
Communications.
    </textarea>
</form>
```

Браузърът би интерпретирал кода по следния начин:

Атрибути на елемента `textarea`:

Атрибут	Стойност	Описание
autofocus (new)	autofocus	Определя, че текстовото поле трябва да вземе фокуса, когато страницата се зареди
cols	number	Задава видимата ширина на текстовото поле в брой символи
disabled	disabled	Указва, че текстовото поле трябва да бъде забранено
form (new)	form_id	Задава един или повече формуляри, към които текстовото поле принадлежи
maxlength (new)	number	Определя максималния брой на знаците в текстовото поле
name	text	Задава име на текстовото поле
placeholder (new)	text	Задава кратък запис, който описва очакваната стойност на текстовото поле
readonly	readonly	Указва, че текстовото поле е само за четене
required (new)	required	Уточнява, че текстовото поле трябва да бъде попълнено
rows	number	Задава височината на текстовото поле в брой редове
wrap (new)	hard, soft	Определя как текста в текстовото поле да бъде предаван, когато се изпраща

5.4. Елементи <select> и <option>

Падащите менюта позволяват избор на един или няколкото предложени варианта, подобно на кутийките с отметки. Единствената разлика е във външния вид – падащото меню показва само една от възможностите, а другите стават видими, едва когато се клик-

не върху менюто. Менюта от вида падащи списъци се създават с тага `<select>`, като отделните елементи на това меню се задават от тага `<option>`.

Атрибути на `<select>`:

Атрибут	Стойност	Описание
autofocus (new)	autofocus	Указва, че списъкът от падащото меню трябва автоматично да вземе фокуса, при зареждане на страницата
disabled	disabled	Указва, че списъкът от падащото меню трябва да бъде забранен
form (new)	form_id	Определя принадлежност на полето към един или повече формуляри
multiple	multiple	Указва, че повече от една опция може да бъде избрана наведнъж
name	name	Определя име на списъка от падащото меню
required (new)	required	Указва, че потребителят трябва да избере стойност, преди да изпрати формуляра
size	number	Определя броя на видимите опции в списък с падащото меню

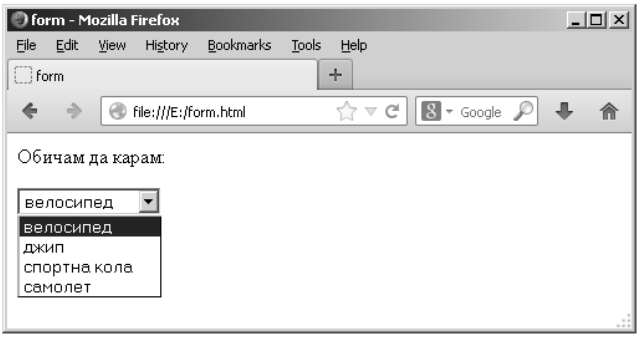
Атрибути на `<option>`:

Атрибут	Стойност	Описание
disabled	disabled	Забранява дадения елемент
label	text	Дефинира заглавие на елемента, което предефинира текста след тага (който, ако не се постави този атрибут, служи за заглавие)
selected	selected	Маркира дадения елемент
value	text	Дефинира стойността на елемента, която да се изпрати на сървъра при изпращане на формата

Ето един пример за формиране на падащо меню, с помощта на елементите `select` и `option`:

```
<p> Обичам да карам: </p>
<form>
   <select name="vehicle">
      <option value="bike">велосипед</option>
      <option value="Jeep">джип</option>
      <option value="sport_car">спортна кола</option>
      <option value="airplane">самолет</option>
   </select>
</form>
```

Браузърът би интерпретирал кода по следния начин:



Избор на повече от един елемент се реализара чрез натиснати клавиши *Shift* или *Ctrl* и съответен избор на елементи от менюто.

Тагът `<optgroup>` се използва за групиране на свързани елементи в падащи списъци. При наличие на дълъг списък от опции, групирането на сродни опции ще улесни избора на потребителя. Възможните атрибути на тага optgroup са `disabled` и `label`. Атрибутът `label` определя етикет за избор на група.

```
<select>
  <optgroup label="French Cars">
    <option value="citroen">Citroen</option>
    <option value="renault">Renault</option>
  </optgroup>
  <optgroup label="Italian Cars">
    <option value="ferrari">Ferrari</option>
    <option value="lamborghini ">Lamborghini </option>
  </optgroup>
</select>
```

Браузърът би интерпретирал кода по следния начин:

Атрибутът `disabled` забранява избор на групирани сродни елементи.

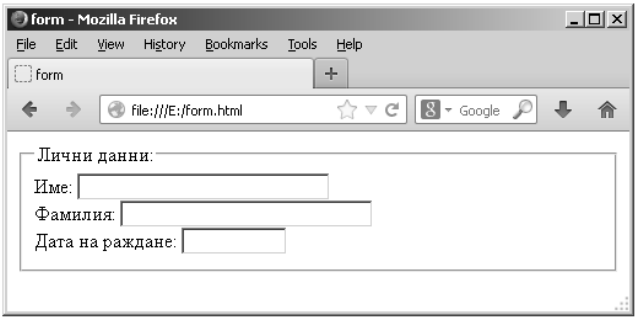
```
<select>
  <optgroup label="French Cars">
    <option value="citroen">Citroen</option>
    <option value="renault">Renault</option>
  </optgroup>

  <optgroup label="Italian Cars" disabled>
    <option value="ferrari">Ferrari</option>
    <option value="lamborghini">Lamborghini</option>
  </optgroup>
</select>
```

Браузърът би интерпретирал кода по следния начин:

Тагът `<fieldset>` се използва за групиране на свързани елементи във формуляр. Чрез тага `<fieldset>` е възможно да се очертае поле около свързаните елементи. Във връзка с този таг се използва и тага `<legend>`, който определя заглавие на елемента `<fieldset>`, както е показано:

```
<form>
  <fieldset>
    <legend>Лични данни:</legend>
    Име: <input type="text" size="30"> <br>
    Фамилия: <input type="text" size="30"> <br>
    Дата на раждане: <input type="text" size="10">
 </fieldset>
</form>
```

Браузърът би интерпретирал кода по следния начин:



Изпращане на информация от формуляр към електронна поща

Един лесен начин за изпращане на данните от попълнения формуляр е използването на електронната поща. Въведените данни се изпращат не към сървър, където те да бъдат обработени, а до конкретен получател с определен имейл адрес:

```
<h3>Изпращане на e-mail до someone@example.com:</h3>
<form action="MAILTO:someone@example.com" method="post"
enctype="text/plain">
Име: <input type="text" name="name" value="your name"><br>
E-mail: <input type="text" name="mail" value="your email"><br>
Коментар: <input type="text" name="comment" value="your comment"
size="50"><br><br>
   <input type="submit" value="Send">
   <input type="reset" value="Reset">
</form>
```

Браузърът би интерпретирал кода по следния начин:

Глава 6. Използване на рамки и мултимедия. Нови елементи в HTML5. Различия между HTML и XHTML

В тази глава са разгледани особеностите на използването на вградените рамки, като начин за показване съдържанието на две или повече уеб страници едновременно в един прозорец на браузъра. Описани са възможностите за вграждане на различни мултимедийни формати в уеб документи, както и използването на вградените рамки за визуализиране на мултимедийни елементи. Направено е сравнение между синтактичните особености на версиите HTML 4.01 и HTML5. Резюмирани са основните синтактични различия на езиците HTML и XHTML.

6.1. Използване на рамки

Възможно е визуализирането на съдържанието на 2 или повече уеб страници еднорвеменно чрез използването на т. нар. „вградени рамки“. Тя позволява съдържанието на дадена уеб страница да бъде вградено в съдържанието на друга уеб страница. Тагът, определящ вградена рамка, е `<iframe>` и има следния синтаксис:

```
<iframe src="URL"> </ iframe>
```

Атрибутът `src` на елемента `<iframe>` посочва URL адреса на документа, който ще се визуализира в рамката. Атрибутът `align` прави възможно подравняването на вградената рамка в документа, като възможните стойности са *top, middle* или *bottom*. За да позволим на текстовия поток да тече около рамката, могат да се използват сойностите *left* или *right* за атрибута `align`. Рамката се премества към левия или десния край на текстовия поток съответно, а останалото съдържание на документа ще заобикаля рамката. Стойността *center* поставя рамката в средата на документа, а текстът преминава над и под нея. Един пример за използване на вградена рамка:

```
<iframe src="iframe_txt.html" name="iframe_a" width="35%" height="75%"
align="right"></iframe>
<p><a href="http://www.w3schools.com/html/html_iframe.asp"
target="iframe_a">отваряне на сайта w3schools във вградена рамка с име
“iframe_a”</a></p>
<p><b>Note:</b> Because the target of the link matches the name of the
iframe, the link will open in the iframe.</p>
```

Браузърът би интерпретирал кода по следния начин:

Атрибути на елемента `iframe`:

Атрибут	Стойност	Описание
align	left,right, top, middle, bottom	Указва подравняването на <iframe> спрямо околните елементи. Отхвърлена в HTML 4.01. Не се поддържа в HTML5.
frameborder	1, 0	Задава дали да се покаже на границата около <iframe>. Не се поддържа в HTML5.
height	pixels	Задава височината на <iframe>
longdesc	URL	Задава страница, която съдържа описание на съдържанието на <iframe>. Не се поддържа в HTML5.
marginheight	pixels	Задава горната и долната граница на съдържанието на <iframe>. Не се поддържа в HTML5.
marginwidth	pixels	Задава лявото и дясното поле от съдържанието на <iframe>. Не се поддържа в HTML5.

Атрибут	Стойност	Описание
name	text	Задава име на <iframe>
sandbox (new)	"", allow-forms, allow-same-origin, allow-scripts, allow-top-navigation	Позволява набор от допълнителни ограничения за съдържанието в <iframe>
scrolling	yes, no, auto	Определя дали да се показват плъзгачите в <iframe>. Не се поддържа в HTML5.
seamless (new)	seamless	Указва, че <iframe> трябва да изглежда така, като ли че е част от документа
src	URL	Определя адреса на документа, който трябва да се вгради в <iframe>
srcdoc (new)	HTML_code	Определя HTML съдържанието на страницата, което да се покаже в <iframe>
width	pixels	Задава ширината на <iframe>

Тагът <iframe> поддържа също и глобални атрибути, описани в Приложение 1.

6.2. Използване на мултимедия

Най-лесният начин за предоставяне на мултимедийни елементи е с помощта на обикновена хипервръзка, която позволява изтеглянето им:

```
<a href="music.mp3"> изтегляне песен </a>
```

Това понякога е за предпочитане пред вграждането на файл в дадена страница. След като файлът бъде изтеглен, той ще бъде асоцииран с подходяща програма на компютъра на потребителя за възпроизваждане. Вграждането на файл означава, че мултимедията се възпроизвежда в средата на браузъра. За да стане това е необходимо браузърът да използва съответен плъгин (Plug-ins). Плъгинът е софтуерен компонент, който добавя специфични функции към съществуващото софтуерно приложение. Много от използваните плъгини идват с новите браузъри. Поради това новите версии на браузърите поддържат различни мултимедийни формати. В HTML за вграждането на мултимедийни елементи се използват следните тагове:

Таг	Описание
<embed>	Определя вграден обект
<object>	Определя вграден обект
<param>	Дефинира параметър за даден обект
<audio> (new)	Дефинира звуково съдържание
<video> (new)	Дефинира видео или филм
<source> (new)	Дефинира множество медийни ресурси за мултимедийните елементи <video> и <audio>
<track> (new)	Определя текстов списък за елементите <video> и <audio>

Ролята на елемента <object> е да подпомага HTML документа при намирането на подходящи плъгини за конкретното мултимедийното съдържание. Примери за добре известни плъгини са Adobe Flash Player и QuickTime. Плъгините могат да бъдат добавени към уеб страниците чрез <object> или <embed>. Повечето плъгини позволяват на потребителя контрол върху настройките за звука – назад, напред, пауза, стоп и възпроизвеждане.

Атрибути на елемента embed:

атрибут	стойност	описание
height (new)	*pixels*	Задава височината на вграденото съдържание
src (new)	*URL*	Определя адреса на външния файл за вграждане
type (new)	*MIME_type*	Указва MIME типа на вграденото съдържание
width (new)	*pixels*	Задава ширината на вграденото съдържание

Тагът <embed> поддържа също и глобални атрибути.

Атрибути на елемента object:

атрибут	стойност	описание
align	top, bottom, middle, left, right	Указва подравняването на елемента <object> според околните елементи. Отхвърлена в HTML 4.01. Не се поддържа в HTML5.
archive	*URL*	Разделени с интервал списъци на URL към архивите. Архивите съдържат ресурси, свързани с обекта. Не се поддържа в HTML5.
border	*pixels*	Задава дебелина на линията около <object>. Отхвърлена в HTML 4.01.Не се поддържа в HTML5.
classid	*class_ID*	Определя стойността на class ID, както е зададена в системния регистър на Windows или URL. Не се поддържа в HTML5.
codebase	*URL*	Определя къде да се намери кодът за обекта. Не се поддържа в HTML5.
codetype	*MIME_type*	Определя вида медия на кода, посочен от Classid атрибута. Не се поддържа в HTML5.

атрибут	стойност	описание
data	*URL*	Указва URL адреса на ресурса, който да бъде използван от обекта.
declare	declare	Определя, че обектът трябва да бъде само деклариран, не създаден или инсталиран, докато това не е необходимо. Не се поддържа в HTML5.
form (new)	*form_id*	Определя един или повече формуляри, към които обектът принадлежи
height	*pixels*	Задава височината на обекта
hspace	*pixels*	Задава празното поле от ляво и дясно на обекта. Отхвърлена в HTML 4.01. Не се поддържа в HTML5.
name	*name*	Задава име за обекта
standby	*text*	Определя текста, който да се покаже, докато обектът се зарежда. Не се поддържа в HTML5.
type	*MIME_type*	Указва типа MIME на данните, посочени в атрибута.
usemap	*#mapname*	Указва името на страна на клиента на изображението карта, която да бъде използвана с обекта
vspace	*pixels*	Задава празно пространство отгоре и отдолу на даден обект. Отхвърлена в HTML 4.01. Не се поддържа в HTML5.
width	*pixels*	Задава ширината на обекта

Тагът `<object>` поддържа също и глобални атрибути (Приложение 7).

В HTML могат да се използват различни видео формати:

Формат	Файл	Описание
AVI	.avi	AVI (*Audio Video Interleave*) е разработен от Microsoft. AVI се поддържа от всички компютри, работещи под Windows, както и от популярните уеб браузъри. Не винаги е възможно да се възпроизведе на компютри с различни от Windows операционни сестеми.
WMV	.wmv	WMV (*Windows Media Video*) е разработен от Microsoft. WMV е общ формат в Интернет, но това не може да се възпроизвежда на различни от Windows операционни сестеми, без допълнително инсталирани компоненти. Някои по-нови версии на WMV не могат да се възпроизвеждат на операционни сестеми, различни от Windows, тъй като няма съвместим плеър.
MPEG	.mpg .mpeg	MPEG (*Moving Pictures Expert Group*) е най-популярният формат в Интернет. Той се поддържа от различни платформи и от всички основни браузъри.
QuickTime	.mov	QuickTime е разработен от Apple. QuickTime е общ формат в Интернет, но QuickTime филми не могат да бъдат възпроизведени на компютър под Windows без допълнително инсталирани компоненти.

Формат	Файл	Описание
RealVideo	.rm .ram	RealVideo е разработен от Real Media. RealVideo позволява стрийминг на видео (онлайн видео, Internet TV) с ниски честотни ленти и намалено качеството на видеото.
Flash	.swf .flv	Flash е разработен от Macromedia. Flash изисква допълнителен компонент за възпроизвеждане.
MP4	.mp4	Mpeg-4 (MP4) е новият формат за Интернет. YouTube препоръчва използването на MP4. YouTube приема множество формати, а след това ги превръща в .flv или .mp4 за разпространение.

HTML може да използва и различни формати за звук:

Формат	Файл	Описание
MIDI	.mid .midi	MIDI (*Musical Instrument Digital Interface*) е формат за електронни музикални устройства, като синтезатори и PC звукови карти. MIDI файловете не съдържат звук, а цифрови музикални инструкции (ноти), които могат да бъдат възпроизвеждани от електроника (като звукова карта на компютър). Тъй като MIDI файловете съдържат само инструкции, те са изключително малки по размер. Например, файл с размер от 23K, може да се възпроизвежда почти пет минути. MIDI се поддържа от много софтуерни системи/платформи. Форматът MIDI се поддържа и от всички популярни Интернет браузъри.
MP3	.mp3	MP3 файловете са действително звуковата част от MPEG файлове. MPEG първоначално е бил разработен за видео от Moving Pictures Experts Group. MP3 е най-популярният формат за музика. Системата за кодиране съчетава добра компресия (малки файлове) с високо качество.
RealAudio	.rm .ram	RealAudio е разработен Real Media. Този формат позволява стрийминг на аудио (онлайн музика, Интернет радио) с ниски честотни ленти и намалено качество на звука.
WAV	.wav	WAVE (по-известен като WAV) е разработен от IBM и Microsoft. Форматът WAV е съвместим с Windows, Macintosh и Linux операционни системи.
WMA	.wma	WMA (*Windows Media Audio*) може да се сравнява по качество с MP3 и е съвместим с повечето плеъри, с изключение на IPOD. WMA файлове могат да бъдат предоставени като непрекъснат поток от данни, което го прави практичен в предоставянето на Интернет радио или музика.

Вграждането на звук и възпроизвеждането му чрез QuickTime плеър има следния синтаксис:

```
<object width="420" height="360"
classid="clsid:02BF25D5-8C17-4B23-BC80-D3488ABDDC6B"
codebase="http://www.apple.com/qtactivex/qtplugin.cab">
   <param name="src" value="liar.wav">
   <param name="controller" value="true">
</object>
```

Атрибутът `classid` предоставя информация, която браузърът използва, за да се разбере как трябва да бъде изпълнен обектът. Това гарантира, че браузърът ще го възпроизведе коректно. Този атрибут има стойност URL на документа или вътрешна справка под формата на `"classid:object-id"`, както е показано в примера погоре (`"clsid:02BF25D5-8C17-4B23-BC80-D3488ABDDC6B"`). Атрибутът `codebase` задава URL адреса, определен в `classid` атрибута, и заменя всеки URL, разположен в заглавната част на документа.

В показания пример `"codebase= http://www.apple.com/qtactivex/qtplugin.cab"` е зададен пълният път до ресурса.

Вграждането на видео и възпроизвеждането му чрез QuickTime плеър има следният синтаксис:

```
<object width="420" height="360"
classid="clsid:02BF25D5-8C17-4B23-BC80-D3488ABDDC6B"
codebase="http://www.apple.com/qtactivex/qtplugin.cab">
   <param name="src" value="movie.mp4">
   <param name="controller" value="true">
</object>
```

За вграждането на SWF (Shockwave Flash или Small Web File) файл и възпроизвеждането му чрез Adobe Flash плеър, може да се използва следния синтаксис.

```
<object width="400" height="40"
classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"
codebase="http://fpdownload.macromedia.com/
pub/shockwave/cabs/flash/swflash.cab#version=8,0,0,0">
   <param name="SRC" value="bookmark.swf">
       <embed src="bookmark.swf" width="400" height="40">
       </embed>
</object>
```

За възпроизвеждане на видео чрез Windows Media плеър, синтаксисът е следния:

```
<object width="100%" height="100%"
type="video/x-ms-asf" url="3d.wmv" data="3d.wmv"
classid="CLSID:6BF52A52-394A-11d3-B153-00C04F79FAA6">
   <param name="url" value="3d.wmv">
   <param name="filename" value="3d.wmv">
   <param name="autostart" value="1">
   <param name="uiMode" value="full">
   <param name="autosize" value="1">
   <param name="playcount" value="1">
```

```
      <embed type="application/x-mplayer2" src="3d.wmv" width="100%"
height="100%" autostart="true" showcontrols="true"
pluginspage="http://www.microsoft.com/Windows/MediaPlayer/">
      </embed>
</object>
```

Най-общо казано, възпроизвеждането на аудио и видео в HTML не е лесно. Съществуват проблеми, свързани с различните браузъри, които поддържат различни формати. Ако даден браузър не поддържа файловия формат, няма да се възпроизведе без плъгин. Ако плъгинът не е инсталиран на компютъра на потребителя, аудиото няма да се възпроизведе. Ако конвертирате файла в друг формат, мултимедията (аудио или видео) отново няма да се възпроизведе. Поради тези проблеми, в HTML5 се добавят много нови синтактически характеристики, част от които са преназначени да направят лесно вграждането и манипулирането на мултимедийното съдържание, без да използват допълнителни плъгини.

Използването на елементите за `<audio>` и `<video>` в HTML5 определят аудио и видео съдържанието и работят във всички съвременни браузъри. Синтаксисът и за двата елемента е доста опростен и има вида:

```
<audio controls>
    <source src="horse.mp3" type="audio/mpeg">
    <source src="horse.ogg" type="audio/ogg">
Your browser does not support this audio format.
</audio>
```

```
<video width="320" height="240" controls>
    <source src="movie.mp4" type="video/mp4">
    <source src="movie.ogg" type="video/ogg">
    <source src="movie.webm" type="video/webm">
Your browser does not support the video tag.
</video>
```

Най-доброто решение за вграждане и възпроизвеждане на видео е да се използва комбинацията от HTML5 + `<video>` елемент + `<embed>` елемент:

```
<video width="320" height="240" controls>
    <source src="movie.mp4" type="video/mp4">
    <source src="movie.ogg" type="video/ogg">
    <source src="movie.webm" type="video/webm">
       <object data="movie.mp4" width="320" height="240">
```

```
        <embed     src="movie.swf"     width="320"     height="240">
        </embed>
      </object>
</video>
```

Атрибути на елемента `audio`:

Атрибут	**Стойност**	**Описание**
autoplay (new)	autoplay	Указва, че звукът ще започне веднага след като се зареди
controls (new)	controls	Указва, че аудио контролите се визуализират (Play/Pause бутони и т.н.)
loop (new)	loop	Указва, че звукът ще се повтаря всеки път, след като свърши
muted (new)	muted	Указва, че звукът ще бъде заглушен
preload (new)	auto, metadata, none	Указва дали и как аудиото се зарежда, когато страницата се зарежда
src (new)	*URL*	Указва URL адреса на аудио файла

Тагът `<audio>` поддържа глобални атрибути и атрибути за събития, описани в Приложения 1 и 7.

Атрибути на елемента `video`:

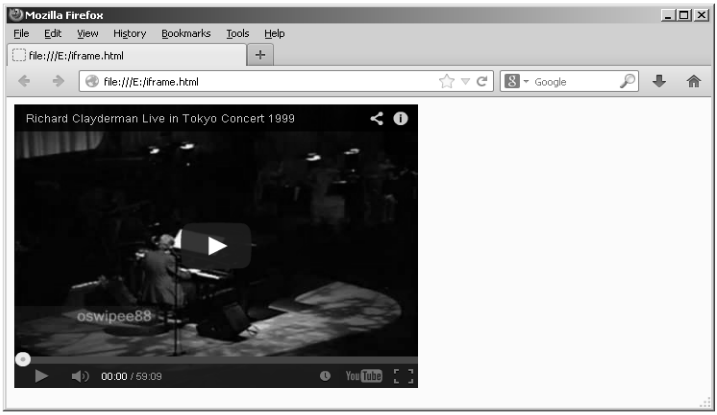
Атрибут	**Стойност**	**Описание**
autoplay (new)	autoplay	Определя дали видеото ще се стартира веднага, след като се зареди
controls (new)	controls	Определя дали да има бутони за управление на възпроизвеждането
height (new)	*pixels*	Определя височината на видео плеъра
loop (new)	loop	Определя дали че видеото ще се повтаря всеки път, след като свърши
muted (new)	muted	Определя заглушаване на звука на видеото
poster (new)	*URL*	Определя показване на изображение, докато се зарежда видеото или докато потребителя не натисне бутона play
preload (new)	auto, metadata, none	Указва дали и как видеото се зарежда, когато страницата се зарежда.
src (new)	*URL*	Задава URL на видео файла
width (new)	*pixels*	Определя ширината на видео плеъра

Тагът `<video>` също поддържа глобални атрибути и атрибути за събития, описани в Приложения 1 и 7.

Въпреки всички тези тагове и атрибути, може би най-лесният начин да се възпроизведе видео в HTML документ е използването на сървъри за видео споделяне като *VBOX7* или *YouTube*. След като качим видеото на някой мултимедиен сървър, вграждането и възпроизвеждането става лесно чрез използването на вградените рамки и има следния систаксис:

```
<iframe width="420" height="315"
src="http://www.youtube.com/embed/QUGGX0-fP8A" frameborder="0"
allowfullscreen></iframe>
```

В браузър, кодът ще се визуализира по следния начин:



6.3. Движещ се текст чрез <marquee>

Посредством <marquee> може да създадем движещ се хоризонтално или вертикално текст в уеб страницата. Движението може да се управлява с помощта на рзлични атрибути. Този елемент не е част от нито една HTML спецификация, но повечето съвременни браузъри го поддържат.

Атрибути на елемента marquee:

Атрибут	Описание
align	Определя вертикалното или хоризонталното подравняване.
behavior	Определя как текстът ще се движи по екрана.
bgcolor	Определя цвета на фона.
dir	Определя насочеността на текст, без предварително определена посока.
height	Определя височината.
hspace	Определя хоризонталното пространството около елемента.
loop	Колко пъти на текстът трябва да се превърти.
scrollamount	Определя размера в пиксели на частта от текста, която трябва да се придвижи при всяко движение.
scrolldelay	Определя скоростта на движение на тескта. Може да има стойност като 10 и т.н.
vspace	Определя вертикалното пространството около елемента.

Примерен код за движещ се текст (отдясно наляво):

```
<marquee>
движещ се текст от дясно на ляво </marquee>
```

Пример за текст, който ще се движи отляво надясно:

```
<marquee direction="right">
текст, който ще се движи от ляво на дясно </marquee>
```

Пример за текст, който ще се движи отдолу нагоре:

```
<marquee direction="up">
текст, който ще се движи отдолу нагоре </marquee>
```

6.4. Нови елементи в HTML5

HTML 4.01 става стандарт през 1999 г. и от неговото приемане до сега има елементи, които са остарели, или елементи, които не се използват по начина, по който са били предназначени. Всички тези промени са отразени в HTML5. HTML5 е следващата голяма ревизия на HTML стандарта, която ще замени HTML 4.01, XHTML 1.0 и XHTML 1.1. HTML5 е стандарт за структуриране и представяне на съдържанието за World Wide Web. HTML5 включва нови елементи за рисуване на графики, добавяне на мултимедийно съдържание, подобряване на структурата на страницата, по-добро управление, геолокация и т.н.

По отношение на синтаксиса HTML5 дава много гъвкавост и поддържа тагове, написани с главни букви; кавичките не са задължителни за атрибутите; стойностите на атрибутите не са задължителни; затварянето на празни елементи не е задължително.

Използва се прост синтаксис за уточняване на кодирането на знаците, както следва:

```
<meta charset="UTF-8">
```

HTML 5 премахва допълнителната информация при вмъкването на скрипт и синтаксисът се редуцира до следния запис:

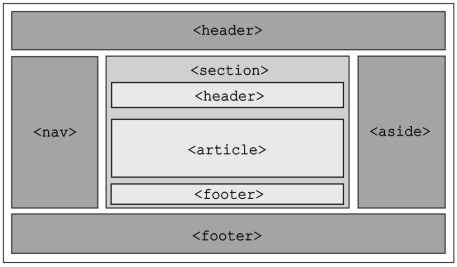
```
<script src="scriptfile.js"></script>
```

В HTML 5 извикването на външен CSS файл има също редуциран синтаксис:

```
<link rel="stylesheet" href="stylefile.css">
```

Има нови елементи в HTML5, които позволяват по-добро структуриране на уеб страниците чрез използване на следните тагове:

Таг	Описание
<article>	Определя независимо и самостоятелно съдържание.
<aside>	Дефинира съдържание отделно от съдържанието на страницата.
<bdi>	Определя част от текст, който се показва в обратна посока.
<command>	Определя команден бутон, който може да бъде извикан от потребителя.
<details>	Определя допълнителни детайли, които потребителят може да види или да скрие.
<dialog>	Определя диалогов прозорец или прозорец.
<summary>	Определя видимо заглавие за <details>.
<figure>	Задава автономно съдържание, като илюстрация, диаграма, снимка, и т.н.
<figcaption>	Дефинира надпис за елемент <figure>.
<footer>	Определя долен колонтитул за документ или раздел.
<header>	Определя горен колонтитул за документ или раздел.
<mark>	Дефинира маркиран/осветен текст.
<meter>	Дефинира скаларно измерване в известен диапазон (progress bar).
<nav>	Дефинира навигационни връзки.
<progress>	Представя процеса на зъвършване на задача.
<ruby>	Определя йероглиф (за източно-азиатските азбуки).
<rt>	Дефинира обяснение за йероглиф (за източно-азиатските азбуки).
<rp>	Определя какви да се покаже в браузъра, когато не се поддържат йероглифи.
<section>	Дефинира секция в документ.
<time>	Дефинира дата/час.
<wbr>	Дефинира възможност за прекъсване на линия.



Фиг. 6.1. Структуриране чрез средствата на HTML5

По-доброто структуриране на документа може да се реализира чрез използване на новите елементи в HTML5, както е показано на фиг. 6.1.

Елементът <article> определя независимо и самостоятелно съдържание. В повечето случаи, този елемент има заглавие и понякога долен колонтитул. Ето и пример за стилизиране на елемента <article>:

```
<article>
    <h1>Ябълка</h1>
    <p><b>Ябълката</b> е дърво от рода <i>Malus</i>, принадлежащ към
семейство Розоцветни (Rosaceae), и е едно от най-широко култивираните
дървета....</p>
</article>
```

Тагът `<article>` се поддържа от Internet Explorer 9+, Firefox, Opera, Chrome и Safari.

Където е подходящо, елементът `<section>` може да съдържа `<article>` елемент. Каква е разликата между `<article>` и `<section>` в HTML5? Елементът `<article>` е специален вид `<section>`. Той има по-специфично семантично значение от `<section>` в смисъл, че е независим и самостоятелен блок от свързаното съдържание. Можем да използваме `<section>`, но използването на `<article>` дава по-семантичен смисъл на съдържанието.

Елементът `<header>` е нов в HTML5. Той определя заглавието на документ или на отделна секция. Елементът `<header>` се използва като контейнер за уводно съдържание или набор от връзки за навигация. Може да се използват няколко `<header>` елемента в един документ. Освен това `<header>` не може да бъде поставен в елементи като `<footer>`, `<address>` или друг `<header>`.

