МІНІСТЕРСТВО ОСВІТИ І НАУКИ, МОЛОДІ ТА СПОРТУ УКРАЇНИ ДРОГОБИЦЬКИЙ ДЕРЖАВНИЙ ПЕДАГОГІЧНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Кафедра інформаційних систем і технологій

Карпин Д.С. Наум О.М. Пелещишин А.М.

WEB- ДИЗАЙН

Тексти лекцій

для студентів освітньо-кваліфікаційного рівня “Бакалавр”
галузі знань 0501 Інформатика та обчислювальна техніка
базового напряму 6.050101 Комп’ютерні науки

Дрогобич 2013

УДК
ББК

Рекомендовано до друку вченою радою
Дрогобицького державного педагогічного університету імені Івана Франка
(протокол № ___ від _____ 2013 р.)

Рецензенти:

Карпин Д.С. Наум О.М. Пелещишин А.М. "Тексти лекцій з дисципліни "Web-дизайн"" для студентів освітньо-кваліфікаційного рівня "Бакалавр", галузі знань 0501 Інформатика та обчислювальна техніка базового напряму 6.050101 Комп'ютерні науки / Карпин Д.С. Наум О.М. Пелещишин А.М. – Дрогобич : РВВ ДДПУ імені І. Франка, 2013 – 246 с.

У навчальному посібнику висвітлено короткий курс матеріалу, необхідного для вивчення основних теоретичних відомостей з Web – дизайну по темах HTML, CSS та JavaScript та наводяться зразки контрольних питань для організації перевірки чи самоперевірки засвоєння матеріалу.

Тексти лекцій призначені галузі знань 0501 Інформатика та обчислювальна техніка базового напряму 6.050101 Комп'ютерні науки Зміст лекцій відповідає програмі затвердженій Дрогобицьким державним педагогічним університетом імені Івана Франка. Можуть бути використані викладачами в якості дидактичного матеріалу, а також для самостійного вивчення і підвищення кваліфікації.

Бібліографія 21 назва.

Зміст

Вступ

Уміння створювати якісні й цікаві веб-сайти наразі стає однією з найважливіших складових інформаційної культури людини, адже від того, як вона зможе представити у світовому інформаційному просторі себе, коло своїх професійних та особистих інтересів або ж реалізувати в Інтернеті той чи інший проект, пов'язаний із професійною діяльністю, багато в чому залежить успішність її кар'єри. Такі вміння вже не пов'язані з певною професією, вони необхідні для будь-якого активного члена сучасного суспільства

Слід зазначити, що у навчальній літературі поняття веб-дизайну часто підмінюється поняттям веб-програмування та супутніх технологій.

Метою курсу “Web-дизайн” є формування у студентів системи спеціальних знань і навичок web-дизайну

Навчальний посібник написано відповідно до програми навчальної дисципліни “Web- дизайн” для студентів освітньо-кваліфікаційного рівня “Бакалавр”, галузі знань 0501 Інформатика та обчислювальна техніка, базового напряму 6.050101 Комп’ютерні науки затвердженої вченою радою Дрогобицького державного педагогічного університету імені Івана Франка (протокол № 12 від 18.12.2009 р.).

У навчальному посібнику автор намагався систематизувати передбачений навчальною програмою матеріал, подати його з урахуванням особливостей модульно-рейтингової системи та сучасних методологічних підходів.

Структура навчального посібника включає три частини: «Гіпертекстова мова розмітки HTML», «Каскадні таблиці стилів CSS» даних» та «Мова JavaScript».

У першій частині розглядаються основи гіпертекстової мови розмітки HTML, форматування текстів, створення гіперпосилань, вставка рисунків, створення списків та таблиць. Наведено як фізичні так і логічні теги форматування і описана специфікація основних тегів.

У другій частині розглядаються основи роботи з касакдними таблицями стилів CSS. Наведенні способи використання стилів в документі, розясненні

різні види селекторів. А також описана специікація основних властивостей та принципи роботи блокової моделі.

У третій частині розглядається скриптова мова JavaScript. Описано основи синтаксису, основні конструкції та події. Наведено обєктну модель документа та робота з її елементами. А також основи скриптової роботи з формами.

Викладання курсу і контроль за його вивченням здійснюється на основі модульно-рейтингової системи. Формами організації навчального процесу з даної дисципліни є *лекції, лабораторні заняття та самостійна робота студентів.* Питання, що виносяться на ту чи іншу форму навчального процесу, визначаються в робочій програмі курсу з урахуванням специфіки факультету та кількості годин, відведених на його вивчення.

1 Гіпертекстова мова розмітки HTML

(6 год)

Питання лекції

1. Поняття HTML, тегів, структури доумента.
2. Гіперпосилання, вставка рисунків, фізичне та логічне форматуваня
3. Побудова списків і таблиць з допомогою HTML.

Питання на самостійну роботу

1 Карти сайтів
2 Фрейми
3 Мультимедійні можливості HTML.

Основні поняття: HTML, гіперпосилання, фізичне та логічне форматування, теги.

1.1 Специфікація HTML

Мова HTML придбала популярність в середині 90-х років, завдяки експоненціальному зростанню мережі Інтернет. До цього часу назріла необхідність стандартизації мови, оскільки різні компанії, що розробляли програмне забезпечення для доступу в Інтернет, пропонували свої варіанти інструкцій HTML, число яких все зростало і зростало.

Настала пора прийти до якоїсь єдиної угоди при застосуванні тегів мови HTML.

Роботу із створення специфікації HTML узяла на себе організація, World Wide Web Consortium (скорочено - W3C). В її завдання входило складання специфікації, що відображає сучасний рівень розвитку можливостей мови з урахуванням різноманітних пропозицій компаній розробників браузерів. Так, в листопаді 1995 р. явилась специфікація HTML 2.0, призвана формалізувати практику використання HTML, що склалася до кінця 1994 р.

Схема затвердження специфікацій полягає в наступному. Консорціум W3C випускає проект специфікації, після обговорення якого випускається так званий чорновий, робочий (draft) варіант специфікації і пропонує його до обговорення на певний період. Після

періоду обговорення робочий варіант специфікації може стати рекомендацією, офіційно визнаним варіантом специфікації HTML.

Незабаром після специфікації 2.0 була випущена робоча версія специфікації 3.0, термін закінчення періоду обговорення якої закінчився у вересні 1995 р.

Ця специфікація так і не була прийнята як офіційна рекомендація. У неї планувалося включити велику різноманітність тегів і можливостей, специфічних для окремих браузерів, проте Консорціум W3C не знайшов можливості розробити хорошу специфікацію для такого великого числа інструкцій.

Після довгих роздумів в травні 1996 р. був випущений проект HTML 3.2.

Проект ґрунтувався на частині тегів, наявних у версії 3.0, які показували стабільність в роботі. У вересні 1996 р. після декількох місяців обговорення версія 3.2 стала пропонованою специфікацією, а в січні 1997 р. - офіційною рекомендацією.

Липень 1997 року ознаменувався виходом пропонованої специфікації HTML 4.0, яка в грудні 1997 р. стала офіційною рекомендацією. На сьогоднішній день це основна з прийнятих специфікацій.

У специфікації HTML 4.0 ключовою ідеєю стало відділення опису структури документа від опису його представлення на екрані монітора. Досвід показує, що розділення структури і представлення документа зменшує витрати на підтримку широкого спектру платформ, середовищ і т.д., а також полегшує внесення виправлень до документів. Відповідно до цієї ідеї слід ширше користуватися методами опису представлення документа за допомогою таблиць стилів, замість того, щоб задавати конкретні дані про форму представлення вперемішку із змістом документа.

Для реалізації цієї ідеї в специфікації HTML 4.0 ряд тегів, використовуваних для безпосереднього задання форми представлення HTML-елементів, відмінені. До скасованих з цієї причини тегів відносяться <CENTER>, <FONT>, <BASEFONT>, <S>, <STRIKE>, <U>. Серед інших скасованих тегів відзначимо <ISINDEX>, <APPLET>, <DIR>, <MENU>. Замість скасованих тегів пропонуються альтернативні варіанти реалізації відповідних можливостей.

Поняття скасованого (deprecate) тега полягає в наступному. Якщо в даній специфікації мови тег названий скасованим, то це означає, що

браузери повинні продовжувати підтримку таких тегів, але їх використання не рекомендується. У наступних специфікаціях ці теги, можливо, будуть переведені в розряд застарілих (obsolete). Застарілі теги можуть більш не підтримуватися браузерами. У специфікації HTML 4.0 застарілими названо всього три теги: <XMP>, <PLAINTEXT> і <LISTING>.

Офіційні відомості про специфікацію HTML завжди можна отримати з Web-сайту Консорціуму W3C за адресою http://www.w3.org/TR/. Специфікація 4.0 знаходиться за адресою http://www.w3.org/TR/REC-html40-971218.

Відмітимо, що за логікою речей офіційна специфікація повинна грати роль керівної і напрямної сили, забезпечуючи однакову форму представлення інформації різними браузерами. Це ідеальний варіант, до якого слід прагнути. На ділі все йде не так добре. Постійно з'являються нові ідеї, що реалізовуються компаніями-розробниками в своїх браузерах. Вдалі ідеї приживаються, а потім підхоплюються іншими розробниками. Частина можливостей так і залишається специфічними особливостями окремого браузера. Вдалі розробки у результаті потрапляють в специфікацію і стають загальноприйнятими. Таким чином, процес удосконалення можливостей браузерів і уточнення специфікації йде безперервно, роблячи взаємний вплив один на одного.

Зараз Консорціум всесвітньої павутини розробляє HTML версії 5. Чорновий варіант специфікації мови з'явився в Інтернеті 20 листопада 2007. Паралельно ведеться робота щодо подальшого розвитку HTML під назвою XHTML (англ. Extensible Hypertext Markup Language - «розширювана мова розмітки гіпертексту»). Поки XHTML за своїми можливостями можна порівняти з HTML, проте пред'являє більш суворі вимоги до синтаксису. Як і HTML, XHTML є підмножиною мови SGML, проте XHTML, на відміну від попередника, заснований на XML. Варіант XHTML 1.0 був схвалений в якості Рекомендації Консорціуму всесвітньої павутини 26 січня 2000.

Планована специфікація XHTML 2.0 розриває сумісність зі старими версіями HTML і XHTML, що не дуже влаштовує деяких веб-розробників і виробників браузерів. Групою WHATWG (англ. Web Hypertext Application Technology Working Group) розробляється специфікація Web Applications 1.0, часто неофіційно звана «HTML 5», яка розширює HTML (втім, маючи і сумісний з XHTML 1.0 XML-синтаксис) для кращого подання семантики різних типових сторінок,

наприклад форумів, сайтів аукціонів, пошукових систем, онлайн-магазинів і т. д., які не дуже вдало вписуються в модель XHTML 2.

1.2 Структура документа

Першим тегом, з якого слід починати опис документів HTML, є тег <HTML>. Він повинен завжди починати опис документа, а завершувати опис документа повинен тег </HTML>. Ці теги означають, що рядки, які знаходяться між ними, представляють єдиний HTML-документ.

Сам по собі документ є звичайним текстовим ASCII-файлом.

Без цих тегів браузер або інша програма перегляду, можливо, буде не в змозі ідентифікувати формат документа і правильно його інтерпретувати.

Найчастіше тег <HTML> використовується без параметрів. У попередніх версіях використовувався параметр VERSION, відмінний специфікацією HTML 4.0. На зміни цьому параметру прийшов тег <!DOCTYPEX>

Більшість сучасних браузерів можуть зрозуміти документ, який і не містить тегів <HTML> і </HTML>, все ж їх вживання бажане.

Між парою тегів <HTML> і </HTML> розміщається сам документ. Документ може складатися з двох розділів - розділу заголовка (що починається тегом <HEAD>) і розділу змістовної частини документа (що починається тегом <BODY>) .Для документів, що описують фреймові структури, замість розділу BODY використовується розділ FRAMESET (з тегом <FRAMESET>) Проте фрейми теги не виправдали себе і зараз майже не використовуються.

1.2.1 Розділ документа HEAD

Розділ документа HEAD визначає його заголовок і не є обов'язковим тегом, проте добре складений заголовок може бути вельми корисний. Завданням заголовка є представлення необхідної інформації для програми, що інтерпретує документ. Теги, які знаходяться всередині розділу HEAD (окрім назви документа, що описується за допомогою тега <TITLE>), не відображаються на екрані.

Розділ заголовка відкривається тегом <HEAD>. Переважно цей тег слідує відразу ж за тегом <HTML>. Закриваючий тег </HEAD> показує кінець цього розділу. Між згаданими тегами розташовується решта тегів розділу заголовка.

Назва документа

Тег-контейнер <TITLE> є єдиним обов'язковим тегом заголовка і служить для того, щоб дати документу назву. Вона зазвичай показується в заголовку вікна браузера. Тег <TITLE> не слід плутати з назвою файлу документа; навпаки, він є текстовим рядком, абсолютно незалежним від імені і місцезнаходження файлу, що робить його вельми корисним. Ім'я ж файлу жорстко визначається операційною системою комп'ютера, на якому він зберігається.

Назва документа записується між тегами <TITLE> і </TITLE> і є рядком тексту. В принципі, назва може мати необмежену довжину і містити будь-які символи окрім деяких зарезервованих. На практиці слід обмежитися одним рядком, зважаючи на те, що назва відображається в заголовку вікна браузера. Також слід пам'ятати про те, що залишиться від назви документа при мінімізації вікна браузера.

По замовчуванню текст, що міститься в назві документа, використовується при створенні закладки (bookmark) для документа. Тому, для більшої інформативності, слід уникати безликих назв (Home Page, Index і т.д.).

Подібні слова, використовувані як назва закладки, зазвичай абсолютно даремні. Назва документа повинна стисло характеризувати його зміст. Відмітимо, що при відображенні на екрані документів з фреймовою структурою, коли в кожен з фреймів завантажується окремий документ, що має свою назву, на екрані буде видно тільки назву головного документа. Проте, задавати назву окремих документів, призначених для завантаження у фрейми, також рекомендується.

Зв'язок з іншими документами

Часто HTML-документи зв'язані між собою, тобто мають посилання один на одного. Посилання можуть бути як абсолютні, так і відносні. І ті та інші мають недоліки. Абсолютні посилання можуть бути дуже громіздкими і переставати працювати, якщо переміщений молодший за ієрархією документ. Відносні посилання легше вводити і оновлювати, але і цей зв'язок обривається, якщо переміщений старший за ієрархією документ. Обидва види зв'язків можуть порушитися при перенесенні документа з одного комп'ютера на інший.

Часто трапляється, що користувач завантажив на свою машину великий документ і відключився від мережі для його докладного

вивчення. Всі посилання в локальній копії документа перестануть працювати. Для їх "реанімації" прийдеться знов звернутися до оригіналу документа, що знаходиться на видаленому комп'ютері.

На щастя, розробники HTML передбачили цю проблему і додали два теги, <BASE> і <LINK>, які включаються в заголовок для того, щоб зв'язок між документами не порушувався.

Тег <BASE>

Тег <BASE> служить для вказування повної базової URL-адреси документа. З його допомогою відносне посилання продовжує працювати, якщо документ переноситься в інший каталог або навіть на інший комп'ютер. Тег <BASE> працює аналогічно команді path MS-DOS, що дозволяє програмі перегляду визначити посилання на документ, навіть якщо вона знаходиться в старшому за ієрархією документі, розташованому на іншому комп'ютері.

Тег <BASE> має один обов'язковий параметр HREF, після якого вказується повна URL-адреса документа. Нижче показаний приклад використання тега <BASE>.

```
<HTML>
  <HEAD>
    <TITLE> Вказівка базового адреса</TITLE>
    <BASE HREF="//www.my_host.ru/~sergeev">
  </HEAD>
  <BODY>
    <IMG SRC=/gifs/news.gif ALT="News">
  </BODY>
</HTML>
```

Тег <BASE> вказує браузеру, де шукати файл. У випадку, якщо користувач працює з локальною копією файлу і його машина не відключена від мережі, зображення піктограми News буде знайдено і відображене у вікні браузера.

Тег <LINK>

Навіть якщо тег <BASE> дозволяє знайти файл, залишається відкритим питання про зв’язок документів. Важливість цих відносин зростає пропорційно зростанню складності ваших документів. Для того, щоб підтримувати логічний зв'язок між ними, в HTML введений тег <LINK>.

Тег <LINK> вказує на зв'язок документа, що містить даний тег і іншого документа або об'єкту. Він складається з URL-адреси і параметрів, що конкретизують зв’язки документів. Заголовок документа може містити будь-яку кількість тегів <LINK>. Таблиця 1.2.1 описує параметри тега <LINK> і їх функції.

Таблиця 1.2.1Параметри тега <LINK>.

Параметр	Значення
HREF	Вказує на URL-адресу іншого документа
REL	Визначає відношення між біжучим та іншим документом
REV	Визначає відношення між іншим та біжучим документом
TYPE	Вказує тип і параметр біжучої таблиці стилів

Приведемо приклади тега <LINK> з параметрами:

```
<LINK REL="contents"HREF =".../toc.html">
<LINK
HREF="mailto:sergeev@mall.ifmo.ru"REV="made">
```

Перший рядок вказує на зв'язок з файлом змісту документа (toc.html - table of contents) з прямим відношенням contents. Другий рядок описує зв'язок з URL-адресою автора документа (із зворотним відношенням made).

Між документами може існувати безліч різних відносин.

Приклади інших значень параметра REL: bookmark, copyright, glossary, help, home, index, toc, next, previous. Параметр REV може також приймати значення: author, editor, publisher, owner.

Тег <META>

Розробка нових специфікацій мови розмітки гіпертексту займає чималий час, і за цей час компанії, що розробляють браузери, встигають випустити декілька версій своїх продуктів. Тому в розділ заголовка може бути доданий ще один тег <META>, що дозволяє авторам документа визначати інформацію, що не має відношення до HTML.

Ця інформація використовується браузером для дій, які не передбачені поточною специфікацією HTML.

Приклад:

```
<META HTTP-EQUIV="refresh" CONTENT="60"
URL="www.my_host.ru/homepage.html">
```

Браузери зрозуміють цей запис як інструкцію чекати 60 секунд, а потім завантажити новий документ. Така інструкція часто використовується при зміні місцерозташування документів.

Невеликий документ з приведеним рядком може бути залишений на старому місцерозташуванні документа для автоматичного посилання на його нове місцерозташування.

Наступний рядок:

```
<META HTTP-EQUIV="refresh" CONTENT="60">
```

Інструктує браузер перезавантажувати сторінку кожні 60 секунд. Це може бути корисно, якщо дані на сторінці часто оновлюються, наприклад, у разі відстежування котирувань акцій.

Стало вельми популярним застосування елементу <META> для рішення деяких типових задач. Як приклад можна привести вказівку ключових слів, використовуваних пошуковими системами. Цей спосіб дозволяє включати в індекс документа додаткові слова, які можуть явно не входити в його зміст. Для цього в тегу <META> в якості значення параметра NAME вказується ім'я деякої властивості. А за допомогою параметра CONTENT указується значення даної властивості, наприклад:

```
<META NAME="author" CONTENT="Іван Іваненко">
```

Специфікація HTML не визначає яких-небудь конкретних імен властивостей, записуваних в тегу <META>. Однак є декілька часто вживаних властивостей, наприклад, description, keywords, author, robots

```
<META NAME="description" CONTENT="Опис
можливостей мови HTML 4.0">
<META NAME="keywords" CONTENT="тег, гіпертекст,
HTML, браузер" >
```

Таблиця 1.2.2 Параметри тега <META>.

Параметр	Значення
HTTP-EQUIV	Визначає властивість для тега
NAME	Забезпечує додатковий опис тегу. Якщо

	він опущений то вважається, що він дорівнює HTTP-EQUIV.
URL	Визначає адрес документа для властивості.
CONTENT	Визначає значення властивості, що повертається

Ще одне важливе призначення тега <META>- це вказівка кодування тексту. Так, для тексту українською мовою в кодуванні Windows потрібно записати наступний рядок:

```
<META HTTP-EQUIV="Content-
Type"CONTENT="text/html; charset=Windows-1251">
```

Інші елементи заголовка

У розділі заголовка документа можуть бути присутніми ще два теги - <STYLE> і <SCRIPT>. Їх призначення зв'язане з використанням таблиць стилів в документі і записом скриптів. Ці питання детально розглянемо далі.

1.2.2 Розділ документа BODY

У цьому розділі документа розташовується його змістовна частина. Більшість тегів, що розглядаються далі в цьому розділі і подальших, повинна розташовуватися в даному розділі документа. Тут ми розглянемо лише деякі загальні питання.

Розділ документа BODY повинен починатися тегом <BODY> і завершуватися тегом </BODY>, між якими мається в своєму розпорядженні весь вміст даного розділу. Строго кажучи, наявність цих тегів не є обов'язковою, оскільки браузери можуть визначити початок змістовної частини документа по контексту. Проте їх вживання рекомендується.

Тег <BODY> має ряд параметрів, жоден з яких не є обов'язковим. Перелік параметрів приведений в Таблиця 1.2.3.

Таблиця 1.2.3 Параметри тега <BODY>.

Параметр	Значення
ALINK	Визначає колір активного посилання
BACKGROUND	Вказує URL-адресу зображення, що використовується як фонове
BOTTOMMARGIN	Встановлює границю нижнього поля

	в пікселях
BGCOLOR	Визначає колір фону
BGPROPERTISE	Якщо встановлено FIXED фонове зображення не прокручується
LEFTMARGIN	Встановлює границю лівого поля в пікселях
LINK	Визначає колір ще не відвідуваного посилання
RIGHTMARGIN	Встановлює границю правого поля в пікселях
SCROLL	Встановлює наявність чи відсутність ліній прокрутки вікна браузера
TEXT	Визначає колір тексту
TOPMARGIN	Встановлює границю верхнього поля в пікселях
VLINK	Визначає колір вже відвідуваного посилання

У мові HTML колір визначається цифрами в шістнадцятизначному коді.

Колірна система базується на трьох основних кольорах - червоному, зеленому і синьому - та позначається RGB. Для кожного кольору задається шістнадцятизначне значення в межах від 00 до FF, що відповідає діапазону 0 – 255 у десятковому численні. Потім ці значення об'єднуються в одне число перед яким ставиться символ #. Наприклад, число #800080 позначає фіолетовий колір. Щоб не запам'ятовувати сукупності цифр, замість них можна користуватися назвами кольорів.

1.2.3Форматування тексту

У даному розділі будуть розглянуті можливості форматування окремих символів тексту документа.

Теги рівня блоку і рядкові теги.

Деякі HTML-теги, які можуть з'являтися в розділі BODY, називають тегами рівня блоку (block level), в той час як інші рядковими (inline) тегами або, кажучи по-іншому, тегами рівня тексту (text level), хоча таке розділення тегів по рівнях до певної міри умовно.

Відмінність рівнів HTML-тегів полягає в наступному: теги рівня блоку можуть містити послідовні теги і інші теги рівня блоку, тоді як рядкові теги містять тільки дані і інші послідовні теги. Блокові теги описують крупніші структури документів, в порівнянні з рядковими тегами.

Логічне і фізичне форматування

Для форматування тексту HTML-документів передбачена ціла група тегів, яку можна умовно розділити на теги логічного і фізичного форматування.

Теги логічного форматування позначають (своїми іменами) структурні типи своїх текстових фрагментів, наприклад, такі як: програмний код (тег <CODE>), цитата (тег <CITE>), абревіатура (тег <ABBR>) і т.д. З допомогою тегів <EM> і <STRONG> можна, наприклад, відзначити окремі фрагменти як виділені, або сильно виділені. Відмітимо, що йдеться про структурну розмітку, яка не впливає на конкретне екранне відображення фрагмента браузером. Тому така розмітка і називається логічною. Фрагменти з логічним форматуванням браузери відображають на екрані певним чином, заданим по замовчуванню. Вигляд відображення ніяк не зв'язано з структурним типом фрагмента (ім'ям тега логічного форматування), але може бути легко перевизначений.

Теги фізичного форматування визначають формат відображення вказаного в них фрагмента тексту у вікні браузера. Наприклад, для відображення фрагмента курсивом можна використовувати тег курсиву <I>.

Між розробниками HTML-документів довгий час йшли спори про переваги і недоліки того або іншого підходу. З виходом специфікації HTML 4.0 ці спори завершилися на користь застосування логічного форматування, оскільки був проголошений принцип відділення структури документа від його представлення. Дійсно, тільки на базі логічного форматування можна гнучко управляти представленням документа, використовуючи сучасні методи (засновані на таблицях стилів, документах, що динамічно змінюються, і т.д.).

Проте, на даний момент може вільно використовуватися і фізичне форматування. У специфікації HTML 4.0 деякі теги фізичного форматування не рекомендуються для застосування, проте, поки вони все ще підтримуються всіма браузерами. Відмітимо, що деякі теги логічного форматування, покликані замінити окремі теги фізичного

форматування, розпізнаються не всіма браузерами, що робить їх застосування вкрай незручним. Прикладом може служити логічний тег <DEL>, який рекомендується використовувати замість фізичного тега <STRIKE>.

Теги, що розглядаються нижче, відносяться до тегів рівня тексту, тобто покликані, в основному, розмічати невеликі групи символів. Деякі теги можуть задавати розмітку і на рівні блоку.

1.2.4Теги логічного форматування тексту

Тег <ABBR>

Тег <ABBR> відмічає свій текст як абревіатуру (ABBReviation).

```
<ABBR> НЛО </ABBR> неопізнаний літальний об’єкт
```

Даний тег зручно використовувати у поєднанні з параметром TITLE, як значення якого можна вказати повну форму запису абревіатури. Тоді візуальні браузери при наведенні курсору на текст, розмічений тегом <ABBR>, будуть видавати повне найменування у вигляді підказки, що з'являється.

Тег <ACRONYM>

Тег <ACRONYM>, як і тег <ABBR>, використовується для відмітки абревіатур. Цим тегом рекомендується відзначати так звані акроніми тобто іншомовні слова, що складаються з абревіатур. Тег <ACRONYM> можливо в майбутньому стане використовуватися для невізуального відображення елементів, наприклад при мовному синтезі.

Даний тег зручно використовувати у поєднанні з параметром TITLE, як значення якого можна вказати повну форму запису абревіатури. Тоді візуальні браузери при наведенні курсору на текст, розмічений тегом <ACRONYM>, будуть видавати повне найменування у вигляді підказки, що з'являється.

```
<ACRONYM TITLE="Дрогобицький державний
педагогічний університет"> ДДПУ </ACRONYM>
```

Тег <CITE>

Тег <CITE> використовується для виділення цитат або назв книг і статей, посилань на інші джерела і т.д. Браузерами такий текст зазвичай виводиться курсивом.

Приклад:

```
<CITE>Вчитися вчитися і ще раз вчитися</CITE>
хороша цитата
```

Тег <CODE>

Тег <CODE> відмічає свій текст як невеликий фрагмент програмного коду.

Як правило, відображається моноширинним шрифтом. Цей тег не слід плутати з тегом <PRE>, являючись елементом рівня блоку, який слід використовувати для відмітки великих фрагментів (лістингів) коду.

Наприклад:

```
Для додавання двох чисел слід написати <CODE>z
= x + y; </CODE>
```

Є ще одна відмінність у використанні тегів <CODE> і <PRE>. В коді програм часто буває важливою наявність декількох пробілів, що йдуть підряд. Їх відображення буде збережено тільки при використання тега <PRE>.

Тег <DEL>

Тег <DEL> відмічає свій текст як видалений. Цей елемент корисно використовувати для відмітки змін, що вносяться до документа від версії до версії.

Тег <DEL> може використовуватися як елемент рівня тексту і як елемент рівня блоку.

Тег має два необов'язкові параметри: CITE і DATETIME. Значення параметра CITE повинне бути URL-адресою документа, що пояснює причини видалення даного фрагмента.

Параметр DATETIME вказує дату видалення у форматі: YYYY-MM-DDThh:mm:ssTZD, що визначає рік, місяць, число, години, хвилини і секунди видалення, а також часовий пояс (Time Zone) .Наприклад:

```
Приклад написання <DEL DATETIME=2010-04-
11T20:12:00+0.00> тега DEL</DEL>
```

Текст, помічений тегом <DEL> звичайно відображається перекресленим текстом. У специфікації HTML 4.0 цьому тегу

віддається перевага перед тегом фізичного форматування <STRIKE> чи <S>, що позначають перекреслений текст.

Тег <DFN>

Тег <DFN> відмічає свій текстовий фрагмент як визначення (DeFinitioN).

Наприклад, цим тегом можна відзначити який-небудь термін, коли він зустрічається в тексті вперше.

Приклад:

```
<DFN>HTML</DFN>- це гіпертекстова мова розмітки
```

Відображається по замовчуванню курсивом.

Тег <INS>

Тег <INS> відмічає свій текст як вставку (INSertion). Цей елемент корисно використовувати для позначення змін, що вносяться до документа від версії до версії. Тег <INS> може використовуватися як елемент рівня тексту і як елемент рівня блоку.

Тег має два необов'язкові параметри: CITE і DATETIME. Значення параметра CITE повинне бути URL-адресою документа, що пояснює подробиці внесених доповнень.

Параметр DATETIME указує дату вставки у форматі: YYYY-MM-DDThh:mm:ssTZD, що визначає рік, місяць, число, години, хвилини і секунди вставки, а також часовий пояс (Time Zone).

```
Приклад написання <INS DATETIME=2010-04-
11T20:12:00+0.00> тега INS</INS>
```

Текст, помічений тегом <INS>, просто відображається підкресленим текстом.

Тег <EM>

Тег <EM>(EMphasis - виділення, підкреслення) використовується для виділення важливих фрагментів тексту. Браузери зазвичай відображають такий текст курсивом.

Приклад:

```
Приклад виділення <EM> окремих слів </EM>
текста
```

Рекомендується використовувати даний тег замість тега фізичного форматування <i>.

Тег <KBD>

Тег <KBD> відмічає текст як той що вводиться користувачем з клавіатури. Зазвичай відображається моноширинним шрифтом, наприклад:

```
Щоб запустити текстовий редактор, надрукуйте
:<KBD>notepad</KBD>
```

Рекомендується використовувати даний тег замість тега фізичного форматування <TT>.

Тег <Q>

Тег <Q> відмічає короткі цитати в рядку тексту. На відміну від тега рівня блоку <BLOCKQUOTE> при відображенні не виконується відділення розміченого тексту порожніми рядками. Зазвичай відображається курсивом.

```
А це приклад <Q>рядкової цитати</Q>
```

Тег має параметр CITE значення якого можна вказати джерело цитати.

Тег <SAMP>

Тег <SAMP> відмічає текст як зразок (SAMPle). Звичайне використання цього тега - позначення тексту, що повертається програмами (sample output). Використовується також для виділення декількох символів моноширинним шрифтом.

Рекомендується використовувати даний тег замість тега фізичного форматування <TT>. Наприклад:

```
В результаті роботи програми буде надруковано:
<SAMP>Hello,World!</SAMP>.
```

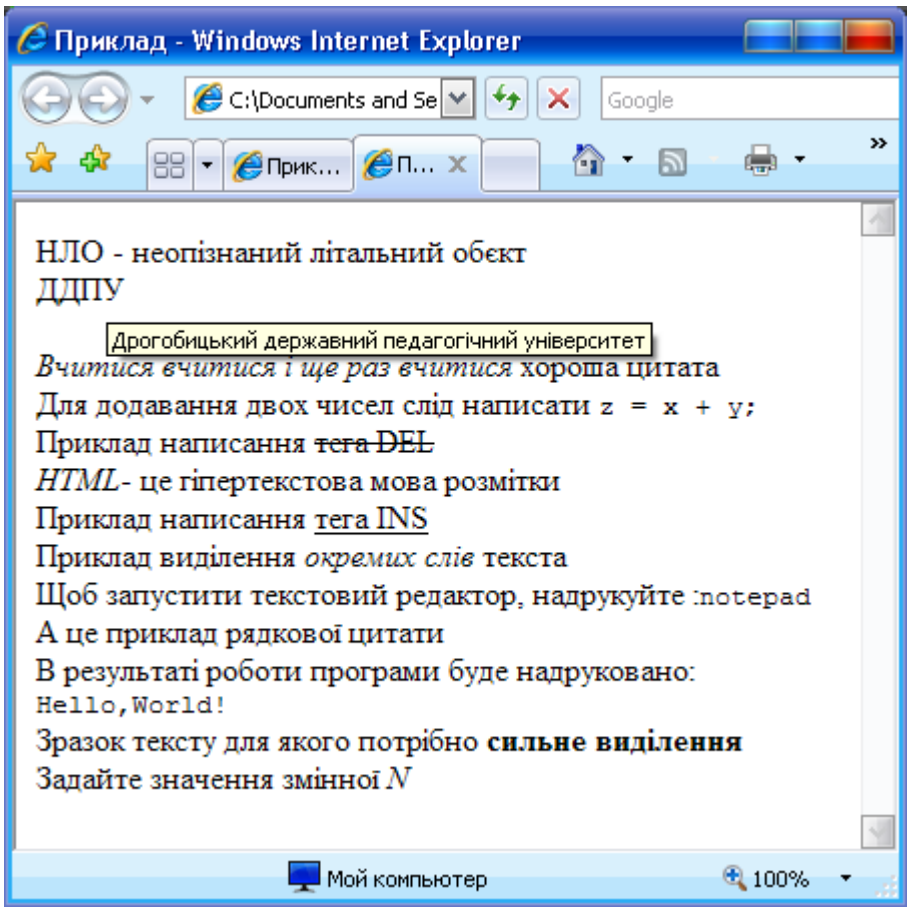

Рис 1.2.1 Теги логічного форматування тексту

Тег <STRONG>

Тег <STRONG>, як правило, використовується для виділення важливих фрагментів тексту. Браузери зазвичай відображають такий текст напівжирним шрифтом.

Приклад:

```
Зразок тексту для якого потрібно <STRONG>
сильне виділення </STRONG>
```

Рекомендується використовувати даний тег замість тега фізичного форматування <B>. Тегом <STRONG> переважно розмічають важливіші фрагменти тексту, ніж ті, що розмічені тегом <EM>.

Тег <VAR>

Тег <VAR> розмічає імена змінних програм. Зазвичай такий текст відображається курсивом.

Приклад:

```
Задайте значення змінної <VAR>N</VAR>
```

1.2.5Теги фізичного форматування тексту

Тег <B>

Тег <B> відображає текст напівжирним шрифтом. Для більшості випадків замість цього тега рекомендується використовувати тег логічного форматування <STRONG> Наприклад:

```
Це <B>напівжирний</B> шрифт
```

Тег <I>

Тег <I> відображає текст курсивом. Для більшості випадків замість цього тега рекомендується використовувати теги <EM>, <DFN>, <VAR> чи <CITE>, оскільки останні краще відображають призначення тексту, що виділяється. Наприклад:

```
Виділення <I>курсивом</I>
```

Тег <TT>

Тег <TT> відображає текст моноширинним шрифтом. Для більшості випадків замість цього тега краще використовувати теги <CODE>, <SAMP> або <KBD>.

Приклад:

```
Це <TT> моноширинньй </TT> шрифт.
```

Тег <U>

Тег <U> відображає текст підкресленим. Відмінений тег. Замість нього рекомендується використовувати теги <STRONG> чи <CITE>.

Наприклад:

```
Приклад <U> підкреслення </U> тексту.
```

Теги <STRIKE> і <S>

Теги <STRIKE> і <S> відображають текст, перекреслений горизонтальною лінією. Відмінений тег. Замість нього слід використовувати тег <DEL>. Наприклад:

```
Приклад <STRIKE>переклресленого</STRIKE>тексту.
```

Тег <BIG>

Тег <BIG> виводить текст шрифтом більшого (чим непомічена частина тексту) розміру. Замість даного елементу краще

використовувати <STRONG> чи теги заголовків, наприклад, <H3>. Більшість браузерів підтримують вкладені теги <BIG>, однак використовувати такий підхід не рекомендується.

Наприклад:

```
Шрифт <BIG>більшого</BIG> розміру.
```

Тег <SMALL>

Тег <SMALL> виводить текст шрифтом меншого розміру. Оскільки в HTML немає тега, протилежного по дії тегу <EM>, то для цих цілей можна застосовувати тег <SMALL>. Більшість браузерів підтримують вкладені теги <SMALL>, проте використовувати такий підхід не рекомендується.

Наприклад:

```
Шрифт <SMALL>меншого</SMALL> розміру.
```

Тег <SUB>

Тег <SUB> зсуває текст нижче за рівень рядка і виводить його (якщо можливо) шрифтом меншого розміру. Зручно використовувати для математичних індексів. Наприклад:

```
Приклад шрифту для <SUB>нижнього</SUB>індексу.
```

Тег <SUP>

Тег <SUP> зсуває текст вище за рівень рядка і виводить його (якщо можливо) шрифтом меншого розміру. Зручно використовувати для завдання степенів чисел в математиці. Наприклад:

```
Приклад шрифту для <SUP> верхнього </SUP>
індексу.
```

Тег <BLINK>

Тег <BLINK> відображає миготливий текст. Цей тег не входить в специфікацію HTML і підтримується тільки браузером Netscape. Досвідчені розробники вкрай рідко вдаються до використання цього тега, оскільки наявність на сторінці миготливих символів дратує багато користувачів.

Тег <SPAN>

Тег-контейнер <SPAN> являється аналогом тега рівня блоку <DIV>. Може використовуватися в тих випадках, коли потрібно

відзначити фрагмент тексту для завдання його властивостей, і при цьому не вдається використовувати ніякий інший структурний тег форматування.

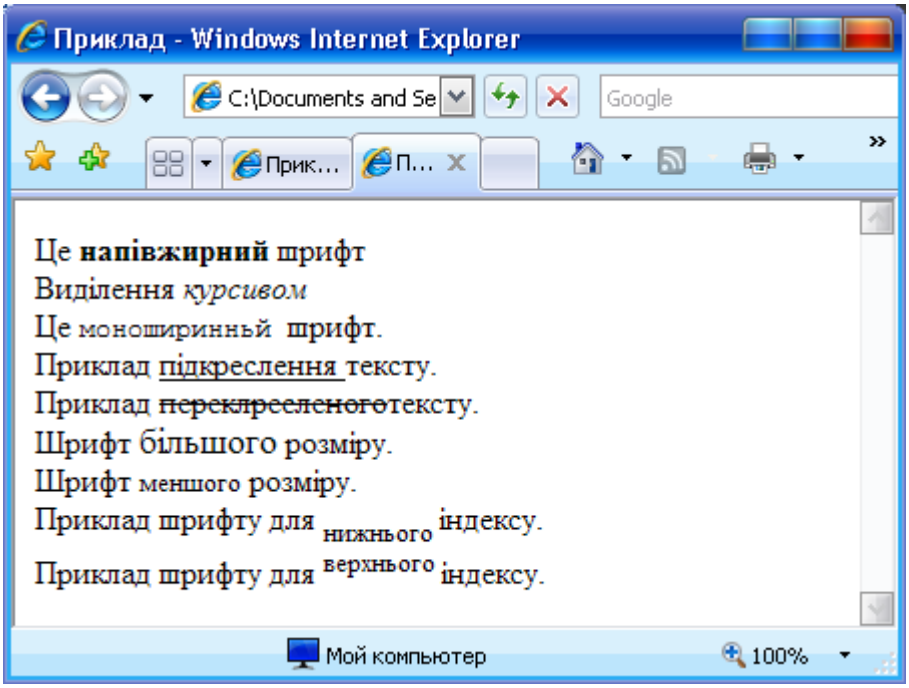

Рис 1.2.2 Теги фізичного форматування тексту

Теги форматування можуть бути вкладеними один в одного. При цьому потрібно уважно стежити, щоб один контейнер знаходився цілком в іншому контейнері.

На Рис 1.2.3 показаний приклад використання вкладення елементу курсиву в елемент напівжирного шрифту. Використаний наступний фрагмент HTML-коду:

```
Це <B> напівжирний </B> шрифт.
<P>
Це <I> курсив </I>.
</P>
А тут текст <B><I> напівжирний і курсив
</I></B>
```

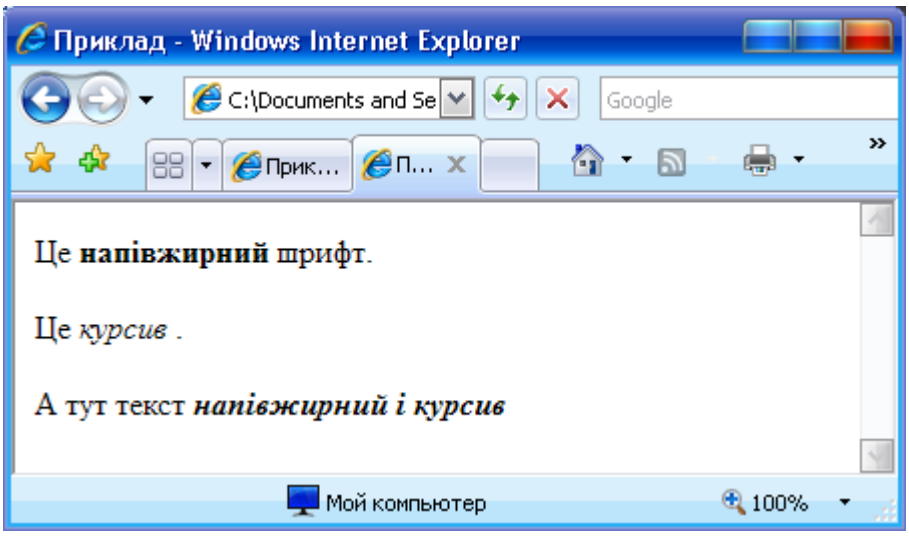

Рис 1.2.3 Результат вкладання тегів форматування тегів

Тег <FONT>

Тег <FONT> вказує параметри шрифту. Він відноситься до тегів фізичного форматування рівня тексту.

Призначення параметрів шрифту безпосередньо в тексті документа порушує основну ідею розділення змістовної частини документа і опису форми представлення документа. Тому в специфікації HTML 4.0 даний тег, а також тег <BASEFONT> відносяться до відмінених. Їх подальше застосування не рекомендується.

Не дивлячись на ці попередження, мабуть, для найпростіших документів фізичне форматування можна вважати допустимим.

Тег <FONT> відноситься до рядкових елементів, тому не може включати елементи рівня блоку, наприклад, <P> чи <TABLE>.

Для тега можуть задаватися наступні параметри: FACE, SIZE і COLOR.

FACE

Параметр FACE служить для вказівки типу шрифту, яким програма перегляду користувача виводитиме текст (якщо такий шрифт є на комп'ютері). Значенням даного параметра служить назва шрифту, яка повинна в точності співпадати з назвою шрифту, що є у користувача. Якщо такого шрифту не буде знайдено, то дана вказівка буде проігнорована і буде використаний шрифт, встановлений по замовчуванню.

Можна вказати як один, так і декілька назв шрифтів, розділяючи їх комами. Це вельми корисна властивість, оскільки в різних системах можуть бути майже ідентичні шрифти, що називаються по-різному.

Іншою важливою якістю є завдання переваги використання шрифтів. Список шрифтів є видимим зліва направо. Якщо на комп'ютері користувача немає шрифту, вказаного в списку першим, то робиться спроба знайти наступний шрифт і т.д.

Приведемо приклад використання параметра FACE:

```
Текст, записаний шрифтом по замовчуванню.
<BR>
<FONT FACE="Verdana", "Arial", "Helvetica">
  Приклад завдання назви шрифту.
</FONT>
```

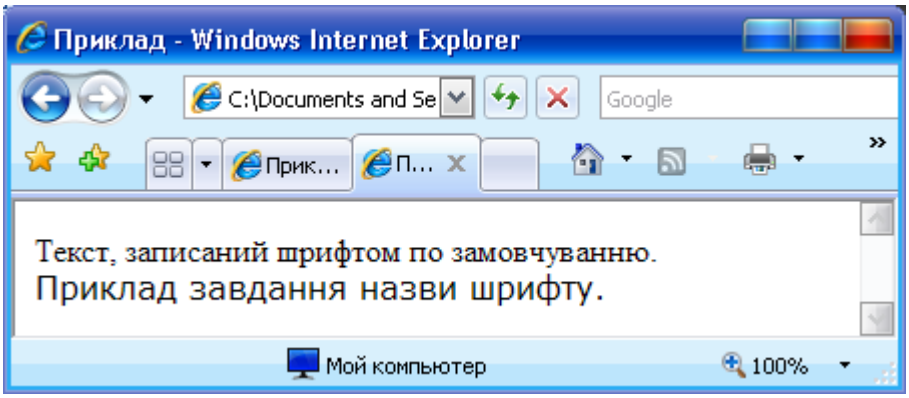

Рис 1.2.4 Приклад використання параметру FACE

На Рис 1.2.4 показано відображення прикладу. У прикладі як віддається перевага указується шрифт Verdana, при його відсутності буде використаний шрифт Arial і т.д.