```
<article>
   <header>
    <h1>Internet Explorer 9</h1>
    </header>
    <p>Windows Internet Explorer 9 (съкратено като IE9) е пуснат офи-
циално на 14 март 2011 в 21:00 часа. </p>
</article>
```

Тагът `<aside>` е нов в HTML5. Този таг определя част от съдържанието на документа, отделно от съдържанието на целия документ. Това съдържание трябва да бъде свързано със съдържанието на целия документ.

```
<p>Аз и моето семейство посетихме Рилския манастир миналата седми-
ца.</p>
<aside>
    <h4>Рилски манастир</h4>
    <p>Рилският манастир е ставропигиален манастир, намиращ се в Юго-
западна България, област Кюстендил, община Рила. Основан е през X век
от св. Иван Рилски, в горното течение на Рилска река.</p>
</aside>
```

Тагът <aside> се поддържа от Internet Explorer 9+, Firefox, Opera, Chrome, и Safari. Internet Explorer 8 и по-ранните версии не поддържат този таг.

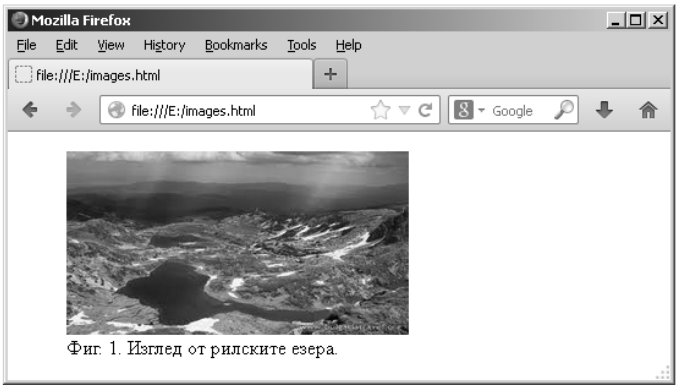
Тагът <figure> е нов в HTML5. Този таг определя самостоятелно съдържание, като фигури, диаграми, снимки и др. Тъй като съдържанието на елемента <figure> е свързано с основния поток, позицията му е независима от основния поток и ако той бъде отстранен, това не трябва да оказва въздействие върху потока на документа.

```
<figure>
    <img src="rilski_manastir.jpg" alt="Рилски манастир" width="500"
height="300">
</figure>
```

Тагът <figcaption> е също нов в HTML5. Този таг дефинира надпис за елемента <figure>. Тагът <figcaption> може да бъде поставен като първи или последен елемент на <figure>.

```
<figure>
   <img src="rilski_ezera.jpg" alt="рилските езера" width="500"
height="300">
   <figcaption>Фиг. 1. Изглед от рилските езера.</figcaption>
</figure>
```

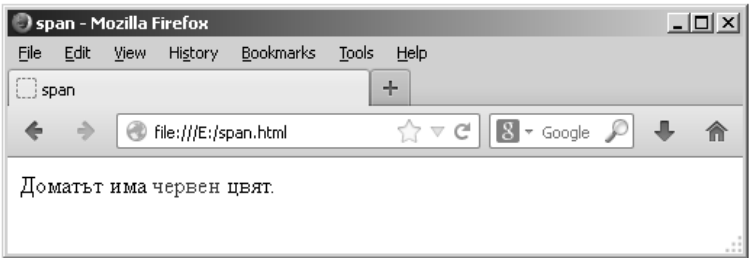
В браузър, кодът ще се визуализира по следния начин:



Тагът се използва за групиране на редови елементи в даден документ. Този таг не дава визуална промяна сам по себе си. Тагът осигурява начин за добавяне на свойство за част от текста или за част от документа, например:

```
<p>Доматът има <span style="color:red">червен</span> цвят.</p>
```

В браузър кодът ще се визуализира по следния начин:

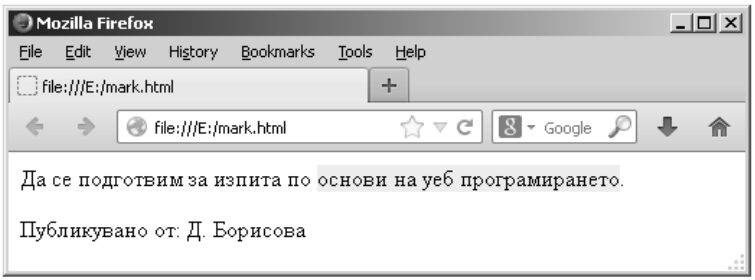


Тагът `<footer>` определя долния колонтитул за документ или раздел. Този елемент е нов в HTML5 и обикновено съдържа информация за автора на документа, за авторските права, за условията за ползване, за контакти и т.н. Можем да имаме няколко елемента `<footer>` в един документ.

Следващият пример илюстрира използването на елемента `mark` и елемента `footer`:

```
<p>Да се подготвим за изпита по <mark>основи на уеб програмиране-
то</mark>.</p>
<footer>
    <p>Публикувано от: Д. Борисова</p>
</footer>
```

В браузър кодът ще се визуализира по следния начин:



Следващите елементи от вер. HTML 4.01 са премахнати в HTML5: `<acronym>`; `<applet>`; `<basefont>`; `<big>`; `<center>`; `<dir>`; `<font>`; `<frame>`; `<frameset>`; `<noframes>`; `<strike>`; `<tt>`.

6.5. Основни различия между HTML и XHTML

В HTML няма значение дали таговете се изписват с големи или малки букви, но в XHTML има изискване таговете да се изписват само с малки букви. Например, тагът за нов ред в HTML може да се изпише и с главни букви `<BR />`, но в XHTML е задължително буквите да са малки: `<br />`.

HTML не изисква затваряне в кавички на стойностите на командните атрибути. В XHTML съществува изискване за поставяне на двойни кавички. Например, атрибутът за ширина на таблица, *width*, ако трябва да има стойност 150 пиксела, може да се изпише в HTML така: `width=150`, но в XHTML стойността на атрибута задължително трябва да е в кавички, т.е. `width="150"`.

Почти всички тагове в HTML се въвеждат по двойки – отварящ и затварящ таг. Съществуват и няколко команди, които нямат затварящ таг. В XHTML е задължително всички команди да имат затварящи тагове. Ако командата е без затварящ таг, тогава в отварящия таг се изписва и наклонена надясно черта за затваряне на тага, като тази черта трябва да е на една стъпка разстояние от края на текста на командата. Пример: командата за нов ред `<br>` няма затварящ таг, затова в XHTML нейния вид е: `<br />`.

На почти всички атрибути в HTML се задава някаква стойност, но съществуват атрибути, които нямат стойност. В XHTML е задължително на всички атрибути да се задава стойност. Щом един атрибут няма стойност в HTML, тогава в XHTML като стойност се задава самото име на атрибута.

Атрибутът *id* заменя атрибута *name*:

```
<img src="picture.gif" id="picture1" />
```

Lang приставката трябва да фигурира във всеки *xhtml* документ, ако страницата съдържа език, различен от английския, т.е. ако документът е на български език тя трябва да присъства. Например:

```
<div lang="bg" xml:lang="bg">Това е български език :)</div>
```

`<!DOCTYPE>` декларацията се отнася до *Document Type Definition* (DTD). DTD определя правилата за маркиращия език и неговата версия, така че браузърът да интерпретира коректно съдържанието на документа.

xmlns атрибутът определя *XML пространство от имена* за да-

ден документ, който е задължителен в XHTML документи.

Вместо `<html>` се използва следният код:

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="bg" lang="bg">
```

Всеки XHTML документ съдържа 3 основни части: *DOCTYPE, head* и *body*. Следователно, основната структура на *XHTML* документа ще има вида:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Title of document</title>
</head>
  <body>
        ............
  </body>
</html>
```

Основните различия между XHTML и HTML могат да се обобщят като:

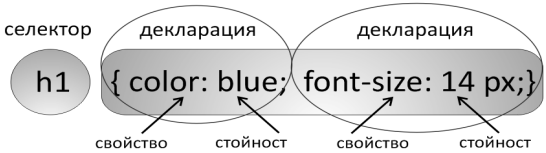
- XHTML изисква добавяне на `<!DOCTYPE>`;
- XHTML изисква добавяне на `xmlns`;
- XHTML изисква командите да се изписват задължително с малки букви;
- XHTML изисква всички атрибути да се поставят в кавички;
- XHTML изисква всички тагове да се затварят;
- XHTML изисква всички атрибути да имат някакви стойности;
- В XHTML *id* се заменя *name* (Елементите с *id* могат да бъдат използвани само веднъж. Поради това, ако ще се ползват повече пъти, трябва да се ползва *class*.).

Глава 7. Форматиране чрез CSS: основни понятия, деклариране и използване на стилове

CSS е съкращение от *Cascading Style Sheets*. CSS е език за описание на стилове – използва се основно за описване стила на документ, написан на език за маркиране (HTML, XML). Официално спецификацията на CSS се поддържа от W3C (*World Wide Web Consortium*). CSS позволява да се определи как да изглеждат елементите на една HTML страница: шрифтове, размери, цветове, фонове и др. CSS кодът се състои от последователност от стилови правила, всяко от които представлява селектор, следван от свойства и стойности. CSS 2.1 има повече от 90 свойства. CSS е удобен начин за форматиране на текстове, шрифтове, списъци, таблици и др., които ще бъдат описани в тази глава.

7.1. CSS – синтаксис

Всяко CSS правило има две основни части: селектор и една или повече декларации. Ако трябва да сме още по-прецизни, всяка декларация се състои от комбинацията на свойство и стойност. Синтаксисът на правилото е следният:



От лявата страна на правилото се намира селекторът. Това е частта от правилото, указваща частта от документа, за която трябва да се приложат стиловете. В този случай са избрани елементите `h1`. От дясната страна на правилото се намира декларацията. Това е комбинация от CSS свойството и стойността му. Свойството е стилов атрибут, на който задаваме стойност. Всяко свойство има стойност. CSS декларацията винаги завършва с точка и запетая, а групите от декларации са заградени от фигурни скоби:

```
p {color:red;text-align:center;}
```

За да се направи CSS кодът по-разбираем, всяка една декларация може да бъде на отделен ред:

```
p
{
color:red;
text-align:center;
}
```

Коментарите се използват за обясняване на CSS кода, и могат да помогнат при редактиране на изходния код на по-късен етап. Коментарите се игнорират от браузърите. CSS коментарите започват с наклонена черта и звездичка (/*) и завършват със звездичка и наклонена черта (*/):

```
/*This is a comment*/
p
{ text-align:center;
/*This is another comment*/
color:black; font-family:arial;
}
```

Можем да вмъкнем по три различни начина CSS кода в HTML документа:

- от външен файл, който представлява самостоятелен CSS файл, и за който е необходимо обръщение от HTML документа;
- деклариране на кода вътре в документа – блок със стилове в документа, затворен от таговете `<style>` и `</style>` и поставен в заглавната част на HTML документа (`<head>`);
- вграден стил на даден елемент – чрез деклариране на CSS кода в реда.

За да се използва външен CSS файл, е необходимо в HTML документа да се добави обръщение към конкретния CSS файл в частта `<head>`, като се използва следният синтаксис:

```
<link rel="stylesheet" type="text/css" name="име" href="адрес">
```

Тук `rel="stylesheet"` указва какъв документ да се търси (в случая *stylesheet*), `type="text/css"` указва какъв е типът на файла, който ще се използва, а `href="адрес"` указва къде се намира файлът. За CSS файл (`style.css`), който се намира в текущата директория, заедно с HTML файла, синтаксисът има вида:

```
<link rel="stylesheet" type="text/css" href="style.css">
```

При деклариране на кода в заглавната част на HTML документа (`<head>`) се използва тагът `<style>`:

```
<html>
<head>
   <STYLE>
   <!--
  #red { color: red }
  #blue { color: blue }
   -->
   </STYLE>
</head>
<body>
<p id="red">Червен текст.</p>
<p id="blue">Син текст.</p>
<body>
<html>
```

При използването на вграден стил на елемента, кодът се добавя в отварящия таг на конкретния елемент:

```
<p style="text-align: center">
това е центриран текст</p>
```

7.2. CSS свойства за форматиране на текст

В тази част е представено форматирането на текст с помощта на CSS свойствата. За форматирането на текстови елементи могат да се използват следните свойства:

- *color* – задава цвят на текста;
- *direction* – задава посоката на текста;
- *letter-spacing* – определя разстоянието между буквите;
- *word-spacing* – определя разстоянието между думите;
- *text-indent* – отместване на първия ред на параграфа;
- *text-align* – определя подравняването на текста на документа;
- *text-decoration* – задава подчертаването или зачертаването на текст;
- *text-transform* – преобразува текста в главни или малки букви;
- *white-space* – определя как да се обработват празните символи в елементите;
- *text-shadow* – използва се за задаване на сянка около текст.

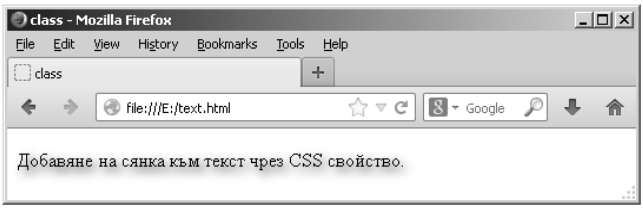
Част от свойствата за форматиране на текст са описани в глава 2. Синтаксисът за задаване на сянка на текст е:

```
text-shadow: h-shadow v-shadow blur color
```

където `h-shadow` определя позицията на хоризонталната сянка (задължително), `v-shadow` определя позицията на вертикалната сянка (задължително), `blur` определя разстоянието на размазването (не е задължително), `color` определя цвета на сянката (не е задължително).

```
<p style="text-shadow:4px 4px 8px blue;">
CSS свойството за добавяне на сянка към текст.</p>
```

Резултатът ще бъде:



7.3. CSS свойства за форматиране на списъци

Възможностите на CSS за форматиране на списъци позволяват задаването на различни булети при неподредените и различна номерация при подредените списъци. Има пет CSS свойства, чрез които може да се контролира форматирането на списъци:

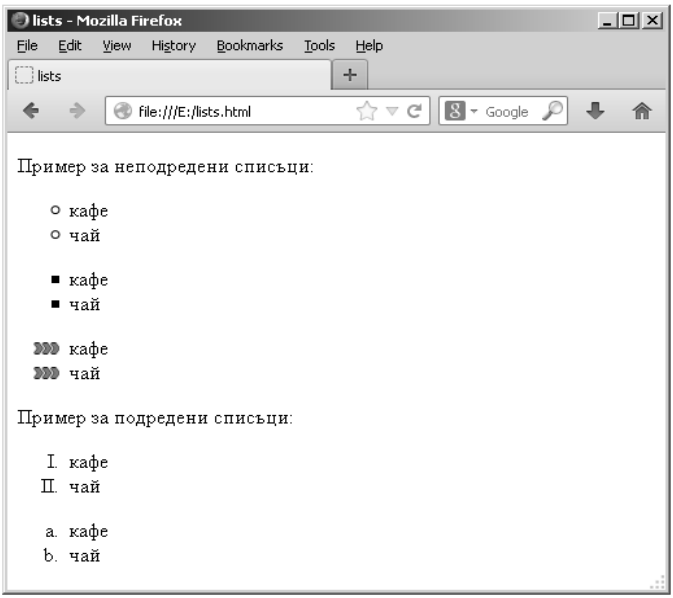
- *list-style-type* – позволява промяна на типа на булета;
- *list-style-position* – промяна на позицията на булета по отношение на съдържанието на списъчния елемент;
- *list-style-image* – задава изображение за булет вместо стандартните символи;
- *list-style* – служи за задаване на повече от едно свойство;
- *marker-offset* – задава разстоянието между маркера и текста в списъка.

Следващият пример илюстрира форматирането на неподредени и подредени списъци чрез свойствата на CSS:

```
<style>
  ul.a {list-style-type:circle;}
  ul.b {list-style-type:square; }
  ul.c {list-style-image:url('img.jpg');}
  ol.d {list-style-type:upper-roman;}
  ol.e {list-style-type:lower-alpha;}
</style>

<p> Пример за неподредени списъци:</p>
```

```
<ul class="a">
  <li>кафе</li>
  <li>чай</li>
</ul>
<ul class="b">
  <li>кафе</li>
  <li>чай</li>
</ul>

<ul class="c">
  <li>кафе</li>
  <li>чай</li>
</ul>

<p> Пример за подредени списъци:</p>
<ol class="d">
  <li>кафе</li>
  <li>чай</li>
</ol>
<ol class="e">
  <li>кафе</li>
  <li>чай</li>
</ol>
```

В браузър, кодът ще се визуализира по следни начин:



7.4. CSS свойства за форматиране на таблици

Форматирането на таблици чрез CSS се реализира чрез задаване на различни свойства за таблица. Възможни са следните форматирания на таблици чрез свойствата:

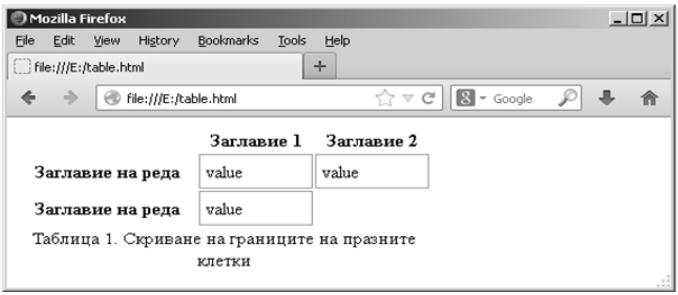
- *border-collapse* – контролира границите на съседните клетки – дали границите на клетките, които се докосват трябва да поддържат своя стил. Възможни стойности са: *collapse*, *separate*, *inherit*;
- *border-spacing* – задава разстоянието, което трябва да се появи между клетките на таблицата;
- *caption-side* – контролира разположението на елемента `<caption>`. Възможни стойности са: *top*, *bottom*, *inherit*;
- *empty-cells* – определя дали границата трябва да бъде показана, ако клетката е празна;
- *table-layout* – позволява на браузърите да ускорят оформлението на таблицата, с помощта на първото зададено свойство за ширина, преди да се зареди цялата таблица.

Следващият пример илюстрира скриване на границите на празните клетки в таблицата чрез свойствата на CSS.

```
<style type="text/css">
table.empty{
    width:350px;
    border-collapse:separate;
    empty-cells:hide;
}
td.empty{
    padding:5px;
    border-style:solid;
    border-width:1px;
    border-color:#999999;
}
caption {caption-side:bottom;}
</style>
<table class="empty">
<caption>Таблица 1. Скриване на границите на празните клетки</caption>
<tr>
  <th></th>
  <th>Заглавие 1</th>
  <th>Заглавие 2</th>
</tr>
<tr>
  <th>Заглавие на реда</th>
  <td class="empty">value</td>
  <td class="empty">value</td>
```

```
</tr>
<tr>
  <th> Заглавие на реда </th>
  <td class="empty">value</td>
  <td class="empty"></td>
</tr>
</table>
```

В браузър кодът ще се визуализира по следни начин:

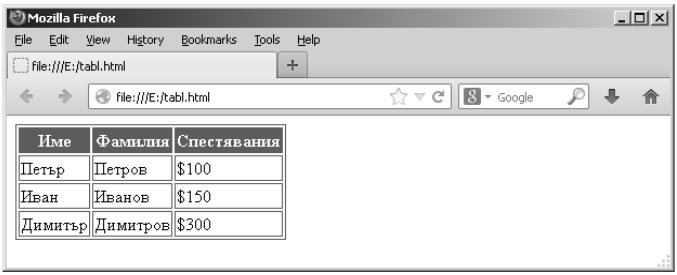


Пример за форматиране на рамката на таблицата в цвят и част от клетките също в цвят чрез свойствата на CSS:

```
<style>
  table, td, th
  {border:1px solid green;}
  th{background-color:green;color:white;}
</style>
<table>
<tr>
  <th>Име</th>
  <th>Фамилия</th>
  <th>Спестявания</th>
</tr>
<tr>
  <td>Петър</td>
  <td>Петров</td>
  <td>$100</td>
</tr>
<tr>
  <td>Иван</td>
  <td>Иванов</td>
<td>$150</td>
</tr>
<tr>
  <td>Димитър</td>
  <td>Димитров</td>
```

```
  <td>$300</td>
</tr>
</table>
```

В браузър кодът ще се визуализира по следни начин:



7.5. CSS свойства за форматиране на шрифт

Чрез CSS свойството за форматиране на шрифт може да определи семейството на шрифта, размерът и стилът на текста. За форматирането на шрифт, могат да се използват следните свойства:

- *font-family* – дефинира семейството шрифове, които да се използват от уеб браузъра за текст;
- *font-style* – определя стила на шрифта за текст;
- *font-variant* – указва, че текстът трябва да бъде показан в шрифт с главни малки букви;
- *font-weight* – определя колко плътен трябва да бъде текстът;
- *font-size* – определя размера на шрифта;
- *font* – определя семейството на шрифта, плътността, размера и стила на текста.

Пример за задаване на семейство шрифтове; като възможни стойности може да се използва всяко име на фамилия шрифтове:

```
<p style="font-family:arial,garamond,serif;">
Този текст ще се визуализира в arial, garamond или в серифен шрифт по
подразбиране, в зависимост от това кои шрифтове са инсталирани на ва-
шия компютър.
</p>
```

Пример за задаване на размер на шрифта; като възможни стойности могат да се използват: *small*, *x-small*, *xx-small*, *smaller*, *medium*, *large*, *x-large*, *xx-large*, *larger* и *size* в пиксели или в проценти:

```
<p style="font-size:20px;"> Този размер на шрифта е 20 пиксела</p>
<p style="font-size:small;"> Този размер на шрифта е small</p>
<p style="font-size:large;"> Този размер на шрифта е large</p>
```

Пример за задаване на повече от едно свойства, които да действат върху текста едновременно:

```
<p style=" font:italic small-caps bold 15px georgia;">
Прилагане на повече от едно свойства върху текста едновременно. </p>
```

7.6. CSS свойство за цвят

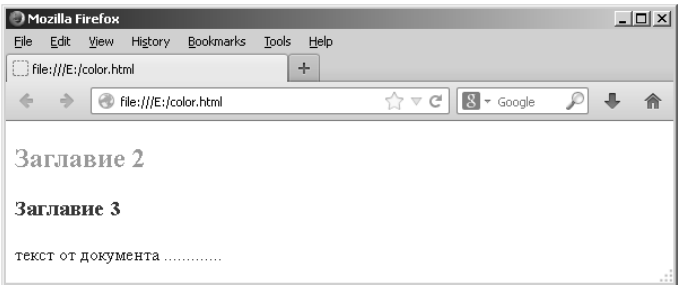
Свойството `color` определя цвета на предния план на даден HTML елемент. Това означава, че може да се променя цветът на текстовото съдържание. Цветът може да бъде зададен чрез неговото име, в шестнайсетичен код или чрез rgb код, както е показано:

- color: red;
- color: #00ff00;
- color: rgb(255,0,0)

В следващия пример ще зададем син цвят на текста в документа, на всички заглавия от тип *h2* – `#00ff00` цвят, а на заглавията от тип *h3* – цвят червен, т. е. `rgb(255,0,0)`:

```
<style>
    body {color:blue;};
    h2 {color:#00ff00;}
    h3 {color:rgb(255,0,0);}
</style>
```

В браузър кодът ще се визуализира по следния начин:



За определянето на цветовете на всички елементи от един и същ тип в HTML документа (напр. всички *хипервръзки*, всички заг-

лавия от тип *h1* и др.) могат да се използват класове или идентификатори, които са разгледани в т. 7.9.

7.7. CSS свойства за форматиране на фон

Свойствата за форматиране на фона позволяват да се зададе определен цвят като фон на различните HTML елементи или да се използва изображение като фон. Могат да се използват следните свойства:

- *background-color* задаване на цвят на фона;
- *background-image* задаване на фоново изображение;
- *background-repeat* контролира повторението на фоновото изображение;
- *background-position* контролира позицията на фоновото изображение;
- *background-attachment* контролира превъртането на фоновото изображение;
- *background* използва се за определяне на повече от едно фоново свойство.

Пример, който задава цвят на фона на определен елемент.

```
<p style="background-color:yellow;">
Този текст има жълт фон.</p>
```

Ето един пример, който показва повтаряне на фоновото изображение:

```
<table style="background-image:url(/images/pattern1.gif);
              background-repeat: repeat;">
   <tr>
  <td>Тази таблица има фоново изображение, което ще се повтаря до нейното цялостно запълване.</td>
   </tr>
</table>
```

Може да се използва стойност *no-repeat*, ако не е необходимо изображението да се повтаря – в този случай изображението ще се покаже само веднъж. По подразбиране стойността е *repeat*.

Пример за повтарящо се фоново изображение само по хоризонтала:

```
<table style="background-image:url(/images/pattern1.gif);
              background-repeat: repeat-x;">
   <tr>
```

```
  <td>Таблица с повтарящо се фоново изображение само по хоризонта-
ла</td>
   </tr>
</table>
```

Пример, който указва позицията на фоново изображение 100 пиксела от лявата страна и 200 пиксела надолу от върха.