SIZE

Цей параметр служить для вказівки розмірів шрифту в умовних одиницях від 1 до 7. Конкретний розмір шрифту залежить від використовуваної програми перегляду. Прийнято вважати, що розмір "нормального" шрифту відповідає значенню 3.

Налаштування розмірів шрифту, використовуваних по замовчуванню, а також величини абсолютної зміни розмірів шрифту залежать від браузерів.

Розмір шрифту вказується як абсолютною величиною (SIZE=2), так і відносною (SIZE=+1). Останній спосіб часто використовується у поєднанні із завданням базового розміру шрифту за допомогою тега <BASEFONT>.

Приведемо приклад, в якому використані різні способи призначення розмірів шрифтів. Відображення прикладу показане на рис.1.6.

```
<FONT SIZE =1>Шрифт розміру 1</FONT><BR>
<FONT SIZE =-1>Шрифт розміру 2</FONT><BR>
<FONT SIZE =3>Шрифт розміру 3</FONT><BR>
<FONT SIZE =4>Шрифт розміру 4</FONT><BR>
<FONT SIZE =5>Шрифт розміру 5</FONT><BR>
<FONT SIZE =+3>Шрифт розміру 6</FONT><BR>
<FONT SIZE =7>Шрифт розміру 7</FONT><BR>
```

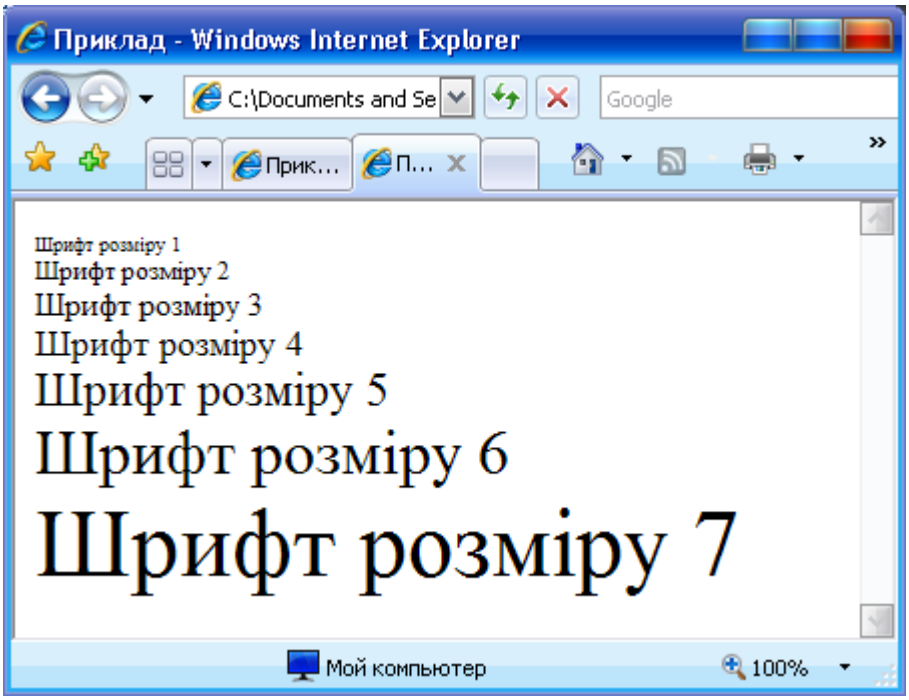

Рис 1.2.5 Приклад різних розмірів шрифта

COLOR

Цей параметр встановлює колір шрифту, який може задаватися за допомогою стандартних імен або у форматі #RRGGBB. Приведемо приклад документа з різноколірним текстом.

```
<FONT COLOR=green> Текст зеленого кольору
</FONT><BR>
<FONT COLOR=#FF0000> Текст червоного кольору
</FONT><BR>
```

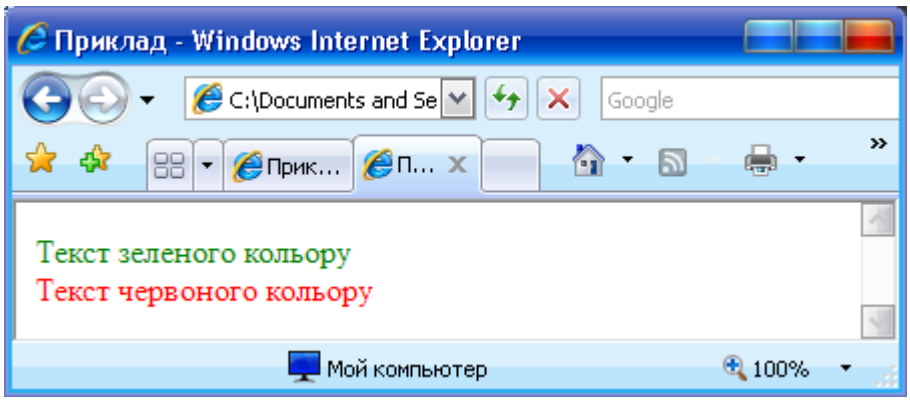

Рис 1.2.6 Приклад зміни кольору шрифта

Тег <BASEFONT>

Тег <BASEFONT> використовується для вказівки розміру, типу і кольору шрифту, використовуваного в документі по замовчуванню. Ці значення обов'язкові для всього документа, проте можуть в потрібних місцях перевизначитися за допомогою тега <FONT>. Після закриваючого тега </FONT> дія тега <BASEFONT> відновлюється. Значення параметрів шрифтів, використовуваних по замовчуванню, можуть неодноразово перевизначитися в документі, тобто тег <BASEFONT> може з'являтися в документі будь-яку кількість разів.

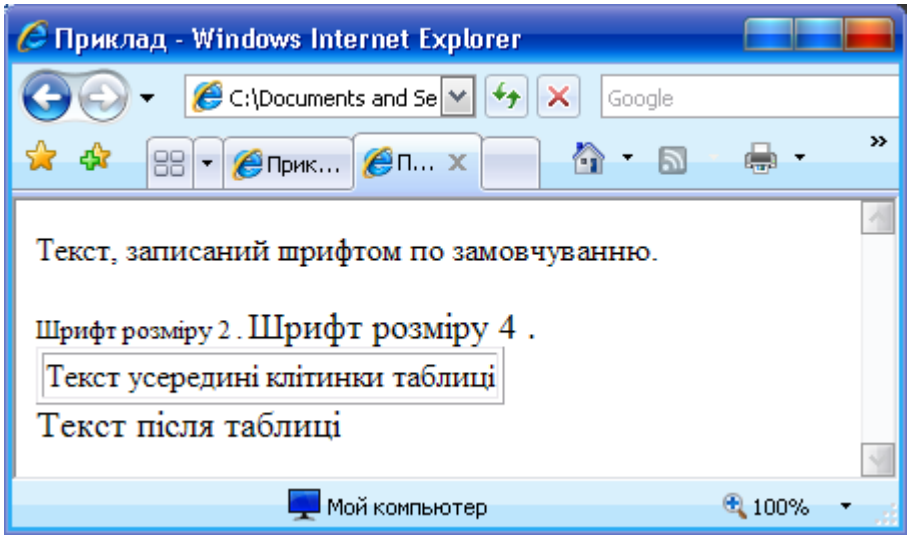

Рис 1.2.7 Приклад використання тегу BASEFONT

```
Текст, записаний шрифтом по замовчуванню.
<BASEFONT SIZE=2>
<P>
Шрифт розміру 2 .
<BASEFONT SIZE=4>
Шрифт розміру 4 .
<TABLE BORDER=1>
  <TR>
```

```
    <TD> Текст усередині клітинки таблиці </TD>
  </TR>
</TABLE>
Текст після таблиці
```

У приведеному прикладі двічі перевизначається розмір шрифту, використовуваного по замовчуванню. Спочатку він рівний 3 (по замовчуванню). Потім встановлюється рівним 2, далі - 4. Зверніть увагу на відображення даного прикладу (Рис 1.2.7). Видно, що для таблиць призначення тега <BASEFONT> не діє. Це характерно для багатьох браузерів, хоча формально порушує ідею застосування тега.

1.2.6Форматування документа

Розділення на абзаци

Одним з перших правил складання практично будь-яких документів є розбиття його тексту на окремі абзаци, що виражають закінчену думку. HTML-документи не є виключенням з цього правила. При створенні документів за допомогою текстових редакторів розбиття на абзаци виконується введенням символу переносу рядка. У HTML-документах символи переносу рядка не приводить до утворення нового абзацу.

Мова HTML припускає, що автор документа нічого не знає про комп'ютер свого читача. Читач має право встановити будь-який розмір вікна і користуватися будь-яким з шрифтів, що є у нього. Це означає, що місце перенесення в рядку визначається тільки програмою перегляду і установками кінцевого користувача. Оскільки символи переносу рядка оригінального документа ігноруються, то текст, що відмінно виглядав у вікні редактора автора документа, може перетворитися на нечитабельний текст у вікні програми перегляду.

Уникнути цієї неприємності дозволяє тег розділення на абзаци <P>. Перед початком кожного абзацу тексту слід помістити тег <P>. Закриваючий тег </P> не обов'язковий. Браузери зазвичай відокремлюють абзаци один від одного порожнім рядком.

Таблиця 1.2.4 Значення параметра align

Параметр	Значення
LEFT	Вирівнює текст по лівій границі браузера
CENTER	Вирівнює текст по центру браузера

RIGTH	Вирівнює текст по правій границі браузера
JUSTIFY	Вирівнює текст по ширині браузера

Перенесення рядка

При відображенні текстових документів в браузері місце переходу рядка в межах абзацу визначається автоматично залежно від розміру шрифтів і розміру вікна перегляду. Перенесення рядка може здійснюватися тільки по символах-роздільниках слів (наприклад, пробілам). Іноді в документах потрібно задати примусове перенесення рядка, що реалізовується незалежно від параметрів налаштувань браузера. Для цього служить тег примусового перенесення рядка
, який не має відповідного закриваючого тега. Включення тега
 в текст документа забезпечить розміщення подальшого тексту з початку нового рядка.

На відміну від тега абзацу <P> при використанні тега
 не буде утворений порожній рядок.

Теги <NOBR> і <WBR>

Бувають ситуації, коли потрібно виконати операцію протилежного призначення - заборонити перехід рядка. Для цього існує тег-контейнер <NOBR>. Текст, розмічений цим тегом, гарантовано розташовуватиметься в одному рядку, незалежно від її довжини. Якщо рядок, що при цьому виводиться, виходитиме за межі вікна перегляду браузера, то з'явиться горизонтальна лінія прокрутки.

Розмічуючи текст за допомогою тега нерозривному рядка <NOBR> можна отримати дуже довгі рядки. Щоб цього уникнути, можна вказати браузеру читача місце можливого переходу рядка, що буде виконано тільки при необхідності (так званий "м'який" перехід рядка). Це можна здійснити, поставивши в потрібному місці тексту тег <WBR> (Word BReak), який також, як і тег
, не потребує закриваючого тегу.

Заголовки всередині HTML-документа

Разом з назвою всього документа, на Web-сторінці можуть використовуватися заголовки для окремих частин документа. Ці заголовки можуть мати шість різних рівнів (розмірів) і є фрагментами тексту, які виділяються на екрані при відображенні сторінки браузером.

Для розмітки заголовків використовуються теги <H1>, <H2>, <H3>, <H4>, <H5> і <H6>. Ці теги вимагають відповідного закриваючого тега. Заголовок з номером 1 є найкрупнішим (заголовок верхнього рівня), а з номером 6 - найдрібнішим. Теги заголовка є елементами рівня блоку, тому за допомогою них не можна розмічати окремі слова тексту для збільшення їх розміру. При використанні тегів заголовків здійснюється вставка порожнього рядка до і після заголовка.

В тегах заголовків можуть використовуватися параметри горизонтального вирівнювання ALIGN. Можливі значення параметра співпадають з параметрами вирівнювання тега абзацу <P> (Таблиця 1.2.4).

Приклад використання заголовків різного рівня з різним вирівнюванням (рис.1.9):

```
<H1> Заголовок розміру 1 </H1>
<H2> Заголовок розміру 2 </H2>
<H3 ALIGN="center"> Заголовок розміру 3 </H3>
<H4 ALIGN="right"> Заголовок розміру 4 </H4>
<H5> Заголовок розміру 5 </H5>
<H6> Заголовок розміру 6 </H6>
Основний текст документа
```

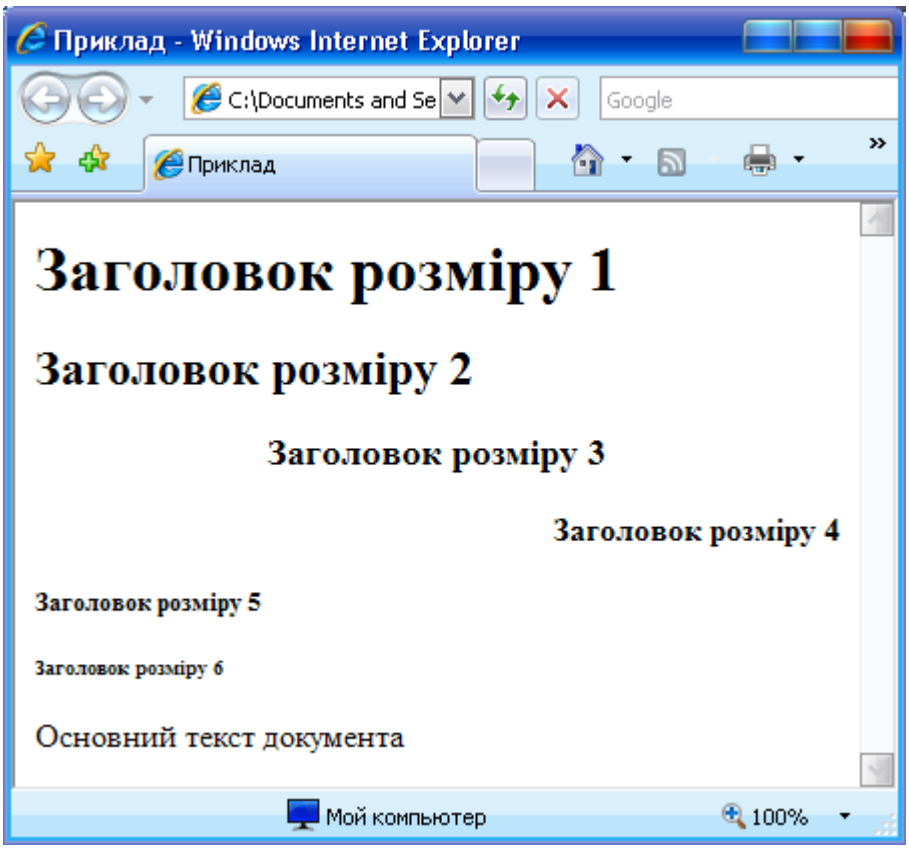

Рис 1.2.8 Відображення заголовків різного рівня

Горизонтальні лінії

Іншим методом розділення документа на частини є проведення горизонтальних ліній. Вони візуально підкреслюють закінченість тієї або іншої області сторінки.

Тег <HR> дозволяє провести рельєфну горизонтальну лінію у вікні більшості програм перегляду. Цей тег не є контейнером, тому не вимагає закриваючого тега. До і після лінії автоматично вставляється порожній рядок. Параметри тега <HR> представлені в Таблиця 1.2.5

Таблиця 1.2.5 Параметри тега <HR>

Параметр	Значення
ALIGN	Вирівнює лінію по краях або центру; має значення LEFT, CENTER, RIGHT
WIDTH	Встановлює довжину лінії в пікселях або відсотках від ширини вікна браузера
SIZE	Встановлює товщину лінії в пікселях
NOSHADE	Скасовує рельєфність лінії
COLOR	Вказує колір лінії. Використовується формат RGB або стандартне ім'я

Приклад використання горизонтальної лінії Рис 1.2.9

```
<HR ALIGN="center" width="50%" NOSHADE>
```

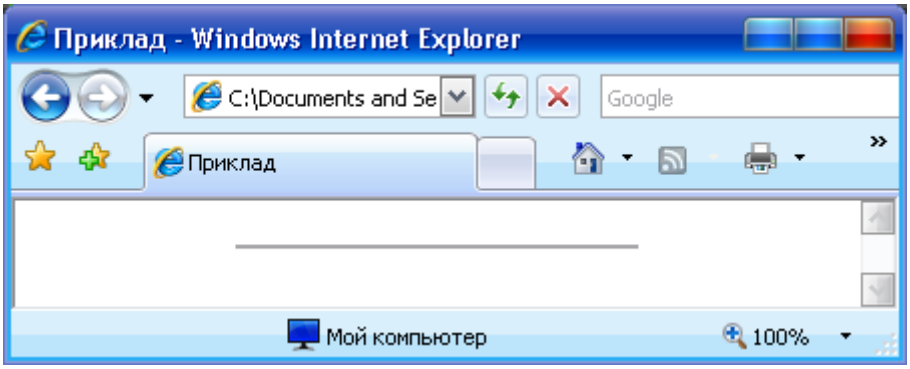

Рис 1.2.9 Приклад горизонтальної лінії

Використання тексту, що заздалегідь відформатований

Для розбиття тексту по абзацах і забезпечення примусового перенесення рядка слід користуватися спеціальними тегами. Проте бувають випадки, коли в HTML-документ необхідно включити текст, що вже має форматування, виконане традиційним способом за допомогою символів перенесення рядка, необхідної кількості пробілів, символів табуляції і т.д. Для вирішення таких завдань передбачений спеціальний тег-контейнер <PRE>, що визначає заздалегідь форматований текст. Текст, розмічений тегом <PRE>, буде відображатися у такому вигляді, як він виглядає в звичайному текстовому редакторові. Для відображення завжди використовуватиметься моноширинний шрифт. При цьому можна контролювати виведення документа програмою перегляду.

Тег <DIV>

Тег-контейнер <DIV> являється елементом рівня блоку, служить для виділення фрагмента документа. Метою цього виділення є зазвичай зміна параметрів фрагмента за допомогою призначення стилів. Приведемо приклад:

```
<DIV STYLE="color: green">
(Фрагмент документа)
</DIV>
```

В даному випадку всі текстові елементи виділеного фрагмента відображатимуться зеленим (green) кольором. Аналогом тега <DIV> рівня тексту є елемент <SPAN>.

Тег <CENTER>

Тег-контейнер <CENTER> призначений для горизонтального вирівнювання всіх елементів посередині вікна браузера. Він має рівень блоку і його корисно використовувати для центрування таких елементів, як таблиці.

Включення коментарів в документ

У HTML-документ можна включати коментарі, які не будуть видимі читачеві. Вони можуть складатися з довільної кількості рядків і повинні починатися тегом <!-- і закінчуватися тегом -->. Все, що знаходиться усередині цих тегів, при прогляданні сторінки не відображатиметься на екрані.

Коментарі зазвичай використовуються авторами документа для заміток, призначених тільки для власного використання. Текст коментарів не відображається на екрані браузера, проте передається разом з документом і цілком може бути переглянутий читачами. Більшість браузерів надають можливість проглядання початкового коду документа.

Коментарі в HTML застосовуються також для того, щоб "заховати" від браузера скрипти у випадку, якщо він не в змозі розпізнати їх.

Тег <BLOCKQUOTE>

Бувають випадки, коли в текст HTML-документа необхідно включити яку-небудь довгу цитату. Для виділення цитат з основного тексту існує тег <BLOCKQUOTE>. Він є контейнером і може містити будь-які теги форматування.

На відміну від тега <Q>, призначеного для виділення коротких цитат в рядку текста, <BLOCKQUOTE> являється тегом рівня блоку. Текст, розмічений даним тегом, при відображенні відділяється від основного тексту порожніми рядками і, як правило, виводиться з невеликим відступом управо.

```
Оновний текст документа
<BLOCKQUOTE>
Приклад відображення блокової цитати, яка
відділяється пустими рядками і відступає вправо
</BLOCKQUOTE>
Оновний текст документа
```

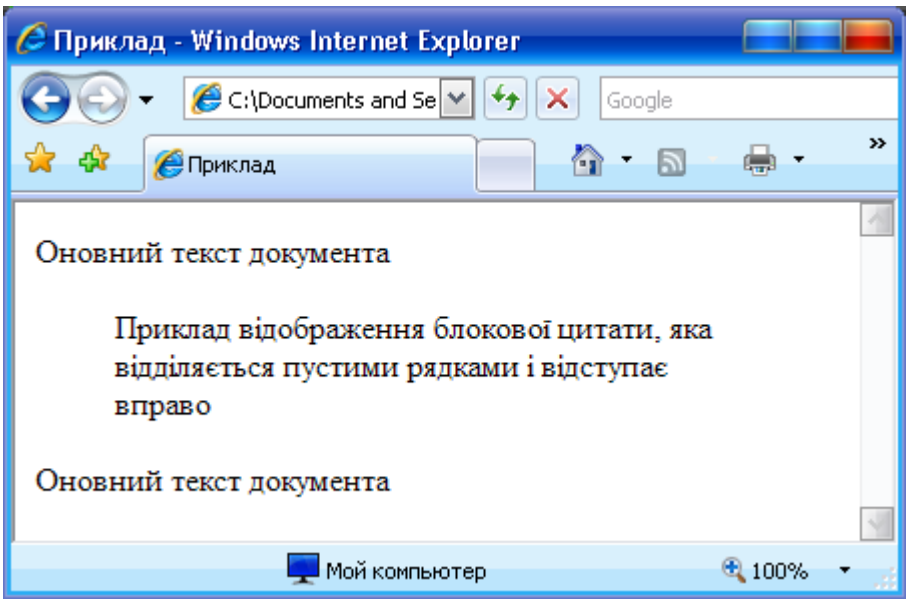

Рис 1.2.10 Приклад відображення блокової цитати

Тег <ADDRESS>

Тег <ADDRESS> використовується для ідентифікації автора документа і для вказівки адреси автора. Сюди ж зазвичай поміщаються відомості про авторські права. Цей елемент розташовується або на початку, або в самому кінці документа.

Текст, що знаходиться між цими тегами, зазвичай відображається браузерами курсивом.

Спеціальні символи

Деякі спеціальні символи не входять в базову частину таблиці кодів ASCII. До них відносяться букви алфавітів частини європейських мов, математичні і деякі інші символи. Деякі символи, безпосередньо введені в HTML-документ, будуть інтерпретовані не так, як задумав автор. До них відносяться символи "<" і ">", що зазвичай використовуються для вказування тегів мови.

У таких випадках можна вводити потрібні символи у HTML-документ за допомогою спеціальних кодів. Ці коди складаються з символу амперсанда (&) і наступним за ним ім'ям символу або його десятковим чи шістнадцятковим значенням. Закінчуватися спеціальний символ повинен знаком "крапка з комою".

У специфікації HTML приводяться цілі таблиці із спеціальними символами і їх значеннями.

Спеціальний символ	Значення
&lt	Знак «менше»
&gt	Знак «більше»

` `	Нерозривний пробіл
`&copy`	Знак «копірайт»
`&amp`	Амперсанд
`&quot`	Знак «лапки»

Всі символи можуть бути також задані своїми кодами. Наприклад, символ нерозривного пробілу має код `160`. Він може записуватися в десятковому вигляді як ` `.

1.3 Гіперпосилання

Одним з найважливіших понять для HTML-документів є посилання.

Сама назва - HTML, мова розмітки гіпертексту, указує на принцип організації таких документів. Посилання є єдиною можливістю перейти від одного документа до іншого.

Гіпертекстовий документ - це документ, що містить посилання на інші документи, що дозволяють за допомогою натиснення кнопки миші швидко переміщатися від одного документа до іншого. Часто подібні посилання можна побачити і у файлах допомоги сучасних програмних продуктів. За основу гіпертексту взятий принцип організації енциклопедичних словників, де в багатьох статтях є посилання на інші.

Існує багато типів мультимедійних об'єктів, які можуть бути розміщені на Web-сторінці. У сучасних гіпертекстових документах як додаток до самого тексту часто використовують різноманітну графіку, відео- і аудіооб'єкти, а як посилання часто застосовують зображення.

1.3.1Організація посилань

Посилання складається з двох частин. Перша з них - це те, що ви бачите на Web-сторінці; вона називається покажчик посилання (anchor). Друга частина, що дає інструкцію браузеру, називається адресною частиною посилання (URL-адреса). Клацаючи мишею по покажчику посилання, браузер завантажує документ, адреса якого дається URL-адресою. Покажчиком посилання може бути слово, група слів або зображення. Зовнішній вигляд посилання залежить від його типу, способів створення і установок програми перегляду читача. Вказівники бувають двох типів - текстові і графічні.

Текстові вказівники зазвичай є словом або декількома словами, виділеними на екрані підкресленням. Колір текстового вказівника може регулюватися автором і установками програми перегляду.

Приведемо приклад запису для текстового вказівника посилання:

```
<A HREF="example.html">Цей текст є вказівником
</A>
```

Як посилання можна використовувати графічне зображення. За принципом дії графічні посилання нічим не відрізняються від текстових.

Вони не підкреслюються і не виділяються кольором, а для їх виділення браузери зазвичай навколо такого зображення малюють рамку. Приклад графічного вказівника посилання:

```
<A HREF="example.html"><IMG
SRC="picture.gif"></A>
```

Другою частиною посилання є URL-адреса. Це не що інше, як адреса Web-сторінки, яка буде завантажена при клацанні мишею на вказівнику. Вказівка адреси може бути відносною або абсолютною. Якщо в URL-адресі не вказується повний шлях до файлу, то таке посилання називається відносним. В цьому випадку визначення місцерозташування файлів виконується з урахуванням місцерозташуванням документа, в якому є таке посилання. Наприклад, якщо браузер завантажив сторінку, що знаходиться за адресою `http://www.mysite.com/page`, то відносний вказівник `/picture` вказує адресу `http://www.mysite.com/page/picture`, тобто підкаталог, розташований на цьому ж комп'ютері.

1.3.2Правила запису посилань

Для організації посилання необхідно повідомити браузеру, що є вказівником посилання, а також вказати адресу документа, на який посилається вказівник. Обидві дії виконуються за допомогою тега `<A>`.

1.3.3Тег <A>

Тег `<A>` має єдиний параметр `HREF`, значенням якого є URL-адреса. Покажчик може бути як відносним, так і абсолютним, наприклад, `http://www.server.com/home/index.htm`. Цей тег є контейнером, тому необхідно поставити закриваючий тег `</A>`:

```
<A HREF=URL-адрес> Текстовий вказівник
посилання </A>
```

Вказівник посилання може бути відносним або абсолютним. Для полегшення роботи з відносними вказівниками посилань введений тег `<BASE>`. Він розташовується на початку документа в розділі `HEAD` і містить URL-адресу відносно якої в документі побудована вся адресація. Цей вказівник впливає на будь-який тег документа, в

якому використовується відносна адресація. Якщо тег <BASE> відсутній, то адресація будується відносно адреси поточного документа.

1.3.4Внутрішні посилання

Окрім посилань на інші документи, часто буває корисно включити посилання на різні частини поточного документа. Наприклад, великий документ читається краще, якщо він має зміст з посиланнями на відповідні розділи.

Для побудови внутрішнього посилання спочатку потрібно створити вказівник, що визначає місце призначення. Наприклад, якщо потрібно зробити посилання текст певного розділу документа, потрібно розмістити там покажчик і дати йому ім'я за допомогою параметра NAME тега <A>. При цьому параметр HREF не використовується, і браузер не виділяє вміст тега <A>. Наприклад:

```
<A NAME=chapter_5></A>
```

В приведеному прикладі відсутній вміст тега <A>. Переважно саме так і роблять, оскільки тут немає необхідності якось виділяти текст, а потрібно лише вказати місцерозташування.

Після того, як місце призначення визначене, можна приступити до створення посилання на нього. Для цього, замість вказівки в параметрі HREF адреси документа, як це робилося раніше, помістимо туди ім'я посилання з префіксом #, що говорить про те, що це внутрішнє посилання.

```
<A HREF="#chapter_5"> Розділ 5</A>
```

Тепер, якщо користувач клацне кнопкою миші на словах "розділ 5" браузер виведе відповідну частину документа у вікні перегляду.

1.3.5Посилання на документи різних типів

Коли користувач клацає мишею на посиланні, що вказує на іншу Web-сторінку, вона виводиться безпосередньо у вікні браузера. Якщо ж посилання вказує на документ іншого типу, програма перегляду приймає документ і потім вирішує, що з ним робити далі. Наступними діями браузера можуть бути:

- Браузер знає цей тип документа і вміє з ним працювати. Наприклад якщо створити посилання на графічний файл формату GIF і користувач клацнув мишею на цьому посиланні, його програма перегляду очистить вікно і завантажить вказане

зображення. В деяких випадках браузер може додатково використовувати програмний модуль, що підключається (plug-in), без якого задача не була б вирішена.

- Браузер не розпізнає тип прийнятого документа і не знає, що з ним робити далі. В цьому випадку він звернеться до допоміжних програм, що є на машині користувача. Якщо відповідна програма знайдеться, браузер запустить її і передасть їй отриманий документ для обробки. Наприклад, якщо користувач клацне на посиланні на відеофайл формату AVI, браузер завантажить файл, знайде програму для демонстрації AVI-файлів і запустить її. Відеофайл буде показаний в додатковому невеликому вікні.

1.3.6 Посилання на інші ресурси Інтернету

Web-простір є лише частиною мережі Інтернет. Інші ресурси почали своє існування задовго до народження WWW, тому накопичили вже багато інформації і мають чималу аудиторію. Тому, розробляючи свою персональну сторінку або документ, можливо, необхідно включити посилання і на інші ресурси.

Ресурси Інтернету вельми різноманітні за формою і змістом. Хоча HTML припускає можливість створення своїх власних версій цих ресурсів за допомогою механізму обробки даних форм, є простіші шляхи до взаємодії з системами UseNet, Telnet, FTP, e-mail і іншими.

Створити посилання на електронну пошту так само просто, як і на іншу сторінку. Для цього замість URL-адреси слід вказати адресу електронної пошти, якій передує слово `mailto`:

```
<A HREF="mailto:mail@mail.ua"> Присилайте ваші
відгуки і пропозиції </A>.
```

Це посилання не буде нічим відрізнятися від решти гіпертекстових посилань вашого документа. Те ж саме можна сказати і про посилання на інші ресурси Інтернету.

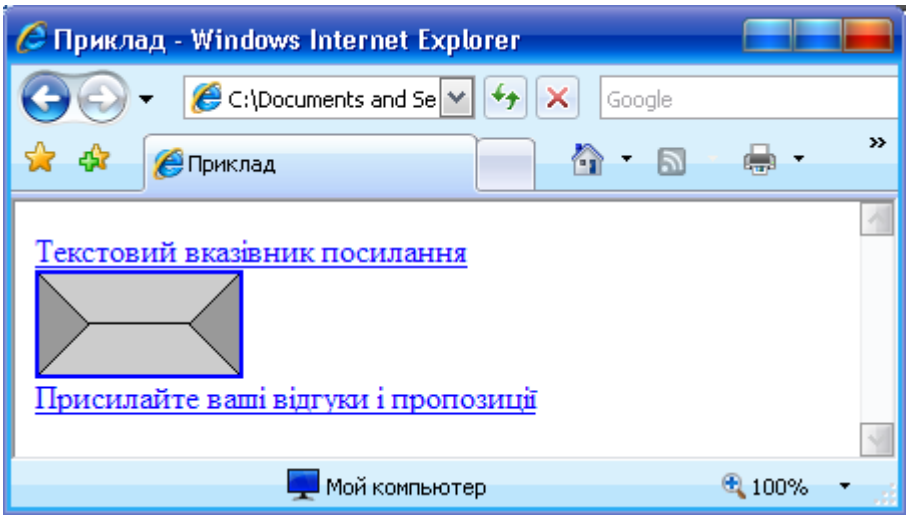

Рис 1.3.1 Приклади відображення гіпепосилань

1.4 Графіка в HTML

Для того щоб вставити в Web-сторінку зображення, необхідно або намалювати його, або взяти вже готове. У будь-якій програмі рисування можна створити просте зображення і зберегти його у відповідному форматі. Якщо програма цей формат не підтримує, необхідно перетворити файл в потрібний формат. Існує безліч програм, призначених для перетворення одного графічного формату в інший. Взяти ж картинки можна з різних програмних пакетів або з інших Web-сторінок в Internet, що містять бібліотеки вільного доступу художніх зображень. Коли браузер виводить Web-сторінку з зображенням, відповідний графічний файл тимчасово зберігається в пам'яті комп'ютера. У більшості браузерів є команда, яка дозволяє зберегти файл на локальному диску. Існує також безліч інших варіантів отримання графічних файлів.

Розглянемо як вставити зображення в Web-сторінку. Тегом HTML, який змушує браузер виводити зображення, є <IMG> з обов'язковим атрибутом SRC (SouRCe, джерело). Файл представляє собою ім'я виведеного графічного файлу. Завершального тега не потрібно. Приклад вставки зображення:

```
<IMG SRC="image.gif" ALT="рисунок">
```

Зображення на Web-сторінці можуть використовуватися в якості гіпертекстових посилань, як і звичайний текст. Читач клацає на зображенні і відправляється на іншу сторінку або переходить до іншого зображення. Для позначення зображення як гіпертекстової мітки використовується той же тег <A>, що і для тексту, але між <A> і </A> вставляється тег зображення <IMG>:

```
<A HREF="URL-адрес"> <IMG SRC="image.gif"> </A>
```

При цьому зображення, що використовується як гіпертекстове посилання, обводиться додатковою рамкою.

1.4.1Атрибути тегу IMG

Тег зображення має один обов'язковий атрибут SRC і необов'язкові: ALT, ALIGN, USEMAP, HSPACE, VSPACE, BORDER, WIDTH, HEIGHT.

Атрибут SRC

Вказує файл зображення і шлях до нього; зображення повинно бути додано в браузер і розміщене в тому місці документа, де розташований тег зображення.

Атрибут ALT

Дозволяє вказати текст, який буде виводитися замість зображення браузерами, нездатними відображати графіку. У деяких випадках при недостатній швидкості користувачі відключають відображення графіки. Наявність назв замість картинок полегшує сприйняття Web-сторінок в такому режимі.

Атрибут ALIGN

Визначає розташування зображення щодо навколишнього його тексту. Можливі значення аргументу.

- `top` - вирівнює верх зображення по верхньому краю самого високого елемента в рядку навколишнього тексту;
- `middle` - вирівнює центр зображення по базовій лінії рядка навколишнього тексту;
- `bottom` - вирівнює нижній край зображення по базовій лінії рядка навколишнього тексту;
- `left` - визначає обтікання тексту зображення. Зображення розташовується вздовж лівої межі документу, а подальші рядки тексту огинають його справа;
- `right` - визначає обтікання текстом зображення. Зображення розташовується уздовж правої межі документа, а наступні рядки тексту огинають його сторінки.

Атрибут USEMAP

Якщо присутні атрибут `USEMAP` та теги `<MAP>`, зображення стає графічною картою, або "графічним меню". Якщо клацнути кнопкою миші на активній області зображення, для якого визначено атрибут `USEMAP`, відбудеться гіпертекстовий перехід до інформаційного ресурсу, встановленому для цієї області. Більш докладно це питання буде розглядатися в наступному розділі.

Атрибут BORDER

Цілочисельне значення аргументу визначає товщину рамки навколо зображення. Якщо значення дорівнює нулю, рамка відсутня.

Атрибут HSPACE

Цілочисельне значення цього атрибута задає горизонтальну відстань між вертикальною межею сторінки і зображенням, а також між зображенням і текстом що його обтікає.

Атрибут VSPACE

Цілочисельне значення цього атрибута задає вертикальну відстань між рядками тексту і зображенням.

Атрибути WIDTH і HEIGHT

Обидва атрибути задають цілочисельні значення розмірів зображення по горизонталі і по вертикалі відповідно. Це дозволяє зменшити час завантаження сторінки з графікою. Браузер відразу відводить рамку для зображення і продовжує завантажувати текст на сторінку. Поки завантажується графіка, користувач може почати читати текст. Визначити розмір зображення неважко, для цього достатньо скористатися будь-якою програмою перегляду графічних файлів, наприклад ACDSee або графічним редактором Corel PhotoPaint чи Adobe Photoshop. Відкрийте файл у графічному редакторі та визначте розмір картинки в пікселах. У тезі зображення варто вказати ширину і висоту зображення.

```
<IMG SRC = "image.gif" ALT = "зображення" WIDTH
= "100" HEIGHT = "200" HSPACE = "10" VSPACE =
"10" BORDER = "2" ALIGN = "left">
```

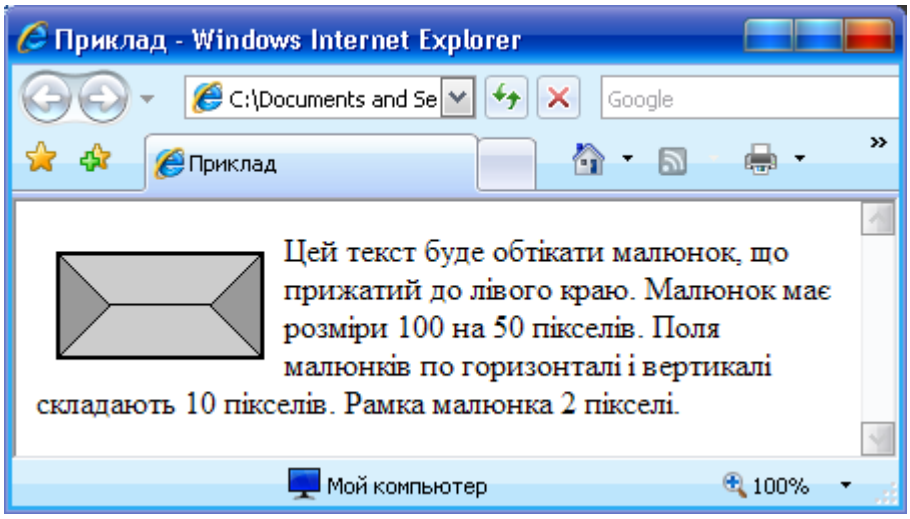

Рис 1.4.1 Вставка малюнку в HTML - документ

1.4.2Графічні формати

Спочатку HTML підтримував роботу із двома графічними форматами: GIF і JPEG, останнім часом все більше поширення

знаходить формат PNG. Кожен з форматів має свої переваги й недоліки, які визначають область його застосування.

GIF - Graphics Interchange Format

Формат GIF був розроблений комерційною Online-службою CompuServe для передачі графіки по мережі між різними платформами. GIF підтримує до 256 кольорів, що є істотним обмеженням формату. GIF практично непридатний для передачі фотографічних зображень.

У той же час формат GIF має низку властивостей, які визначили його широке застосування при розробці Web-сторінок:

Прозорість. Зображення у форматі GIF можуть мати прозорі області, що дозволяє відійти від стандартних прямокутних форм і створювати елементи сторінки довільної форми. Щоправда це неповноцінна прозорість. Прозорим буде лише один певний колір.

Чергування рядків. Графічний файл може бути записаний таким чином, що при відображенні спочатку буде завантажуватися кожен четвертий рядок, розтягнутий на три сусідніх. Природньо, якість відображення буде доволі низькою. Потім, по мірі завантаження наступних рядків, зображення прийме остаточний вигляд. Така схема дозволяє користувачеві ще до завершення завантаження графічного файлу зрозуміти, що саме представлено на зображенні.

Анімація. Форматом GIF передбачена можливість створення анімованих зображень. Фактично це кілька зображень, записаних в один файл і чергуються із заданим інтервалом часу.

Область застосування GIF-файлів обмежується і застосованим алгоритмом стиснення зображень, чия ефективність якого залежить від різноманітності кольорів зображення наявності одноколірних областей. Найбільшу економію об'єму файлу дають зображення з невеликою кількістю кольорів і великими одноколірними областями. Тому формат GIF широко використовується для відображення графічних меню і кнопок, а також для створення технічних ілюстрацій (графіки, схеми, діаграми і т.п.). У той же час використання формату GIF для відображення повнокольорових фотографій практично не знаходить застосування внаслідок малої кількості передаваних кольорів і низьку ефективність алгоритму стиснення для таких зображень.

JPEG - Joint Photographic Experts Group

Формат JPEG був розроблений для передачі фотографій між різними платформами. Завдяки підтримці 24-бітової колірної палітри (мільйони кольорів) формат JPEG отримав широке розповсюдження для відображення ілюстрацій високої якості. При цьому використовуваний в JPEG алгоритм стиснення дозволяє варіювати розмір завантажуваного файлу в залежності від необхідної якості відображення ілюстрації.

Формат JPEG відрізняється від інших графічних форматів перш за все тим, що він використовує метод стиснення "з втратами". JPEG частково ідентифікує і видаляє ту інформацію, яка несуттєва для сприйняття зображення. У результаті JPEG може досягати високого рівня стиснення без помітних втрат у якості зображення.

Метод стиснення "з втратами" має багато реалізацій. JPEG досягає істотного стиснення за рахунок відкидання тієї графічної інформації, яка зазвичай не проявляється в реальних зображеннях. Однак при стисненні за допомогою JPEG зображень з чіткими контурами лінії починають помітно "тремтіти".

Так як JPEG припускає стиснення з втратами, при створенні файлів необхідно бути уважним. Більшість програм, що створюють такі файли, дозволяють задавати значення параметра якості зображення. Зазвичай воно варіює від нуля до ста. Нижні значення дозволяють при стисненні JPEG відкидати більше інформації, в результаті чого виходять файли меншого розміру. У свою чергу, високі значення обмежують кількість інформації, якою можна знехтувати під час стиснення.

PNG - Portable Network Graphics

Останнім часом все більшого поширення набуває формат PNG, який був розроблений для мережевих ілюстрацій з метою замінити формати JPEG і GIF і об'єднав в собі всі їхні переваги.

Формат PNG підтримує 24 -, 32 - і 48-бітову колірну палітру, 8 - і 16 - бітову палітру сірого, 8-бітову налаштовувану палітру. Реалізований в PNG алгоритм стиснення дозволяє стискати зображення краще ніж GIF (від 5 до 25%). Для нього реалізований механізм черезрядкового виведення першого зображення з використанням двовимірної чергування рядків, що дозволяє забезпечити мінімальні витрати часу на виведення першого зображення.

Крім того, в PNG реалізований ефективний механізм контролю цілісності файлу, що дозволяє виявити помилки при передачі по мережі. PNG підтримує прозорість.

1.5 Списки

У мові HTML передбачений спеціальний набір тегів для відображення інформації у вигляді списків. Приведемо ряд випадків, для яких використання списків зручно:

- Об'єднання фрагментів інформації в єдину структуру для надання легкого для читання вигляду;
- Опис складних покрокових процесів;
- Розташування інформації в стилі змісту, пункти якого вказують на відповідні розділи документа.

У мові HTML передбачені наступні основні типи списків: маркований, нумерований і список визначень. Для реалізації списків різних типів використовуються наступні теги: <UL>, <OL>, <DL>, <DIR>, <MENU>. За допомогою різних типів вбудованих в документ списків можуть бути реалізовані різні можливості, а також застосуватись вкладеніодин в одного списки.

1.5.1Маркований список

Одним з типів списків, реалізованих в мові HTML, є маркований список. Інакше, списки такого типу називають ненумерованими або невпорядкованими. Остання назва часто використовується як формальний переклад назви відповідного тега <UL>, за допомогою якого і організовуються списки такого типу в HTML-документах (UL - Unordered List, неврегульований список).

У маркованому списку для виділення його елементів використовуються спеціальні символи, що називають маркерами списку. Вигляд маркерів списку визначається браузером, причому при створенні вкладених списків браузери автоматично змінюють вигляд маркерів різного рівня вкладеності.

Теги <UL> і <LI>

Для створення маркованого списку необхідно використовувати тег-контейнер <UL> </UL>, всередині якого розташовуються всі елементи списку. Відкриваючий і закриваючий теги списку забезпечують перенесення рядка до і після списку, відокремлюючи, таким чином, список від основного вмісту документа, тому тут немає необхідності використовувати теги абзацу <P> або примусового переносу рядка
.

Кожен елемент списку повинен починатися тегом <LI> (LI - List Item, елемент списку). Тег <LI> не потребує відповідного закриваючого тега хоча його наявність в принципі не забороняється. Браузери зазвичай при відображенні документа починають кожен новий елемент списку з нового рядка.