```
<table style="background-image:url(/images/pattern1.gif);
              background-position:100px 200px;">
   <tr>
   <td> Тази таблица има фоново изображение, позиционирано на 100 пик-
сели отляво и 200 пиксели отгоре.</td>
   </tr>
</table>
```

Пример за фиксирано фоново изображение:

```
<p    style="background-image:url(/images/pattern1.gif);    background-
attachment:fixed;">
Този параграф има фиксирано фоново изображение.</p>
```

Пример за определяне на повече от едно фоново свойство:

```
<p style="background:url(/images/pattern1.gif) repeat fixed;">
Този параграф има фиксирано и повтарящо се фоново изображение.</p>
```

7.8. CSS3 свойства за форматиране на текст в колони

CSS3 въвежда ново свойство за оформление в много колони. То позволява определяне на броя на колоните, респективно на ширината им, разстоянието помежду им, стила и цвета на линиите между тях. Това се осъществава чрез следните свойства:

- *column-count* задава броя на колоните, на които елементът трябва да бъде разделен;
- *column-gap* задава разстоянието между колоните;
- *column-rule* определя ширината, стила и цвета на линиите между колоните.

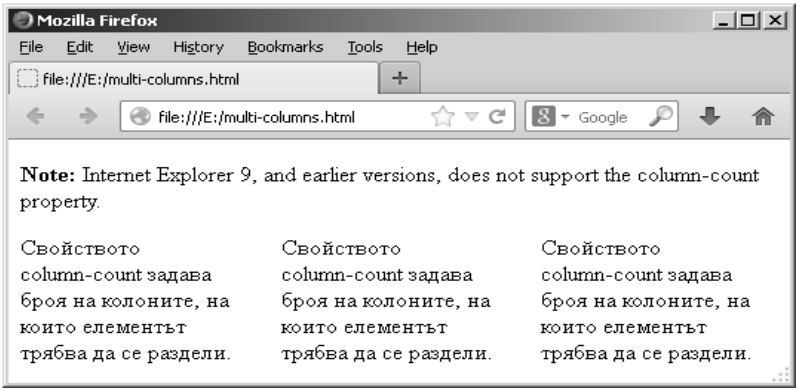
Определянето на броя или на ширината на колоните е важно при оформлението чрез много колони. Колоните трябва да бъдат зададени или чрез свойството `column-width` или чрез `column-count`, но никога не трябва да се използват заедно. И двете свойства по подразбиране са `auto`. Internet Explorer 10 и Opera поддържат свойството за многоколонно форматиране. Firefox изиксва префикса `-moz-`; Chrome и Safari

изискват префикса `-webkit-`. Internet Explorer 9 и по-ранните версии не поддържат форматиране в много колони.

Пример за използване на свойството `column-count` за определяне на броя на колоните:

```
<style>
     .newspaper{
 -moz-column-count:3;    /* Firefox */
 -webkit-column-count:3; /* Safari and Chrome */
column-count:3;
      }
</style>
<p><b>Note:</b> Internet Explorer 9, and earlier versions, does not
support the column-count property.</p>
<div class="newspaper">
Свойството column-count задава броя на колоните, на които елементът
трябва да се раздели. Свойството column-count задава броя на колоните,
на които елементът трябва да се раздели. </div>
```

В браузър кодът ще се визуализира по следни начин:

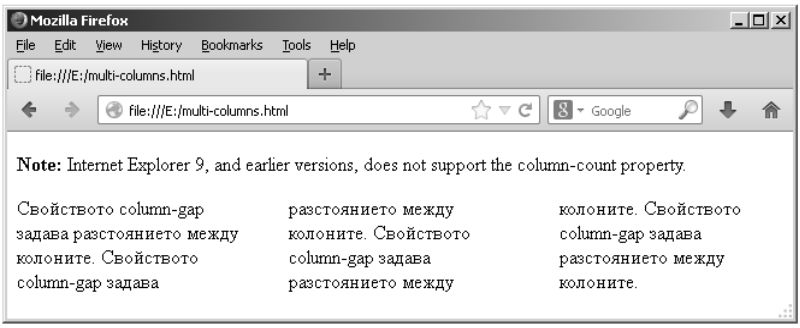


Пример за използване на свойството `column-gap`, което определя разстоянието между колоните:

```
<style>
.newspaper{
 -moz-column-count:3;    /* Firefox */
 -webkit-column-count:3; /* Safari and Chrome */
column-count:3;
 -moz-column-gap:40px;    /* Firefox */
 -webkit-column-gap:40px; /* Safari and Chrome */
column-gap:40px;
}
```

```
</style>
<p><b>Note:</b> Internet Explorer 9, and earlier versions, does not
support the column-count property.</p>
<div class="newspaper">
Свойството column-gap задава разстоянието между колоните. Свойството
column-gap задава разстоянието между колоните. Свойството column-gap
задава разстоянието между колоните. </div>
```

В браузър кодът ще се визуализира по следни начин:



Други свойства на многоколонното форматиране са:

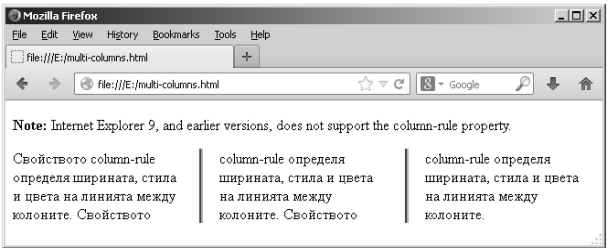
- *column-fill* определя как се запълват колоните;
- *column-rule-color* задава цвета на линиите между колоните;
- *column-rule-style* задава стила на линиите между колоните;
- *column-rule-width* задава дебелината на линиите между колоните;
- *column-span* задава броя на колоните, които да бъдат обединени;
- *column-width* задава ширината на колоните;
- *columns* определя повече от едно свойство на форматирането в много колони.

Пример за използване на свойството `column-rule`, което определя ширината, стила и цвета на линиите между колоните:

```
<style>
     .newspaper{
     -moz-column-count:3;    /* Firefox */
     -webkit-column-count:3; /* Safari and Chrome */
     column-count:3;
     -moz-column-gap:40px;   /*Firefox */
     -webkit-column-gap:40px;   /*Safari and Chrome */
     column-gap:40px;
```

```
     -moz-column-rule:4px outset #ff00ff;   /*Firefox */
     -webkit-column-rule:4px outset #ff00ff; /*Safari&Chrome */
     column-rule:4px outset #ff00ff;
     }
</style>
<p><b>Note:</b> Internet Explorer 9, and earlier versions, does not
support the column-rule property.</p>
<div class="newspaper">
Свойството column-rule определя ширината, стила и цвета на линиите
между колоните. Свойството column-rule определя ширината, стила и цве-
та на линиите между колоните. </div>
```

В браузър кодът ще се визуализира по следни начин:



7.9. CSS Селектори

Селекторите са един от най-важните аспекти на CSS, тъй като те се използват за избор на елементи от HTML страницата, така че те да бъдат стилизирани. Селекторите показват към кои елементи на HTML документа трябва да бъде прилаган съответният стил. В CSS съществуват два вида селектори: `class` и `id`.

`ID` селекторите се използват за определяне на стилът за един уникален елемент. Тъй като `id` атрибутите трябва да имат уникални стойности, то `id` селекторът не трябва да съвпада с повече от един елемент в документа. `ID` селекторът използва `id` атрибут на HTML елемент и се дефинира с #, посредством синтаксиса:

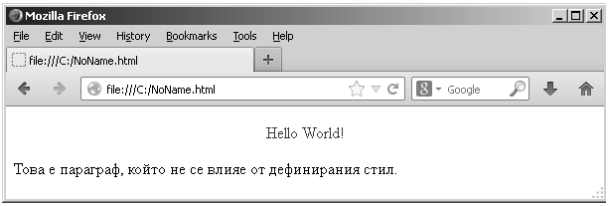
```
#id {attribute: value}
```

Нека дефинираме стил като използваме `id` селектор с името "`paragraph1`":

```
<style>
#paragraph1 {text-align:center;color:red;}
</style>
```

```
<p id="paragraph1">Hello World!</p>
<p>Това е параграф, който не се влияе от дефинирания стил.</p>
```

В браузър кодът ще се визуализира по следни начин:



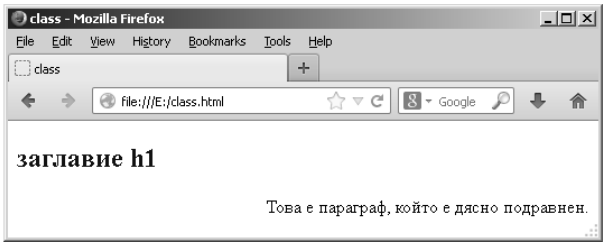
Избирането на елементи на базата на техните `class` имена е техника много често срещана в CSS. CSS позволява да се задават собствени класове за различните селектори. След като един клас е дефиниран, всички селектори, на които е присвоен този клас, ще показват един и същ ефект. Използва се следният синтаксис:

```
selector.class {attribute: value}
```

Ето един пример, в който е направен клас за заглавията *h1* с червен цвят и клас за параграфи *p*, чийто текст да се позиционира вдясно. Името на класа е произволно, но за удобство е използвано *blue*. Следователно кодът ще изглежда така:

```
<style>
    h1.blue {color: #0000ff}
    p.right {text-align: right}
</style>

<h1 class="blue">заглавие h1</h1>
<p class="right">
Това е параграф, който е дясно подравнен.</p>
```

В браузър кодът ще се визуализира по следни начин:



Класовете могат да бъдат декларирани и по друг начин, а именно – без необходимост от обвързване с конкретен селектор. В този случай, синтаксисът е:

```
.class {attribute: value}
```

Както се вижда, тук отсъства самият селектор. Така например вместо кода:

```
h1.red {color: #ff0000}
p.right {text-align: right}
```

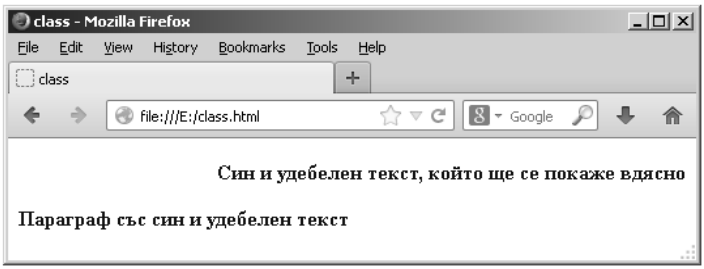
може да се използва следният опростен запис:

```
.red {color: #ff0000}
.right {text-align: right}
```

Всичко останало е както в горния пример, при което ефектът в HTML страницата ще бъде същият. Към декларирания клас за позициониране в дясната част (`.right`) на страницата може да добавим следния `id` (`#bluebold`):

```
<style type="text/css">
      .right {text-align: right}
      #bluebold {color:#0000ff; font-weight:bold}
</style>

<p class="right" id="bluebold">Син и получерен текст, който ще се по-
каже вдясно</p>
<p id="bluebold"> Параграф със син и получерен текст</p>
```

Ефектът от този код ще бъде удебелен текст със син цвят. В случая названието *bluebold* е произволно и е избрано само, за да подсказва за ефекта, може да бъде и друго. Можем да използваме и код без декларирания клас, като в този случай, ефектът ще бъде син и удебелен текст, но без позициониране на параграфа вдясно:



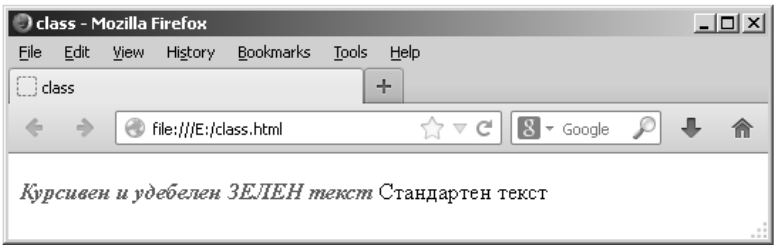
Разлика мужду селекторите `class` и `id`. На първо място класовете могат да бъдат присвоявани на неограничен брой елементи. От друга страна, предполага се, че идентификаторите (`id`) могат да се използват само веднаж в тялото на даден HTML документ. По този начин те напомнят за стойностите на атрибута `name` във формиращите елементи от вида `input`. Всяка стойност на `name` трябва да бъде уникална, така е и при стойностите на `id`. Друга разлика между класовете и идентификаторите е, че идентификаторите имат по-голяма тежест при определянето на стила на даден елемент.

Контекстуалните селектори представляват комбинация от няколко селектора, като зададеният ефект се проявява в зависимост от тяхната подредба. Синтаксисът на контекстуалните селектори е:

```
1-ви селектор 2-ри селектор... {атрибут: стойност}
```

Следващият пример показва ефекта от удебелен, курсивен и зелен на цвят текст, разположен между таговете `<i><b>` и `</b></i>` и стандартен текст за останалата част от текста в параграфа:

```
<style type="text/css">
  p i b {color:#00ff00}
</style>

<p><i><b>Курсивен и удебелен ЗЕЛЕН текст</b></i> Стандартен текст</p>
```



Трябва да се спазва последователността на декларираните селектори, в противен случай ефектът няма да се прояви, т.е. ако напишем:

```
<p><b><i>Текст</b></p></i>
```

като ефект няма да се получи зелен цвят, тъй като е нарушена декларираната последователност на селекторите `<i>` и `<b>`.

Няколко селектора могат да бъдат подредени като се отделят със запетаи и им се зададе еднакъв ефект, т.е. атрибути с еднакви стойности. Пример за групиране на селектори:

```
<style type="text/css">
  h1,h2,h3,p,del {color: #ff0000}
</style>
```

В случая са подредени селекторите за първите 3 по големина заглавия, за параграф и за зачертаване на текст, като на всички тях е зададен червен цвят. Ако в HTML документа запишем:

```
<del>ЧЕРВЕН ТЕКСТ С ЧЕРТА ПО СРЕДАТА</del>
```

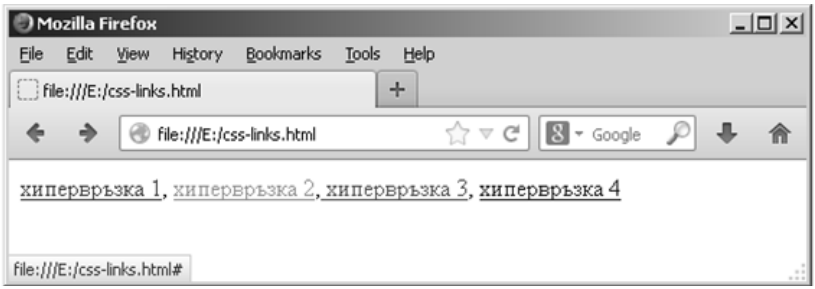
този текст ще се покаже не само с черта посредата, какъвто е по принцип ефектът от тага `<del>`, но и в червен цвят.

Псевдокласовете и псевдоелементите позволяват присвояването на класове структури, които могат и да не съществуват в документа, или на неща, които зависят от състоянието на определени елементи или от самия документ. С други думи, стиловете се прилагат за части от документа, които не са базирани на структурата на документа и които не могат да бъдат разбрани при разглеждането на форматирането на документа. Псевдокласовите селектори се използват за добавяне на специални ефекти към някои селектори. Синтаксисът на псевдо-класовете е:

```
selector:pseudo-class {property:value;}
```

CSS класовете могат да се използват с псевдо-класове. Ето един пример за показване на хипервръзките в различен цвят:

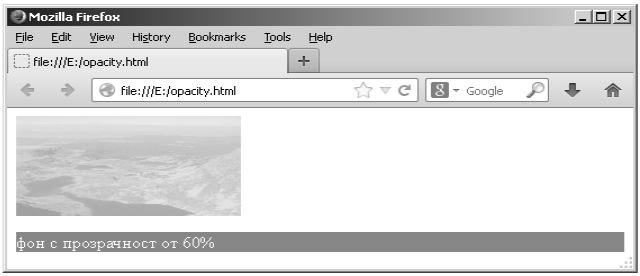
```
<style>
    a:link  {color:#FF0000;}    /* unvisited link */
    a:visited {color:#00FF00;}  /* visited link  */
    a:hover  {color:#FF00FF;}   /* mouse over link */
    a:active {color:#0000FF;}   /* selected link */
</style>
```



7.10. CSS свойство за прозрачност

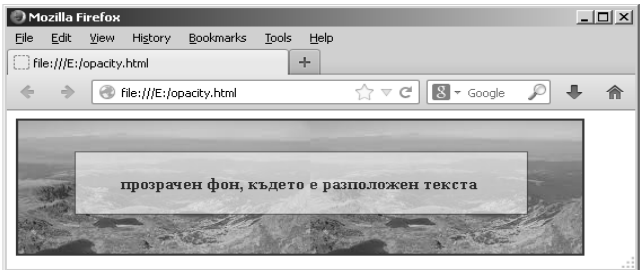
Създаването на прозрачни изображения с CSS е лесно. CSS3 свойството за прозрачност е *opacity*, което е част от препоръката на W3C. Свойството *opacity* може да има стойност от 0.0 до 1.0, като 0.0 е 100% прозрачен, а 1.0 е 100% непрозрачен. Нека да създадем изображение с 40% прозрачност и фон с прозрачност 60%:

```
<img src="rila.jpg" border="0" width="260" height="200" alt=""
style="filter:alpha(opacity=40); opacity:0.4;">
<p style="background-color:red; color:#fff; opacity:0.6;
opacity:0.6;"> fff</p>
```



Нека направим прозрачен фона, където е разположен текстът:

```
<style>
div.background {width:500px; height:250px; background:url(rila.jpg)
repeat; border:2px solid red;}
div.transp {width:400px; height:180px;  margin:30px 50px; background-
color:#ffffff; border:1px solid red; opacity:0.6;
filter:alpha(opacity=60); /* For IE8 and earlier */}
div.transp p { margin:30px 40px; font-weight:bold;
color:#000000;}</style>
<div class="background"><div class="transp"><p> прозрачен фон, където
е разположен текста </p></div>
</div>
```

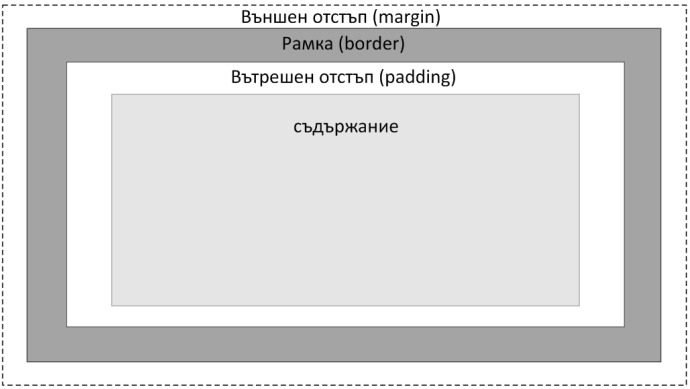


Глава 8. Оформление на HTML документ чрез CSS

В CSS се приема, че всеки елемент генерира едно или повече правоъгълни полета, наричани „елементни полета“ или „контейнери“. За да разберем начина на обработване на блоковите елементи (параграфи, списъци и др.) е необходимо да бъде описан т. нар. *box* модел. В тази глава са представени свойствата за форматиране на контейнерите – външни и вътрешни полета, рамка и контур. Показани са схемите за позициониране на елементите, възможностите за задаване на плаващи елементи. Накрая са илюстрирани няколко различни оформления на уеб страници, заедно със съответните програмни решения, комбиниращи възможностите на HTML и CSS.

8.1. CSS Box модел

Моделът box се прилага към блоковите елементи. В CSS2.1, блоковите елементи могат до бъдат само правоъгълни. Габаритните размери на ниво блоков елемент се изчисляват като се вземат предвид височината и ширината на областта, съдържанието, както всички отстъпи (вътрешен отстъп – *padding* и външен отстъп – *margin*) и рамката (*border*), които се прилагат към елемента (фиг. 8.1):



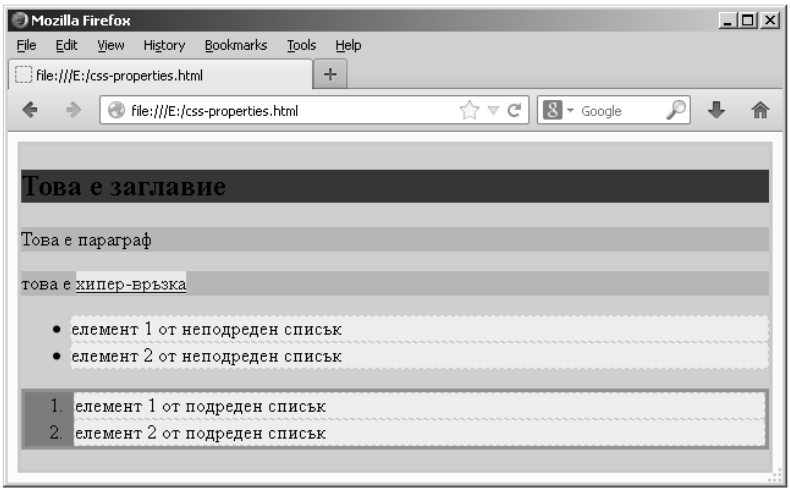
Фиг. 8.1 Визуализация на модела box

- *съдържание (content)* – това е съдържанието на самия елемент: може да е текстът между началния и крайния таг или да е друг елемент;
- *вътрешен отстъп (padding)* – това е разстоянието между ръба на съдържанието и вътрешната страна на рамката;
- *рамка (border)* – вътрешният ръб на рамката съвпада с ръба на вътрешния отстъп, а от външния ѝ ръб – започва външният отстъп;
- *външен отстъп (margin)* – това е разстоянието от външния ръб на рамката до заобикалящите елементи.

Ето един пример за форматиране на блокови елементи чрез CSS свойства:

```
<style>
    html { background: #FFFFCC; }
    body { border: 3px solid cyan; background: #FFCC66; }
    h2 { background: red; }
    p  { background: lime; }
    ol { border: 3px solid #F88008; background: #333300; }
    li { border: 1px dashed #CCCC99; background: yellow; }
    a {  background: #99FFFF; }
</style>
```

В браузър кодът ще се визуализира по следния начин:



CSS свойствата на модела box се използват за определяне на разстоянието в и около HTML елементите, техните граници, както и дру-

гите аспекти на позиционирането. *Border, margin* и *padding* имат по 4 свойства: *top*, *right*, *bottom* и *left*. Свойствата *width* и *height* регулират ширината и височината на съдържанието за елемент от тип блок.

Използваните стойности в CSS могат да се разделят на: абсолютни (px, pt, cm, …); относителни (%, em); или auto (default), при които браузърът определя размера. В последния случай, изчисляването зависи от вида на браузъра, от дефиницията на документа и от типа на съдържанието. В Приложение 6 са показани възможните стойности с кратко описание.

8.2. CSS свойства за форматиране на външния отстъп

Свойството *margin* задава пространство около HTML елементите. Външният отстъп (*margin*) няма цвят и е напълно прозрачен. Използваните стойности могат да бъдат както положителни, така и отрицателни, при което може да се получи припокриване на съдържанието. Стойностите на свойството *margin* не се наследяват от дъщерните елементи. При наличието на съседни горен и долен външни отстъпи, резултатът ще бъде обединяване, така че разстоянието между блоковете няма да бъде сбор от съответните полета, а ще е по-голямото от двете или ще има размера на едно от тях, ако двете са еднакви.

Има четири свойства за определянето на външния отстъп на елемент:

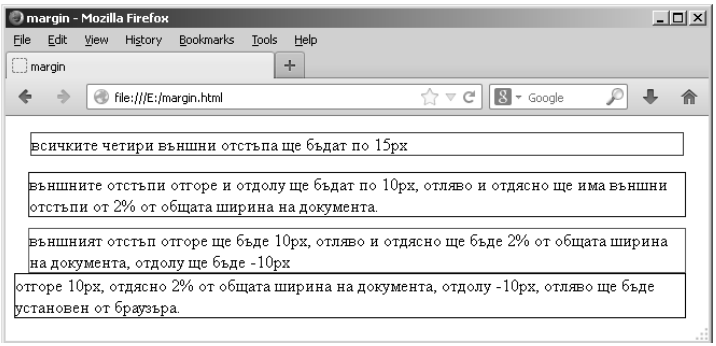
- *margin* определя повече от едно свойство за външния отстъп в една декларация;
- *margin-bottom* задава долната граница на външния отстъп;
- *margin-top* задава горната граница на външния отстъп;
- *margin-left* задава лявата граница на външния отстъп;
- *margin-right* задава дясната граница на външния отстъп.

Ето един пример на задаване на външни отстъпи (*margin*):

```
<p style="margin: 15px; border:1px solid red;">
всичките четири външни отстъпа ще бъдат по 15px </p>

<p style="margin:10px 2%; border:1px solid blue;">
външните отстъпи отгоре и отдолу ще бъдат по 10px, отляво и отдясно ще
има външни отстъпи от 2% от общата ширина на документа.</p>
```

```
<p style="margin: 10px 2% -10px; border:1px solid green;"> външният
отстъп отгоре ще бъде 10px, отляво и отдясно ще бъде 2% от общата ши-
рина на документа, отдолу ще бъде -10px </p>

<p style="margin: 10px 2% -10px auto; border:1px solid black;"> отгоре
10px, отдясно 2% от общата ширина на документа, отдолу -10px, отляво
ще бъде установен от браузъра.</p>
```

В браузър кодът ще се визуализира по следния начин:



8.3. CSS свойства за форматиране на вътрешен отстъп

Свойството *padding* позволява да се определи колко място трябва да се появи между съдържанието на елемента и неговата граница. Стойността на свойството *padding* се задава като дължина, процент, или се наследява. Има пет свойства, които определят вътрешния отстъп на даден елемент:

- *padding-bottom* задава долния вътрешен отстъп;
- *padding-top* задава горния вътрешен отстъп;
- *padding-left* задава левия вътрешен отстъп;
- *padding-right* задава десния вътрешен отстъп;
- *padding* задава повече от един вътрешен отстъп.

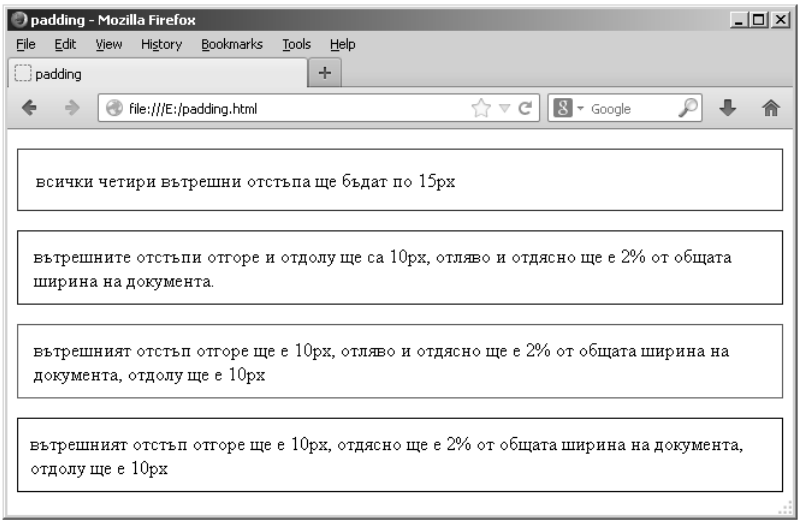
Ето един пример за определяне на долния вътрешен отстъп *padding-bottom* на елемента параграф, зададен в пиксели/проценти:

```
<p style="padding-bottom: 15px; border:1px solid red;">
Това е параграф, с определен отдолу вътрешен отстъп</p>

<p style="padding-bottom: 5%; border:1px solid black;">
Това е друг параграф, с определен отдолу вътрешен отстъп в проценти</p>
```

Пример за определяне на повече от един вътрешен отстъп:

```
<p style="padding: 15px; border:1px solid red;">
всичките четири вътрешни отстъпа ще бъдат по 15px</p>

<p style="padding:10px 2%; border:1px solid blue;">
вътрешните отстъпи отгоре и отдолу ще са 10px, отляво и отдясно ще са
2% от общата ширина на документа.</p>

<p style="padding: 10px 2% 10px; border:1px solid green;"> вътрешният
отстъп отгоре ще е 10px, отляво и отдясно ще е 2% от общата ширина на
документа, отдолу ще е 10px </p>

<p style="padding: 10px 2% 10px 10px; border:1px solid black;"> вът-
решният отстъп отгоре ще е 10px, отдясно ще е 2% от общата ширина на
документа, отдолу ще е 10px</p>
```

В браузър кодът ще се визуализира по следния начин:



8.4. CSS свойства за форматиране на рамка

Свойството *border* позволява да се форматира рамката на елемент от HTML. Има три свойства за форматиране на рамка:

- *border-color* задава цвят на рамката;
- *border-style* определя стил рамката чрез стойностите *none*, *hidden*, *dotted*, *dashed*, *solid*, *double*, *groove*, *ridge*, *inset*, *outset*, *inherit*;

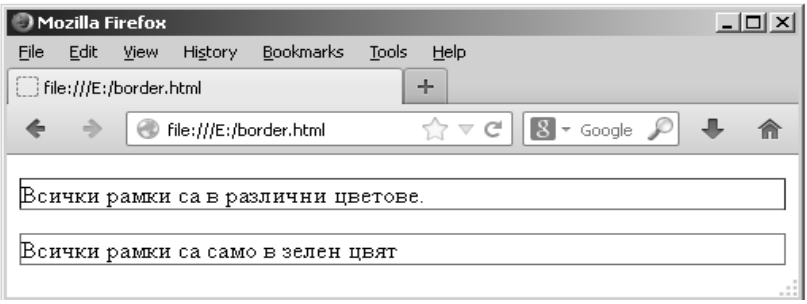
- *border-width* задава ширина/дебелина на рамката.

Свойството *border-color* позволява да се променя цветът на рамката за даден елемент. Може да се променя цветът на всичките четири страни (отдолу, отляво, отгоре и отдясно), като се използват следните свойства: *border-bottom-color, border-top-color, border-left-color* и *border-right-color*.

Пример, който използва ефекта на всички тези свойства:

```
<style type="text/css">
p.example1{
   border:1px solid;
   border-bottom-color:#009900; /* Green */
   border-top-color:#FF0000;    /* Red */
   border-left-color:#330000;   /* Black */
   border-right-color:#0000CC;  /* Blue */
}
p.example2{ border:1px solid; border-color:#009900;}
</style>

<p class="example1">Всички рамки са в различни цветове.</p>
<p class="example2">Всички рамки са само в зелен цвят.</p>
```

В браузър кодът ще се визуализира по следния начин:



Свойството *border-style* позволява да се избере един от следните стилове на рамката: *none* (еквивалент на *border-width:0*), *solid*, *dotted*, *dashed*, *double*, *groove*, *ridge*, *inset*, *outset* или *hidden*.

Индивидуално стилът на рамката може да се променя с помощта на свойствата: *border-bottom-style*, *border-top-style*, *border-left-style*, *border-right-style*.

В следващия пример е показана рамка с различни стилове за всичките 4 страни:

```
<p    style="border-width:4px;    border-top-style:solid;border-bottom-
style:dashed;  border-left-style:groove;  border-right-style:double;">
Това е рамка с четири различни стила.</p>
```

Свойството *border-width* позволява да се определи дебелината на рамката. Стойността на това свойство може да бъде дължина в px, pt или cm или да е зададена чрез *thin*, *medium* или *thick*. Индивидуално дебелината на рамката може да се променя чрез свойствата: *border-bottom-width*, *border-top-width, border-left-width, border-right-width*.

В следващия пример е показана плътна рамка с 4 различни дебелини за всичките 4 страни:

```
<p style="border-bottom-width:4px; border-top-width:10px; border-left-
width: 2px; border-right-width:15px; border-style:solid;">
Това е плътна рамка с 4 различни дебелини.</p>
```

Свойството `border` позволява да се зададе повече от едно свойство, например – цвят, стил и ширина на рамката:

```
<p style="border:4px solid red;">
Този пример показва кратък запис на повече от едно свойство на рамка-
та.</p>
```

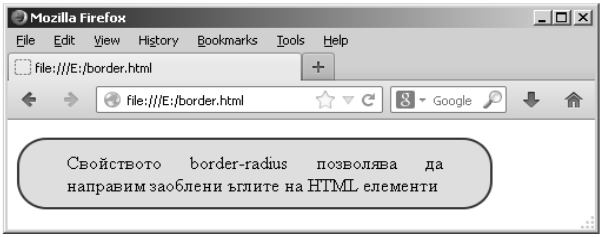
Свойството `border-radius`, поддържано от CSS3, позволява да направим лесно закръглени ъгли на различни елементи от HTML документа. Синтаксисът е следният:

```
border-radius: 1-4 length|% / 1-4 length|%;
```

Първият набор от стойности (1-4) определя хоризонталните радиуси за четирите ъгъла. Като опция се предлага втори набор от стойности, разделени от "/", които определят вертикалните радиуси за четирите ъгъла. Ако използваме само един набор от стойности, те ще се отнасят както за хоризонталните радиуси, така и за вертикалните радиуси. Нека дефинираме следния стил за параграф:

```
<style>
p {border:2px solid red; padding:10px 40px;
background:#CAE0FF; width:320px; border-radius:25px;}
</style>

<p align="justify">Свойството border-radius позволява да направим за-
облени ъглите на HTML елементи</p>
```

Използването на така дефинирания стил ще доведе до следния резултат:



Отделните радиуси могат да се определят чрез следните свойства: *border-top-left-radius; border-top-right-radius; border-bottom-right-radius* и *border-bottom-left-radius*.

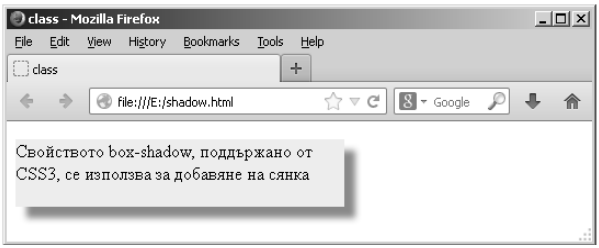
Свойството `box-shadow`, поддържано от CSS3, се използва за добавянето на сянка. Синтаксисът е:

```
box-shadow: h-shadow v-shadow blur spread color
```

където `h-shadow` определя позицията на хоризонталната сянка (задължително), `v-shadow` определя позицията на вертикалната сянка (задължително), `blur` определя разстоянието на размазването (не е задължително), `spread` определя размера на сянката (не е задължително), `color` определя цвета на сянката (не е задължително).