Приведемо приклад HTML-документа, що використовує маркований список, відображення якого браузером показане на Рис 1.5.1 Маркований списокрис.2.1.

```
<UL>
  <B>Пори року</B>
  <LI>Зима</LI>
  <LI>Весна</LI>
  <LI>Літо</LI>
  <LI>Осінь</LI>
</UL>
```

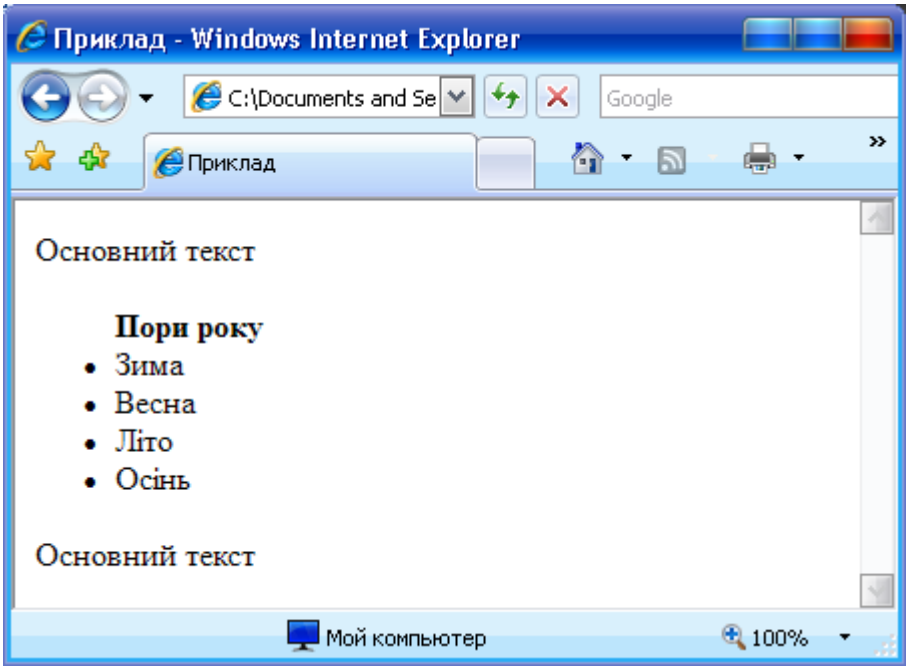

Рис 1.5.1 Маркований список

У тегу <UL> можуть бути вказано два параметри: COMPACT і TYPE.

Параметр COMPACT записується без значень і застосовується для вказівки браузеру, що даний список слід виводити в компактному вигляді. Наприклад, може бути зменшений шрифт або відстань між рядками списку і т.д.

Параметр TYPE може приймати наступні значення: disc, circle і square. Цей параметр використовується для

примусового завдання виду маркерів списку. Конкретний вид маркера залежатиме від використовуваного браузера.

Типовими варіантами відображення є наступні:

- `disc` - маркери відображаються зафарбованими кружками;
- `circle` - маркери відображаються не зафарбованими колами;
- `square` - маркери відображаються зафарбованими квадратиками.

Значенням, що використовується по замовчуванню, `disc`. Для вкладених маркованих списків на першому рівні по замовчуванню використовується значення `disc`, на другому - `circle`, на третьому і далі - `square`.

Параметр `TYPE` з тими ж значеннями може використовуватися для вказівки вигляду маркерів окремих елементів списку. Для цього параметр `TYPE` з відповідним значенням дозволено вказувати в тегу елементу списку `<LI>`.

```
<UL>
  <LI TYPE="disc">Круг</LI>
  <LI TYPE="circle">Коло</LI>
  <LI TYPE="square">Квадрат</LI>
</UL>
```

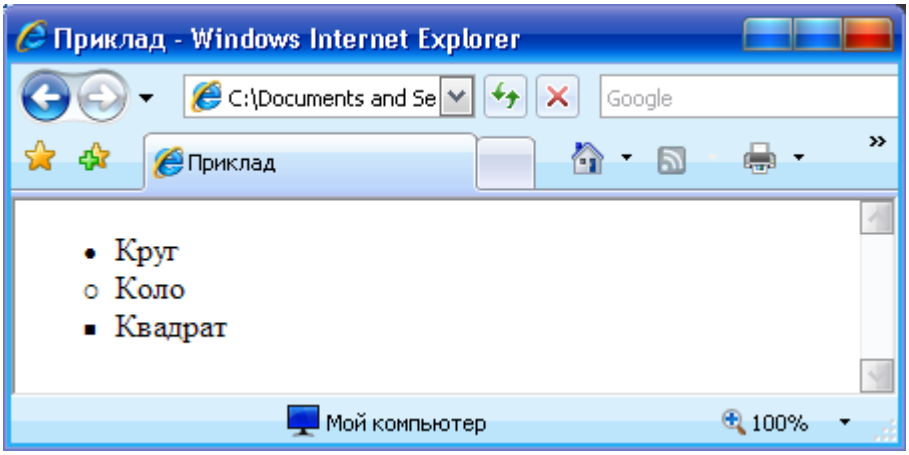

Рис 1.5.2 Різні типи маркерів

1.5.2Нумерований список

Іншим типом списків, реалізованих в мові HTML, є нумерований список. Інакше списки такого типу називають впорядкованими. Остання назва часто використовується як формальний переклад назви відповідного тега `<OL>`, за допомогою якого і організовуються списки такого типу в HTML-документах (OL - Ordered List, впорядкований список).

Списки даного типу зазвичай є впорядкованою послідовністю окремих елементів. Відмінністю від маркованих списків є те, що в нумерованому списку перед кожним його елементом автоматично проставляється порядковий номер. Вид нумерації залежить від браузера і може задаватися параметрами тегів списку. У іншому реалізація нумерованих списків багато в чому схожа на реалізацію маркованих списків.

Теги <OL> і <LI>

Для створення нумерованого списку слід використовувати тег-контейнер `<OL> </OL>`, всередині якого розташовуються всі елементи списку. Відкриваючий і закриваючий теги списку забезпечують перенесення рядка до і після списку, відокремлюючи таким чином список від основного вмісту документа.

Як і для маркованого списку, кожен елемент нумерованого списку повинен починатися тегом `<LI>`.

Приведемо приклад HTML-документа, що використовує нумерований список. Його вигляд показаний на Рис 1.5.3 Нумерований список.

```
<OL>
  <LI>Один</LI>
  <LI>Два</LI>
  <LI>Три</LI>
</OL>
```

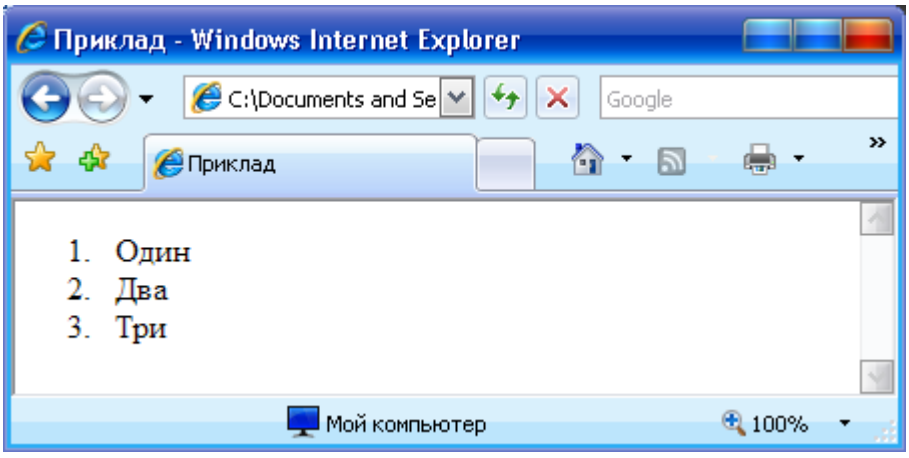

Рис 1.5.3 Нумерований список

У тегу `<OL>` можуть бути вказані наступні параметри: `COMPACT`, `TYPE` і `START`.

Параметр COMPACT має такий же зміст, що і у маркованих списків. Параметр TYPE використовується для завдання вигляду нумерації списку. Може приймати наступні значення:

- A - задає маркери у вигляді прописних латинських букв;
- a - задає маркери у вигляді рядкових латинських букв;
- I - задає маркери у вигляді великих римських цифр;
- i - задає маркери у вигляді маленьких римських цифр;
- 1 - задає маркери у вигляді арабських цифр.

По замовчуванню завжди використовується значення 1, тобто нумерація з допомогою арабських цифр. Це стосується і вкладених нумерованих списків.

Тут, на відміну від маркованих списків, браузери по замовчуванню не роблять різної нумерацію на різних рівнях вкладеності списків.

Відмітимо, що після номера елементу списку завжди додатково виводиться знак "крапка".

Параметр TYPE з тими ж значеннями може вживатися для вказування вигляду нумерації окремих елементів списку. Для цього параметр TYPE з відповідним значенням дозволено вказувати в тегу елементу списку <LI>.

Параметр START тега <OL> дозволяє почати нумерацію списку не з одиниці.

Як значення параметра START завжди повинне указуватися натуральне число, незалежно від виду нумерації списку. Приведемо приклад:

```
<OL TYPE=A START=5>.
```

Такий запис визначає нумерацію списку з прописної латинської букви "E". Для інших видів нумерації запис START=5 задасть нумерацію, відповідно, з числа "5", римскої цифри "V" і т.д.

Зміну вигляду нумерації списку і значень номерів допустимо проводити і для будь-якого елементу списку. Тег <LI> для нумерованих списків дозволяє використовувати параметри TYPE і VALUE. Параметр TYPE може приймати такі ж значення, як і для тега <OL>.

Значення параметра VALUE тега <LI> дозволяє змінити номер даного елементу списку. При цьому змінюється нумерація і всіх подальших елементів. Типовим застосуванням є списки з пропуском деяких елементів.

```
<OL START="5">
  <LI TYPE="A">Великі латинські букви</LI>
  <LI TYPE="a">Малі латинські букви</LI>
  <LI TYPE="I">Великі римські цифри</LI>
  <LI TYPE="i">Малі римські цифри</LI>
  <LI TYPE="1" VALUE="10">Цифри з заданням
конкретного номеру</LI>
</OL>
```

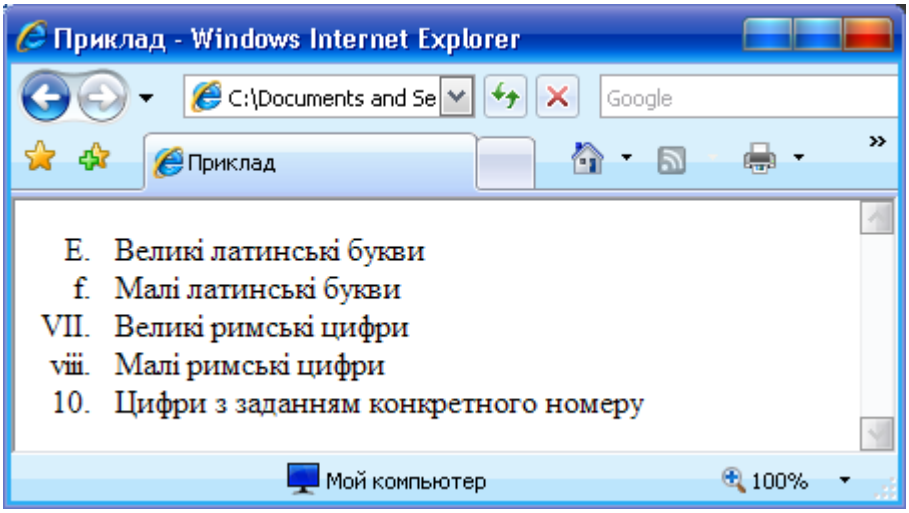

Рис 1.5.4 Різні типи нумерованих списків

1.5.3Список визначень

Списки визначень, так звані словники визначень спеціальних термінів, є особливим видом списків. На відміну від інших типів списків, кожен елемент списку визначень завжди складається з двох частин. У першій частині елементу списку записується термін, якому дається визначення, а в другій частині - текст у формі статті, що розкриває значення терміну.

Списки визначень задаються за допомогою тега-контейнера <DL> (Definition List). Усередині контейнера тегом <DT> (Definition Term) позначається визначуваний термін, а тегом <DD>(Definition Description) - абзац з його визначенням. Для тегів <DT> і <DD> можна не записувати відповідні закриваючі теги.

Загалом, список визначень записується таким чином:

```
<DL>
  <DT> Термін
  <DD> Визначення терміну
</DL>
```

У тексті після тега <DT> не можуть використовуватися елементи рівня блоку такі як, наприклад, теги абзацу <P> чи заголовків <H1> - <H6>. Як правило, текст визначуваного терміну повинен розташовуватися в одному рядку.

Текст, що містить визначення терміну, виводиться, починаючи з наступного рядка (або через рядок для деяких браузерів) після визначення терміна з відступом вправо. В інформації, поміщеній після тега <DD>, можуть розташовуватися елементи рівня блоку. Звідси слідує, зокрема, що списки визначень можуть бути вкладеними.

У тезі <DL> може бути вказаний параметр COMPACT, призначення якого аналогічне вище описаним спискам.

Приведемо приклад HTML-документа, в якому використаний список визначень:

```
<DL>
<DT> HTML
<DD>Гіпертекстова мова розмітки
<DT> CSS
<DD> Каскадні таблиці стилів
<DT> JavaScript
<DD> Мова програмування клієнтських сценаріїв
</DL>
```

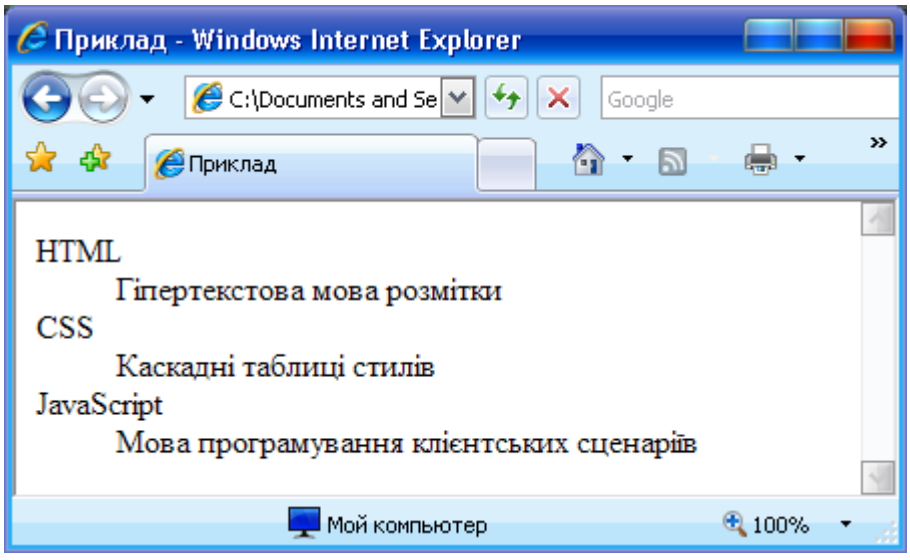

Рис 1.5.5 Список визначень

1.5.4Вкладені списки

Бувають випадки, коли в елемент списку одного типу потрібно включити цілий список такого ж типу або іншого. При цьому будуть

організовані багаторівневі або вкладені списки. У HTML допустимо довільне вкладення різних типів списків.

```
<UL>
  <LI>Весна</LI>
  <OL>
    <LI>Березень</LI>
    <LI>Квітень</LI>
    <LI>Травень</LI>
  </OL>
  <LI>Осінь</LI>
  <OL>
    <LI>Вересень</LI>
    <LI>Жовтень</LI>
    <LI>Листопад</LI>
  </OL>
</UL>
```

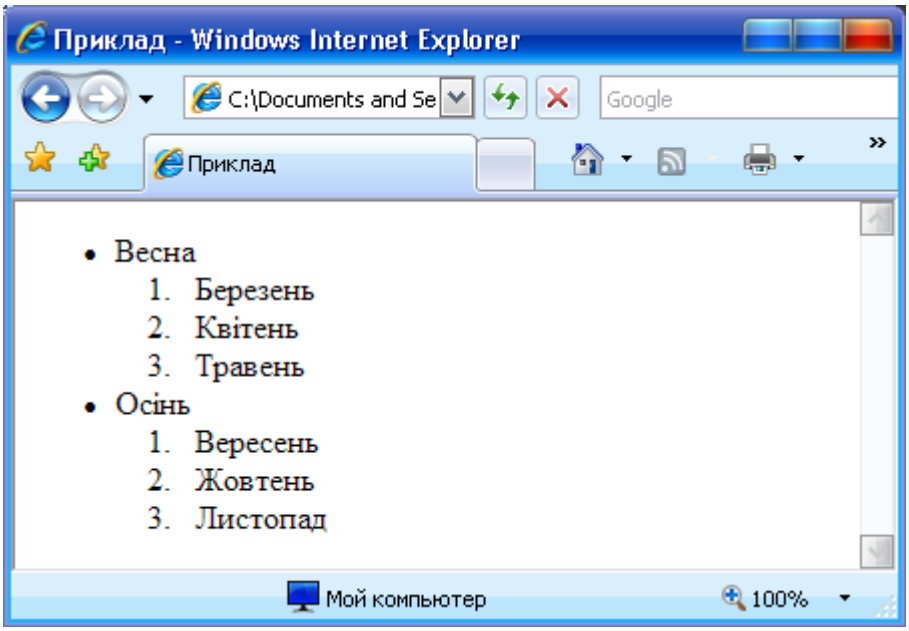

Рис 1.5.6 Вкладені списки

1.6 Таблиці в HTML

1.6.1Створення простих HTML-таблиць

Розглянемо спочатку мінімальний набір тегів і їх параметрів, необхідний і достатній для створення нескладних таблиць, а потім перейдемо до їх детального опису.

Опис таблиць повинен розташовуватися всередині розділу документа <body>. Документ може містити довільне число таблиць, причому допускається вкладеність таблиць одна в одну. Кожна таблиця повинна починатися тегом <table> і завершуватися тегом </table>. Всередині цієї пари тегів розташовується опис вмісту таблиці. Будь-яка таблиця складається з одного або декількох рядків, в кожен з яких задаються дані для окремих клітинок.

Кожен рядок починається тегом <tr> (Table Row) і завершується тегом </tr>. Окрема клітинка в рядку обрамляється парою тегів <td> і </td> (Table Data) або <th> і </th> (Table Header). Тег <th> використовується зазвичай для елементів-заголовків таблиці, а <td> для клітинок-даних. Відмінність у використанні полягає лише в типі шрифту, використовуваного по замовчуванні для відображення вмісту клітинок, а також розташуванню даних усередині клітинки. Вміст клітинок типу <th> відображається напівжирним (Bold) шрифтом і розташовується по центру (align = center, valign = middle). Клітинки, визначені тегом <td> по замовчуванню відображають дані, вирівняні вліво (align = left) і посередині (valign = middle) по вертикалі. Теги <td> і <th> не можуть з'являтися поза описом рядка таблиці <tr>.

Кількість рядків в таблиці визначається числом відкриваючих тегів <tr>, а кількість стовпців - максимальною кількістю <td> чи <th> серед всіх рядків. Частина клітинок може не містити ніяких даних, такі клітинки описуються парою тегів, що йдуть підряд <td>, </td>. Якщо одна або декілька клітинок, розташованих в кінці якого-небудь рядка, не містять даних, то їх опис може бути опущений, а браузер автоматично додасть необхідну кількість порожніх клітинок.

Таблиця може мати заголовок, який складається з пари тегів <caption> і </caption>. Опис заголовка таблиці повинен

розташовуватися усередині тегів <table> і </table> відразу ж після тега <table> і до першого <tr>.

По замовчуванню текст заголовка таблиці розташовується над нею (align = top) і центрується в горизонтальному напрямі.

Приведемо приклад простої таблиці, що складається з двох рядків і двох стовпців, зображення якої показане на рис.4.3.

```
<table border>
  <tr>
    <td> Клітинка 1 рядок 1</td>
    <td> Клітинка 2 рядок 1</td>
  </tr>
  <tr>
    <td> Клітинка 1 рядок 2</td>
    <td> Клітинка 2 рядок 2</td>
  </tr>
</table>
```

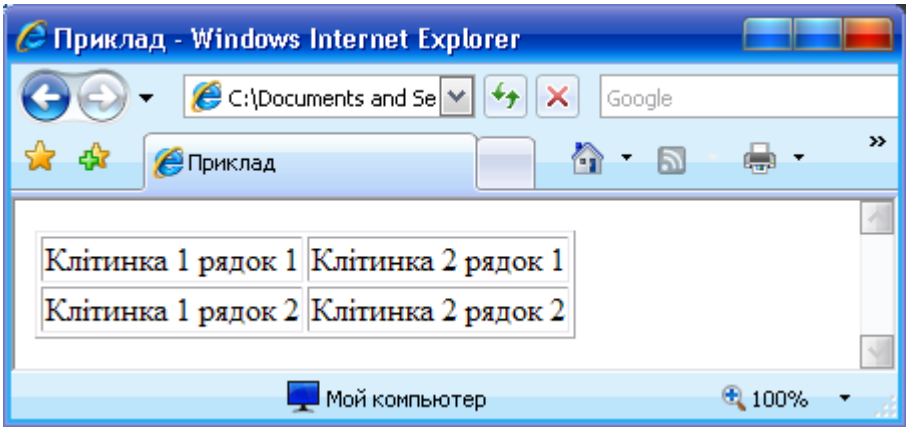

Рис 1.6.1 Приклад простої таблиці

1.6.2Теги оформлення таблиць

Розглянемо призначення різних параметрів, які можуть використовуватися в тегах, що описують таблиці.

Заголовок таблиці <CAPTION>

Тег заголовка таблиці <caption> має єдиний параметр align, що приймає значення top (заголовок над таблицею) або bottom (заголовок під таблицею). Якщо параметр align , то значення по замовчуванню align = top. Якщо використовувати вирівнювання заголовка по горизонталі то використовується параметр align, що приймає значення left, center, right по лівому по центру і по правому краю відповідно. Тоді для

вирівнювання заголовка по вертикалі використовується параметр valign.