```
<style>
p {width:300px;
height:100px;
background-color:yellow;
box-shadow: 10px 10px 5px #888888; }
</style>

<p >Свойството box-shadow, поддържано от CSS3, се използва за добавяне
на сянка </p>
```

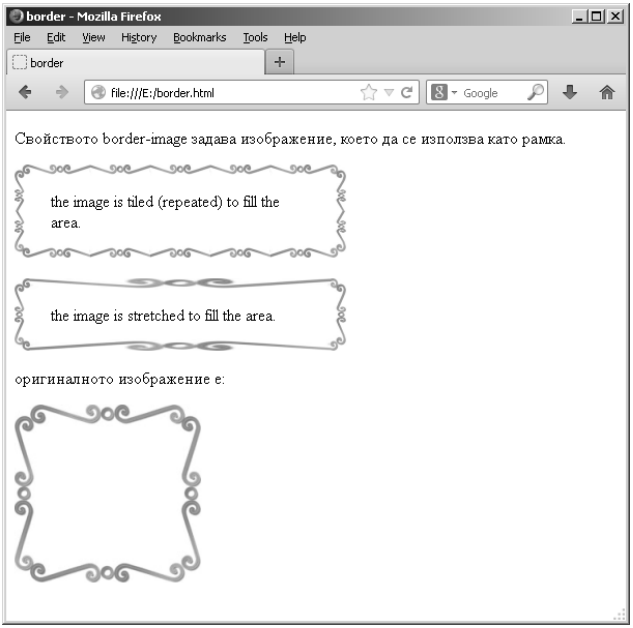
В браузър кодът ще се визуализира по следния начин:



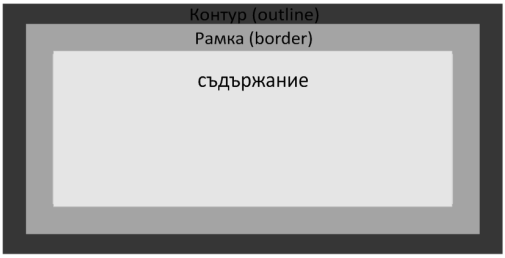
CSS3 свойството `border-image` използва изображение, което да се ползва като рамка на HTML елемент:

```
<style>
div {border:15px solid transparent;
width:250px;
padding:10px 20px;}

#round {
-webkit-border-image:url(border.png) 30 30 round; /*Safari 5*/
-o-border-image:url(border.png) 30 30 round; /*Opera10.5-12.1*/
border-image:url(border.png) 30 30 round;}

#stretch {
-webkit-border-image:url(border.png) 30 30 stretch; /*Safari5*/
-o-border-image:url(border.png) 30 30 stretch; /*Opera 10.5-12.1*/
border-image:url(border.png) 30 30 stretch;}
</style>

<p>Свойството border-image задава изображение, което да се използва
като рамка.</p>
<div id="round"> the image is tiled (repeated) to fill the area.</div>
<br>
<div id="stretch">the image is stretched to fill the area.</div>
<p> оригиналното изображение е:</p><img src="border.png">
```

В браузър кодът ще се визуализира по следния начин:



8.5. CSS свойства за форматиране на контура



Фиг. 8.2 Позиция на контура спрямо елемента

Контурна линия (*outline*) е линията, която се изчертава около елементите (извън рамките), за да се открояят елементите. На фиг. 8.2 е показана схематично позицията на контура, спрямо рамката и съдържанието на HTML елемента.

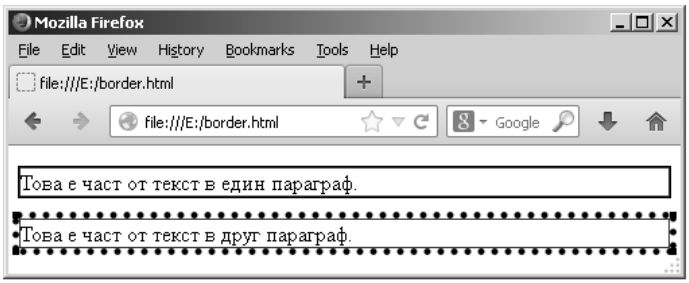
Свойствата на контурната линия са: *style*, *color* и *width*:

- *outline-color* задава цвят на контура;
- *outline-style* задава стил на контура;
- *outline-width* задава дебелина на контура;
- *outline* задава повече от едно свойство.

Пример за използване на свойствата на контура:

```
<style>
 p.one
 {border:1px solid red;outline-style:solid;outline-width:thin;}
 p.two
 {border:1px solid red;outline-style:dotted;outline-width:5px;}
</style>

<p class="one"> Това е част от текст в един параграф.</p>
<p class="two"> Това е част от текст в друг параграф.</p>
```

Използването на дефинираните стилове ще доведе до следния резултат:



8.6. Схеми за позициониране чрез CSS

CSS предлага изобилие от контроли за позициониране на елементите в даден HTML документ. Може да поставим всеки HTML елемент на определено място, което ни харесва. Може да се зададат 4 вида позициониране – *static, relative, absolute, fixed*, *inherit*, задавани чрез свойството *position*. Стойностите на *position* имат следното значение:

- *static* – полето се генерира както обикновено. Блоковите елементи генерират правоъгълно поле, което е част от потока на документа, а вградените полета се генерират в контекста на едно или повече полета за ред;
- *relative* – полето се отмества на някакво разстояние, а елементът запазва формата и пространството, което би имал, ако не беше позициониран. Възможно е позиционираният елемент да застъпи друг елемент. Посоката и степента на отместването се указват чрез свойсвата *top, right, bottom, left*;
- *absolute* – полето е напълно премахнато от потока на документа. Размерът и позицията на елемента се дефинират от свойствата *height, width, top, right, bottom, left*, като са възможни и комбинации с *padding* и *border*;
- *fixed* – полето се позиционира така, сякаш е била зададена стойност absolute, но самият блок е гледната точка. При разглеждане в уеб браузърите, елементът няма да се премества в прозореца на браузъра. Чрез него могат да се реализират заглавия и забележки под линия в документа;
- *inherit* – използва се за явно задаване на наследяването на дадена изчислена стойност от родителския елемент.

Три от схемите за позициониране (*relative, absolute, fixed*) използват 4 различни свойства за описване на страничното отместване на позиционирания елемент. Тези свойства са основните механизми за осъществяване на позиционирането, а именно:

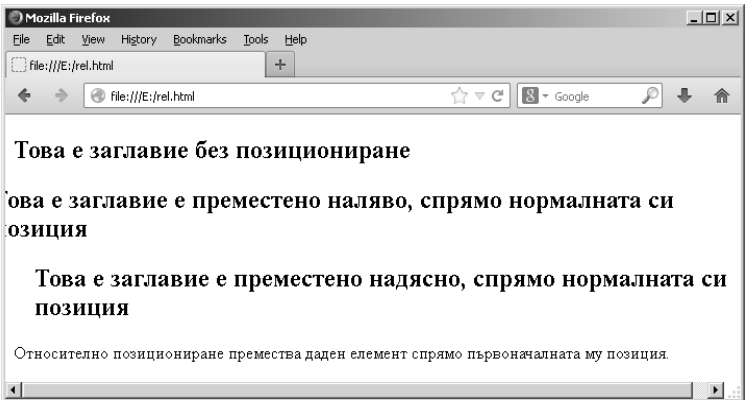
- *top* – определя колко далече трябва да бъде позициониран външният горен край на елемента от върха на съдържащия блок;
- *right* – определя колко далече да бъде позициониран външният десен край на елемента спрямо десния край на съдържащия блок;
- *bottom* – определя колко далече да бъде позициониран външният долен край на елемента от долната страна на съдържащия блок;
- *left* – определя колко далече вдясно (при положителни стойности) или вляво (при отрицателни стойности) да бъде позициони-

ран външният ляв край на елемента спрямо левия край на съдържащия блок.

Пример за относително позициониране на няколко HTML елемента:

```
<style>
  h2.pos_left {position:relative; left:-20px;}
  h2.pos_right{position:relative; left:20px;}
</style>

<h2>Това е заглавие без позициониране </h2>
<h2 class="pos_left">Това заглавие е преместено наляво, спрямо нормалната си позиция</h2>
<h2 class="pos_right">Това заглавие е преместено надясно, спрямо нормалната си позиция</h2>
<p> Относителното позициониране премества даден елемент спрямо първоначалната му позиция.</p>
```

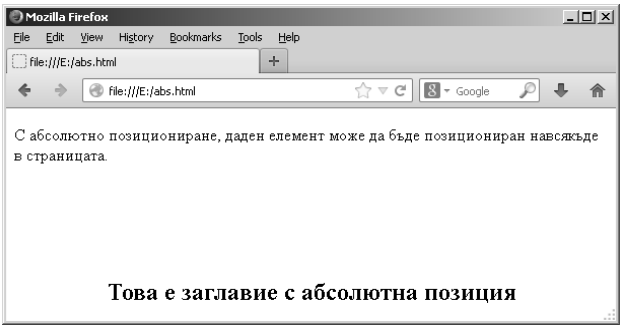
В браузър кодът ще се визуализира по следния начин:



Ето един пример за абсолютно позициониране:

```
<style>
  h2{ position:absolute; left:100px; top:150px;}
</style>

<h2> Това е заглавие с абсолютна позиция </h2>
<p> С абсолютно позициониране даден елемент може да бъде позициониран навсякъде в страницата. </p>
```

В браузър кодът ще се визуализира по следния начин:

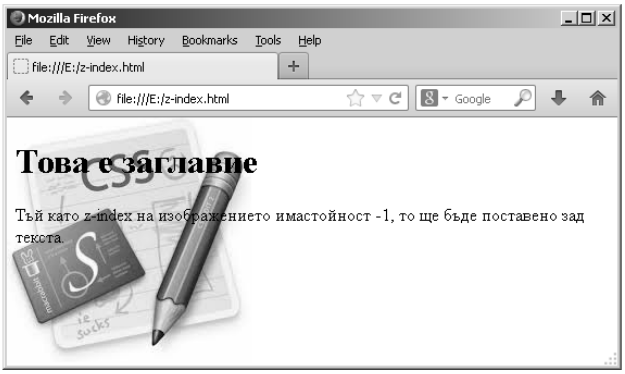


Когато елементите са разположени извън нормалния поток, те могат да се застъпват с други елементи. Свойството *z-index* определя последователността на елементите (кой елемент трябва да бъде поставен отгоре или отдолу на другите елементи). Използвайки подходящи стойности за *position*, елементите могат да се наслагват един върху друг. В такъв случай колкото по-късно в HTML кода е описан даден елемент, на толкова по-предна позиция ще се намира.

Ето един пример за използване на *z-index*:

```
<style>
  img{position:absolute; left:0px;top:0px;z-index:-1;}
</style>

<h1>Това е заглавие</h1>
<img src="image.gif" width="200" height="200" />
<p>Тъй като z-index на изображението има стойност -1, то ще бъде пос-
тавено зад текста.</p>
```

Кодът ще се визуализира в браузъра по следния начин:



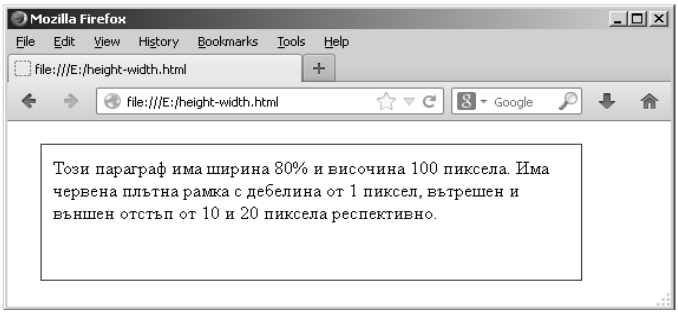
След като вече сме определили къде да бъдат позиционирани елементите, може да определим колко висок и колко широк да бъде съответният елемент. Посредством следните свойства тези размери могат да се контролират:

- *height* – за определяне на височината;
- *width* – за определяне на ширината;
- *line-height* – за определяне на височината на редовете;
- *max-height* – задава максималната височина на един елемент;
- *min-height* – задава минималната височина на един елемент;
- *max-width* – задава максимална ширина на един елемент;
- *min-width* – задава минималната ширина на един елемент.

Свойствата *height* и *width* позволяват да се определи височината и ширината на елементите. Тези свойства могат да имат стойности като дължина, процент или ключова дума *auto*. Ето пример за определяне на ширината и височината на елемента параграф:

```
<p style="width:80%; height:100px; border:1px solid red; padding:10px;
margin:20px;">
Този параграф има ширина 80% и височина 100 пиксела. Има червена плът-
на рамка с дебелина от 1 пиксел, вътрешен и външен отстъп от 10 и 20
пиксела респективно.</p>
```

Кодът ще се визуализира в браузъра по следния начин:



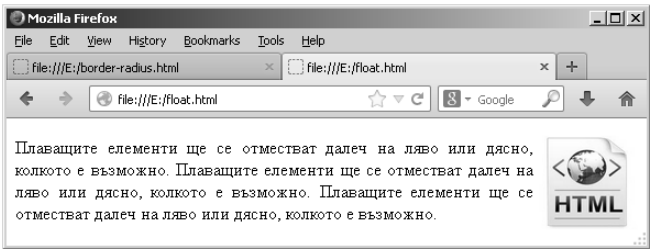
8.7. Плаващи елементи чрез CSS свойството Float

Плаващ елемент означава, че даден елемент може да бъде позициониран само отляво или отдясно, но не горе или долу. Плаващите елементи се отместват наляво или дясно, колкото е възможно, а останалите елементи от основния поток ги заобикалят (т.е. обти-

чат плаващите елементи). Елементите преди плаващия елемент няма да бъдат засегнати. Допустимите стойности за плаващо позициониране на елементите са: *left*, *right*, *none* или *inherit*. Ако изображението е зададено като плаващо вдясно, то останалите елементи ще се разположат от неговата лява страна:

```
<style>
  img {float:right;}
</style>

<p><img src="image.gif" width="95" height="85" /> Плаващите елементи ще се отместват далеч наляво или надясно, колкото е възможно. Плаващите елементи ще се отместват далеч на ляво или дясно, колкото е възможно. Плаващите елементи ще се отместват далеч на ляво или дясно, колкото е възможно. </p>
```

Кодът ще се визуализира в браузъра по следния начин:



Ако поставим няколко плаващи елемента един след друг, те ще бъдат плаващи един спрямо друг. С използването на *float* автоматично се задава ефектът на обтичане – текстът следва формата на плаващия елемент. Ако подобно поведение на текста не е желателно, може да се използва свойството *clear* за разположение на останалите елементи независимо от плаващия обект. *Clear* се използва със следните стойности: *none* – без ефект (обтичането на плаващия елемент продължава), *left* – преустановява се обтичането на плаващия вляво елемент, *right* – не се обтича плаващия вдясно елемент, *both* – няма да се обтичат плаващи както вдясно, така и вляво елементи.

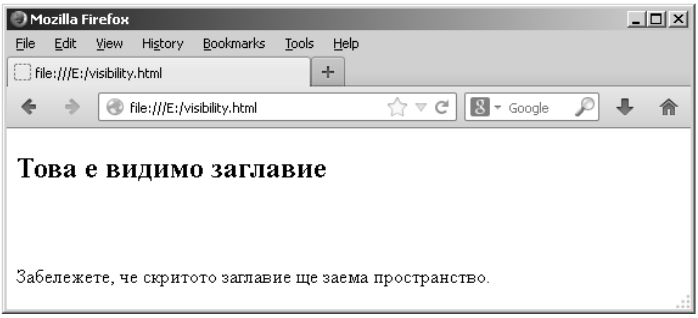
8.8. CSS свойствата Display и Visibility

Свойството *display* определя как да се показва един елемент, а свойството *visibility* уточнява дали един елемент трябва да бъде видим или скрит. Чрез *visibility:hidden* може да скрием елемента, но

въпреки това, той ще заема едно и също място, както и преди. Елементът ще бъде скрит, но все пак ще влияе на оформлението.

```
<style>
  h1.hidden {visibility:hidden;}
</style>

<h1>Това е видимо заглавие</h1>
<h1 class="hidden">Това е скрито заглавие </h1>
<p> Забележете, че скритото заглавие ще заема пространство.</p>
```

Кодът ще се визуализира в браузъра по следния начин:



Свойството *display:none* може да скрие елемента, и той няма да отнеме пространство. Елементът ще бъде скрит, а страницата ще се показва, като ли че елементът не е там:

```
<style>
  h1.hidden {display:none;}
</style>

<h1>Това е видимо заглавие</h1>
<h1 class="hidden">Това е скрито заглавие</h1>
<p>Забележете, че скритото заглавие не заема пространство.</p>
```

Според начина, по който се показват на страницата, елементите могат най-общо да се поделят на блокови и редови. Блоковите елементи (като заглавия *h1~h6*, параграфи *p* и др.) заемат цялата налична ширина на екрана и винаги имат преди и след себе си автоматично включен знак за край на реда. Редовите елементи (като *span*, *a*, *b*, *i*, и др.) от своя страна заемат само толкова пространство, колкото им е необходимо и не включват знак за край на реда.

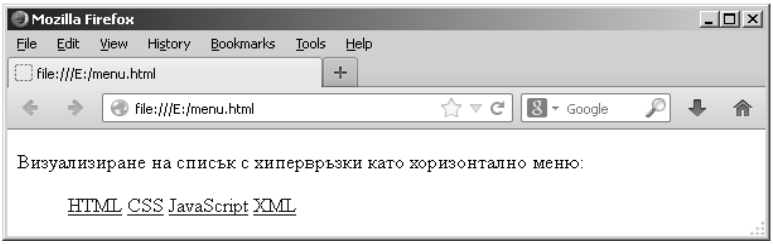
Свойството *display* се използва, за да промени начина на показване на даден елемент, а възможните му стойности са:

- *block* – за представяне на елемента като блоков;
- *inline* – елементът получава свойствата на редови елемент;
- *list-item* – елементът ще бъде показан като елемент от списък;
- *none* – елементът няма да бъде показан.

Ето пример за визуализиране на елементите в ред (*inline*):

```
<style>
  li{display:inline;}
</style>

<p>Визуализиране на списък с хипервръзки като хоризонтално меню:</p>

<ul>
  <li><a href="/html/default.asp" target="_blank">HTML</a></li>
  <li><a href="/css/default.asp" target="_blank">CSS</a></li>
  <li><a href="/js/default.asp" target="_blank">JavaScript</a></li>
  <li><a href="/xml/default.asp" target="_blank">XML</a></li>
</ul>
```

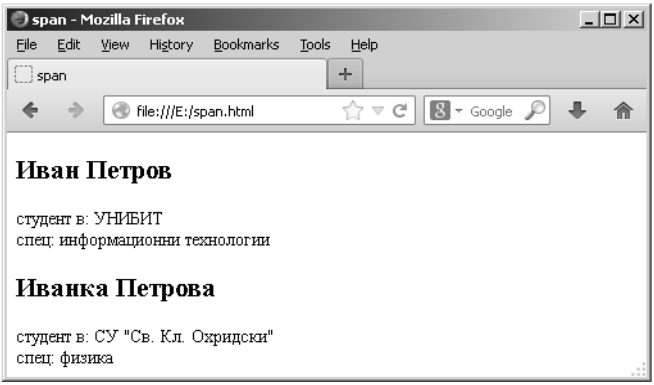
Кодът ще се визуализира в браузъра по следния начин:



Следващият пример илюстрира как *span* елементи (елементи от типа *inline*) могат да се визуализират като блокови елементи с помощта на CSS:

```
<style>
  span{display:block;}
</style>

<h2>Иван Петров</h2>
   <span>студент в: УНИБИТ</span>
   <span>спец: информационни технологии </span>

<h2>Иванка Петрова</h2>
   <span>студент в: СУ "Св. Кл. Охридски"</span>
   <span>спец: физика</span>
```

Кодът ще се визуализира в браузъра по следния начин:



8.9. Създаване на CSS оформление

Първата стъпка при създаването на CSS оформлението е да се определят основните структурни елементи на дизайна. Като основни структурни HTML елементи се използват елементите, формирани от `<div>` таговете. Тези елементи формират правоъгълни контейнери, които могат да бъдат позиционирани и форматирани по подходящ начин със средствата на CSS.

Ще разгледаме един HTML дизайн, илюстриращ оформлението на дедена уеб страница.

Определяме дали оформлението ще изпълва целия прозорец на браузъра или ще има фиксирана ширина. След това определяме `id` на основните контейнери (`<div>`), на които ще зададем определени свойства в CSS файла. Елементът `div` е елемент на ниво блоков елемент, който ще използваме за групиране на други HTML елементи.

Според диаграмата, илюстрирана по-горе, могат да бъдат индентифицирани три основни елемента:

- заглавна част – обикновено включва заглавието на уеб сайта, фоново изображение и името на фирмата (width: 700px; height: 100px);
- основна част – включва по-голямата част от съдържанието на сайта (width: 700px; height: променлива, зависеща от съдържанието);
- долна част – включва информация за авторските права, алтернативен текст за навигация и др. (width: 700px; height: 66px).

Това оформление е добре да бъде центрирано в прозореца на браузъра. Сега вече разполагаме с цялата необходима информация, за да започнем оформлението на дизайна.

В тялото на документа (между таговете `<body> </body>`) трябва да създадем три области (`div`), всяка от които с индивидуален уникален идентификатор `id`. Тези области ще съответстват на основните части на оформлението, които вече идентифицирахме. В HTML документа дефинираме следните 4 области:

```
<body>
   <div id="wrapper">
       <div id="header">Заглавна част </div>
  <div id="content"> Основна част – съдържание </div>
  <div id="footer"> Долна част </div>
   </div>
</body>
```

В CSS файла дефинираме правила за стиловете, които искаме да присвоим на определените области от оформлението:

```
* { padding: 0; margin: 0; }

body { font-family: Arial, Helvetica, sans-serif;
       font-size: 13px;   text-align: center;}

#wrapper { margin: 0 auto; width: 722px; }

#header { width: 700px; height: 100px; padding: 10px;
 margin: 10px 0px 5px 0px; border: 1px double #006699;
 color: #FFFFCC;  font-size: 18px; background: #78A7C6; }
```

```
#content { width: 700px; height: 250px; padding: 10px;
 margin: 0px 0px 5px 0px; border: 1px dotted #006666;
 color: #000000; background: #F2F2E6; text-align: justify; }

#footer { width: 700px; height: 66px; padding: 5px;
 margin: 0px 0px 10px 0px; color: #000000; background: #99CCFF;
 border: 1px solid #CCCCCC; }
```

Уникалният идентификатор `id` се прилага за елементи, които само веднъж ще се използват в страницата. Следователно, за описаните елементи като заглавна част, основна част и долна част е допустимо да се използват идентификатори, докато за хипервръзките би трябвало да използваме класове, тъй като те се срещат повече от един път на една и съща страница.

Нека създадем оформление на HTML документ, съдържащо четири основни области, дефинирани като заглавна част и долна част, а частта между тях да разделим на две като лява колона и дясна колона, както е илюстрирано:

HTML кодът на това представяне ще е следният:

```
<body>
   <div id="wrapper">
       <div id="header">Заглавна част</div>
  <div id="leftcolumn">Лява колона</div>
  <div id="rightcolumn">Основна част – съдържание </div>
  <div id="footer">Долна част </div>
   </div>
</body>
```

В CSS файла дефинираме правила за стиловете, които искаме да присвоим на определените области от оформлението:

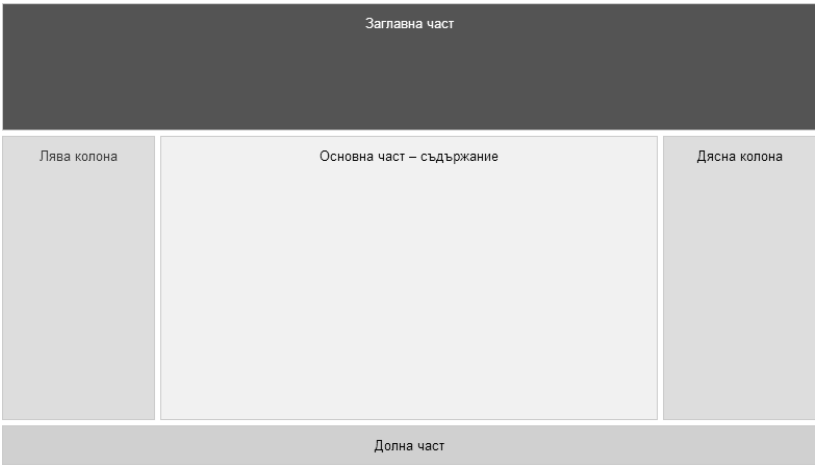
```
* { padding: 0; margin: 0; }

body {font-family: Arial, Helvetica, sans-serif;
      font-size: 13px; text-align: center;}

#wrapper { margin: 0 auto; width: 700px; }

#header { width: 700px; height: 100px; padding: 10px;
 margin: 10px 0px 5px 0px; border: 1px double #006699;
 color: #FFFFCC;  font-size: 18px; background: #78A7C6; }

#leftcolumn { float: left; width: 195px; height: 250px;
 margin: 0px 0px 5px 0px; padding: 10px;color: #000000;
 border: 1px dashed #006699; background: #E7DBD5; }

#rightcolumn { float: right; width: 495px; height: 250px;
 text-align: justify; margin: 0px 0px 5px 0px; padding: 10px;
 color: #000000; border: 1px dotted #006699;
 background: #F2F2E6; display: inline; }

#footer { width: 700px; margin: 0px 0px 10px 0px; padding: 5px;
 color: #000000; background: #99CCFF;
 border: 1px solid #CCCCCC; }
```

Ако е необходимо в лявата колона от дизайна да се разположи съдържанието, то оформлението ще изглежда по следния начин:

HTML кодът в този случай се запазва същия, а в CSS кода трябва да се промени само съответно ширината на лявата и на дясната колони:

```
...
#leftcolumn { float: left; width: 495px; height: 250px;
 margin: 0px 0px 5px 0px; padding: 10px;color: #000000;
 border: 1px dashed #006699; background: #E7DBD5; }

#rightcolumn { float: right; width: 195px; height: 250px;
 text-align: justify; margin: 0px 0px 5px 0px; padding: 10px;
 color: #000000; border: 1px dotted #006699;
 background: #F2F2E6; display: inline; }
...
```

Нека сега създадем HTML дизайн, чието оформление включва заглавна и долна част, а средната част е разделена на три, като лява колона, дясна колона и съдържание, както е показано:



За реализирането на това оформление използваме HTML код, който описва пет основни области на оформлението:

```
<body>
   <div id="wrapper">
       <div id="header"> Заглавна част </div>
  <div id="leftcolumn"> Лява колона </div>
  <div id="content">Основна част – съдържание </div>
  <div id="rightcolumn"> Дясна колона </div>
  <div id="footer"> Долна част </div>
   </div>
</body>
```

В CSS файла дефинираме правилата за стиловете, които искаме да присвоим на определените области от оформлението:

```
* { padding: 0; margin: 0; }

body {font-family: Arial, Helvetica, sans-serif;
      font-size: 13px; text-align: center;}

#wrapper {margin: 0 auto; width: 782px;}

#header {float: left; width: 760px; height: 100px;
 margin: 10px 0px 5px 0px; padding: 10px;
 border: 1px solid #CCCCCC; background: #006699;color: white; }

#leftcolumn {float: left; width: 125px; height: 250px;
 margin: 0px 5px 5px 0px; padding: 10px;
 border: 1px solid #CCCCCC; background: #E7DBD5;color: #333; }

#content {float: left; width: 456px; height: 250px;
 display: inline; margin: 0px 5px 5px 0px; padding: 10px;
 border: 1px solid #CCCCCC; background: #F2F2E6;color: black;}

#rightcolumn { float: left; width: 125px; height: 250px;
 margin: 0px 0px 5px 0px; padding: 10px;
 border: 1px solid #CCCCCC; background: #E7DBD5; color: black;}

#footer { width: 700px; margin: 0px 0px 10px 0px; padding: 5px;
 color: #000000; background: #99CCFF;
 border: 1px solid #CCCCCC; }
```

Нека да създадем оформление, което да изпълва целия прозорец на браузъра, както е показано:

За тази реализация използваме следният HTML код:

```
<div id="container">
  <div id="header">
     <h1>Име на сайта/лого</h1>
  </div>
  <div id="navigation">
     <ul>
       <li><a href="#">Начало</a></li>
       <li><a href="#">За нас</a></li>
       <li><a href="#">Услуги</a></li>
       <li><a href="#">Контакти</a></li>
     </ul>
  </div>
  <div id="content-container">
     <div id="content">
     <h2>Заглавие</h2>
     <p>Първо трябва да се определят основните структурни елементи на
дизайна на HTML документа. Определяме ID на основните контейнери, кои-
то ще асоцииране със съответните свойства в CSS файла. </p>
     </div>
       </div>
  <div id="footer">Copyright © Site name, 20XX</div>
</div>
```

В CSS файла дефинираме правилата за стиловете, които искаме да присвоим на определените области на оформлението:

```
#container {margin: 0 auto;width: 100%;background: #F1F1F1;}

#header {background: #D4E3FF; padding: 20px;
         text-align: center;}

#header h1 {margin: 0;}

#navigation {float: left; width: 100%; background: #0B4353;}

#navigation ul {margin: 0; padding: 0;}

#navigation ul li{list-style-type: none;display: inline;}

#navigation li a {display: block; float: left;
 padding: 5px 10px; color: #FFFF33;
 text-decoration: none; border-right: 1px solid #fff;}

#navigation li a:hover {background: #383;}

#content-container {float: left; width: 100%;
 background: #FFFFCC url(image.gif) repeat-y 68% 0;}

#content {clear: left; float: left; width: 92%;
 padding: 20px 0; margin: 0 0 0 4%; display: inline;
```

```
  text-align:justify;}

#content h2 {margin: 0;}

#footer {clear: left; height: 1%; padding: 10px;
  background: #D4E3FF; text-align: center;}
```

За реализиране на различни оформления на дизайна на HTML документа, може да се използват и наличните Интернет ресурси (виж. http://csslayoutgenerator.com/).

Глава 9. Въведение в JavaScript. Синтаксис, типове данни и променливи, оператори и функции

JavaScript е олекотен, интерпретиран език за програмиране с обектно ориентирани възможности, които позволяват изграждането на интерактивност в статичните HTML страници. JavaScript е разработен от Брендан Ейч, от екипа на Netscape, под името Mocha. По-късно е преименуван на LiveScript и това е официалното име на езика, когато за първи път е пуснат в бета-версиите на браузъра Netscape Navigator 2, а по-късно през 1995 г. (вер. 2 на Netscape Navigator) е преименуван на JavaScript. Microsoft също въвежда подобен на JavaScript език т. нар. JScript в своя Internet Explorer 3. Езикът JavaScript се поддържа от всички интернет браузъри и е стандартизиран от Европейската асоциация на производителите на компютри (ECMA).

В тази глава е описан синтаксисът на езика JavaScript, разполагането му в HTML документите, поддържаните типове данни и декларирането на променливите. Показани са типовете оператори и дефинирането на функциите.

9.1. JavaScript – обща информация

Кодът на JavaScript се поставя в HTML документа посредством таговете `<script> ... </script>`. JavaScript кодът може да бъде вмъкнат навсякъде в рамките на уеб страницата, но се препоръчва поставянето му в секцията `<head>`. Тагът `<script>` уведомява браузъра да започне тълкуването на текста между отварящия и затварящия таг като скрипт. Пример за JavaScript код:

```
<script ...>
  JavaScript code
</script>
```

Тагът `<script>` има два важни атрибута:

- *language* – определящ какъв е скриптовият език, който се използва. Обикновено стойността му е *"javascript"*, въпреки че пос-

ледните версии на HTML (XHTML) преустановяват използването на този атрибут;

- *type* – показващ, че това е скриптов език и неговата стойност трябва да е *"text/javascript"*.

Следователно, отварящият таг на JavaScript ще има вида:

```
<script language="javascript" type="text/javascript">
   JavaScript code
</script>
```

Нека с помощта на JavaScript скрипт напишем "*Hello World*" в уеб страницата.