Як заголовок таблиці в більшості випадків використовується простий текст, що регламентується специфікацією HTML, проте насправді між тегами <caption> і </caption> допустимо записувати будь-які HTML-елементи, що використовуються в розділі <body>. Приведемо приклад запису заголовка таблиці:

```
<table border>
  <caption valign="bottom" align="right">Проста
таблиця</caption>
  <tr>
    <td> Клітинка 1 рядок 1</td>
    <td> Клітинка 2 рядок 1</td>
  </tr>
  <tr>
    <td> Клітинка 1 рядок 2</td>
    <td> Клітинка 2 рядок 2</td>
  </tr>
</table>
```

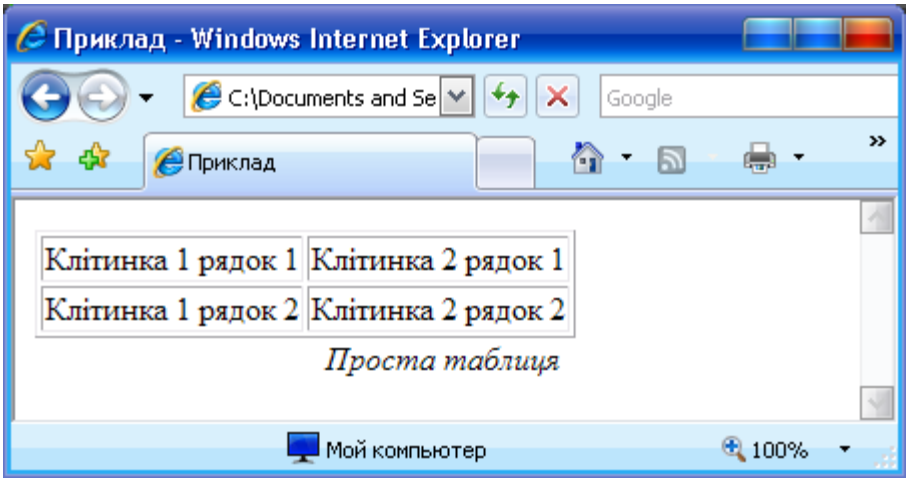

Рис 1.6.2 Приклад використання заголовку

Параметри тега <TABLE>

Основним тегом, вживаним при створенні таблиць, є тег <table>.

Він може використовуватися з кількома параметрами, кожний з яких можна опускати. Набір допустимих параметрів залежить від браузера. Згідно специфікації HTML в тегу <table> можуть використовуватися наступні параметри: border, cellspacing, cellpadding, width, align та інші.

Параметр BORDER

Параметр `border` керує виглядом рамки навколо кожної клітинки, і навколо всієї таблиці. По замовчуванню рамки не малюються, і на екрані користувач побачить лише рівно розташований текст елементів таблиці. Існує немало ситуацій, коли використання таблиць без рамок цілком виправдано, наприклад, для багатоколоночних списків, реалізованих за допомогою таблиць, або задання точного взаємного розташування малюнків і тексту. Проте в більшості випадків для традиційного використання таблиць її клітинки корисно відокремити один від одного лініями сітки, що полегшує сприйняття і розуміння інформації, що міститься в таблиці.

Для додавання в таблицю рамок необхідно включити в код `<table>` параметр `border`, який може мати числове значення.

```
<table border="5">
  <tr>
    <td> Клітинка 1 рядок 1</td>
    <td> Клітинка 2 рядок 1</td>
  </tr>
  <tr>
    <td> Клітинка 1 рядок 2</td>
    <td> Клітинка 2 рядок 2</td>
  </tr>
</table>
```

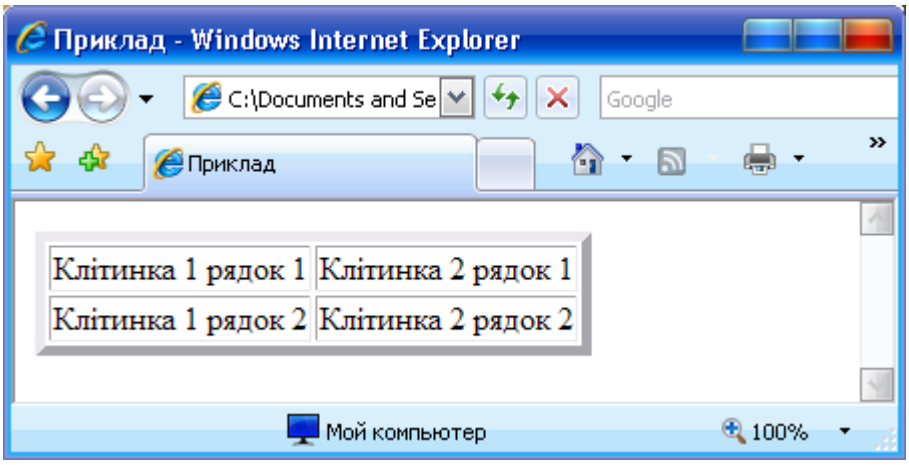

Рис 1.6.3 Таблиця з товщиною рамки 5

Параметр CELLSPACING

Форма запису параметра: `cellspacing = num`, де `num` - числове значення параметра в пікселях, яке не може бути опущене. Величина `num` визначає відстань між суміжними клітинками

(точніше між рамками клітинок) як по горизонталі, так і по вертикалі. По замовчуванню значення приймається рівним 2. Відмітимо, що традиційно у видавничих системах суміжні елементи таблиці мають спільну межу. У HTML-таблицях по замовчуванню між ними залишається місце, що добре видно на приведеному вище малюнку (Рис 1.6.3) .При завданні `cellspacing = 0` рамки суміжних осередків зіллються і створять враження єдиної сітки таблиці.

```
<table border="5" cellspacing="10">
  <tr>
    <td> Клітинка 1 рядок 1</td>
    <td> Клітинка 2 рядок 1</td>
  </tr>
  <tr>
    <td> Клітинка 1 рядок 2</td>
    <td> Клітинка 2 рядок 2</td>
  </tr>
</table>
```

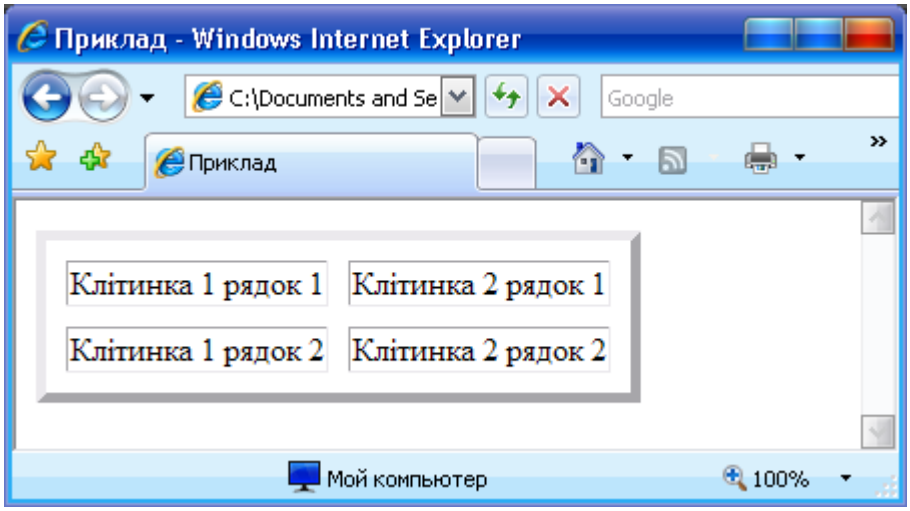

Рис 1.6.4 Таблиця з великим значенням cellspacing

Параметр CELLPADDING

Форма запису параметра аналогічна `cellspacing`. Величина `num` визначає розмір вільного простору (відступу) між рамкою клітинок і даними усередині клітинки. По замовчуванню значення приймається рівним одиниці. Установка параметра `cellpading` рівним нулю може привести до того, що деякі частини тексту клітинки, будуть прилипати до рамки, що виглядає не дуже естетично.

```
<table border="5" cellpadding="10">
```

```
  <tr>
    <td> Клітинка 1 рядок 1</td>
    <td> Клітинка 2 рядок 1</td>
  </tr>
  <tr>
    <td> Клітинка 1 рядок 2</td>
    <td> Клітинка 2 рядок 2</td>
  </tr>
</table>
```

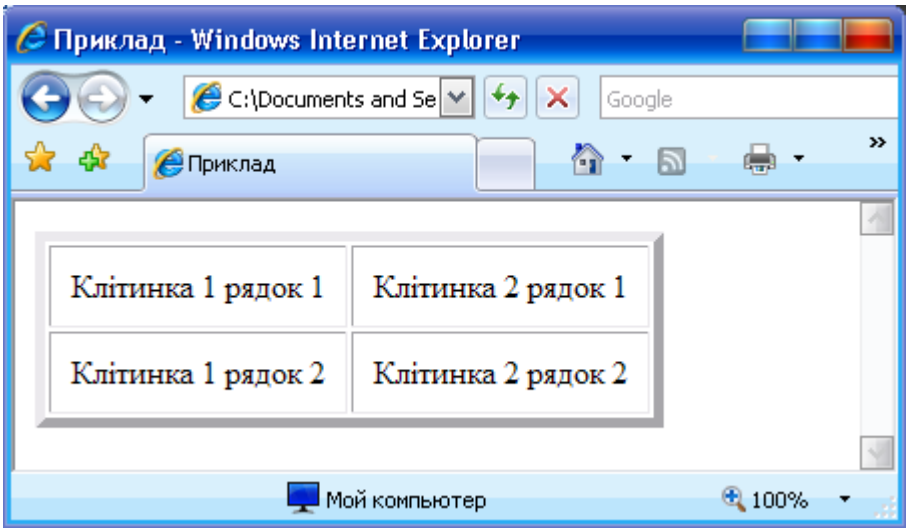

Рис 1.6.5 Приклад таблиці з великим cellpadding

Дія параметрів `cellpading` і `cellspacing` дуже схоже один на одного. Для таблиці без рамок зміна того або іншого параметра приводить до одного і того ж результату. Обидва параметри впливають на відповідні відступи одночасно по горизонталі і по вертикалі. На жаль, роздільного управління горизонтальними і вертикальними відступами так, як це зроблено, наприклад, для відступів від зображень (параметри `hspace` і `vspace` тега `<img>`), не передбачено.

Всі три параметри - `border`, `cellpading` і `cellspacing` діють незалежно один від одного, якщо який-небудь з них опущений, то береться його значення по замовчуванню. Зокрема, якщо опущені всі перераховані параметри, то мінімальна відстань між даними в суміжних клітинках буде дорівнювати 6 пікселям. Це значення складається з двох пікселів для `cellspacing`, одного пікселя для `cellpading` і одного пікселя для рамки кожної з комірок. Найбільш компактна таблиця буде отримана завданням наступного опису:

```
<table border = 0 cellpadding = 0 cellspacing =
0>
```

Тільки у такому варіанті клітинки будуть розташовані впритул один до одного.

Параметри WIDTH і HEIGHT

При відображенні таблиць їх ширина і висота автоматично обчислюються браузером і залежать від багатьох чинників: значень параметрів, заданих в описі всього документа, даної таблиці, окремих її рядків і клітинок вмісту клітинок, а також параметрів, що задаються при перегляді документа в тому або іншому браузері, наприклад, типу і розмірів шрифту, розмірів вікна перегляду і ін. При відображенні розрахунок розмірів таблиць виконується автоматично з урахуванням цих чинників, при цьому робиться спроба представити таблицю в найбільш зручному вигляді - розташувати таблицю так, щоб вона поміщалася у вікні перегляду. Загальна схема перегляду великих документів, як правило, зводиться до лінійної прокрутки вмісту документа по вертикалі і читання тексту, що перетинається різними таблицями, зображеннями і т.д. Це відноситься як до HTML-документів, так і до звичайних документів, що створюються в будь-яких текстових редакторах. Більшість текстових редакторів, так і HTML-браузерів автоматично форматують текст так, щоб довжина рядків не перевершувала ширину вікна перегляду. Це дозволяє уникнути необхідності горизонтальної прокрутки документа. Аналогічні дії робляться браузерами з таблицями - по можливості форматувати їх так, щоб ширина таблиці не перевершувала ширина вікна перегляду. Можна зробити висновок, що ширина таблиць є важливішим, першорядним параметром, розрахунок якого виконується в першу чергу в порівнянні з висотою.

В більшості випадків динамічне визначення розмірів таблиці дає в результаті ефективне використанням реальної площі вікна перегляду. Проте буває необхідно примусово указувати ширину або висоту таблиці. Для цієї мети використовуються параметри `width` (ширина таблиці) і `height` (висота таблиці) тега `<table>`. Форма запису: `width = num` або `width = num%`, де `num` - числове значення ширини всієї таблиці в пікселях або у відсотках від всього розміру вікна. Відмітимо, що допустимо задавати значення, більші 100%

Аналогічні параметри можуть задаватися і для окремих клітинок. Відмітимо, що задання конкретного значення параметра, наприклад

width=200, не означає, що таблиця у будь-якому випадку матиме вказану ширину, а лише визначає рекомендовану ширину, яка буде отримана при можливості.

Конкретні алгоритми настройки таблиць для різних браузерів можуть дещо відрізнятися.

Параметр ALIGN

Даний параметр тега <table> визначає горизонтальне розташування таблиці в області перегляду. Допустимі значення - left (вирівнювання вліво) center (вирівнювання по центру) і right (вирівнювання вправо). По замовчуванню таблиці вирівняні по лівому краю. Параметр align в тегу <table> не тільки визначає місцерозташування таблиці, але і дозволяє виконати обтікання таблиці текстом з протилежного боку аналогічно обтіканню картинок. Обтікання таблиці текстом з двох боків не передбачається ні в яких випадках. Для точнішого управління обтіканням слід використовувати тег
 з параметром clear так само, як це виконується для <img>. Якщо параметр align опущений, то місце справа і/або зліва від таблиці завжди буде порожнім незалежно від її ширини.

```
Текст, що обтікає таблицю. Для демонстрації
обтікання таблиці, що має вирівнювання по
правому краю
<table border="1" align="right">
  <tr>
    <td> Клітинка 1 рядок 1</td>
    <td> Клітинка 2 </td>
  </tr>
  <tr>
    <td> Клітинка 1 </td>
    <td> Клітинка 2 рядок 2</td>
  </tr>
</table>
```

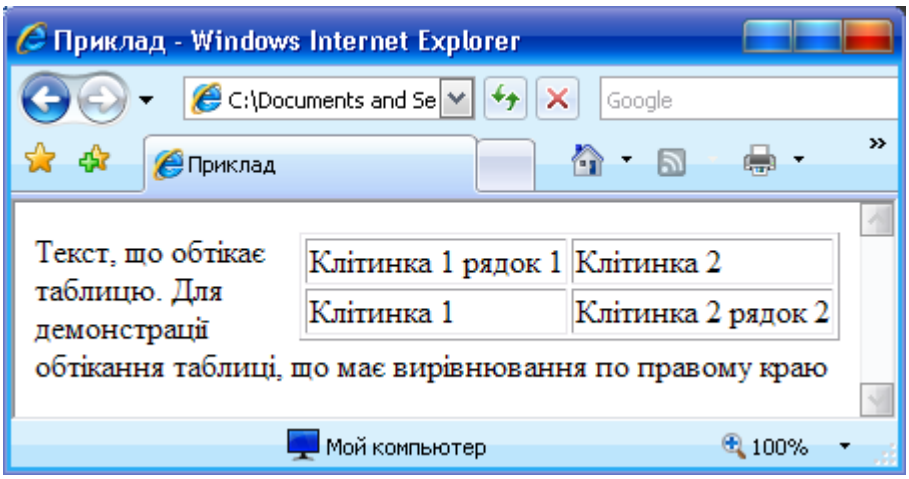

Рис 1.6.6 Приклад обтікання таблиці

Форматування даних усередині таблиці

Кожну окрему клітинку усередині таблиці можна розглядати як область для незалежного форматування. Всі правила, які діють для управління відображенням тексту, можуть бути використані для форматування тексту усередині клітинки. Усередині клітинки допустиме використання практично всіх елементів HTML, які можуть з'являтися усередині тіла документа `<body>`, зокрема теги, управління розташуванням тексту - `<p>`, `<br>`, `<hr>`, коди заголовків - від `<h1>` до `<h6>`, теги форматування символів - `<b>`, `<i>`, `<strong>`, `<big>`, `<em>`, `<font size>`, `<font color>`, теги вставки графічних зображень `<img>`, гіпертекстових посилань `<a>` і т.д.

Для форматування даних усередині елементів таблиці передбачені наступні параметри.

Параметри вирівнювання вмісту комірок - `align` і `valign`. Можуть застосовуватися в кодах `<tr>`, `<td>` і `<th>`. Параметр горизонтального вирівнювання `align` може приймати значення `left`, `right` і `center` (по замовчуванню `left` для `<td>` і `center` для `<th>`). Параметр вертикального вирівнювання `valign` може приймати значення `top` (по верхньому краю), `bottom` (по нижньому краю), `middle` (посередині), `baseline` (по базовій лінії). По замовчуванню - `middle`. Вирівнювання по базовій лінії забезпечує прив'язку тексту окремого рядка у всіх клітинках до єдиної лінії. Задання параметрів вирівнювання на рівні коду `<tr>` задає вирівнювання для всіх клітинок даного рядка, при цьому в кожного окремого рядка можуть бути визначені свої параметри, що перевизначають дію параметрів, заданих в `<tr>`.

Приведемо приклад таблиці, в якій дані в клітинках першого стовпця вирівняні управо, другого стовпця - по центру, а третього - вліво (значення по замовчуванню):

```
<table border="1" width="90%" align="center">
  <tr>
    <td align="right"> Клітинка 1</td>
    <td align="center"> Клітинка 2</td>
    <td> Клітинка 3 </td>
  </tr>
  <tr>
    <td align="right"> Клітинка 4</td>
    <td align="center"> Клітинка 5</td>
    <td> Клітинка 6 </td>
  </tr>
</table>
```

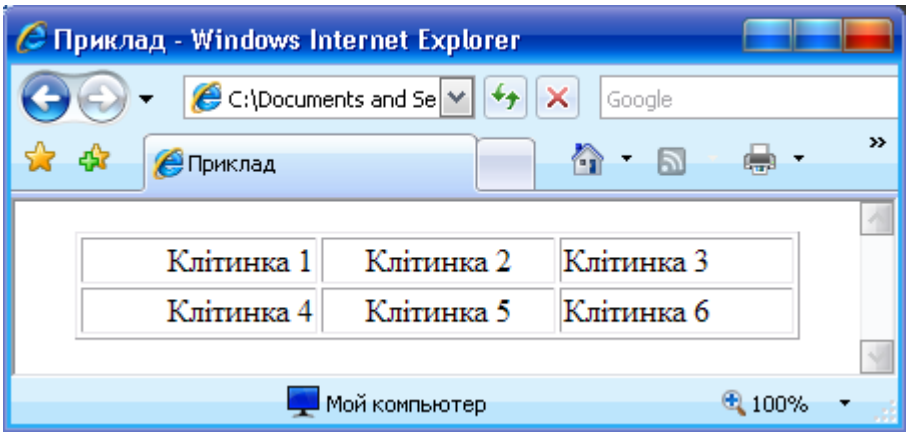

Рис 1.6.7 Вирівнювання контенту в клітинках

Параметр `nowrap` відключає можливість автоматичного розбиття тексту клітинки на рядки. Може застосовуватися в кодах `<tr>`, `<td>` і `<th>`. Слід уникати невиправданого застосування цього параметра, оскільки це може значно скоротити можливості динамічної зміни розмірів таблиць і погіршити їх сприйняття. В більшості випадків досить застосувати `nowrap` для окремих клітинок, що дійсно вимагають заборони перенесення слів на новий рядок. Перенесення слів здійснюється тільки по роздільниках між словами (пробілам), і у ряді випадків для заборони розриву тексту в окремих місцях слід замість символу пробілу задавати код нерозривного пробілу ` ` (NonBreaking Space). Як приклади можна привести випадки, де розрив не рекомендується - між числовим значенням і одиницями вимірювання даної величини; між

прізвищем і ініціалами. Так, текст 650 км. або Іваненко Б.Н. рекомендується записувати У вигляді `650 км` `Іваненко Б.Н.`

Параметри `width` і `height` можуть застосовуватися в кодах `<td>` і `<th>`. Їх синтаксис аналогічний синтаксису цих параметрів для тега `<table>`. Їх значення визначає ширину або висоту комірки, для якого записані дані параметри. Значення можуть задаватися в пікселях або у відсотках від розмірів всієї таблиці. Якщо в колонці значення ширини вказане лише в одній клітинці, то дане значення стає шириною всієї колонки. Якщо таких вказівок декілька, то вибирається максимальне значення. Ті ж властивості характерні і для рядків.

Атрибути обєднання клітинок

Для складних таблиць характерна потреба в об'єднанні декількох суміжних клітинок по горизонталі або по вертикалі в одну. Дана можливість реалізується за допомогою параметрів `colspan` (COLumn SPANning) і `rowspan` (ROW SPANning), що задаються в кодах `<td>` або `<th>`. Форма запису: `colspan = num`, де `num` - числове значення, що визначає, на скільки стовпців слід розширити біжучу клітинку по горизонталі. Застосування параметра `rowspan` аналогічно, тільки тут вказується кількість рядків, які повинна захопити біжуча клітинка по вертикалі. По замовчуванню для цих параметрів встановлюється значення, рівне одиниці. Допустиме одночасне завдання значень обох параметрів для однієї клітинки. Правильне встановлення значень цих параметрів може виявитися не дуже простим завданням

На Рис 1.6.8 показаний приклад відображення таблиці, отриманий по наступному HTML-коду:

```
<table border="1" width="90%" align="center">
  <tr>
  <td rowspan="2"> Клітинка, що охоплює два
рядки </td>
  <td colspan="2"> Клітинка, що охоплює два
стовпці </td>
  </tr>
  <tr>
    <td> Клітинка 3</td>
```

```
    <td> Клітинка 4</td>
  </tr>
  <tr>
    <td> Клітинка 5</td>
    <td> Клітинка 6</td>
    <td> Клітинка 7</td>
  </tr>
</table>
```

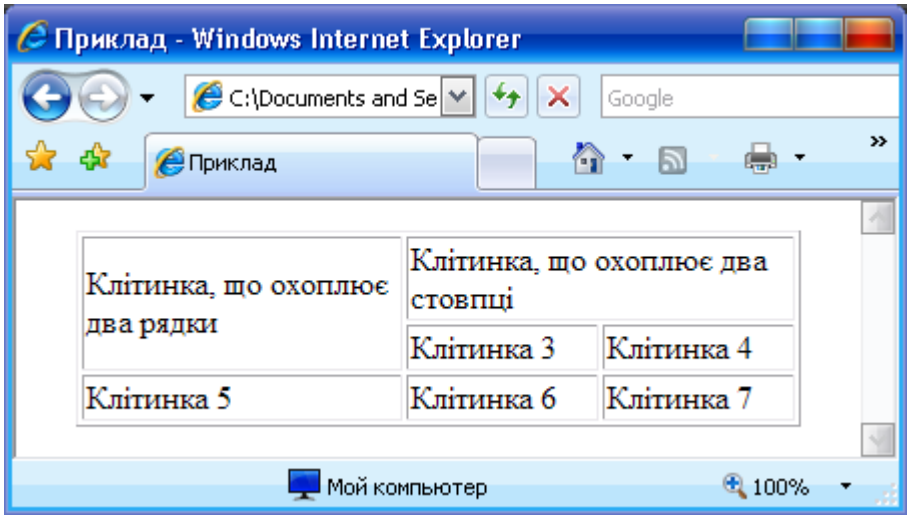

Рис 1.6.8 Приклад об'єднання клітинок

Неуважне завдання значень параметрів об'єднання клітинок може привести до їх взаємного перекриття і конфліктів, при яких результат непередбачуваний. Характерне застосування об'єднання клітинок - спільний заголовок для декількох суміжних колонок або рядків.