```
<html>
<body>
<script language="javascript" type="text/javascript">
<!--
    document.write("Hello World!")
//-->
</script>
</body>
</html>
```

Добавихме HTML коментар, който заобикаля JavaScript кода. По този начин скриваме JavaScript кода от браузър, неподдържащ JavaScript. Коментарът завършва с "//-->". Тук "//" означава коментар от JavaScript, като част от кода. Извикваме функцията *document.write*, която е една от основните функции на JavaScript. Тя прави възможно отпечатването на зададения в скобите и заграден от кавичките в уеб страницата текст. Следователно, изпълнението на горния скрипт ще даде следния резултат:

```
Hello World!
```

Празни символи. JavaScript игнорира интервалите, табулациите и новите редове в програмния код. Следователно интервали, табулации и нови редове могат да се използват за по-доброто структуриране на JavaScript кода.

Точка и запетая не са задължителни. Простите записи в JavaScript обикновено са последвани от знака точка и запетая, точно както в езиците C, C ++ и Java. JavaScript обаче позволява пропускане на точката и запетаята, ако записите са поставени на отделни редове.

```
<script language="javascript" type="text/javascript">
<!--
   var1 = 10
   var2 = 20
//-->
</script>
```

Когато използваме запис на един ред (по-кратък запис), точката и запетаята са задължителни:

```
<script language="javascript" type="text/javascript">
<!--
   var1 = 10; var2 = 20;
//-->
</script>
```

Добра практика в JavaScript програмирането е винаги да се използва точката и запетаята.

Малки и главни букви: JavaScript прави разлика между главни и малки букви. Това означава, че ключовите думи на езика, променливите, имената на функциите, както и другите идентификатори, трябва винаги да бъдат изписвани по един и същ начин – само с малки, само с големи или с главна буква. Например, променливите *name*, *Name* и *NAME* ще бъдат интерпретирани като 3 различни променливи в JavaScript. Следователно, трябва да бъдем много прецизни при въвеждането на имената на променливите и на функциите в JavaScript.

Коментари в JavaScript. JavaScript поддържа стила на езика C и на езика C++ по отношение на коментарите:

- Всеки текст между `//` до края на реда ще се интерпретира като коментар и се игнорира от JavaScript.
- Всеки текст между символите `/*` и `*/` се интерпретира като коментар. Този коментар може да се простира на няколко реда.
- JavaScript разпознава също отварящия запис за HTML коментар `<!--`. JavaScript интерпретира това като коментар от един ред, точно както го прави коментарът `//`.
- Затварящият запис за HTML коментар `-->` не се интерпретира от JavaScript, така че трябва да бъде написано като `//-->`.

Например:

```
<script language="javascript" type="text/javascript">
<!--
  // Това е просто коментар, подобен на коментарите в C++
  /*
   * Това е многоредов коментар в JavaScript
   * Записът е подобен на задаването на коментари в
   * програмния език C
   */
  //-->
</script>
```

9.2. Разполагане на JavaScript в HTML документ

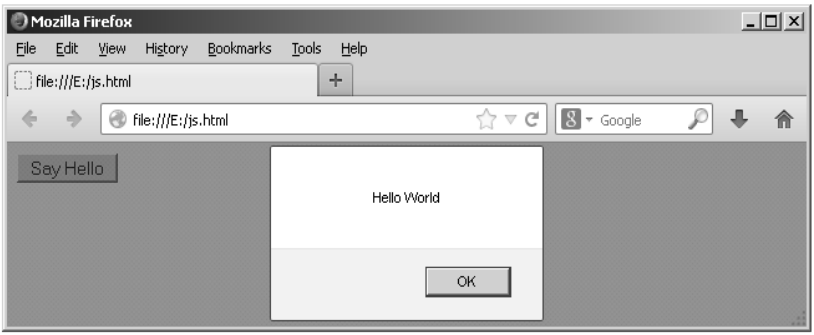
Има известна гъвкавост по отношение на разполагането на JavaScript кода в HTML документа. Най-често използваните начини за разполагане на JavaScript в HTML документа са следните:

- JavaScript в секцията `<head>...</head>`.
- JavaScript в секцията `<body>...</body>`.
- JavaScript в `<body>...</body>` и `<head>...</head>`.
- JavaScript във външен файл и включен в секцията `<head>...</head>`.

Сега ще покажем няколко примера за различно разполагане на JavaScript. Използване на JavaScript в `<head>...</head>` секцията. Ако искаме изпълнението на скрипта да реагира на събитие, например на кликане с мишката от потребителя, то би трябвало този скрипт да се постави в *head* секцията, както е показано:

```
<html>
<head>
<script type="text/javascript">
   <!--
   function sayHello() {
        alert("Hello World")
   }
   //-->
</script>
</head>
<body>
<input type="button" onclick="sayHello()" value="Say Hello" />
</body>
</html>
```

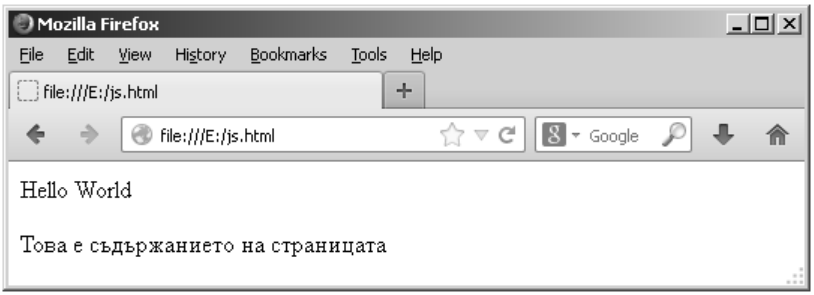
Активирането на бутона *„Say Hello“* ще доведе до следния резултат – показване на прозорец с надпис *„Hello World“*:



Използване на JavaScript в секцията `<body>...</body>`. Ако е необходимо скриптът да се изпълни при зареждане на страницата, така че да се генерира съдържание на нея, неговото място е в секцията *body* на документа. В този случай няма да има дефиниране на функция за използването на JavaScript:

```
<html>
<head> </head>
<body>
<script type="text/javascript">
   <!--
   document.write("Hello World")
   //-->
</script>
<p>Това е съдържанието на страницата</p>
</body>
</html>
```

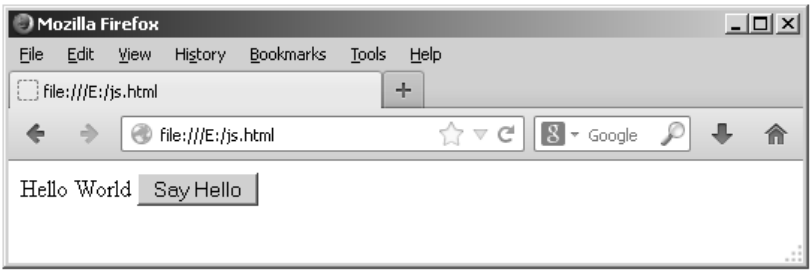
Това ще доведе до следния резултат:



Използване на JavaScript в секциите `<body>` и `<head>`. Части от JavaScript кода може да се поставят в секциите `<head>` и `<body>`:

```
<html>
<head>
   <script type="text/javascript">
      <!--
      function sayHello() {
          alert("Hello World")
      }
      //-->
   </script>
</head>
<body>
   <script type="text/javascript">
   <!--
       document.write("Hello World")
   //-->
</script>
<input type="button" onclick="sayHello()" value="Say Hello" />
</body>
</html>
```

Изпълнението на горния код ще доведе до следния резултат:



JavaScript във външен файл. Когато започнем да работим по-интензивно с JavaScript, най-вероятно ще открием, че има случаи, в които се налага използване на идентичен JavaScript код в няколко страници на един сайт. Разбира се, няма ограничение да се поддържа идентичен код в множество HTML файлове. Има по-добра възможност за съхраняване на JavaScript във външен файл, който да бъде включен в HTML. Ето един пример, който показва как да бъде включен външен JavaScript файл (например, *filename.js*) в HTML кода, използвайки атрибута *src* на тага `<script>`:

```
<html>
<head>
<script type="text/javascript" src="filename.js" ></script>
</head>
```

```
<body>
.......
</body>
</html>
```

9.3. Типове данни и променливи

Една от основните характеристики на даден език за програмиране е наборът от типове данни, които той поддържа. Това е типът на стойностите, които могат да бъдат представени и манипулирани в език за програмиране. JavaScript позволява да се работи с три примитивни типове данни:

- числа – като 123; 120.50; 6.02e11;
- низове – като "This text string";
- булеви (логически) – като *true* или *false*.

Всички числа в JavaScript се представят като числа с плаваща запетая. JavaScript представя числата в 64-битов формат с плаваща точка, определен от стандарта IEEE 754. Низът в JavaScript е последователност от произволви букви, цифри и други символи. Логическият тип данни има две възможни стойности, представени в JavaScript с ключовите думи *true* и *false*. Тези стойности означават истина или лъжа, включено или изключено, да или не и всичко друго, което може да бъде представено с един бит информация.

JavaScript определя още два тривиални типа данни, *null* и *undefined*, всеки от който определя само една стойност. Ключовата дума *null* е специална стойност, която означава „без стойност“. Ако една променлива има стойност *null*, то тя не съдържа валидна стойност от какъвто и да е тип. *Undefined* е стойността, която се връща, ако се използва недекларирана или неинициализирана променлива, или ако се използва несъществуващо свойство на обекта.

За променливите в JavaScript може да се мисли като за наименувани контейнери. Данните могат да се поставят в тези контейнери, а достъпът до тях става чрез името на контейнера. Преди да се използва дадена променлива в JavaScript, трябва предварително да се декларира. Променливи се декларират чрез ключовата дума *var*, както е показано:

```
<script type="text/javascript">
<!--
    var money;
```

```
    var name;
//-->
</script>
```

Може също така да се декларират множество променливи чрез една и съща ключовата дума *var*:

```
<script type="text/javascript">
<!--
    var money, name;
//-->
</script>
```

Съхраняването на стойност в променлива се нарича „инициализация на променливата“. Може да се направи инициализация на променливата по време на нейното създаване или на по-късен етап, когато е необходимо тя да бъде използвана.

Нека да създадем променлива с име *money* и след това да ѝ присвоим стойност 2000.50. Променливата *name* получава стойност по време на нейната декларация, както е показано:

```
<script type="text/javascript">
<!--
    var name = "Ali";
    var money;
    money = 2000.50;
//-->
</script>
```

Ключовата дума *var* може да се използва за деклариране или инициализация само веднъж в съществуването на всяка променлива в даден документ. Не се допуска два пъти да се декларира една и съща променлива.

Числовите променливи се представят като числа, които не са заградени в кавички. Числата в Java Script се представят в 64-битов формат с плаваща запетая. Java Script не прави разлика между цели числа и числа с плаваща запетая. Целите числа могат да бъдат представяни в осмична или шестнайсетична бройна система. Литералът в осмичната бройна система започва с 0 (нула), а този в шестнайсетичната с 0x:

```
var x = 56;      //десетично представяне
var y = 34e-5;  // десетично представяне – научен формат
var z = 034;     // осмично представяне
var s = 0x23F;  // шестнайсетично представяне
```

Низовите променливи представляват букви, цифри или други символи, както и техни комбинации, поставени в кавички (всичко в кавичките се интерпретира като текст):

```
var text = "Hello wolrd!"
var text1 = "40";
vat text2 = "Низова променлива + 40";
```

Възможно е използването на комбинация от единични и двойни кавички:

```
var text = 'Използване на "кавички" в низовата променлива';
```

Масивите са тип променливи, които съдържат множество елементи. Ето една структура на празен масив с неопределен брой елементи, където `masiv` е името на масива, а `new Array()` е начинът на деклариране му:

```
masiv=new Array()
```

Вместо празни скоби, можем да зададем броя на елементите в масива:

```
masiv=new Array(10)
```

Броят на елементите винаги трябва да бъде цяло неотрицателно число. В горния случай създадохме масив с име `masiv` и 10 елемента със стойност *NULL*, т.е. празни елементи. Ако не зададем предварително броя на елементите в масива, това можем да направим по-късно, като задаваме стойност на всеки елемент поотделно. Самото индексиране на елементите става с квадратни скоби [ ], като първият индекс е нула. Така първият елемент в масив с десет елемента е `masiv[0]`, а последният – `masiv[9]`. Например: масив със седем елемента и зададени стойности – дните от седмицата:

```
masiv=new Array(7)
masiv[0]="понеделник"
masiv[1]="вторник"
masiv[2]="сряда"
masiv[3]="четвъртък"
masiv[4]="петък"
masiv[5]="събота"
masiv[6]="неделя"
```

Същото нещо можем да направим по по-кратък начин:

```
masiv = new Array ("понеделник","вторник", "сряда",
                   "четвъртък", "петък","събота","неделя")
```

Така вече всеки елемент от масива има някаква стойност. Тези стойности могат да бъдат променяни по всяко време, както и типа на елемента. Например, ако напишем `masiv[5]=122`, то шестият елемент ще бъде от целочислен тип, докато останалите ще са от низов тип. За осигуряване на достъп до елементите на масива може да се използва на цикъл:

```
masiv=new Array ("понеделник","вторник", "сряда", "четвъртък"
                 "петък","събота","неделя")
for(i=0;i<7;i++)
{
     document.write(masiv[i] + "<br>")
 }
```

Този скрипт ще покаже на екрана всеки един елемент от масива, като след всеки елемент ще минава на нов ред.

JavaScript е типизиран език. Това означава, че променливата в JavaScript може да съдържа стойност от всякакъв тип данни. За разлика от много други езици, в JavaScript не е нужно да се укаже при самото деклариране на променливата от какъв тип ще бъде. Стойността на типа на променливата може да се променя по време на изпълнението на програмния код и JavaScript се грижи за това автоматично. Променливите в JavaScript имат два обхвата на действие:

- **глобални променливи** – когато променливата има глобален обхват, тя може да се дефинира навсякъде в JavaScript кода;
- **локални променливи** – локалната променлива е видима само в рамките на функцията, където тя е дефинирана.

В рамките на тялото на функцията, локалната променлива има предимство пред глобална променлива със същото име. Ако декларираме локална променлива или параметър на функция със същото име като глобалната променлива, това прави възможно ефективното скриване на глобалната променлива. Следващият пример обяснява твърдението:

```
<script type="text/javascript">
<!--
   var myVar = "global"; // Declare a global variable
   function checkscope( ) {
      var myVar = "local";  // Declare a local variable
      document.write(myVar);
```

```
}
//-->
</script>
```

Изпълнението на скрипта води до следния резултат:

```
local
```

Имена на променливи в JavaScript. При задаване на имената на променливите в JavaScript трябва да се спазват следните правила:

- Не трябва да се използват ключовите думи в JavaScript, като имена на променливи, функции, методи, етикети или някакви други имена на обекти. Тези ключови думи са:

abstract	else	instanceof	switch
boolean	enum	int	synchronized
break	export	interface	this
byte	extends	long	throw
case	false	native	throws
catch	final	new	transient
char	finally	null	true
class	float	package	try
const	for	private	typeof
continue	function	protected	var
debugger	goto	public	void
default	if	return	volatile
delete	implements	short	while
do	import	static	with
double	in	super	

- Имената на променливите в JavaScript не трябва да започват с цифра (0-9). Трябва да започват с букви или долна черта. Например, *123test* е невалидно име на променлива, но *_123test* е валидно име.
- JavaScript прави разлика между главни и малки букви. Така например, *Name* и *name* са две различни променливи.

9.4. Обекти

JavaScript е обектноориентиран език. Обектът е съставен тип данни, който съдържа произволен брой свойства. Всяко едно свойство притежава име и стойност. Обектите в JavaScript се описват заедно със свойствата и методите им. Методите на обектите се отнасят до начина на изпълнение на самия обект.

Нека разгледаме пример за обект от реалния свят, какъвто е *прозореца*. Методите на обекта *прозорец* ще бъдат отваряне и затва-

ряне, а свойствата му – например ширина и височина. Вече използвахме един обект заедно със метода му в почти всички примери дотук. Това е обектът *document* заедно с метода му *write*. Операторът за достъп до именуваното свойство на даден обект е разделителната точка ".", например *document.write* или *image.border* и др. В JavaScript най-често срещаният обект е прозореца на браузъра, познат под името *window*, който може да се отваря и затваря с помощта на JavaScript. Можем да използваме JavaScript и за извеждане на информация в статус бара (status bar) на прозореца на браузъра.

Ето и някои от най-често срещаните обекти, заедно с методите и свойствата им:

обект	свойства	методи
document	bgColor; image; location; title	write; writeln; open
image	border; height; width; src	
window	location; history; frames; name	close; open; prompt; scroll

Можем да четем или да записваме стойностите на свойството на обекта *ob* по следния начин:

```
ob.x=1;
ob.y=2;
ob.total=ob.x + ob.y;
```

Достъпът до свойствата на обектите може да се осъществи и чрез използването на масив:

```
ob["x"] = 1;
ob["y"] = 2;
```

9.5. Оператори

Изразите в JavaScript се формират като се комбинират стойности на литерали и променливи посредством операторите. В JavaScript се поддържат следните типове оператори:

- аритметични оператори;
- оператори за присвояване;
- оператори за сравнение;
- логически (или релационни) оператори;
- побитови оператори.

За да илюстрираме действията на **аритметичните оператори** ще приемем, че променливата A има стойност 10, а променливата B има стойност 20, тогава:

Оператор	Описание	Пример
+	Сумира два операнда	A + B ще даде 30
-	Изважда втория операнд от първия	A - B ще даде -10
*	Умножава двата операнда	A * B ще даде 200
/	Разделя числителя на знаменателя	B / A ще даде 2
%	Целочислено деление – резултатът е целочисления остатък от делението	B % A ще даде 0
++	Оператор за увеличаване на стойността с единица	A++ ще даде 11
--	Оператор за намаляване на стойността с единица	A-- ще даде 9

Операторът събиране (+) работи както с числови променливи, така и с низове. Например: "a" + 10 ще даде "a10":

```
var num1=a;
var num1="10";
var sum=num1+num2; //събират се като низове
window.alert(sum);
```

Оператори за присвояване в JavaScript:

Оператор	Описание	Пример
=	Оператор за присвояване. Присвоява стойността от десния операнд към стойността на левия операнд.	C = A + B ще присвои стойност A + B на C
+=	Присвояване със съответната операция за събиране.	C += A е еквивалентно на C = C + A
-=	Присвояване със съответната операция за изваждане.	C -= A е еквивалентно на C = C – A
*=	Присвояване със съответната операция за умножение.	C *= A е еквивалентно на C = C * A
/=	Присвояване със съответната операция за деление.	C /= A е еквивалентно на C = C / A
%=	Взима целочисления остатък от делението, а след това я присвоява на променливата.	C %= A е еквивалентно на C = C % A

Операторите за сравнение често се използват с условни конструкции и цикли за извършване на дадено действие, само ако е изпълнено определено условие. Тъй като тези оператори сравняват две стойности, те връщат стойностите *true* или *false*, в зависимост от стойностите от двете страни на оператора. Приемаме, че променливата A има стойност 10, а променливата B има стойност 20, то тогава:

Оператор	Описание	Пример
==	Проверява дали стойностите на два операнда са равни или не, ако да – условието се превръща в истина.	(A == B) не е вярно
!=	Проверява дали стойностите на два операнда са равни или не, ако стойностите не са равни то условието се превръща в истина.	(A != B) е вярно
>	Проверява дали стойността на левия операнд е по-голяма от стойността на десния операнд, ако е да – условието се превръща в истина.	(A > B) не е вярно
<	Проверява дали стойността на левия операнд е по-малка от стойността на десния операнд, ако е да – условието се превръща в истина.	(A < B) е вярно
>=	Проверява дали стойността на левия операнд е по-голяма или равна на стойността на десния операнд, ако е да – условието се превръща в истина.	(A >= B) не е вярно
<=	Проверява дали стойността на левия операнд е по-малка или равна на стойността на десния операнд, ако е да – условието се превръща в истина.	(A <= B) е вярно

Логическите оператори позволяват да се сравняват променливи (изрази) и връщат *true* или *false*, в зависимост от стойностите им. Нека приемем, че променливата X има стойност 20, а променливата Y има стойност 10, то тогава:

Оператор	Описание	Пример
&&	Логически оператор AND. Връща true, ако стойностите от двете страни на оператора са true.	(X <10 && Y>1) е вярно
||	Логически оператор OR. Връща true, ако поне една стойност от двете страни на оператора е true.	(X==5 || X==5) не е вярно
!	Логически оператор NOT. Връща обратната стойност на стойността от дясната страна на оператора.	!(X==Y) е вярно

9.6. Функции

Функцията представлява група от многократно използван програмен код, който може да бъде извикан от всяко място в програмата. Това подпомага писането на модулен код. Дефинирането на функция в JavaScript се реализира чрез използването на ключовата дума `function`, следвана от уникално име на функцията, списък с параметри (може и да е празен) и оператори във фигурни скоби, като се следва основният синтаксис:

```
<script type="text/javascript">
<!--
function functionname(parameter-list)
{
  statements
}
//-->
</script>
```

Пример за проста функция с име *sayHello*, която няма параметри:

```
<script type="text/javascript">
<!--
function sayHello()
  {   alert("Hello there");
  }
//-->
</script>
```

За да се извика функцията някъде от скрипта, е необходимо да се запише името на тази функция:

```
<script type="text/javascript">
<!--
    sayHello();
//-->
</script>
```

Досега разгледахме функции без параметри, но съществуват и предимства при предаването на параметри, когато извикваме функцията. Предаваните параметри могат да бъдат манипулирани и върнати със съответните им стойности. Могат да бъдат задавани множество параметри, разделени със запетая.

Пример: Нека направим малка промяна във функцията *sayHello*, като добавим два параметъра:

```
<script type="text/javascript">
<!--
function sayHello(name, age)
{
   alert( name + " is " + age + " years old.");
}
//-->
</script>
```

Използването на + оператора позволява да свържем заедно низ и число. JavaScript позволява добавяне на числа в низове. Сега мо-

жем да извикаме тази функция по следния начин:

```
<script type="text/javascript">
<!--
    sayHello('Ivan', 21 );
//-->
</script>
```

Изпълнението на JavaScript кода, след извикване на функцията `sayHello('Ivan', 21 )`, ще доведе до следния резултат:

JavaScript функцията може да има оператор *return*, за връщане на стойности. Това е необходимо, ако искаме да се върне стойност на променливата, обработена в тялото на функцията. Тази декларация трябва да бъде последна в декларирането на функцията. Например можем да предадем две числа във функция и след това да очакваме от функцията да върне тяхното умножение, след изпълнението на функцията.

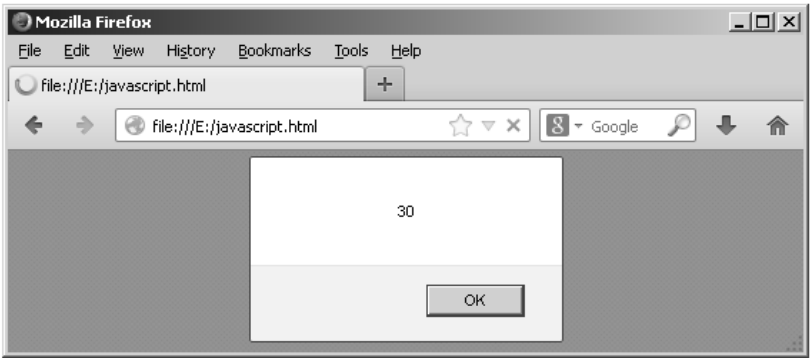
Пример за функция с два параметъра и връщане на получения резултат от изпълнението:

```
<script type="text/javascript">
<!--
function concatenate(first, last)
{
   var full;
   full = first + last;
   return  full;
}
//-->
</script>
```

Сега можем да извикаме тази функция по следния начин:

```
<script type="text/javascript">
<!--
   var result;
   result = concatenate(9, 21);
   alert(result );
//-->
</script>
```

Изпълнението на JavaScript кода, след извикване на функцията `concatenate (9, 21)`, ще доведе до следния резултат:



Глава 10. JavaScript – условия, цикли, събития, обекти Date и Math, регулярни изрази

В тази глава са описани някои от основните действия (събития), които се извършват върху уеб страницата и които могат да бъдат манипулирани със средствата на JavaScript. Представен е синтаксисът на често използваните цикли, изпълняващи поредица от команди до удовлетворяването на дадено условие. Показани са възможностите на обектите Date и Math, както и използването на т.нар. „регулярни изрази“ за проверка на информацията, изпращана чрез формулярите.

10.1. JavaScript if...else конструкции

При писането на програми има ситуации, когато трябва да се избере един от два възможни клона. Това може да бъде реализирано от условните конструкции, които позволяват на програмата да взема правилни решения и да изпълнява правилни действия. JavaScript поддържа условни конструкции, които се използват за извършване на различни действия, базирани на различни условия. Тук ще обясним конструкцията `if...else`. JavaScript поддържа следните форми на конструкцията `if...else`:

- `if`;
- `if...else`;
- `if...else if...`

Операторът `if` е фундаментален управляващ опертор, който позволява на JavaScript да взема решения и да изпълнява оператори, съгласно някакво условие. Синтаксисът е:

```
if (expression)
{
   Statement(s) to be executed if expression is true
}
```

Трябва да отбележим, че изразът, в който се задава условието с оператора за сравнение, се поставя в кръгли скоби (*expression*), а операторите, която трябва да се изпълнят, ако е изпълнено условието, са

поставени във фигурни скоби {*Statement(s)*}. Такъв вид конструкция връща като резултат булева стойност от израза в кръглите скоби, чиято истинност се проверява, т.е. върнатият резултат е *true* или *false*. Ако резултатът е *true*, тогава се изпълнява операцията, зададена в големите скоби {}, а ако не – операторите във фигурните скоби се пропускат и се изпълнява операторът след последната фигурна скоба.

В повечето случаи се налага използването на оператори за сравнение за вземането на решения. Например:

```
<script type="text/javascript">
<!--
    var age = 20;
    if( age > 18 ){
       document.write("<b>Отговаря на изискванията да бъде шо-
фьор</b>");
    }
//-->
</script>
```

Изпълнението на скрипта ще доведе до следния резултат:

```
Отговаря на изискванията да бъде шофьор
```

Конструкцията `if...else` е друга форма за условен оператор, който позволява изпълнение с по-голям контрол. Синтаксисът е:

```
if (expression){
    Statement(s) to be executed if expression is true
} else{
    Statement(s) to be executed if expression is false
}
```

Тази конструкция се тълкува по следния начин: Ако поставеното условие `if` *(expression)* е изпълнено (връща резултат *true*) се изпълнява оператора/ите в следващите фигурни скоби {}. Ако то не е изпълнено (връща резултат *false*) – тогава се изпълнява оператора/ите във фигурните скоби, в секцията `else` {}. Например:

```
<script type="text/javascript">
<!--
   var age = 15;
   if( age > 18 ){
      document.write("<b>Отговаря на изискванията за шофьор </b>");
   }else{
       document.write("<b>Не отговаря на изискванията за шофьор
</b>");
   }
```

```
//-->
</script>
```

Изпълнението на скрипта ще доведе до следния резултат:

```
Не отговаря на изискванията за шофьор
```

Конструкцията `if...else if...` е следващото ниво на управление, което позволява на JavaScript да вземе правилно решение след няколко последователни условия. Синтаксисът е:

```
if (expression 1){
   Statement(s) to be executed if expression 1 is true
}else if (expression 2){
   Statement(s) to be executed if expression 2 is true
}else if (expression 3){
   Statement(s) to be executed if expression 3 is true
}else{
   Statement(s) to be executed if no expression is true
}
```

Този код представлява последователност от `if` оператори, където всеки следващ `if` е част от `else` условието на предишния оператор. Блокът оператори във фигурните скоби се изпълнява в зависимост от това, кое условие е вярно. Ако няма такова условие, тогава се изпълнява блокът във фигурните скоби на последния `else`. Например:

```
<script type="text/javascript">
<!--
var book = "математика";
if( book == "история" ){
    document.write("<b>Историческа книга</b>");
}else if( book == "maths" ){
    document.write("<b>Математическа книга</b>");
}else if( book == "economics" ){
    document.write("<b>Икономическа книга</b>");
}else{
   document.write("<b>Друга книга</b>");
}
//-->
</script>
```

Изпълнението на скрипта ще доведе до следния резултат:

```
Математическа книга
```

10.2. JavaScript switch оператор

Могат да се използват няколко `if...else if` оператора, за да се осъществи многократно разклоняване на изпълнението на програмата. Това не винаги е най-доброто решение, особено когато разклоненията зависят от стойността на някаква променлива. От версия 1.2 JavaScript за такива случаи е въведен по-ефективният оператор `switch`. Основният синтаксис на оператора `switch` включва израз за оценка и няколко възможни оператори за разклонения `case`, в зависимост от стойността на израза. JavaScript интерпретаторът проверява всяко разклонение `case` дали съвпада със стойността на израза. Ако съвпада, се изпълнява блокът оператори за този `case`. Ако не съвпада, не се изпълнява никое `case`, но може да се зададе условие `default`, което ще се изпълни в този случай.