Приведемо приклад коду HTML (відображення якого показане на Рис 1.6.9) у якому розтяжні клітинки сформовані некоректно.

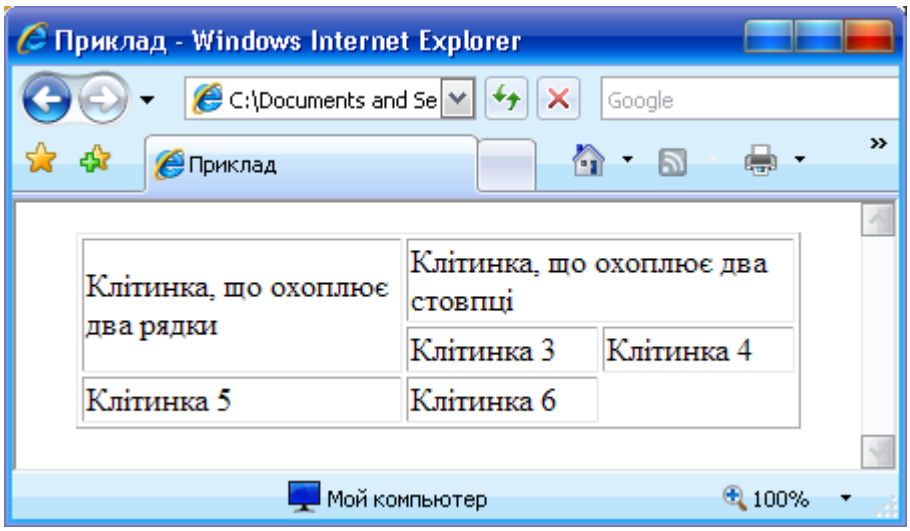

Рис 1.6.9 Приклад некоректного об'єднання клітинок

```
<table border="1" width="90%" align="center">
```

```
<tr>
<td rowspan="2"> Клітинка, що охоплює два рядки
</td>
<td colspan="2"> Клітинка, що охоплює два
стовпці </td>
</tr>
<tr>
<td> Клітинка 3</td>
<td> Клітинка 4</td>
</tr>
<tr>
<td> Клітинка 5</td>
<td> Клітинка 6</td>
</tr>
</table>
```

Фон таблиці

Параметр `bgcolor` задає колір підкладки всієї таблиці, окремих рядків або осередків. Може зустрічатися в тегах `<table>`, `<tr>`, `<td>` і `<th>`. Форма запису така ж, як і для тега `<body>`, а саме: `bgcolor = значення`, де як значення задається колір в форматі RGB - або його назва.

```
<table border>
  <tr>
    <td bgcolor="red"> Клітинка 1 </td>
    <td bgcolor="#00FF00"> Клітинка 2 </td>
  </tr>
  <tr>
    <td bgcolor="#0000FF"> Клітинка 1 </td>
    <td bgcolor="yellow"> Клітинка 2 </td>
  </tr>
</table>
```

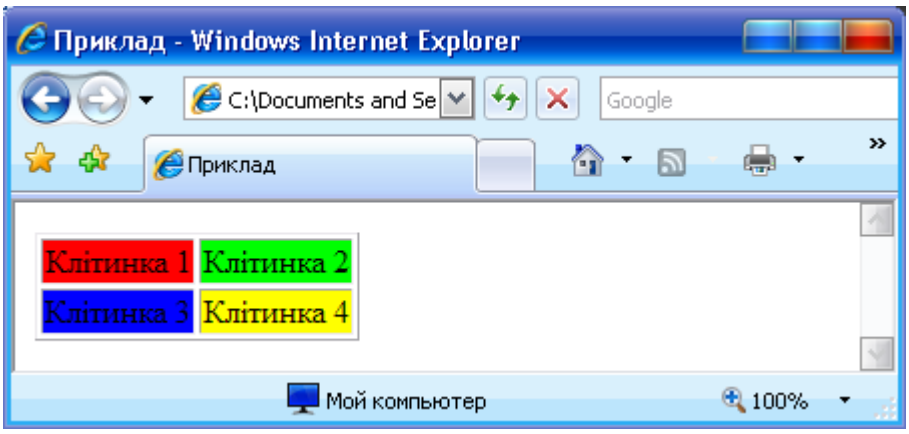

Рис 1.6.10 Приклад зміни кольору фону клітинок

Визначення фонового малюнка для таблиці

Браузери дозволяють використовувати параметр `background`, що визначає фоновий малюнок для таблиці так, як це може бути зроблено для всього HTML-документа.

Цей параметр може задаватися в тегах `<table>`,`<td>` і `<th>`.

1.6.3Особливості побудови таблиць

Вкладені таблиці

Окремі елементи таблиці можуть містити практично будь-які теги мови і дані, дозволені в розділі `<body>` документа. Зокрема, всередині елемента таблиці може бути цілком розміщена інша таблиця. Такі таблиці називаються вкладеними. Правила їх побудови не відрізняються від побудови таблиць і не потребують окремого опису.

Відображення порожніх клітинок в таблицях

Однією з особливостей представлення таблиць різними браузерами є відображення порожніх клітинок. Згідно опису мови всі браузери повинні доповнювати рядки порожніми клітинками, якщо в якому-небудь рядку їх кількість задана меншою, ніж в решті рядків. Крім того, в будь-якому місці таблиці можуть знаходитися клітинки, що не містять даних. Існує відмінність між порожніми клітинками і клітинками, що містять невидимі дані. У порожніх клітинках усередині пари тегів `<td>` і `</td>` не міститься ніякої інформації або один або більше пробілів, які не трактуються як дані. Клітинки, що містять невидимі дані, наприклад, можуть містити код ` ` або код перенесення рядка `<br>`, або будь-який текст, колір якого

співпадає з кольором фону клітинок Навколо порожніх клітинок не промальовувалася рамка. Приклад таблиці з другою порожньою клітинкою і третьою клітинкою що містить нерозривний пробіл приведений на Рис 1.6.11.

```
<table border="1">
  <tr>
    <td> Клітинка 1</td>
    <td>  </td>
  </tr>
  <tr>
    <td></td>
    <td> Клітинка 4</td>
  </tr>
</table>
```

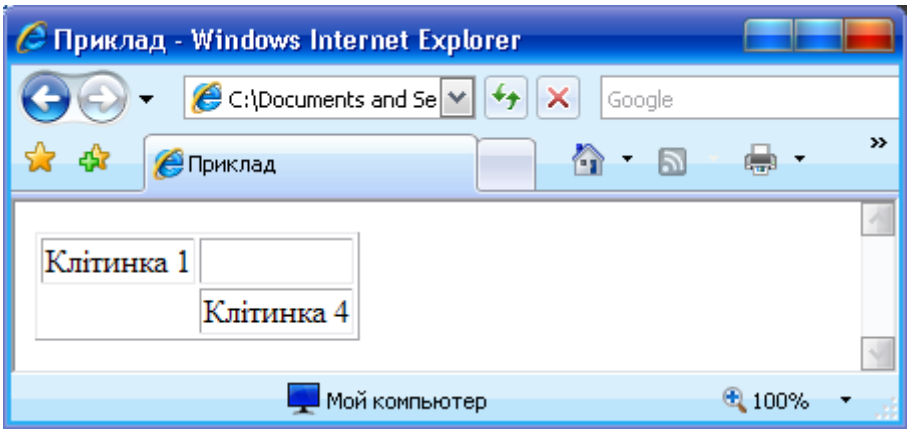

Рис 1.6.11 Приклад різного відображення порожніх клітинок

Вирівнювання даних в стовпцях таблиці

Характерною проблемою при створенні таблиць є визначення параметрів вирівнювання для окремих рядків або стовпців. Для вирівнювання вмісту всіх клітинок поточного рядка досить задати необхідні параметри в коді `<tr>`. Проте частіше необхідно забезпечити однакове вирівнювання для всіх елементів одного стовпця, оскільки в більшості випадків в стовпці розташовуються однорідні дані.

Передбачені спеціальні теги - `<col>` і `<colgroup>`. Ці теги повинні розташовуватися відразу ж за описом `<table>` перед першою появою тега `<tr>`.

Параметрами тегів `<col>` і `<colgroup>` можуть бути `span`, що визначає кількість суміжних колонок, на які розповсюджується дія

значень параметрів, і `align`, що визначає горизонтальне вирівнювання даних у всіх клітинках відповідного стовпця (або стовпців). Допустимими значеннями параметра `align` є `left`, `right` і `center`. Для параметра `span` значення по замовчуванню рівне одиниці.

Тег `<colgroup>` додатково дозволяє задавати параметр `valign`, що визначає вертикальне вирівнювання даних в клітинках. Допустимі значення параметра `valign` є `middle`, `top` і `bottom`.

Відмінність між тегами `<clogroup>` і `<col>` полягає в тому, що перший з них, крім визначення параметрів вирівнювання даних для стовпців, виконує також умовне об'єднання декількох стовпців в групу. Ефект такого об'єднання виявляється при використанні параметра `rules`, який описується нижче. По замовчуванню всі стовпці таблиці вважаються однією групою. Тег `<col>` повинен використовуватися тільки для визначення вирівнювання даних в окремих стовпців в групі.

Приведемо приклад: нехай необхідно побудувати таблицю, що містить 6 стовпців, причому дані перших трьох з них повинні бути вирівняні вправо, а наступних трьох - посередині. Для вирішення цього завдання слід записати такий фрагмент HTML-коду:

```
<table>
<colgroup span ="3" align="right">
<colgroup span ="3" align="center">
(дані для таблиці)
</table>
```

Інший приклад. Нехай в попередній таблиці перші дві колонки повинні бути вирівняні управо, а третя - посередині, причому всі три колонки необхідно об'єднати в групу. Наступні три колонки також повинні бути об'єднані в групу і мати вирівнювання, аналогічне першій групі.

Для вирішення цього завдання слід записати такий фрагмент HTML-коду:

```
<table>
<colgroup>
<col span ="2" align="right">
<col align="center">
<colgroup>
<col span ="2" align="right">
```

```
<col align="center">
(дані для таблиці)
</table>
```

В даному прикладі тег <colgroup> визначає налаштування окремих стовпців даної групи. При цьому в тегу <colgroup> при необхідності могли б бути задані параметри вирівнювання, значення яких розповсюджуються на всі стовпці даної групи. Значення параметрів, задані в тегу <col>, перевизначають значення з тега <colgroup>. Зауважимо, що в тегу <colgroup> в даному прикладі, на відміну від попереднього, відсутній параметр span. Тут його вживання безглузде, оскільки кількість елементів в групі визначатиметься наступними за тегом <colgroup> тегами <col>. Тому будь-яке задане значення параметра span тега <colgroup> буде перевизначено.

На мал. 4.17 показаний результат реалізації приведеного вище коду, а також варіант відображення такої таблиці із записом RULES=GROUPS в тегу <TABLE>, з якого видно сенс об'єднання в групи.

Визначення кольору рамок таблиці

Ще декілька параметрів, характерних тільки для Microsoft Internet Explorer, дозволяють вибирати колір рамок таблиць - BORDERCOLOR, BORDERCOLORLIGHT і BORDERCOLORDARK. Ці параметри можуть задаватися в тегах <TABLE>, <TD>, <TH> і <TR>. Як значення цих параметрів може використовуватися назва кольору або його шістнадцятизначне значення. Параметр BORDERCOLOR визначає колір всіх елементів рамок таблиці, а інші два параметри задають колір окремих рамок, що становлять, перевизначене значення BORDERCOLOR. Параметр BORDERCOLORLIGHT забарвлює в заданий колір лівий і верхній краї всієї таблиці і відповідно правий і нижній краї кожної клітинки. Другий параметр BORDERCOLORDARK задає кольори протилежних країв. За рахунок поєднання дії цих параметрів таблиця виглядатиме дещо піднятою над поверхнею сторінки або заглибленою. Все залежить від вибраного поєднання кольорів.

Теги структуризації таблиці <THEAD>, <TBODY> і <TFOOT>

Браузер Microsoft Internet Explorer дозволяє використовувати ряд нових тегів для структуризації таблиць і гнучкого управління промальовуванням рамок і ліній сітки.

Теги <THEAD>, <TBODY> і <TFOOT> більш строго задають структуру опису таблиці, виділяючи клітинки заголовка таблиці, основний вміст таблиці і підсумковий рядок. Ці теги можуть зустрічатися тільки в описі таблиць усередині пари тегів <TABLE> і </TABLE>.

Теги <THEAD> і <TFOOT> використовуються для опису верхнього і нижнього колонтитулів таблиці. Ці теги можуть зустрічатися в таблиці не більше одного разу. Завершальний тег для них можна опускати. Використання даних тегів зручне при створенні великих таблиць, що виходять за межі однієї сторінки.

Тег <TBODY> може зустрічатися багато разів в описі таблиці, при цьому потрібне використання завершального тега </TBODY>. Цей тег виконує логічне групування даних так само, як і тег <COLGROUP>, що виконує групування суміжних стовпців.

При використанні нових тегів з'являється можливість гнучкіше управляти рамками і лініями сітки таблиці.

Управління промальовуванням рамок навколо таблиці здійснюється параметром FRAME тега <TABLE>, а ліній сітки таблиці - параметром RULES. Наприклад, стає можливим провести тільки вертикальні лінії між колонками і замість рамки навколо всієї таблиці дати горизонтальні лінії зверху і знизу таблиці.

Параметр FRAME може приймати наступні значення:

BOX або BORDER - рамка малюється зі всіх чотирьох боків

ABOVE - тільки з верхнього боку

BELOW - тільки з нижнього боку

HSIDES - малюється нижня і верхня сторона

VSIDES - малюється ліва і права сторона

LHS - тільки з лівого боку

RHS - тільки з правого боку

VOID - таблиця без зовнішніх рамок

Параметр RULES управляє промальовуванням внутрішніх ліній сітки таблиці і може приймати наступні значення:

ALL - малюються всі внутрішні лінії

GROUPS - малюються тільки лінії, що розділяють групи

ROWS - малюються лінії, що розділяють рядки

COLS - малюються лінії, що розділяють стовпці

NONE - внутрішні лінії не малюються

Приклад: <TABLE BORDER FRAME=HSIDES RULES=GROUPS>.

Приведемо приклад повного HTML-коду, що створює таблицю з використанням описаних можливостей:

```
<HTML>
<HEAD>
<TITLE> Виділення заголовка і підсумкового
рядка </TITLE>
</HEAD>
<BODY>
<TABLE BORDER=5 WIDTH=100% RULES=GROUPS
FRAME=HSIDES>
<COLGROUP ALIGN=CENTER>
<COLGROUP ALIGN=CENTER>
<COLGROUP ALIGN=CENTER>
<CAPTION><H3>
Приклад гнучкого управління лініями <BR> сітки
таблиці </H3>
</CAPTION>
<THEAD>
<TR>
<TH> Заголовок стовпця 1 </TH>
<TH> Заголовок стовпця 2 </TH>
<TH> Заголовок стовпця 3 </TH>
</TR>
</THEAD>
<TBODY>
<TR> <TD> Дані </TD><TD> Дані </TD><TD> Дані
</TD></TR>
<TR> <TD> Дані </TD><TD> Дані </TD><TD> Дані
</TD></TR>
<TR> <TD> Дані </TD><TD> Дані </TD><TD> Дані
</TD></TR>
<TR> <TD> Дані </TD><TD> Дані </TD><TD> Дані
</TD></TR>
```

```
<TR> <TD> Дані </TD><TD> Дані </TD><TD> Дані
</TD></TR>
<TR> <TD> Дані </TD><TD> Дані </TD><TD> Дані
</TD></TR>
</TBODY>
<TFOOT>
<TR> <TD> Підсумок </TD><TD> Підсумок </TD><TD>
Підсумок </TD></TR>
</TFOOT>
</TABLE>
</BODY>
</HTML>
```

В даному прикладі, відображення якого браузером представлене на рис.4.18 показується один з можливих варіантів управління лініями сітки і рамками навколо таблиці. Навколо таблиці малюється рамка завтовшки 5 пікселів (BORDER=5) тільки з верхнього і нижнього боку (FRAME=HSIDES).

Усередині таблиці малюються лінії сітки, що розділяють групи даних (RULES=GROUPS). Групи даних визначені, по-перше, наявністю трьох тегів <COLGROUP ALIGN=CENTER>, кожен з яких оголошує окремий стовпець таблиці групою. По-друге, теги <THEAD>, <TBODY> і <TFOOT> також розбивають дані таблиці на групи, що визначає промальовування внутрішніх горизонтальних ліній.

Завдання числа стовпців таблиці

Браузер Microsoft Internet Explorer (а також браузер Netscape версії 4.x) дозволяє задавати в тегу <TABLE> параметр COLS, значення якого визначає число стовпців в таблиці. Запис цього параметра дозволяє прискорити процедуру верстки таблиці при відображенні в браузері, оскільки з'являється можливість визначити кількість стовпців до закінчення завантаження коду опису таблиці. На даний момент включення цього параметра ніяк не впливає на хід завантаження документа.

Вертикальне вирівнювання таблиць

Останній параметр тега <TABLE>, властивий тільки Microsoft Internet Explorer, це - VALIGN, що визначає вертикальне

вирівнювання таблиці щодо тексту. Його дія подібна до такого ж параметру для зображень.

1.6.4Альтернатива табличного представлення

Підтримка таблиць стала широко поширеною властивістю Web-браузерів, так що практично не залишилося причин, із-за яких слід було б уникати їх використання. Проте, розглянемо можливі варіанти альтернативного представлення даних, які можна використовувати замість таблиць або на додаток до них.

Деякі інші способи, що не використовують поняття таблиць:

Використання преформатованого тексту. Цей спосіб традиційно використовувався в ранніх версіях мови HTML, коли підтримки таблиць ще не існувало. Його вживання і до теперішнього часу не втратило актуальності, оскільки такі тексти правильно відображатимуться будь-якими браузерами, у тому числі і чисто текстовими.

Використання зображення, що містить таблицю. Таблиця може бути створена будь-яким текстовим редактором або навіть відображена Web-браузером і потім збережена як картинка в одному з графічних форматів.

Це не кращий варіант, оскільки при цьому втрачається вся гнучкість динамічного налаштування відображення таблиць. Крім того, виникає необхідність зберігання додаткового файлу із зображенням, розмір якого до того ж, як правило, буде значно більший, чим розмір тексту, що описує HTML-таблицю. Можлива область застосування - таблиці строго певних розмірів, для яких недопустима залежність її відображення від яких-небудь зовнішніх чинників (шрифтів, режимів роботи браузера і т.п.).

Використання списків замість таблиць. Для простих випадків замість організації таблиць цілком можливо обійтися одним з видів списків наявних в мові HTML.

1.7 Контрольні запитання

- Для чого призначені html-дескриптори?
- Опишіть структуру html-документа.
- Як створити абзац тексту і вирівняти його?
- Як виділити текст заголовком певного рівня? Для чого застосовуються заголовки?
- Як виділити текст курсивом, напівжирним? Підкреслити?
- Як встановити колір, тип шрифту і розмір тексту?
- Що таке гіперпосилання? Як вставити в документ текстове посилання?
- Як вставити малюнок?
- Які теги використовуються для створення маркованих списків?
- Які теги використовуються для створення нумерованих списків?
- Які теги використовуються для створення таблиць?
- Поясніть призначення властивостей rowspan і colspan тега <td>?

1.8 Перелік навчально-методичної літератури.

1. Основи веб-дизайну / Пасічнік О.Г., Пасічнік О.В., Стеценко І.В. – Видавнича група BHV 2009, . – 336 с.
2. Основи будування сайтів / В. Манако, Д. Манако, О.Данилова, О.Войченко – Шкільний світ 2006, . – 120 с.
3. HTML 4.0. В подлиннике / А. Матросов, А. Сергеев, М. Чаунин – БХВ-Петербург, 2008, – 678с.
4. Спецификация HTML 4.01 / W3C – 1999.

2 Каскадні таблиці стилів CSS

(6 год)

Питання лекції

1. Поняття CSS, селектор, псевдокласи, псевдоелементи, групування, каскадування.
2. CSS правило. Властивості CSS
3. Блокова модель, позиціонування, нормальний потік, плаваюча модель, абсолютне та відносне позиціонування

Питання на самостійну роботу

1. CSS шари.
2. Спливаючі елементи.
3. Web-стандарти та перевірка коду.

Основні поняття: селектор, псевдокласи, псевдоелементи, групування, каскадування, нормальний потік, плаваюча модель, абсолютне та відносне позиціонування.