```
switch (expression)
{
  case condition 1: statement(s)
                    break;
  case condition 2: statement(s)
                    break;
   ...
  case condition n: statement(s)
                    break;
  default: statement(s)
}
```

Операторът `break` прекъсва изпълнението на оператора `switch`. Ако се изпусне операторът `break`, проверките продължават до края на оператора `switch`. Ето един пример за използване на оператора `switch`:

```
<script type="text/javascript">
<!--
var grade='A';
document.write("Начало на switch <br />");
switch (grade)
{
  case 'A': document.write("Добра работа <br />"); break;
  case 'B': document.write("Много добра <br />"); break;
  case 'C': document.write("Приета <br />"); break;
  case 'D': document.write("Не толкова добра <br />"); break;
  case 'F': document.write("Неуспешна <br />"); break;
  default:  document.write("Неизвестна степен <br />")
}
document.write("Край на switch");
//-->
</script>
```

Изпълнението на скрипта ще доведе до следния резултат:

```
Начало на switch
Добра работа
Край на switch
```

10.3. JavaScript while цикъл

В една програма може да се налага изпълнението на едно действие многократно, т.е. в цикъл. В такива случаи се използват оператори, редуциращи програмния код чрез използването на един и същи код многократно. Цикълът `while` е такъв пример, а неговият синтаксис е:

```
while (expression)
       {
       Statement(s) to be executed if expression is true
       }
```

Операторите във фигурните скоби се изпълняват последователно и многократно дотогава, докато изразът в кръглите скоби на `while` е валиден – т.е. докато се оценява като *true*. Когато този израз стане *false*, цикълът се прекъсва. Пример:

```
<script type="text/javascript">
<!--
   var count = 0;
   document.write("Начало на цикъла" + "<br />");
   while (count < 10){
     document.write("Текуща стойност: " + count + "<br />");
     count++;
   }
   document.write("Край на цикъла!");
//-->
</script>
```

Изпълнените на скрипта ще доведе до следния резултат:

```
Начало на цикъла
Текуща стойност: 0
Текуща стойност: 1
Текуща стойност: 2
Текуща стойност: 3
Текуща стойност: 4
Текуща стойност: 5
Текуща стойност: 6
Текуща стойност: 7
```

```
Текуща стойност: 8
Текуща стойност: 9
Край на цикъла!
```

Конструкцията `do...while` е подобна на `while` цикъла с тази разлика, че проверката на състоянието се случва в края на цикъла. Това означава, че цикълът ще бъде изпълнен най-малко веднъж, дори ако условието е невярно. Синтаксисът е:

```
do{
   Statement(s) to be executed;
      } while (expression);
```

Трябва да се обърне внимание, че точката и запетаята се използват в края на `do...while`. Нека да запишем горния например в термините на конструкцията `do...while`.

```
<script type="text/javascript">
<!--
   var count = 0;
   document.write("Начало на цикъла " + "<br />");
   do{
     document.write("Текуща стойност: " + count + "<br />");
     count++;
   }while (count < 0);
   document.write("Край на цикъла!");
//-->
</script>
```

Изпълнението на скрипта ще доведе до следния резултат:

```
Начало на цикъла
Текуща стойност: 0
Край на цикъла!
```

10.4. JavaScript for цикъл

Тук ще илюстрираме друг, често използван цикъл, наречен `for` цикъл. Цикълът `for` е най-компактната форма на цикъл и включва следните три важни части:

- инициализация на цикъла (*initialization*) – инициализиране на брояч с начална стойност. Инициализацията се изпълнява преди началото на цикъла;
- условие за тест (*test condition*) – условието, което ще се тества дали е вярно или не. Ако то е вярно, тогава кодът вътре в цикъла ще се изпълни, в противен случай ще се осъществи изход от цикъла;

- оператор за увеличаване или намаляване на брояч (*iteration statement*).

Синтакситът на цикъла `for` се записва в един ред, като съставляващите го части се разделят с точка и запетая:

```
for (initialization; test condition; iteration statement)
{
     Statement(s) to be executed if test condition is true
}
```

Следващият пример показва използването на `for` цикъл:

```
<script type="text/javascript">
<!--
   var count;
   document.write("Начало на цикъла" + "<br />");
   for(count = 0; count < 10; count++){
     document.write("Текуща стойност: " + count );
     document.write("<br />");
   }
   document.write("Край на цикъла!");
//-->
</script>
```

Изпълнението на скрипта ще доведе до резултат, който е подобен на `while` цикъла:

```
Начало на цикъла
Текуща стойност: 0
Текуща стойност: 1
Текуща стойност: 2
Текуща стойност: 3
Текуща стойност: 4
Текуща стойност: 5
Текуща стойност: 6
Текуща стойност: 7
Текуща стойност: 8
Текуща стойност: 9
Край на цикъла!
```

10.5. Оператори break и continue

JavaScript осигурява пълен контрол за управление както на циклите, така и на `switch` операторите. Възможно е да има ситуация, когато трябва да се излезе от един цикъл, без той да се изпълни докрай. Може да има и ситуация, когато е необходимо да се про-

пусне част от кода в цикъла и да се започне следващата итерация на цикъла. Управлението на тези ситуации може да се реализира посредством операторите `break` и `continue`. Тези оператори се използват за незабавен изход от всеки цикъл или съответно за започване на следващата итерация на цикъла.

Операторът `break`, който бе въведен при оператора `switch`, се използва за изход от цикъла по-рано, като се излиза от затварящите кода фигурни скоби.

Следващият пример илюстрира използването на оператора `break` в цикъла `while`. Следва да се отбележи как цикълът се прекъсва по-рано, когато `x` достига 5 и се изпълнява операторът `document.write (..)` точно под затварящата фигурна скоба:

```
<script type="text/javascript">
<!--
var x = 1;
document.write("Начало на цикъла<br /> ");
while (x < 20)
{
  if (x == 5){
     break;  // breaks out of loop completely
  }
  x = x + 1;
  document.write( x + "<br />");
}
document.write("Изход от цикъла!<br /> ");
//-->
</script>
```

Изпълнението на този скрипт ще доведе до следния резултат:

```
Начало на цикъла
2
3
4
5
Изход от цикъла!
```

Операторът `continue` изисква започване на следващата итерация на цикъла и пропускане на останалата част от кода. Когато операторът `continue` се достигне, изпълнението на програмата ще продължи в условието за проверка и ако то е истина, ще започне следващата итерация, в противен случай се излиза от цикъла.

Следващият пример илюстрира използването на оператора `continue` в цикъла `while`. Следва да се отбележи как операторът

`continue` се използва, за да пропусне отпечатването, когато индексът на променливата `x` достигне стойност 5:

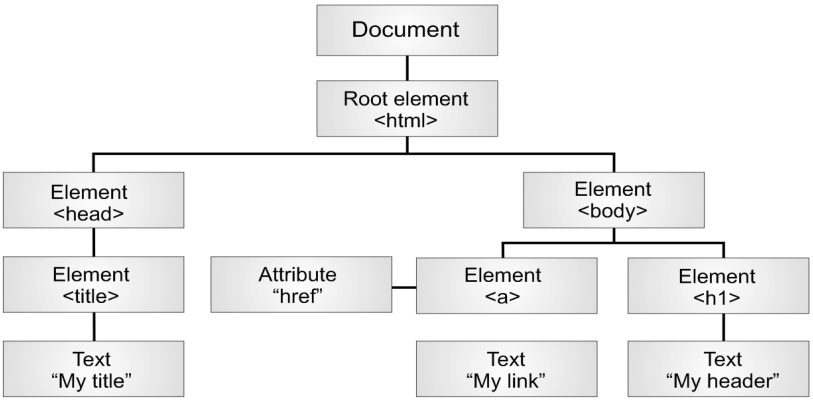
```
<script type="text/javascript">
<!--
var x = 1;
document.write("Начало на цикъла <br /> ");
while (x < 10)
{
  x = x + 1;
  if (x == 5){
     continue;  // skill rest of the loop body
  }
  document.write( x + "<br />");
}
document.write("Изход от цикъла!<br /> ");
//-->
</script>
```

Изпълнението на скрипта ще доведе до следния резултат:

```
Начало на цикъла
2
3
4
6
7
8
9
10
Изход от цикъла!
```

10.6. Функции за обработка на събития

Събитията са действия, които потребителят изпълнява при взаимодействие с уеб страницата. Преместването на курсора на мишката върху даден елемент (обект) е събитие, активирането на левия бутон на мишката също е събитие, самото зареждане на страницата е събитие и т.н. Други примери за събития – активирането на произволен клавиш, затварянето на прозорец, промяна на размерите на прозореца и т.н. Събитията са част от обектния модел на документа (*Document object model – DOM*) и всеки HTML елемент има определен набор от събития, които JavaScript може да обработва. Обектният модел на документа е конструиран като дърво на обектите (Фиг. 10.1):



Фиг. 10.1. Обектен модел на документа (DOM)

Посредством обектния модел, JavaScript получава възможността за създаване на динамичен HTML документ:

- JavaScript може да променя всички HTML елементи в страницата;
- JavaScript може да променя всички HTML атрибути в страницата;
- JavaScript може да променя всички стилове на CSS в страницата;
- JavaScript може да премахне съществуващите HTML елементи и атрибути;
- JavaScript може да добави нови HTML елементи и атрибути;
- JavaScript може да реагира на всички съществуващи HTML събития в страницата;
- JavaScript може да създава нови събития в HTML страницата.

Нека сега разгледаме няколко примера, за да обясним връзката между събитието и JavaScript:

Събитие **onClick** – това е най-често използваният тип събитие, което се случва, когато потребителят кликне върху левия бутон на мишката. Този тип събитие може да бъде използвано за валидиране, за показване на съобщение и др. Например:

```
<html>
<head>
<script type="text/javascript">
   <!--
```

```
   function sayHello()
      { alert("Hello World")
      }
   //-->
</script>
</head>
<body>
<input type="button" onclick="sayHello()" value="Hello" />
</body>
</html>
```

Това ще доведе до следния резултат: когато кликнем върху бутона *Hello,* тогава `OnClick` събитие ще се случи, което ще задейства функцията `sayHello()`.

Събитието **onSubmit** е друг важен тип събитие, което се случва при изпращането на формуляр. Това събитие може да се използва за проверка на коректността на въведените данни в елементите на формуляра. Ето един пример за неговото използване:

Нека извикаме функцията `validate()` преди изпращането на данните от формуляра към сървъра. Ако функцията `validate()` върне `true`, формулярът ще бъде изпратен, в противен случай данните от формуляра няма да бъдат изпратени:

```
<html>
<head>
<script type="text/javascript">
   <!--
   function validation() {
      all validation goes here
   .........
      return either true or false
   }
   //-->
</script>
</head>
<body>

<form method="POST" action="form.cgi" onsubmit="return validate()">
.......
<input type="submit" value="Submit" />
</form>
</body>
</html>
```

Събитията **onMouseOver** и **onMouseOut**. Тези два типа събития ще помогнат за създаването на красиви ефекти с изображения

или дори с текст. Събитието `onmouseover` се случва, когато мишката стъпи върху елемент, а `onmouseout` се случва, когато мишката напусне този елемент. Следващият пример показва как един елемент реагира, когато мишката попадне върху него:

```
<html>
<head>
<script type="text/javascript">
   <!--
   function over()
      { alert("Mouse Over");
      }
   function out() {
      alert("Mouse Out");
   }
   //-->
</script>
</head>
<body>

<div onmouseover="over()" onmouseout="out()">
   <h2>Елемент, който реагира на местонахождението на мишката
   </h2>
</div>
</body>
</html>
```

В Приложение 7 са описани атрибутите и събитията, към които JavaScript може да присвои различни функции за изпълнение при настъпване на събитието.

10.7. Обект Date

Обекът `Date` е вграден в JavaScript и служи за работа с дата и час. Стандартът ECMAScript изисква от обекта `Date` да може да представя дата и час, до милисекунди точност, в рамките на 100 000 000 дни преди или след 01.01.1970 г. Обектите `Date` се създават с `new Date()`, както е показано:

```
new Date()
new Date(milliseconds)
new Date(datestring)
new Date(year,month,date[,hour,minute,second,millisecond ])
```

Параметрите в скобите не са незадължителни и ако не се зададат, се приемат за нулеви стойности. Ето и тяхното описание:

- **без аргумент:** конструкторът `Date()` създава обект, настроен на текущата дата и час;
- **milliseconds:** когато един числов аргумент се предава, той се приема като вътрешно представяне на датата в милисекунди, като се връща от метода `getTime()`. Например, предавайки аргумента 5000, се създава дата, която представява пет секунди след полунощ на 01.01.1970;
- **datestring:** когато се предава низов аргумент, той се интерпретира като представяне на датата във формат, приет от метод `Date.parse()`;
- **7 аргумента:** за да се използва последният формат на конструктора, показан по-горе, ще разгледаме описанието на всеки един от аргументите:

1. **year:** целочислена стойност, представляваща година. За съвместимост, годината трябва винаги да се задава като 1998, а не като 98;
2. **month:** целочислена стойност, представляваща месеца, започваща с 0 за януари до 11 за декември;
3. **date:** целочислена стойност, представляваща деня на месеца;
4. **hour:** целочислен стойност, представляваща часа на деня (по 24-часова скала);
5. **minute:** целочислена стойност, представляваща минутите;
6. **second:** целочислена стойност, представляваща секундите;
7. **millisecond:** целочислена стойност, представляваща милисекундите.

Примери за инициализаране на дата:

```
var today = new Date()
var d1 = new Date("October 13, 1975 11:13:00")
var d2 = new Date(79,5,24)
var d3 = new Date(79,5,24,11,33,0)
```

Обекът `Date` може да се манипулира чрез методите на обекта `Date`. В следващия пример ще зададем дата 14 февруари 2014 г.:

```
var myDate=new Date();
myDate.setFullYear(2014,1,14);
```

Нека сега датата се отмести с 5 дни напред:

```
var myDate=new Date();
myDate.setDate(myDate.getDate()+5);
```

Да сравним текущата дата с датата 14 февруари 2100 г.:

```
var x=new Date();
x.setFullYear(2100,1,14);
var today = new Date();
   if (x>today)
     { alert("Today is before 14th February 2100");}
   else
     { alert("Today is after 14th February 2100"); }
```

Пример за установяване на определена дата:

```
<script>
     function myFunction()
          {
          var d = new Date();
          d.setFullYear(2020,10,3);
          var x = document.getElementById("demo");
          x.innerHTML=d;
          }
</script>

<p id="demo">Click the button to display a date after changing the
year, month, and day of month.</p>
<button onclick="myFunction()">Try it</button>
```

Пример за определяне на годината:

```
<script>
     function myFunction()
          {
          var d = new Date();
          var x = document.getElementById("demo");
          x.innerHTML=d.getFullYear();
          }
</script>

p id="demo">Click the button to display the full year of todays
date.</p>
<button onclick="myFunction()">Try it</button>
```

В следващия пример ще определим текущата дата:

```
<script>
          var d=new Date();
          document.write(d);
</script>
```

Превръщане на текущата дата в низ, съгласно UTC:

```
<script>
     function myFunction()
          {
          var d = new Date();
```

```
          var x = document.getElementById("demo");
          x.innerHTML=d.toUTCString();
          }
</script>

<p id="demo">Click the button to display the UTC date and time as a
string.</p>
<button onclick="myFunction()">Try it</button>
```

Брой на милисекундите от 1 януари 1970 г. до сега:

```
<script>
     function myFunction()
          {
          var d = new Date();
          var x = document.getElementById("demo");
          x.innerHTML=d.getTime();
          }
</script>

<p id="demo">Click the button to display the number of milliseconds
since midnight, January 1, 1970.</p>
<button onclick="myFunction()">Try it</button>
```

Показване в *Status Bar* на датата, часа и годината:

```
<script language="JavaScript">
     function doClock()
         {
          window.setTimeout( "doClock()", 1000 );
          today = new Date();
          self.status = today.toString();
          }
          doClock()
</script>
```

В статус бара на браузъра ще се покажат датата, часа и годината:

Ето и един пример за показване на часовник в уеб страница:

```
<script>
     function startTime()
        {
        var today=new Date();
        var h=today.getHours();
        var m=today.getMinutes();
```

```
        var s=today.getSeconds();
        // add a zero in front of numbers<10
        m=checkTime(m);
        s=checkTime(s);
        document.getElementById('txt').innerHTML=h+":"+m+":"+s;
        t=setTimeout(function(){startTime();},500);
        }
            function checkTime(i)
            {
            if (i<10)   {   i="0" + i;   }
            return i;
            }
</script>
<body onload="startTime()">
<div id="txt"></div>
</body>
```

Методите `Date`, заедно с техните описания, са дадени в Приложение 8.

10.8. Обект Math

Обектът `Math` предоставя свойства и методи за математически константи и функции. За разлика от другите глобални обекти, `Math` не е конструктор. Всички свойства и методи на `Math` са статични и могат да бъдат извикани с помощта `Math` като обект, без да се създава. Следователно, за да използваме стойнстта на числото π или синусова функция можем да използваме обекта `Math`:

```
var pi_val = Math.PI;
var sin_val = Math.sin(30);
```

Свойства на обекта `Math` и тяхното описание:

Свойство	Описание
E	Ойлерова константа и основа на натуралните логаритми, приблизително 2.718.
LN2	Натурален логаритъм от 2, приблизително 0.693.
LN10	Натурален логаритъм от 10, приблизително 2.302
LOG2E	Логаритъм на E при основа 2, приблизително 1.442
LOG10E	Логаритъм на E при основа 10, приблизително 0.434
PI	Отношение между дължината на окръжност и нейния диаметър, приблизително 3.14159
SQRT1_2	Корен квадратен от 1/2, приблизително 0.707.
SQRT2	Корен квадратен от 2, приблизително 1.414

Списъкът от методи на обекта `Math` и тяхното описание са показани в Приложение 8.

10.9. Регулярни изрази и конструктор RegExp ()

Регулярният израз представлява последователност от символи, започващи и завършващи със символа за наклонена черта (/), а също така и низ, който се предава на конструктора `RegExp()`. Незадължителните модификатори *g* – за глобално търсене и *i* – за търсене без разграничаване на малки и големи букви, могат да се поставят след втория символ /или да се предават към конструктора `RegExp()`. В следващата таблица е показан синтаксисът на регулярните изрази:

Символ	Значение
\n, \r, \t	Символи за нов ред, връщане на каретката, табулация
\\, V, *, \+, \?	Съответства на някой от специалните символи, като се игнорира или променя тяхното специално значение
[...]	Съвпадение с някой от символите в квадратните скоби.
[^…]	Съвпадение със символи, различни от тези в квадратните скоби.
.	Съответства на всеки един символ с изключение на символа за нов ред.
\w	Съответства на всеки буквено-цифров символ.
\W	Съответства на всеки небуквено-цифров символ.
\s	Съответства на празни символи (интервал, табулатор, нов ред).
\S	Съответства на непразни символи.
\d	Съответства на цифров символ (0-9).
\D	Съответства на нецифров символ.
\^	Съвпадение в началото на низа или в началото на реда.
$	Съвпадение в края на низа или в края на реда.
\b	Съответства на съвпадение в началото/края на думата.
\B	Съвпадение, различно от началото/края на думата.
?	Съвпадение с предходния символ 0 или 1 път. Еквивалентно на {0,1}.
+	Съвпадение с предходния символ 1 или повече пъти. Еквивалентно на {1,}.
*	Съвпадение с предходния символ 0 или повече пъти. Еквивалентно на {0,}.
{n}	Точно *n* на брой съвпадения с предходния символ. *N* е положително цяло число.
{n,m}	Най-малко *n*, но не повече от *m* съвпадения.
a|b	Съответства на един от символите *a* или *b*.
\n	Съответствие със символите, които съвпадат точно с *n*-ия подизраз.
$n	Използва се при замяна на низове, замества текста, който съвпада с *n*-ия подизраз.

Регулярният израз е обект, който описва модела на символите. В JavaScript `RegExp` заедно с `String` служат за определяне на начините, чрез които регулярни изрази се използват за търсене и замяна на текст във функциите. Регулярният израз може да се дефинира чрез конструктора `RegExp()`:

```
var pattern = new RegExp(pattern, attributes);
или
var pattern = /pattern/attributes;
```

Описанието на използването на параметрите при задаване на синтаксиса е както следва:

- **pattern** – низ, който определя модела на регулярния израз или друг регулярен израз;
- **attributes** – допълнителен низ, съдържащ някой от атрибутите "g" – търсене на съвпадение, което намира всички съвпадения, а не спира след първото намерено; "i" – за търсене на съвпадение, което не прави разлика между главни и малки букви; "m" – търсене на съвпадение, съдържащо повече от един ред (напр. ако низът има нов ред или carriage return, операторите ^ и $ ще търсят съвпадение по отношение на новия ред, по отношение на границата на низа).

Скобите ([]) имат специално значение, когато се използват в контекста на регулярните изрази, както е показано:

Израз	Описание
`[...]`	Всеки един символ между скобите.
`[^...]`	Всеки един символ извън скобите.
`[0-9]`	Съвпадение с всяка десетична цифра от 0 до 9.
`[a-z]`	Съвпадение с малките букви от *a* до малки букви *z*.
`[A-Z]`	Съвпадение с всеки символ от главните букви от *A* до *Z*.
`[a-Z]`	Съвпадение с малките букви от *a* до главните *Z*.

Диапазонът, показан по-горе е общ, но може да използва и редуциран диапазон [0-3], за търсене на съответствие на коя да е десетична цифра от 0 до 3, или в интервала [b-v], за търсене на съответствие на коя да е от малките букви, започвайки от *b* до *v*.

Количествени определители: честотата или позицията на символни последователности в скоби, както и единични символи, могат да бъдат обозначени със специален знак. Всеки специален символ има специфичен смисъл. Знаците +, *, ?, и $ следват поредица от знаци, както е показано:

Израз	Описание
p+	Съвпадение с всеки низ, съдържащ най-малко едно *p*.
p*	Съвпадение с всеки низ, несъдържащ *p* или съдържащ един или повече *p*.
p?	Съвпадение с всеки низ, съдържащ едно или повече *p*.
p{N}	Съвпадение с всеки низ, съдържащ последователност от *N-броя p*-та.
p{2,3}	Съвпадение с всеки низ, съдържащ последователност от 2 или 3 броя *p*-та.
p{2, }	Съвпадение с всеки низ, съдържащ последователност от поне 2 броя *p*-та.
p$	Съвпадение с всеки низ, съдържащ *p* в края.
^p	Съвпадение с всеки низ, съдържащ *p* в началото.

Със следващите примери ще се пояснят концепциите за съвпадение на съмволи.

Израз	Описание
[^a-zA-Z]	Съвпадение с всеки низ, който не съдържа някой от символите от *a* до *z* и от *A* до *Z*.
p.p	Съвпадение с всеки низ, съдържащ *p*, последван от произволен символ, който от своя страна е последван от друг *p*.
^.{2}$	Съвпадение с всеки низ, съдържащ точно два символа.
(.*)	Съвпадение с всеки низ ограничен от и .
p(hp)*	Съвпадение с всеки низ, съдържащ *p,* последвано от нула или повече случаи на последователността hp.

С всичко изброено до тук може да конструираме различни функции, използвайки регулярните изрази за проверка на въведената от потребителя информация.

10.10. Проверка за въведена информация

В много от полетата на формулярите потребителят попълва различна информация, която може да съдържа само букви (напр. имена), само цифри (напр. телефон), или и двете (напр. имейл).

В следващия пример ще направим проверка дали са въведени само буквени символи:

```
<script type='text/javascript'>
   function isAlphabet(elem, helperMsg){
  var alphaExp = /^[a-zA-Z]+$/;
  if(elem.value.match(alphaExp))
         { return true; }
         else
             {alert(helperMsg);
```

```
        elem.focus();
        return false;
        }
   }
</script>

<form>
Потребителско име: <input type='text' id='letters'/>
<input type='button'
  onclick="isAlphabet(document.getElementById('letters'),
       'Letters Only Please')"; value='Check Field'; />
</form>
```

Проверка за въведени цифри:

```
<script type='text/javascript'>
   function isNumeric(elem, helperMsg){
  var numericExpression = /^[0-9]+$/;
  if(elem.value.match(numericExpression))
        { return true; }
        else
           {alert(helperMsg);
            elem.focus();
       return false;
      }
   }
</script>

<form>
Само цифри: <input type='text' id='numbers'/>
<input type='button'
  onclick="isNumeric(document.getElementById('numbers'), 'Numbers Only
Please')" value='Check Field' />
</form>
```

Проверка за валидна дължина между 6 и 8 символа, независимо дали са букви или цифри:

```
<script type='text/javascript'>
   function lengthRestriction(elem, min, max){
  var uInput = elem.value;
  if(uInput.length >= min && uInput.length <= max)
        { return true; }
         else
           {alert("Please    enter    between    "    +min+    "    and
                  " +max+ "characters");
             elem.focus();
             return false;
      }
   }
```

```
</script>

<form>
Въведете парола (6-8 символа):
<input type='text' id='restrict'/>
<input type='button'
  onclick="lengthRestriction(document.getElementById ('restrict'), 6,
8)"  value='Check Field'; />
</form>
```

За да сме сигурни, че потребителят действително е направил избор, може да използваме следния код за валидиране:

```
<script type='text/javascript'>
function madeSelection(elem, helperMsg){
  if(elem.value == "Please Choose"){
     alert(helperMsg);
     elem.focus();
     return false;
  }else{
     return true;
  }
}
</script>

<form>
Какво означава HTML?: <select id='selection'>
<option> Моля изберете </option>
<option> Home Tool Markup Language </option>
<option> Hyper Text Markup Language </option>
<option> Hyperlinks and Text Markup Language </option>
</select>
<input type='button'
onclick="madeSelection(document.getElementById('selection'), 'Please
Choose Something')" value='Check Field' />
</form>
```

Проверка за валиден имейл адрес:

```
<script type='text/javascript'>
   function validateForm() {
      var x=document.forms["myForm"]["email"].value;
      var atpos=x.indexOf("@");
      var dotpos=x.lastIndexOf(".");
      if (atpos<1 || dotpos<atpos+2 || dotpos+2>=x.length)
        { alert("Not a valid e-mail address");
          return false; }
        else
           { alert("valid e-mail address");}
    }
```

```
</script>

<form name="myForm" onsubmit="return validateForm();" method="post">
E-mail: <input type="text" name="email">
<input type="submit" value="Submit">
</form>
</body>
```

Приложение 1. HTML глобални атрибути

Атрибут	Описание
accesskey	Задава кратка комбинация от клавиши за активиране на даден елемент.
class	Задава едно или повече имена на класове за един елемент (отнася се за класа в CSS).
contenteditable (new)	Определя дали съдържанието на даден елемент може да се редактира или не.
contextmenu (new)	Задава контекстно меню, което се появява, когато потребителят кликне върху елемента с десния бутон на мишката.
dir	Указва посоката на текста за съдържанието на елемент.
draggable (new)	Определя дали даден елемент е преместваем или не.
dropzone (new)	Определя дали преместваемият обект се копира, премества или се свързва, когато се пусне.
hidden (new)	Указва, че даден елемент все още не е или вече не е от значение.
id	Задава уникален идентификатор (id) за елемент.
lang	Задава езика на съдържанието на елемента.
spellcheck (new)	Указва дали елементът да има свой правопис и проверка за граматика или не.
style	Задава вграден CSS стил за елемент.
tabindex	Задава последователността от табулации.
title	Задава допълнителна информация за елемент.
translate (new)	Определя дали стойността на елемента трябва да бъде преведена, когато страницата се зареди, или не.

(new) – нови глобални атрибути в HTML5

Приложение 2. Тагове в HTML

Таг	Описание
<!--...-->	Определя коментар.
<!DOCTYPE>	Определя типа на документа.
<a>	Определя хипервръзката.
<abbr>	Определя съкращение.
<acronym>	Определя акроним. Не се поддържа от HTML5.
<address>	Определя информация за контакт с автора на документа.
<applet>	Определя вграден аплет. Отхвърлен в HTML 4.01. Не се поддържа от HTML5.
<area>	Определя област от изображението карта
<article> (new)	Определя статия.
<aside> (new)	Определя съдържание, отделно от съдържанието на страницата.
<audio> (new)	Определя звуково съдържание.
<b>	Определя получерен текст.
<base>	Задава база за всички относителни URL адреси в документа.
<basefont>	Задава цвят по подразбиране, размер и шрифт за целия текст в документа. Отхвърлен в HTML 4.01. Не се поддържа от HTML5.
<bdi> (new)	Изолира част от текста, която да бъде форматирана в различна посока от останалия текст извън нея.
<bdo>	Обръща посоката на текста.
<big>	Определя голям текст. Не се поддържа в HTML5.
<blockquote>	Дефинира секция, която е цитирана от друг източник.
<body>	Определя тялото на документа.
 	Определя прекъсване на реда.
<button>	Дефинира бутон за кликване.
<canvas> (new)	Използва се за направата на графика в движение, чрез скриптове (обикновено JavaScript).
<caption>	Определя надпис на таблица.
	Определя центриран текст. Отхвърлена в HTML 4.01. Не се поддържа в HTML5.
<cite>	Дефинира заглавие на произведение.
<code>	Определя част от компютърен код.
<col>	Задава свойство на колона за всяка колона в елемента <colgroup>.
<colgroup>	Определя група от една или повече колони при форматиране на таблица.
<command>	Определя команден бутон, който потребителят може да

Таг	Описание
(new)	извика.
<datalist> (new)	Задава списък от предварително дефинирани опции за входните контроли.
<dd>	Дефинира описание на термин в дефиниращ списък.
<del>	Определя текст, който е изтрит.
<details> (new)	Определя допълнителни детайли, които потребителят може да види или скрие.
<dfn>	Задава определение на термин.
<dialog> (new)	Определя диалогов прозорец.
<dir>	Дефинира списък на директории. Отхвърлена в HTML 4.01. Не се поддържа в HTML5.
	Дефинира секция в документ.
<dl>	Дефинира списък с описание.
<dt>	Дефинира термин в дефиниращ списък.
	Дефинира текста като курсив.
<embed> (new)	Дефинира контейнер за външно приложение.
<fieldset>	Групира свързани елементи във формуляр.
<figcaption> (new)	Дефинира надпис за елемента <figure>.
<figure> (new)	Дефинира самостоятелно съдържание.
<font>	Дефинира шрифта, цвят и размер за текст. Отхвърлена в HTML 4.01. Не се поддържа в HTML5.
<footer> (new)	Дефинира долния колонтитул в долната част на документа или раздела.
<form>	Определя HTML формуляр за въвеждане от потребителя.
<frame>	Дефинира прозорец (рамка) в <frameset>. Не се поддържа в HTML5.
<frameset>	Дефинира набор от рамки. Не се поддържа в HTML5.
<h1> до <h6>	Дефинира заглавия.
<head>	Задава информация за документа.
<header> (new)	Задава текст в горната част на документа или раздела.
<hr>	Задава хоризантална линия.
<html>	Дефинира HTML документ.
<i>	Определя текста в курсив.
<iframe>	Определя вградена рамка.
<img>	Задава изображение.
<input>	Определя входен елемент.
<ins>	Определя текст, който е вмъкнат.
<kbd>	Задава текста в шрифт Courier.
<keygen> (new)	Определя поле за генератор на двойка ключове (за формуляри).
<label>	Определя етикет за <input> елемент.
<legend>	Определя надпис за <fieldset> елемент.
	Определя елемент от списък.
<link>	Определя връзката между документ и външен ресурс

Таг	Описание
	(при връзка към CSS).
<map>	Определя изображение карта.
<mark> (new)	Задава маркиран/осветен текст.
<menu>	Дефинира списък/меню от команди.
<meta>	Задава метаданни за HTML документ.
<meter> (new)	Дефинира скаларно измерване в дене диапазон.
<nav> (new)	Задава връзки за навигация.
<noframes>	Задава алтернативно съдържание за потребители, които не поддържат рамки. Не се поддържа в HTML5.
<noscript>	Задава алтернативно съдържание за потребители, които не поддържат скриптове от страна на клиента.
<object>	Определя вграден обект.
	Определя подреден списък.
<optgroup>	Определя група от свързани опции в падащ списък.
<option>	Определя опция в падащ списък.
<output> (new)	Определя резултата от изчисление.
<p>	Определя параграф.
<param>	Определя параметър за даден обект.
<pre>	Определя предварително форматиран текст.
<progress> (new)	Представя напредъка на изпълнението на някаква операция.
<q>	Определя кратък цитат.
<rp> (new)	Определя какво да се покаже в браузъри, които не поддържат йероглифи.
<rt> (new)	Определя обяснение/произношение на йероглифи (за типография East Asian).
<ruby> (new)	Определя йероглифи (за типография East Asian).
<s>	Форматира текста като зачертан с черта по средата на реда.
<samp>	Форматира текста с шрифт, подобен на компютърна програма.
<script>	Определя скрипт от страна на клиента.
<section> (new)	Определя секция в документ.
<select>	Определя падащото меню.
<small>	Определя по-малък текст.
<source> (new)	Определя множество мултимедийни ресурси за мултимедийни елементи (<video> и <audio>).
	Определя секция в документ.
<strike>	Определя текста като зачертан. Отхвърлен в HTML 4.01. Не се поддържа в HTML5.
	Определя важен текст.
<style>	Определя информация за стила на документа.
<sub>	Определя текста като долен индекс.
<summary> (new)	Определя заглавието за елемента <details>.
<sup>	Форматира текста като горен индекс.

Таг	Описание
<table>	Определя таблица.
<tbody>	Групира съдържанието на тялото в таблица.
<td>	Определя клетка в таблица.
<textarea>	Определя многоредово текстово поле.
<tfoot>	Групира съдържанието на долния текст в таблица.
<th>	Определя заглавна клетка в таблица.
<thead>	Групира съдържанието на горния текст в таблица.
<time> (new)	Определя дата/час.
<title>	Определя заглавие на документа.
<tr>	Определя ред в таблицата.
<track> (new)	Определя списък на песни за мултимедийни елементи (<video> и <audio>).
<tt>	Определя текста в шрифт Courier. Не се поддържа в HTML5.
<u>	Определя текста като подчертан.
	Определя неподреден списък.
<var>	Определя променлива.
<video> (new)	Определя видео или филм.
<wbr> (new)	Определя възможност за прекъсване на реда.

Приложение 3. Символи в HTML

Character	Entity Number	Буквен запис
∀	&#8704;	&forall;
∂	&#8706;	&part;
∃	&#8707;	&exist;
∅	&#8709;	&empty;
∇	&#8711;	&nabla;
∈	&#8712;	&isin;
∉	&#8713;	&notin;
∋	&#8715;	&ni;
∏	&#8719;	&prod;
∑	&#8721;	&sum;
−	&#8722;	&minus;
*	&#8727;	&lowast;
√	&#8730;	&radic;
∝	&#8733;	&prop;
∞	&#8734;	&infin;
∠	&#8736;	&ang;
∧	&#8743;	&and;
∨	&#8744;	&or;
∩	&#8745;	&cap;
∪	&#8746;	&cup;
∫	&#8747;	&int;
∴	&#8756;	&there4;
~	&#8764;	&sim;
≅	&#8773;	&cong;
≈	&#8776;	&asymp;
≠	&#8800;	&ne;
≡	&#8801;	&equiv;
≤	&#8804;	&le;
≥	&#8805;	&ge;
⊂	&#8834;	&sub;
⊃	&#8835;	&sup;
⊄	&#8836;	&nsub;
⊆	&#8838;	&sube;
⊇	&#8839;	&supe;
⊕	&#8853;	&oplus;
⊗	&#8855;	&otimes;
⊥	&#8869;	&perp;
⋅	&#8901;	&sdot;

Гръцки букви, поддържани от HTML

Character	Entity Number	Entity Name
Α	Α	Α
Β	Β	Β
Γ	Γ	Γ
Δ	Δ	Δ
Ε	Ε	Ε
Ζ	Ζ	Ζ
Η	Η	Η
Θ	Θ	Θ
Ι	Ι	Ι
Κ	Κ	Κ
Λ	Λ	Λ
Μ	Μ	Μ
Ν	Ν	Ν
Ξ	Ξ	Ξ
Ο	Ο	Ο
Π	Π	Π
Ρ	Ρ	Ρ
Σ	Σ	Σ
Τ	Τ	Τ
Υ	Υ	Υ
Φ	Φ	Φ
Χ	Χ	Χ
Ψ	Ψ	Ψ
Ω	Ω	Ω
α	α	α
β	β	β
γ	γ	γ
δ	δ	δ
ε	ε	ε
ζ	ζ	ζ
η	η	η
θ	θ	θ
ι	ι	ι
κ	κ	κ
λ	λ	λ
μ	μ	μ
ν	ν	ν
ξ	ξ	ξ
ο	ο	ο
π	π	π
ρ	ρ	ρ
ς	ς	ς
σ	σ	σ
τ	τ	τ
υ	υ	υ
φ	φ	φ

Character	Entity Number	Entity Name
χ	χ	χ
ψ	ψ	ψ
ω	ω	ω
ϑ	ϑ	ϑ
ϒ	ϒ	ϒ
ϖ	ϖ	ϖ

Други символи, поддържани от HTML

Character	Entity Number	Entity Name
Œ	Œ	Œ
œ	œ	œ
Š	Š	Š
š	š	š
Ÿ	Ÿ	Ÿ
ƒ	ƒ	ƒ
ˆ	ˆ	ˆ
˜	˜	˜
–	–	–
—	—	—
‘	‘	‘
’	’	’
‚	‚	‚
“	“	“
”	”	”
„	„	„
†	†	†
‡	‡	‡
•	•	•
…	…	…
‰	‰	‰
′	′	′
″	″	″
‹	‹	‹
›	›	›
‾	‾	‾
€	€	€
™	™ или ™	™
←	←	←
↑	↑	↑
→	→	→
↓	↓	↓
↔	↔	↔
↵	↵	↵
⌈	⌈	⌈

⌉	&#8969;	&rceil;
⌊	&#8970;	&lfloor;
⌋	&#8971;	&rfloor;
◊	&#9674;	&loz;
♠	&#9824;	&spades;
♣	&#9827;	&clubs;
♥	&#9829;	&hearts;
♦	&#9830;	&diams;

Приложение 4. Цветове

Наеменование	16-ичен код	Цвят
AliceBlue	#F0F8FF	
AntiqueWhite	#FAEBD7	
Aqua	#00FFFF	
Aquamarine	#7FFFD4	
Azure	#F0FFFF	
Beige	#F5F5DC	
Bisque	#FFE4C4	
Black	#000000	
BlanchedAlmond	#FFEBCD	
Blue	#0000FF	
BlueViolet	#8A2BE2	
Brown	#A52A2A	
BurlyWood	#DEB887	
CadetBlue	#5F9EA0	
Chartreuse	#7FFF00	
Chocolate	#D2691E	
Coral	#FF7F50	
CornflowerBlue	#6495ED	
Cornsilk	#FFF8DC	
Crimson	#DC143C	
Cyan	#00FFFF	
DarkBlue	#00008B	
DarkCyan	#008B8B	
DarkGoldenRod	#B8860B	
DarkGray	#A9A9A9	
DarkGreen	#006400	
DarkKhaki	#BDB76B	
DarkMagenta	#8B008B	
DarkOliveGreen	#556B2F	
DarkOrange	#FF8C00	
DarkOrchid	#9932CC	
DarkRed	#8B0000	

Наеменование	16-ичен код	Цвят
DarkSalmon	#E9967A	
DarkSeaGreen	#8FBC8F	
DarkSlateBlue	#483D8B	
DarkSlateGray	#2F4F4F	
DarkTurquoise	#00CED1	
DarkViolet	#9400D3	
DeepPink	#FF1493	
DeepSkyBlue	#00BFFF	
DimGray	#696969	
DodgerBlue	#1E90FF	
FireBrick	#B22222	
FloralWhite	#FFFAF0	
ForestGreen	#228B22	
Fuchsia	#FF00FF	
Gainsboro	#DCDCDC	
GhostWhite	#F8F8FF	
Gold	#FFD700	
GoldenRod	#DAA520	
Gray	#808080	
Green	#008000	
GreenYellow	#ADFF2F	
HoneyDew	#F0FFF0	
HotPink	#FF69B4	
IndianRed	#CD5C5C	
Indigo	#4B0082	
Ivory	#FFFFF0	
Khaki	#F0E68C	
Lavender	#E6E6FA	
LavenderBlush	#FFF0F5	
LawnGreen	#7CFC00	
LemonChiffon	#FFFACD	
LightBlue	#ADD8E6	
LightCoral	#F08080	
LightCyan	#E0FFFF	
LightGoldenRodYellow	#FAFAD2	
LightGray	#D3D3D3	
LightGreen	#90EE90	

Наеменование	16-ичен код	Цвят
LightPink	#FFB6C1	
LightSalmon	#FFA07A	
LightSeaGreen	#20B2AA	
LightSkyBlue	#87CEFA	
LightSlateGray	#778899	
LightSteelBlue	#B0C4DE	
LightYellow	#FFFFE0	
Lime	#00FF00	
LimeGreen	#32CD32	
Linen	#FAF0E6	
Magenta	#FF00FF	
Maroon	#800000	
MediumAquaMarine	#66CDAA	
MediumBlue	#0000CD	
MediumOrchid	#BA55D3	
MediumPurple	#9370DB	
MediumSeaGreen	#3CB371	
MediumSlateBlue	#7B68EE	
MediumSpringGreen	#00FA9A	
MediumTurquoise	#48D1CC	
MediumVioletRed	#C71585	
MidnightBlue	#191970	
MintCream	#F5FFFA	
MistyRose	#FFE4E1	
Moccasin	#FFE4B5	
NavajoWhite	#FFDEAD	
Navy	#000080	
OldLace	#FDF5E6	
Olive	#808000	
OliveDrab	#6B8E23	
Orange	#FFA500	
OrangeRed	#FF4500	
Orchid	#DA70D6	
PaleGoldenRod	#EEE8AA	
PaleGreen	#98FB98	
PaleTurquoise	#AFEEEE	
PaleVioletRed	#DB7093	

Наеменование	16-ичен код	Цвят
PapayaWhip	#FFEFD5	
PeachPuff	#FFDAB9	
Peru	#CD853F	
Pink	#FFC0CB	
Plum	#DDA0DD	
PowderBlue	#B0E0E6	
Purple	#800080	
Red	#FF0000	
RosyBrown	#BC8F8F	
RoyalBlue	#4169E1	
SaddleBrown	#8B4513	
Salmon	#FA8072	
SandyBrown	#F4A460	
SeaGreen	#2E8B57	
SeaShell	#FFF5EE	
Sienna	#A0522D	
Silver	#C0C0C0	
SkyBlue	#87CEEB	
SlateBlue	#6A5ACD	
SlateGray	#708090	
Snow	#FFFAFA	
SpringGreen	#00FF7F	
SteelBlue	#4682B4	
Tan	#D2B48C	
Teal	#008080	
Thistle	#D8BFD8	
Tomato	#FF6347	
Turquoise	#40E0D0	
Violet	#EE82EE	
Wheat	#F5DEB3	
White	#FFFFFF	
WhiteSmoke	#F5F5F5	
Yellow	#FFFF00	
YellowGreen	#9ACD32	

Приложение 5. Мерни единици

Единица	Пример	Допълнителна информация
px	font-size:9px	Точно определяне на размера. Не е препоръчително при използването му в border-radius или в други свойства от този тип.
em	font-size:16px; line-height:1.5em /*24px/16*/	Определяне на размера спрямо размера на шрифта на бащиния елемент. Когато шрифт има размер 16px и се приеме за 1em, тогава размер на всеки останал елемент който се определя чрез em се умножава по стойността си. Пример: при размер на шрифта 10px, 1em = 10px; 2em = 20px. Това е най-добрата модерна техника за определяне на размери, поради запазването на съотношението между елементите. Em стойността е приета и за W3C стандарт.
%	font-size:100%	Използва се при сайтовете с подвижен строеж. Пример: при промяната на ширината на екрана, елементът се намира на x% от съседният елемент (margin-left:10%;).
in	font-size:0,09in	Мерната единица инч.
cm	font-size:0.3cm	Мерната единица сантиметър.
mm	font-size:8mm	Мерната единица милиметър.
ex	font-size:16px; line-height:3ex	X-височината на шрифта. (X-височината е приблизително половината от шрифта)
pt	font-size:16pt;	1pt = 1/72in (1/72 инча) = 0,0138888889 инча
pc	font-size:16pc;	Пика. 1pc = 12 points

Приложение 6. CSS Свойства

Свойство	Описание	CSS
@font-face	Правило, което позволява на уебсайтове да се изтеглят и използват шрифтове, различни от „безопасните" шрифтове.	3
@keyframes	Задава анимация.	3
alignment-adjust	Позволява по-прецизно подравняване на елементите.	3
alignment-baseline	Определя как елемент от инлайн ниво е подравнен по отношение на основния елемент.	3
animation	Задава на всички анимирани свойства, с изключение на свойството animation-play-state.	3
animation-delay	Определя кога ще започне анимацията.	3
animation-direction	Определя дали анимацията да се възпроизведе в обратен цикъл.	3
animation-duration	Определя колко секунди или милисекунди отнема изпълнението на анимацията за един цикъл.	3
animation-iteration-count	Определя колко пъти да се изпълни анимацията.	3
animation-name	Задава име за @keyframes анимация.	3
animation-play-state	Определя дали анимацията да стартира или да остане на пауза.	3
animation-timing-function	Задава скоростта на анимацията.	3
appearance	Позволява да се направи един елемент така, че да изглежда като елемент на стандартен потребителски интерфейс.	3
background	Задава всички фонови свойства в една декларация.	1
background-attachment	Определя дали фоновото изображение е фиксирано или се превърта едновременно с останалата част от страницата.	1
background-clip	Задава зоната за оцветяване в заден план.	3
background-color	Задава фоновия цвят на елемент.	1
background-image	Задава фоново изображение.	1
background-origin	Указва зоната за позициониране на фонови изображения.	3
background-position	Задава начална позиция на фоновото изображение.	1
background-	Задава как ще бъде повторено фоновото изображе-	1

Свойство	Описание	CSS
repeat	ние.	
background-size	Определя размера на фоновите изображения.	3
backface-visibility	Определя дали гърба на даден елемент, трябва да се вижда.	3
baseline-shift	Позволява препозициониране на основната линия спрямо доминиращата основна линия.	3
border	Задава всички рамки в една декларация.	1
border-bottom	Задава всички долни рамки в една декларация.	1
border-bottom-color	Задава цвета на долната рамка.	1
border-bottom-left-radius	Определя формата на рамката на долния ляв ъгъл.	3
border-bottom-right-radius	Определя формата на рамката на долния десен ъгъл.	3
border-bottom-style	Задава стила на долната рамка.	1
border-bottom-width	Задава дебелината на долната рамка.	1
border-color	Задава цвета на четирите рамки.	1
border-collapse	Определя дали рамката на таблицата да се вижда или не.	2
border-image	Съкратено свойство за определяне на всички border-image-* свойства.	3
border-image-outset	Определя с колко изображението, използвано като рамка за дадено поле, да се разширява извън полето.	3
border-image-repeat	Определя дали image-border трябва да се повтори, да се закръгли или да се разтегне.	3
border-image-slice	Задава вътрешните отмествания на image-border.	3
border-image-source	Определя изображението, което да се използва като рамка.	3
border-image-width	Задава ширината на image-border.	3
border-left	Задава свойствата за всички леви рамки в една декларация.	1
border-left-color	Задава цвета на лявата рамка.	1
border-left-style	Задава стила на лявата рамка.	1
border-left-width	Задава дебелината на лявата рамка.	1
border-radius	Съкратено свойство за определяне на всички четири border-*-radius свойства.	3
border-right	Задава свойствата за всички десни рамки в една декларация.	1
border-right-color	Задава цвета на дясната рамка.	1
border-right-	Задава стила на дясната рамка.	1

Свойство	Описание	CSS
style		
border-right-width	Задава дебелината на дясната рамка.	1
border-spacing	Задава разстоянието между рамките на съседни клетки.	2
border-style	Задава стила за четирите рамки.	1
border-top	Задава свойствата за всички горни рамки в една декларация.	1
border-top-color	Задава цвета на горната рамка.	1
border-top-left-radius	Определя формата на рамката на най-горния ляв ъгъл.	3
border-top-style	Задава стила на горната рамка.	1
border-top-width	Задава дебелината на горната рамка.	1
border-top-right-radius	Определя формата на рамката на най-горния десен ъгъл.	3
border-width	Задава дебелината за всички рамки.	1
bookmark-label	Определя етикета на bookmark.	3
bookmark-level	Определя нивото на bookmark.	3
bookmark-target	Определя целта на bookmark връзката.	3
bottom	Задава долната позиция на позициониран елемент.	2
box-align	Определя как да се подравнят елементите на контейнерите.	3
box-decoration-break	Определя дали индивидуални контейнери се третират като счупени парчета на един цял контейнер и дали всеки контейнер е обвит сам по себе си с рамка и отместване.	3
box-direction	Определя посоката, в която се показват децата на един контейнер.	3
box-flex	Определя дали децата на един контейнер са гъвкави по размер или не.	3
box-flex-group	Присвоява гъвкави елементи към гъвкави групи.	3
box-lines	Определя дали колоните ще минат на нов ред, когато свърши пространството в родителския контейнер.	3
box-ordinal-group	Определя реда на показване на дъщерните елементите на контейнера.	3
box-orient	Определя дали дъщерните елементи на контейнера следва да се разположат хоризонтално или вертикално.	3
box-pack	Определя хоризонталното положение в хоризонталните контейнери и вертикалното положение във вертикалните контейнери.	3
box-shadow	Добавя една или повече падащи сенки за контейнера.	3
box-sizing	Определя как някои елементи да се поберат в дадена област по определен начин.	3

Свойство	Описание	CSS
caption-side	Задава разположението на надписа на таблицата.	2
clear	Определя страните на елемента, за които не са позволени други плаващи елементи.	1
clip	Отрязва абсолютно позициониран елемент.	2
color	Задава цвета на текста.	1
color-profile	Позволява спецификацията на цвят, различен от подразбиращия се.	3
column-count	Определя броя на колоните на които един елемент трябва да бъде разделен.	3
column-fill	Определя как да се запълват колоните.	3
column-gap	Задава разстоянието между колоните.	3
column-rule	Съкратено свойство за определяне на всички свойства на колоните.	3
column-rule-color	Задава цвета на линията между колоните.	3
column-rule-style	Определя стила на линията между колоните.	3
column-rule-width	Определя ширината на линията между колоните.	3
column-span	Задава на колко колони може да се раздели един елемент.	3
column-width	Определя ширината на колоните.	3
columns	Съкратено свойство за определяне на свойствата на колоните (ширина и брой).	3
content	Използва се с :before и :after псевдо-елементи, за да се вмъкне генерирано съдържание.	2
counter-increment	Увеличава един или повече броячи.	2
counter-reset	Създава или инициализира един или повече броячи.	2
crop	Позволява един заместван елемент да бъде в правоъгълна област на обект вместо в целия обект.	3
cursor	Определя типа на показвания курсор.	2
direction	Определя посоката на текста.	2
display	Определя как да се показва определен HTML елемент.	1
dominant-baseline	Определя мащабирана таблица.	3
drop-initial-after-adjust	Задава долната точка при вертикално понижение за първата буква на текста спрямо долната линия на текста.	3
drop-initial-after-align	Задава линията за подравняване на първата буква на текста при вертикално понижение спрямо долната линия на текста.	3
drop-initial-before-adjust	Задава долната точка при вертикално понижение на първата буква на текст спрямо горната линия на текста.	3
drop-initial-before-align	Задава линията за подравняване на първата буква на текста при вертикално понижение спрямо горната линия на текста.	3
drop-initial-	Управлява частичното потъване на първата буква.	3

Свойство	Описание	CSS
size		
drop-initial-value	Активира ефекта на потъване на първата буква.	3
empty-cells	Определя дали да се показват рамките и фона на празните клетки в таблиците.	2
float	Определя дали контейнерът трябва да бъде плаващ.	1
float-offset	Премества плаващите елементи в обратната посока на която те са били зададени като плаващи.	3
fit	Задава как да се мащабира един заместван елемент когато не са зададени свойства за ширина и височина като auto.	3
fit-position	Определя как да се подравни обект в контейнер.	3
font	Задава всички свойства на шрифта в една декларация.	1
font-family	Определя семейството на шрифта на текста.	1
font-size	Определя размера на шрифта на текста.	1
font-style	Определя стила на шрифта на текста.	1
font-variant	Определя дали текстът трябва да бъде показан в шрифт с малки букви.	1
font-weight	Определя дебелината на шрифта.	1
font-size-adjust	Запазва четивността на текста, когато шрифтът не е подходящ по размер.	3
font-stretch	Избира нормален, кондензиран или разтеглен шрифт от зададеното семейство шрифтове.	3
grid-columns	Определя ширината на всяка колона.	3
grid-rows	Определя височината на всяка колона.	3
hanging-punctuation	Определя дали определен препинателен знак трябва да бъде разположен извън контейнера.	3
height	Задава височината на даден елемент.	1
hyphenate-after	Определя минималния брой символи при пренасяне след тирето.	3
hyphenate-before	Определя минималния брой символи при пренасяне преди тирето.	3
hyphenate-character	Определя низ, който се показва, когато се пренася.	3
hyphenate-lines	Определя максималния брой редове с пренасяне в елемент.	3
hyphenate-resource	Определя списък на външни ресурси, разделени със запетаи, които могат да помогнат на браузъра да определи кога да се пренася.	3
hyphens	Задава как да се разделят думи, за да се подобри оформлението на параграфа.	3
icon	Дава възможност за показване на елемент като икона.	3
inline-box-align	Определя кой ред от многоредов елемент да са подравни с предишен или следващ елемент.	3
image-orientation	Задава въртене на изображение по посока или обратна на часовниковата стрелка.	3

Свойство	Описание	CSS
image-resolution	Определя разрешаваща способност на изображенията.	3
left	Определя лявата позиция на елемент.	2
letter-spacing	Увеличава или намалява разстоянието между буквите в текста.	1
line-height	Определя височината на реда.	1
line-stacking	Съкратено свойство за задаване на line-stacking-strategy, line-stacking-ruby и line-stacking-shift.	3
line-stacking-ruby	Задава метода за подреждане на ред за блокови елементи, съдържащи йероглифи.	3
line-stacking-shift	Задава метода за подреждане на редове в блоков елемент с base-shift.	3
line-stacking-strategy	Задава метода за подреждане на редове, съдържащи подредени контейнери за редове, в блоков елемент.	3
list-style	Задава всички свойства за списък в една декларация.	1
list-style-image	Задава изображение като маркер на елемент от списъка.	1
list-style-position	Задава дали маркерът на елемент от списъка трябва да се появя в или извън потока на съдържанието.	1
list-style-type	Задава типа на маркера на елемент от неподреден списък.	1
marks	Добавя crop и/или cross маркери към документа.	3
margin	Задава всички външни отмествания в една декларация.	1
margin-bottom	Задава долно външно отместване на даден елемент.	1
margin-left	Задава ляво външно отместване на даден елемент.	1
margin-right	Задава дясно външно отместване на даден елемент.	1
margin-top	Задава горно външно отместване на даден елемент.	1
marquee	Използва се за превъртане на част от текста или изображението в хоризонтална или вертикална посока в зависимост от настройките.	3
marquee-direction	Задава посоката на движещо се съдържание.	3
marquee-play-count	Задава колко пъти да се изпълни движението на съдържанието.	3
marquee-speed	Задава колко бързо да се движи съдържанието.	3
marquee-style	Определя стила на движещото се съдържание.	3
mark	Съкратено свойство за задаване на свойствата mark-before и mark-after.	3
mark-after	Позволява на наименовани маркери да бъдат прикрепени към аудио поток.	3
mark-before	Позволява на наименовани маркери да бъдат прикрепени към аудио поток.	3
max-height	Задава максималната височина на елемент.	2
max-width	Задава максималната ширина на елемент.	2

Свойство	Описание	CSS
min-height	Задава минималната височина на елемент.	2
min-width	Задава минималната ширина на елемент.	2
move-to	Принуждава елемента да се отстрани от потока и да се добави на по-късен етап в документа.	3
nav-down	Указва къде да се отиде, когато се използва клавишът със стрелка надолу.	3
nav-index	Определя последователността на табулиране.	3
nav-left	Указва къде да се отиде, когато се използва клавишът със стрелка наляво.	3
nav-right	Указва къде да се отиде, когато се използва клавишът със стрелка надясно.	3
nav-up	Указва къде да се отиде, когато се използва клавишът със стрелка нагоре.	3
opacity	Задава нивото на прозрачност.	3
orphans	Задава минималния брой на редовете, които трябва да се оставят в долната част на страницата, когато прекъсването на страницата се извършва вътре в елемента.	2
outline	Задава всички свойства на контура в една декларация.	2
outline-color	Задава цвета на очертанието.	2
outline-offset	Отмества контура и го показва извън границата.	3
outline-style	Задава стила на контура.	2
outline-width	Задава дебелината на контура.	2
overflow	Указва какво се случва, ако съдържанието препълва контейнера на елемента.	2
overflow-x	Указва дали да се отрежат левите/десните части на съдържанието, ако то препълва областта на елемента.	3
overflow-y	Указва дали да се отрежат горните/долните части на съдържанието, ако то препълва областта на елемента.	3
overflow-style	Задава метод за пренасяне (scroll) на елементите, които препълват страницата.	3
page	Задава определен тип страница, където елементът трябва да се показва.	3
page-break-after	Задава начин на прекъсване на страницата след елемент.	3
page-break-before	Задава начин на прекъсване на страницата преди елемент.	3
page-break-inside	Задава начин на прекъсване на страницата в елемент.	3
padding	Съкратен запис на повече от едно вътрешно отместване в една декларация.	3
padding-bottom	Задава долно отместване на елемент.	3
padding-left	Задава ляво вътрешно отместване на елемент.	3
padding-right	Задава дясно вътрешно отместване на елемент.	3
padding-top	Задава горно вътрешно отместване на елемент.	3
perspective	Задава перспективата за 3D елементи.	3

Свойство	Описание	CSS
perspective-origin	Указва долната позиция на 3D елементи.	3
position	Определя типа на позициониране на елемент (static, relative, absolute или fixed).	
punctuation-trim	Определя дали препинателен знак трябва да бъде изпуснат.	3
quotes	Задава типа на кавичките за вградени цитати.	2
resize	Определя дали даден елемент е мащабируем.	3
right	Определя дясната позиция на елемента.	2
rest	Съкратено свойство за установяване на паузи преди и след възпроизвеждане на съдържанието.	3
rest-after	Задава паузите, които трябва да се спазват след възпроизвеждане на съдържанието на елемента.	3
rest-before	Задава паузите, които трябва да се спазват преди възпроизвеждане на съдържанието на елемента.	3
size	Определя размера и ориентацията на контейнера, в който е разположено съдържанието на страницата.	3
table-layout	Задава алгоритъма, който да се използва при оформлението на таблицата.	2
target	Обобщено свойство, с което може да се задават свойствата target-name, target-new, и target-position.	3
target-name	Указва къде да се отварят хипервръзките целево местоназначение).	3
target-new	Указва дали хипервръзките трябва да се отварят в нов прозорец или в нов раздел на съществуващ прозорец.	3
target-position	Указва къде да се отварят хипервръзките.	3
text-align	Задава хоризонтално подравняване на текст.	1
text-align-last	Описва как последния ред на блок или на ред, преди принудително прекъсване на реда, се подравнява, когато подравняването на текста е "justify".	3
text-decoration	Задава дали текстът да бъде подчертан, зачертан или с черта отгоре.	1
text-height	Задава разстояние между редовете.	3
text-indent	Задава отстъпа на първия ред в текстов блок.	1
text-transform	Управлява показването на главни букви на текст.	1
text-justify	Определя метода на подравняване на буквите, когато е използвано "justify".	3
text-outline	Задава контур на текста.	3
text-overflow	Указва какво да се случи, когато текстът е в повече от размера на елемента, който го съдържа.	3
text-shadow	Добавя сянка на текст.	3
text-wrap	Определя правила за прекъсване на редове от текст.	3
top	Определя горната точка на позиционирания елемент.	2
transform	Отнася се за 2D или 3D преобразувания на един	3

Свойство	Описание	CSS
	елемент.	
transform-origin	Позволява да се промени позицията на трансформираните елементи.	3
transform-style	Определя как вложени елементи се показват в 3D пространство.	3
transition	Съкратено свойство за определяне на 4 свойства за преход.	3
transition-property	Указва името на CSS свойството на ефекта на преход.	3
transition-duration	Задава колко секунди или милисекунди продължава ефекта на прехода.	3
transition-timing-function	Задава скоростта на ефекта на прехода.	3
transition-delay	Указва кога ще започне ефектът на прехода.	3
unicode-bidi	Използвано заедно със свойството за посока, определя дали текстът трябва да бъде заместен за нуждите на поддържане на множество езици в същия документ.	2
vertical-align	Задава вертикално подравняване на елемент.	1
visibility	Задава дали даден елемент е видим или не.	2
voice-balance	Задава баланса между левия и десния канал.	3
voice-duration	Задава продължителността на възпроизвежданото съдържание за избрания елемент.	3
voice-pitch	Определя средната височина (честота) на говорещия глас.	3
voice-pitch-range	Задава отклонението от средната височина на звука.	3
voice-rate	Контролира скоростта на говорене.	3
voice-stress	Определя силата на ударението, което трябва да се прилага.	3
voice-volume	Определя амплитудата на синтезираната реч.	3
white-space	Определя как се управляват празните символи в елемент.	1
width	Задава ширината на елемент.	1
word-spacing	Увеличава или намалява пространството между думите в текста.	1
word-break	Определя правила за прекъсване на думи.	3
word-wrap	Позволява на дълги думи да бъдат пренасяни на следващия ред.	3
widows	Задава минималния брой редове, които трябва да се оставят празни в горната част на страницата, когато се случва прекъсване на страница в елемент.	2
z-index	Определя реда за подреждане на слоевете на елементите.	2

CSS selectors

селектор	Пример	Описание	CSS
.class	.intro	Избира всички елементи с class="intro".	1
#id	#firstname	Избира всички елементи с id="firstname".	1
*	*	Избира всички елементи.	2
[*attribute*]	[target]	Избира всички елементи с атрибут target.	2
[*attribute=value*]	[target=_blank]	Избира всички елементи с target="_blank".	2
[*attribute~=value*]	[title~=flower]	Избира всички елементи, чиито title атрибут съдържа "flower".	2
[attribute|=value]	[lang|=en]	Избира всички елементи, чиито lang атрибут съдържа "en".	2
[*attribute^=value*]	a[src^="https"]	Избира всеки <a> елемент, чиито src атрибут започва с "https".	3
[*attribute$=value*]	a[src$=".pdf"]	Избира всеки <a> елемент, чиито src атрибут завършва с ".pdf".	3
[*attribute***=value*]	a[src*="w3schools"]	Избира всеки <a> елемент, чиито src атрибут съдържа подниза "w3schools".	3
element	p	Избира всички <p> елементи.	1
element,element	div,p	Избира всички елементи и всички <p> елементи.	1
element element	div p	Избира всички <p> елементи в елементите.	1
element>element	div>p	Избира всички <p> елементи, чиито родител е елемент.	2
element+element	div+p	Избира всички <p> елементи, които са разположени веднага след елемент.	2
element1~element2	p~ul	Избира всеки елемент, който е предшестван от <p> елемент.	3
:active	a:active	Избира активната връзка.	1
:after	p:after	Вмъква съдържание след всеки <p> елемент.	2
:before	p:before	Вмъква съдържание преди съдържанието на всеки <p> елемент.	2
:checked	input:checked	Избира всеки маркиран <input> елемент.	3
:disabled	input:disabled	Маркира всеки забранен	3

селектор	Пример	Описание	CSS
		<input> елемент.	
:enabled	input:enabled	Избира всеки позволен <input> елемент.	3
:empty	p:empty	Избира всеки <p> елемент, който няма деца (вкл. тесктовите възли).	3
:first-child	p:first-child	Избира всеки <p> елемент, който е първо дете на своя родител.	2
:first-letter	p:first-letter	Избира първата буква на всеки <p> елемент.	1
:first-line	p:first-line	Избира първия ред на всеки <p> елемент.	1
:first-of-type	p:first-of-type	Избира всеки <p> елемент, който е първи <p> елемент на този родител.	3
:focus	input:focus	Избира input елемент, който е активен в момента.	2
:hover	a:hover	Избира връзката за събитието on mouse over.	1
:lang(*language*)	p:lang(it)	Избира всеки <p> елемент, чиито lang атрибут е равен на "it" (Italian).	2
:last-child	p:last-child	Избира всеки <p> елемент, който е последното дете на своя родител.	3
:last-of-type	p:last-of-type	Избира всеки <p> елемент, който е последният <p> елемент на този родител.	3
:link	a:link	Избира всички непосетени връзки.	1
:not(*selector*)	:not(p)	Избира всеки елемент, който не е <p> елемент.	3
:nth-child(*n*)	p:nth-child(2)	Избира всеки <p> елемент, който е второто дете на своя родител.	3
:nth-last-child(*n*)	p:nth-last-child(2)	Избира всеки <p> елемент, който е второто дете на своя родител, считано от последното дете.	3
:nth-of-type(*n*)	p:nth-of-type(2)	Избира всеки <p> елемент, който е втори <p> елемент на своя родител.	3
:nth-last-of-type(*n*)	p:nth-last-of-type(2)	Избира всеки <p> елемент, който е вторият <p> елемент на своя родител, считано от последното дете.	3
:only-child	p:only-child	Избира всеки <p> елемент, който е единственото дете на своя родител.	3

селектор	Пример	Описание	CSS
:only-of-type	p:only-of-type	Избира елемент <p>, който е единственият <p> елемент на този родител.	3
:root	:root	Избира кореновия елемент на документа.	3
::selection	::selection	Избор на част от елемент, който е избран от потребителя.	3
:target	#news:target	Избира текущия активен #news елемент (кликнат URL, съдържащ това име).	3
:visited	a:visited	Избира всички посетени връзки.	1

Приложение 7. Обработка на събития

В таблицата по-долу са описани атрибутите и събитията, на които те съответстват. Атрибутите на глобалните събития, които могат да се добавят към HTML елементите, са показани в получерно.

Атрибут	Събитие
onAbort	прекъсване на зареждането
onActivate	избран/активен елемен
onAfterPrint	след отпечатване
onAfterUpdate	след актуализация на съдържанието на елемент
onBeforeActivate	преди елементът да се избере като активен
onBeforeCopy	преди копиране в клипборда
onBeforeCut	преди изтриване на селектирано съдържание
onBeforeDeactivate	преди промяна на активния елемент от един обект към друг
onBeforeEditFocus	редактиране на обект непосредствено след получаване на фокус
onBeforePaste	преди поставяне от клипборда
onBeforePrint	преди отпечатване или преглед за отпечатване
onBeforeUnload	преди затваряне на обекта
onBeforeUpdate	преди актуализиране на обекта
onBlur	когато елементът загуби фокус
onBounce	когато съдържанието на елемента *marquee* достигне границата
onCellChange	при промяна на данните в обекта
onChange	промяна на съдържанието на елемента
onClick	натискане на левия бутон на мишката
onContextMenu	натискане на десния бутон на мишката и отворено контекстно меню
onControlSelect	избор на комбинация от бутон Ctrl и друг бутон
onCopy	копиране в клипборда
onCut	изтриване и копиране в клипборда
onDataaVailable	извличане на данни от източника
onDatasetChanged	промяна на данни от източника
onDatasetComplete	налични данни в източника
onDblClick	дукратно натискане на левия бутон на мишката
onDeactivate	промяна на активния елемент
onDrag	процес на влачене
onDragEnd	преустановяване на влаченето
onDragEnter	курсорът на мишката се движи в елемента
onDragLeave	преместване на курсора на мишката извън елемента

Атрибут	Събитие
onDragOver	влачене върху валиден елемент
onDragStart	потребителят започва да влачи селектирания елемент
onDrop	при пускане на влачените данни върху елемента цел
onError	възникване на грешка при зареждането
onErrorUpdate	трансферът на данни е преустановен в резултат на onbeforeupdate
onFilterChange	промяна на състоянието на филтъра (при визуални ефекти)
onFinish	завършени итерации при елемента marquee
onFocus	получаване на фокус
onFocusIn	точно преди получаването на фокус
onFocusOut	точно преди загубата на фокус
onHashChange	промяна в URL, който следва знака #
onHelp	натиснат F1 клавиш при активен прозорец
onKeyDown	натискане на клавиш
onKeyPress	натиснат и отпуснат клавиш
onKeyUp	отпускане на клавиш
onLayoutCompleate	край на процесите, изпълнавани за печат или преглед за печат
onLoad	когато обектът се зареди
onLoseCapture	при загуба на възможност за прехващане с мишката (фокус)
onMessage	изпращане на съобщение посредством метода postMessage
onMouseDown	натиснат бутон на мишката
onMouseEnter	преместване на показалеца на мишката в обекта
onMouseLeave	преместване на показалеца на мишката извън обекта
onMouseMove	движение на показалеца на мишката върху обекта
onMouseOut	показалецът на мишката е извън обекта
onMouseOver	показалецът на мишката е върху обекта
onMouseUp	отпускане на бутона на мишката, когато показалецът е върху обекта
onMouseWheel	промяна на състоянието на скрол-бутона на мишката
onMove	преместване на обект
onMoveEnd	преустановено преместване на обект
onMoveStart	започнало преместване на обект
onOffline	стартиране на режим offline на браузъра
onOnline	стартиране на режим online на браузъра
onPaste	поставяне на съдържанието от клипборда
onProgress	премане на данни от сървъра
onPropertyChange	промяна на свойствата на обект
onReadyStateChage	промяна на състоянието на обект
onReset	инициализиране на данните във форма
onResize	промяна на размера на обект

Атрибут	**Събитие**
onResizeEnd	преустановена промяна на размера на обекта
onResizeStart	стартиране на промяна на размера на обект
onRowEnter	промяна на текущия запис при източника на данни
onRowExit	преди внасяне на промени на текущия запис при източника на данни
onRowsDelete	изтриване на записи с данни
onRowsInserted	добавяне на нови записи с данни
onScroll	превъртане на документа
onSelect	избиране на някакъв текст от потребителя
onSelectionChange	промяна на състоянието на избрания елемент
onSelectStart	началото на процеса на избор
onStart	започва превъртане на marquee елемент и при започване нов цикъл
onStop	потребителят прекъсва зареждането на документа
onStorage	съхраняване
onStorageCommit	изпълнение, когато локалната уеб област се записва на диска
onSubmit	активиране на бутона за изпращане (submit)
onTimeout	при изтекло време за изчакване за обработка на заявката
onUnload	преди браузърът да се освободи от документа

Приложение 8. Вградени функции в JavaScript

`Number` Методи – Обектът `Number` съдържа само методите по подразбиране, които са част от дефиницията на всеки обект.

Метод	Описание
`constructor()`	Връща функцията, която е създала този обект. По подразбиране това е Number обект.
`toExponential()`	Принуждава числото да се показва в експоненциален запис, дори ако числото е в границите, които обикновено JavaScript използва за стандартна нотация.
`toFixed()`	Форматира число с определен брой цифри вдясно от десетичния знак.
`toLocaleString()`	Връща низова стойност на текущо число във формат, който може да варира, в зависимост от настройките на браузъра.
`toPrecision()`	Определя общия брой цифри на числото (включително отляво и отдясно на десетичната запетая), които да се показват.
`toString()`	Връща низ, представляващ стойността на числото.
`valueOf()`	Връща стойността на числото.

`Boolean` Методи

Метод	Описание
`toSource()`	Връща низ, който съдържа източник на Boolean обект, и този низ може да се използва за създаване на еквивалентен обект.
`toString()`	Връща низ на "true" или "false", в зависимост от стойността на обекта.
`valueOf()`	Връща примитивната стойност на Boolean обект.

`String` Методи

Метод	Описание
`charAt()`	Връща символа на определен индекс.
`charCodeAt()`	Връща Unicode код на символа за зададен индекс.
`concat()`	Комбинира текста на два низа и връща нов низ.
`indexOf()`	Връща индекс на String обект на първата поява на определена стойност и -1, ако не е намерен.
`lastIndexOf()`	Връща индекс в String обект на последната

Метод	Описание
	намерена зададена стойност или -1, ако не е намерена такава.
localeCompare()	Връща число, показващо дали низът е преди, след или е същият като зададения сортиран низ.
length()	Връща дължината на низа.
match()	Използва се търсене на съвпадение с регулярен израз в низ.
replace()	Използва се за намиране на съвпадение между регулярен израз и низ, и замяна на съвпадащият подниз с нов подниз.
search()	Изпълнява търсене за съвпадение между регулярен израз и определен низ.
slice()	Извлича част от низ и го връща като нов низ.
split()	Разделя String обект на масив от низове чрез разделяне низа на поднизове.
substr()	Връща броя символи в низ, започвайки от определено място до определен брой символи.
substring()	Връща символите в низ между два индекса на низа.
toLocaleLowerCase()	Символите в низ се превръщат в малки букви, съгласно местното представяне.
toLocaleUpperCase()	Символите в низ се превръщат в главни букви, съгласно местното представяне.
toLowerCase()	Превръща низ в малки букви.
toString()	Връща низ, представляващ посочения обект.
toUpperCase()	Превръща низ в главни букви.
valueOf()	Връща примитивната стойност на посочения обект.

Array Методи

Метод	Описание
concat()	Връща нов масив, съставен от текущия масив и присъединени към него други масиви или стойности.
every()	Връща true, ако всеки елемент от масива удовлетворява зададена функция за условието.
filter()	Създава нов масив с всички негови елементи, за които зададената филтрираща функция връща true.
forEach()	Извиква функция за всеки елемент от масива.
indexOf()	Връща първия индекс на елемента в масива, равен на зададената стойност или на -1, ако не е намерен такъв индекс.
join()	Присъединява всички елементи на масива в низ.
lastIndexOf()	Връща последният (най-голям) индекс на елемента в масива, равен на зададена стойност или на -1, ако такъв не е намерен.
map()	Създава нов масив с елементи, получени като резултат от изпълнение на зададената функция за всеки

Метод	Описание
	елемент от масива.
pop()	Премахва последния елемент от масива и връща този елемент.
push()	Добавя един или повече елементи в края на масива и връща новата дължина на масива.
reduce()	Прилага функцията едновременно за две стойности на масива (отляво надясно) до получаването на една-единствена стойност.
reduceRight()	Прилага функцията едновременно за две стойности на масива (от дясно на ляво) до получаването на една единствена стойност.
reverse()	Обръща реда на елементите от масива – първият става последен, а последният става първи елемент.
shift()	Премахва първия елемент от масива и връща този елемент.
slice()	Извлича част от масива и го връща като нов масив.
some()	Връща true, ако поне един от елементите в масива удовлетворява зададената функция за условие.
toSource()	Дава изходния код на даден обект.
sort()	Сортира елементите на даден масив.
splice()	Добавя и/или премахва елементи от масива.
toString()	Връща низ, представляващ масив, и елементите му.
unshift()	Добавя един или повече елементи в предната част на масив и връща новата дължина на масива.

Date методите

Метод	Описание
Date()	Връща текущата (днешната) дата и час.
getDate()	Връща деня от месеца за определената дата, съгласно локалното време.
getDay()	Връща деня от седмицата за определената дата, съгласно локалното време.
getFullYear()	Връща годината от определената дата, съгласно местното време.
getHours()	Връща часа в определената дата, съгласно локалното време.
getMilliseconds()	Връща милисекундите в определената дата, съгласно локалното време.
getMinutes()	Връща минутите в определената дата, съгласно локалното време.
getMonth()	Връща месеца в определената дата съгласно локалното време.
getSeconds()	Връща секундите в определената дата съгласно локалното време.
getTime()	Връща стойността на определената дата като брой милисекунди от 1 януари 1970 г., 00:00:00 UTC.
getTimezoneOffset()	Връща отместването на времевата зона в ми-

Метод	Описание
	нути за текущото местоположение.
getUTCDate()	Връща ден (дата) от месеца за определена дата, в съответствие с универсалното време.
getUTCDay()	Връща деня от седмицата в определената дата, в съответствие с универсалното време.
getUTCFullYear()	Връща годината в определената дата, в съответствие с универсалното време.
getUTCHours()	Връща часа на определената дата, в съответствие с универсално време.
getUTCMilliseconds()	Връща милисекундите на определената дата, в съответствие с универсалното време.
getUTCMinutes()	Връща минутите в определената дата, в съответствие с универсалното време.
getUTCMonth()	Връща месеца в определената дата, в съответствие с универсалното време.
getUTCSeconds()	Връща секундите в определената дата, в съответствие с универсалното време.
getYear()	Отхвърлен. Връща годината в определената дата, съгласно местното време. Вместо него се използва getFullYear.
setDate()	Задава ден от месеца с определена дата, съгласно локалното време.
setFullYear()	Задава пълна година с определена дата, съгласно локалното време.
setHours()	Задава час на определена дата, съгласно местното време.
setMilliseconds()	Задава милисекунди на определена дата, съгласно локалното време.
setMinutes()	Задава минутите на определена дата, съгласно локалното време.
setMonth()	Задава месеца на определената дата, съгласно локалното време.
setSeconds()	Задава секундите на определена дата, съгласно локалното време.
setTime()	Задава обекта Date на времето, представено в брой милисекунди, от 1 януари 1970 г., 00:00:00 UTC.
setUTCDate()	Задава ден от месеца с определена дата, в съответствие с универсалното време.
setUTCFullYear()	Задава пълна година с определена дата, в съответствие с универсалното време.
setUTCHours()	Задава час на определена дата, в съответствие с универсалното време.
setUTCMilliseconds()	Задава час за определена дата в съответствие с универсалното време.
setUTCMinutes()	Задава минути на определена дата, в съответствие с универсалното време.
setUTCMonth()	Задава месец с определена дата, в съответствие с универсалното време.

Метод	Описание
setUTCSeconds()	Задава секундите на определена дата, в съответствие с универсалното време.
setYear()	Отхвърлен. Задава годината с определена дата, съгласно местното време. Вместо него се използва setFullYear.
toDateString()	Връща датата като разбираем низ.
toGMTString()	Отхвърлен. Преобразува датата в низ, използвайки конвенцията на Internet GMT. Вместо него се използва toUTCString.
toLocaleDateString()	Връща датата в Date като низ, използвайки локалната конвенция за дата.
toLocaleFormat()	Преобразува дата в низ, използвайки низов формат.
toLocaleString()	Преобразува датата в низ, използвайки локалните конвенции.
toLocaleTimeString()	Връща времето в Date като низ, използвайки локалната конвенция за време.
toSource()	Връща низ, който представлява обекта Date; може да се използва за създаването на нов обект.
toString()	Връща низ, който представлява обект, определен от Date.
toTimeString()	Връща времето като разбираем низ.
toUTCString()	Преобразува датата в низ, като използва конвенцията на универсално време.
valueOf()	Връща стойност на обекта Date.

Обектът `Date` определя два статични метода:

Метод	Описание
Date.parse( )	Прави синтактичен анализ на низ за дата и време и връща вътрешното представяне на тази дата в милисекунди.
Date.UTC( )	Конвертира спецификацията на дата в милисекунди.

`Math` Методи

Метод	Описание
abs(x)	Връща абсолютната стойност на числото в скобите.
acos(x)	Връща аркускосинуса (в радиани) на числото.
asin(x)	Връща аркуссинуса (в радиани) на числото.
atan(x)	Връща аркустангенса (в радиани) на числото.
atan2(y,x)	Връща аркустангенса на отношението от аргументи.
ceil(x)	Връща най-малкото цяло число, което е по-голямо от или е равно на числото в скобите.
cos(x)	Връща косинус на число.
exp(x)	Връща E^N, където N е аргумент, E е Ойлеровата константа на натурален логаритъм.

Метод	Описание
floor(x)	Връща най-голямото цяло число, което е по-малко от или равно на числото в скобите.
log(x)	Връща натурален логаритъм на число.
max(x,y,z,..,n)	Връща най-голямото число.
min(x,y,z,..,n)	Връща най-малкото число.
pow(x,y)	Връща x повдигнато на степен y.
random()	Връща псевдо-случайно число между 0 и 1.
round(x)	Връща стойност, закръглена до най-близкото цяло число.
sin(x)	Връща синуса на число.
sqrt(x)	Връща квадратния корен на число.
tan(x)	Връща тангенса на число.
toSource()	Връща низа "Math".

RegExp Методи

Метод	Описание
exec()	Изпълнява търсене за съвпадание на параметър в низ.
test()	Тества за съвпадение на параметър в низ.
toSource()	Връща константа, обозначаваща посочения обект, която може да се използва за създаване на нов обект.
toString()	Връща низ на посочения обект.

Заключение

За публикуването в средата на Интернет се използват езиците уеб за програмиране. Основната част от езиците уеб за програмиране, описани в този учебник, се базира на технологии като HTML, XHTML, CSS, JavaScript и XML. Друга част от уеб програмирането се реализира с помощта на езиците за уеб програмиране от страна на сървъра. Придобитите познания от представения материал в този учебник са една добра основа за изучаване и програмиране от страна на сървъра с помощта на езици като: PHP, ASP.NET, Perl, ASP, Python, JSP и други. Всички тези познания ще допринесат за създаването на ефектни и ефективни уеб приложения, използващи средствата на съвременните информационни технологии.

Литература

1. **Томс, Ж.** Програмиране на WEB дизайн. Нови знания, 2013.
2. **J. Duckett.** HTML & CSS: Design and Build Web Sites. John Wiley & Sons, 2011.
3. **Томс, Ж., В. Джамбазов**. Основи на уеб дизайна. 2004.
4. **Нийдерст, Д.** Web дизайн – накратко. ЗеСТ Прес, 2002.
5. **Госни, Д.** HTML. Професионални проекти. DuoDesign, 2006.
6. **Майер, Е.А.** Cascading Style Sheets – Пълното ръководство. O'Reilly, 2002.
7. **Хетчър, П., Д. Госни.** JavaScript. Професионални проекти. Duo Design, 2005.
8. **HTML Tutorial,** http://www.w3schools.com/html/default.asp
9. **JavaScript Tutorial**, http://www.w3schools.com/js/default.asp
10. **CSS Tutorial**, http://www.w3schools.com/css/default.asp

ОСНОВЫ ВЕБ-ПРОГРАММИРОВАНИЯ

В этом учебнике рассмотрены основы веб-программирования, кратко представлены наиболее часто используемые языки программирования в Интернет-среде. Основная часть изложения рассматривает язык HTML – его теги и атрибуты. В ходе описания сделано сопоставление между актуальной версией языка HTML 4.01 и предстоящей версией HTML 5. Уделено внимание стилизирующим возможностям CSS версий 1, 2 и 3. Показана часть CSS-свойств, использованных совместно с HTML. Описаны возможности форматирования текстов, списки, таблицы, цвета, фон, а также возможности форматирования в колонны. Представлены свойства форматирования контейнера, схемы позиционирования и возможностей для установления плавающих элементов. Показаны различные оформления веб-страниц, комбинирующие возможности CSS и HTML. В конце, вкратце представлен наиболее распространенный язык программирования в Интернете после HTML – JavaScript. Даны основные сведения о синтаксисе, типах данных и переменных, операторы и функции. Описано несколько основных действий (событий), которые осуществляются на веб-страницах и которые могут быть манипулированы средствами JavaScript. Показаны возможности объектов Date и Math, а также использование т. наз. «регулярных выражений» для проверки информации, которая отправляется из формуляров.

Показано много фрагментов программного кода, которые непосредственно могут быть использованы и тестированы.

Учебник предназначен для студентов, изучающих дисциплину «Основы веб-программирования» в факультете «Информационные науки» – УниБИТ, может использоваться и студентами, изучающими дисциплину «Веб-технологии».

WEB PROGRAMMING BASICS

This textbook covers the basics of web programming by describing in short the most commonly used languages when developing for the Internet. The main part of the book covers the HTML language – its tags and attributes. As part of the presentation, a comparison between the current HTML 4.01 version and the upcoming HTML 5 version is made. Attention is paid on the styling capabilities of CSS versions 1, 2 and 3. Part of the CSS capabilities used alongside HTML is presented. The capabilities for formatting text, lists, tables, colors, backgrounds as well as the option to format in columns are described. The capabilities for formatting a container, the positioning schemes and the possibilities for creating floating elements are presented. Different web page layouts are shown, that combine the capabilities of CSS and HTML. In the end, the most widely used Internet programming language apart from HTML – JavaScript is described in brief. Basic information is given regarding the syntax, the types of data and variables as well as the operators and functions. Some of the main actions (events) that are performed on a web page and which can be manipulated with the means of JavaScript are described. The capabilities of the Data and Math objects are shown, as well as the use of the so called "regular expressions" for checking information sent through forms.

Many fragments of programming code that can be used and tested are provided.

The textbook is intended for students majoring in "Information science" and studying "Web programming basics" course at ULSIT, and can also be used by students studying "Web technologies".

ОСНОВИ НА УЕБ ПРОГРАМИРАНЕТО

Българска
Първо издание

Автор:
Доц. д-р Даниела Иванова Борисова

Рецензенти:
Доц. д-р Иван Мустакеров
Проф. д-р Иван Иванов

Редактор и коректор:
Проф. д. ф. н. Цветана Георгиева

Превод резюме от български език
на руски език и на английски език:
„Латина“ ООД

Формат: 64/84/16
Печатни коли: 16
Предпечат и печат: „БПС“ ООД

Издателство „За буквите – О писменехь“
Печат „БПС” ООД

София, 2014

ISBN 978-619-185-052-5

Университет по библиотекознание
и информационни технологии
1784, София, бул. „Цариградско шосе“ 119
www.unibit.bg