2.1 Основи CSS

2.1.1Включення CSS в HTML

Припустимо, у вас є HTML-сторінка до деяких елементів якої треба застосувати стилі. Код сторінки такий:

```
<html>
  <head>
    <title>Приклад</title>
  </head>
  <body>
    <h1>Приклад використання CSS</h1>
    <p>Зараз на даній сторінці будемо пробувати
застосовувати різні стилі.Для прикладу частину
тексту <i>виділимо кусивом</i>, а і ншу частину
<b>виділимо напівжирним</b>.</p>
  </body>
</html>
```

Результат побачимо на Рис 2.1.1

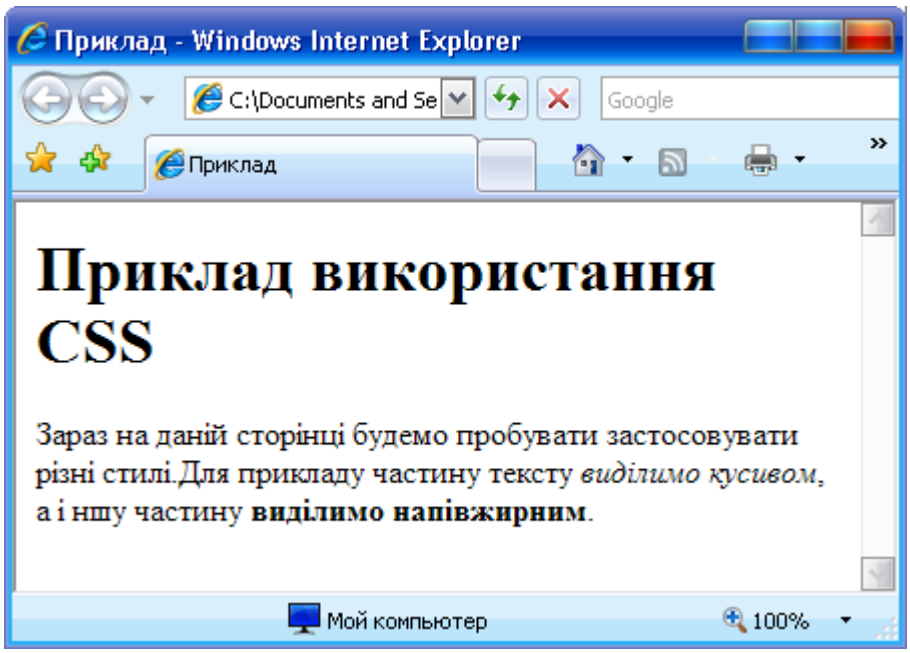

Рис 2.1.1 Приклад простого HTML- документа

Припустимо, зробимо заголовок `<h1>` червоним. Це можна реалізовувати як за допомогою CSS, так і за допомогою HTML. Засобами HTML це робиться так:

```
<h1><font color="#FF0000"> Приклад використання
CSS </font></h1>
```

Якщо використовувати CSS, то застосувати стиль до елемента `<h1>` можна декількома способами. Наприклад, вписати в секцію `head` такий код:

```
<style type = "text/css">
H1 {
  color: #F00
}
</style>
```

Таблиці стилів, включені в сторінку у такий спосіб, називаються *вбудованими*. У даному випадку текст у всіх елементах `<h1>` на цій сторінці будуть червоного кольору.

Крім вбудованих є ще й *внутрішні* стилі. Вони вставляються безпосередньо в потрібний елемент за допомогою атрибуту `style` (зауважте, що в HTML є як елемент `<style>`, так і атрибут `style`):

```
<h1 style="color: #F00"> Приклад використання
CSS </h1>
```

Текст буде червоним тільки в даному елементі <h1> і ніде більше. Потрібно сказати, що внутрішні стилі мають кілька принципових недоліків:

- Застосовуючи цей стиль до будь-якого елементу, не використовується селектор, тому згодом можна не згадати, чому саме цей елемент <h1> має червоний колір, а другий елемент <h1> - чорний;
- Якщо використовувались внутрішні стилі і при цьому захочете поміняти колір заголовків на всіх сторінках сайту, то доведеться міняти їх вручну, як і при зміні HTML-коду, тобто зникає одна з головних переваг CSS: поділ структури документа і його візуального представлення.

Крім того, таблицю стилів можна помістити в окремий файл і підключити його за допомогою елемента <link>. Такі таблиці стилів називаються *зовнішніми*. Помістили наш стиль, застосований до елемента <h1>, в файл style.css, який розміщений у тому ж каталозі, що і HTML-файл. Тоді до HTML-документу його можна підключити наступним чином:

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

Тег <link> обов'язково повинен знаходитися в секції head. Зовнішні таблиці стилів володіють очевидним і дуже важливою перевагою: їх можна підключати до будь-якій кількості HTML-документів. Тому для всього сайту може бути всього одна зовнішня таблиця стилів, яка знаходиться у файлі з розширенням css. У цей файл можна помістити все те, що і у вбудованих таблицях стилів розташоване між тегами <style> </style> тобто в нашому випадку файл style.css містить всього один рядок:

```
H1 {
  color: #F00
}
```

Докладніше розглянемо атрибути тега <link>.

- rel - описує відношення зовнішньої таблиці стилів та документа. Може набувати значення stylesheet і alternate stylesheet. Якщо задано перше з них, то зовнішня таблиця стилів використовується браузером для форматування документа по замовчуванню, якщо ж вказано

друге значення, цю таблицю стилів для форматування документа може вибрати користувач.

- `type` - використовується для вказування мови стилів. Крім CSS існує ще XSL і деякі інші мови, які на даний момент практично не використовуються. Для XSL атрибут `type` повинен мати значення `text/xsl`.
- `href` - у цьому атрибуті вказується шлях до зовнішнього CSS-файлу. Шлях може бути як відносним, так і абсолютним.

Всі вищенаведені атрибути є обов'язковими, але існує ще один необов'язковий атрибут - `media`. Він визначає тип пристрою для якого потрібно застосовувати дану таблицю стилів. Перелік можливих значень цього атрибуту представлений у Таблиця 2.1.1

Таблиця 2.1.1 Значення атрибуту MEDIA

Значення	Застосування таблиць стилів
`all`	Для будь-яких типів пристроїв
`aural`	Для пристроїв з голосовим виводом
`braille`	Для пристроїв з тактильним виводом за допомогою азбуки Брайля
`handheld`	Для переносних пристроїв з маленьким екраном (наприклад, КПК)
`print`	Для виводу сторінки на друк
`projection`	Для виводу сторінки на пристрої типу проектора
`screen`	Для виводу сторінки на кольоровий екран
`tv`	Для виводу сторінки на інтернет-пристрої на основі телевізора

Атрибут `media` по замовчуванню має значення `all`, так що застосовувати його треба тільки в тому випадку, якщо ви використовуєте дану таблицю стилів для певного типу пристроїв.

Є ще один спосіб підключення зовнішніх таблиць стилів за допомогою інструкції `@import`:

```
<style type = "text/css">
H1 {
  color: #F00
}
@import url(style.css)
</style>
```

У цьому випадку відбувається інтегрування вже існуючих стилів і стилів, що знаходяться у файлі style.css.

Підводячи підсумок, запам'ятаємо, що стилі до HTML-сторінці можна підключати за допомогою тега <link> і інструкції @import, розміщувати їх в розділ head між тегами <style></style> І вставляти безпосередньо в елементи за допомогою атрибуту style.

2.1.2Селектори

Для того, щоб вибрати елемент або групу елементів до яких застосовується стиль використовується селектори.

Визначення

Селектором будемо називати конструкцію, що дозволяє вибрати елемент, до якого будуть застосовані дані стилі.

У CSS-1 є порівняно небагато способів вибору елемента, до якого можна застосувати певний стиль.

Селектор по елементу

В якості селектора може виступати назва самого елементу (P, EM, TD, BODY). Наприклад, конструкція:

```
P {
  color: #CCC;
  font-size: 14px
}
```

позначає, що текст всередині всіх тегів <P></P> буде сірого кольору і розміром 14 пікселів.

Аналогічно можна як селектор використовувати будь-який елемент, що є в мові HTML. Але зауважте, що при цьому застосовані до даного селектору стилі будуть поширюватися на всі елементи цього виду, що зустрічаються на сторінці. Наприклад, в наступній таблиці стилів чорний колір задається для всіх елементів <TD> і <P>, а червоний колір для всіх елементів <EM>:

```
TD {
  color: #000
}
P {
  color: #000
```

```
}
EM {
  color: #F00
}
```

Однак найчастіше треба якимось чином виділити елемент з собі подібних. Наприклад, необхідно, щоб деякі заголовки <H1> були синіми з розміром шрифту 18 пікселів, а деякі заголовки <H1> чорними з розміром шрифту 16 пікселів. Цілком очевидно, що якщо як селектор брати назву елемента H1, то нам не вдасться добитися такої диференціації. Для вирішення цієї проблеми в CSS-1 ввели два механізми: класи та унікальні ідентифікатори (або ID).

Спершу розглянемо поняття класу.

Селектор по класу

З назви можна зробити висновок, що цей механізм дозволяє розбивати елементи на класи, які можуть мати зовсім різні стильові рішення. Робиться це в такий спосіб:

```
<P> Тут міститься який-небудь текст, </P>
<P CLASS = "smaller"> А цей абзац буде
виведений дрібнішим шрифтом. </P>
```

Як бачите, прямо в тілі документа ми присвоюємо елементу <P> клас, який носить назву smaller. А стилі до даного класу застосовуються таким чином:

```
P {
  color: #CCC;
  font-size: 14px
}
P.smaller {
  font-size: 12px
}
```

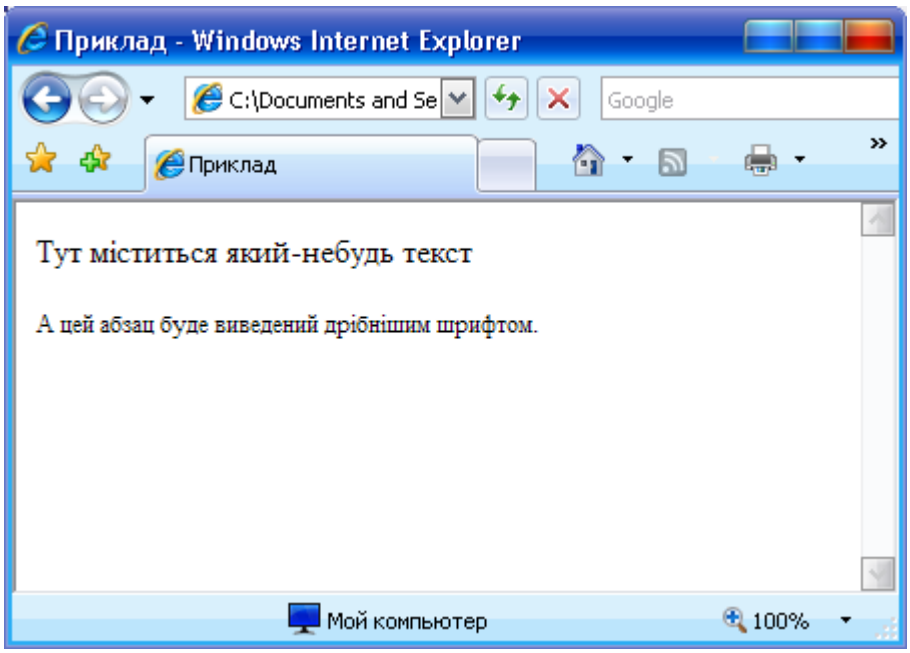

Рис 2.1.2 Використання селекторів по класу

Як бачимо на Рис 2.1.1 перший абзац буде виведений шрифтом розміром 14 пікселів, а другий - шрифтом розміром 12 пікселів. Як бачите, синтаксис схожий на застосовуваний в об'єктно-орієнтованих мовах програмування, доступ до класу здійснюється через точку, наче клас `smaller` є властивістю об'єкта `<P>`.

Треба сказати, що в CSS можна застосовувати даний клас до абсолютно різних елементів. Наприклад:

```
<PRE CIASS="smaller">Тут міститься який-небудь
текст, який збереже своє форматування. </PRE>
<P CLASS = "smaller"> А в цьому абзаці буде
просто текст. </ P>
```

У даному випадку клас `smaller` мають і елемент `<P>`, і елемент `<PRE>`. У таблиці стилів можна вказати, що стиль для класу `smaller` застосовується до всіх елементів, а не лише до елементів `<P>`, як було зазначено раніше. У цьому випадку перед точкою ім'я елемента не ставиться:

```
.smaller {color: #CCC; font-size: 14px}
```

Фактично, селектор по класу, що починається з крапки, застосовується до всіх елементів з даними класом, тобто місце перед точкою позначає будь-який елемент. Тоді текст, що відповідає цим елементам, буде виведений сірим шрифтом розміром 14 пікселів.

А можна диференціювати елементи `<P>` і `<PRE>`, що мають однаковий клас `smaller`:

```
P.smaller {
  color: #CCC;
  font-size: 14px
}
PRE.smaller {
  color: #F00;
  font-size: 12px
}
```

У цьому випадку текст в елементі <PRE> буде виведений червоним шрифтом розміром 12 пікселів, а текст в елементі <P> - сірим шрифтом розміром 14 пікселів.

Селектор по ID

Перейдемо до розгляду унікальних ідентифікаторів, які ми для стислості будемо називати просто ID. За допомогою ID можна застосовувати стиль до точно вибраного елементу, тобто в HTML-коді може бути тільки один елемент з даними ID. Наприклад:

```
<H1 ID=”first”>Унікальний заголовок</H1>
```

Селектор для ID="first" формується, взагалі-то, аналогічно селектору по класах, але замість крапки для адресації використовується знак #:

```
Hl#first {
  color: #00F;
  font-family: Verdana, sans-serif
}
```

Однак ID є більш специфічним, ніж клас. Припустимо, в коді є елемент <H1> виду:

```
<H1 CLASS="redhead" ID="first"> Унікальний
заголовок</H1>
```

а в таблиці стилів клас і ID описані наступним чином:

```
Hl#first {
  color: #00F;
  font-family: Verdana, sans-serif
}
H1.redhead {
  color: #F00;
```

```
  font-family: "Times New Roman", serif
}
```

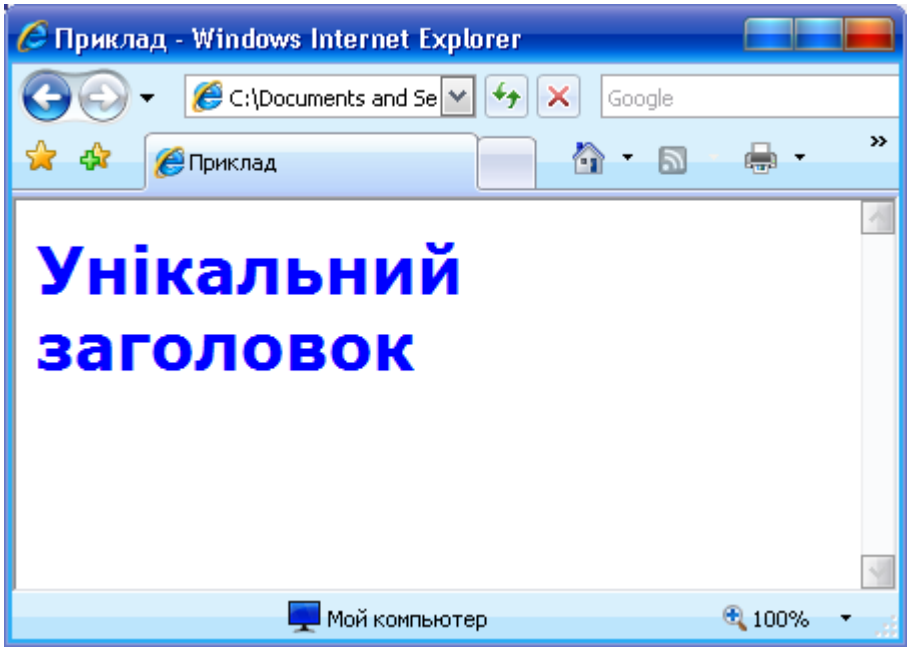

Рис 2.1.3 Спільне використання селекторів по класу і по ідентифікаторі

Бачимо на Рис 2.1.3 заголовок буде виведений на екран синім кольором і шрифтом Verdana. Таким чином, якщо знадобиться виділити з класу який-небудь елемент, можна зробити це за допомогою ID. Звичайно, можна просто ввести новий клас, але іноді від цього страждає логічність коду і сам код важить більше.

До речі кажучи, в HTML-документі може бути тільки один ID, тобто код, наведений нижче, буде некоректним:

```
<P ID=”red”>Абзац тексту</P>
<CODE ID=”red”>="рага">Приклад коду</CODE>
```

Проте всі браузери дозволяють застосовувати стиль до кількох елементів, що містить однаковий ID, що є помилкою, хоча і не грубою.

У CSS-1 є ще один вид селектор - контекстні селектори.

Контекстний селектор

Наприклад, необхідно, щоб у заголовках <H2> фрази, виділені за допомогою елемента <STRONG>, були помаранчевого кольору. Перше, що спадає на думку, написати такі стилі:

```
H2 {
  color: #00F
}
```

```
STRONG {
  color: orange
}
...
<H2>Заголовок
з<STRONG>оранжевим</STRONG>виділенням</H2>
```

Однак у цьому випадку всі елементи <STRONG> на сторінці будуть помаранчевого кольору. Можна ще віднести під всі елементи <STRONG>, які знаходяться всередині елементів <H2>, окремий клас. Тоді код доведеться написати так:

```
<H2>Заголовок з<STRONG CLASS="inhead">
оранжевим </STRONG> виділенням </H2>
```

А таблиця стилів буде наступний:

```
H2 {
  color: #00F
}
.inhead {
  color: orange
}
```

Проте, досить незручно весь час вставляти клас у елемент <STRONG>, тому що це збільшує код. З цієї причини і ввели контекстні селектори, які дозволяють застосувати стиль до певного виду елементів, що знаходяться всередині іншого елемента. У нашому випадку стиль буде виглядати так:

```
H2 STRONG {
  color: orange
}
```

Тут в якості селектора виступають імена елементів через пробіл. Дослівно це можна сказати: "Елементи <STRONG>, які знаходяться всередині елементів <H2>, будуть мати помаранчевий колір". Аналогічно можна застосовувати стилі до елементів будь-якого рівня вкладеності. Наприклад, стиль

```
P CODE EM {
  color: green;
  font-weight: bold
}
```

```
...
<P> Тут йде текст абзацу, а тут починається
<CODE> приклад комп'ютерного коду з <EM>
виділеним </ EM> словом </CODE>, після чого
абзац продовжуєnmcя. </P>
```

виводить елементи <EM>, які знаходяться в абзаці в коді напівжирним шрифтом зеленого кольору.

Ми розглянули чотири способи вибору елемента, до якого буде використовуватись стиль. Зауважте, що всі вони ґрунтуються на розташуванні елемента в дереві документа.

Визначення

Деревом документа називається структура всіх елементів на сторінці з врахуванням їх вкладеності.

Однак, є чимало ситуацій, коли інформації про розташування елемента в документі недостатньо для адресації. Наприклад, у мові HTML немає елемента, який відповідав би першій букві параграфа (так званої буквиці). З цієї причини даний елемент відсутній у дереві документа і звичайним способом неможливо застосувати стиль до першої букві параграфа.

Псевдоелементи

Для вирішення проблеми з буквиці в CSS-1 ввели псевдоелементи `first-letter` і `first-line`, які відповідають першій букві і першому рядку параграфа. Приставка псевдо підкреслює той факт, що дані елементи не є елементами в звичайному розумінні цього слова, тобто в самому тілі документа їх немає, тому і в дереві документа вони не відображаються, але в селекторах їх можна використовувати. Наявність псевдоелементів дозволяє робити буквиці і виділяти перші рядки параграфів без будь-яких хитрощів.

Із застосуванням псевдоелемента `first-letter` буквиця реалізується зовсім просто, в таблиці стилів треба написати:

```
P {
  font: 12px Verdana, Tahoma, sans-serif
}
P:first-letter {
  color: #F00;
  font: bold 36px Verdana, Tahoma, sans-serif;
  float: left
```

```
}
```

А в HTML-коді взагалі не потрібно буде виділяти першу літеру, тобто можна просто написати:

```
<P>Перша буква в наведеному абзаці буде великою
і червоною. </P>
```

Результат такого в коду в браузері Рис 2.1.4

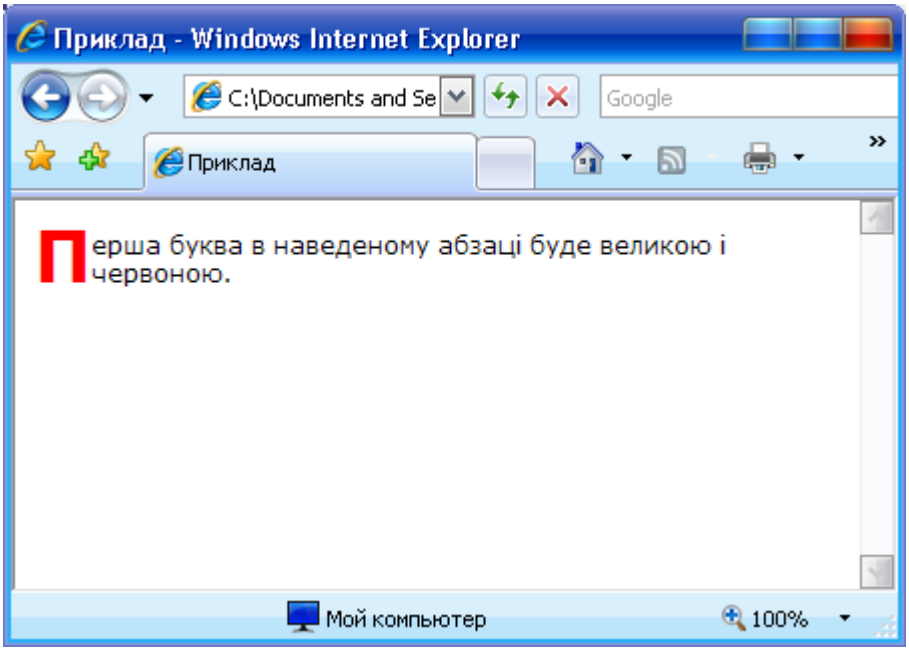

Рис 2.1.4 Приклад використання псевдоелементу

Селектор `p:first-letter` застосовувати стилі до всіх перших літер тексту в елементах `<P>`. Властивість `float:left` вказує на те, що текст повинен обтікати буквиці справа.

Псевдо елемент `first-letter` дозволяє змінити перший рядок абзацу. Результат дії такого коду показаний на Рис 2.1.5:

```
P:first-line {
  color: #F00;
  font: bold 20px Verdana, Tahoma, sans-serif
}
...
<P>В даному абзаці ми змінимо вигляд першого
рядка</P>
```

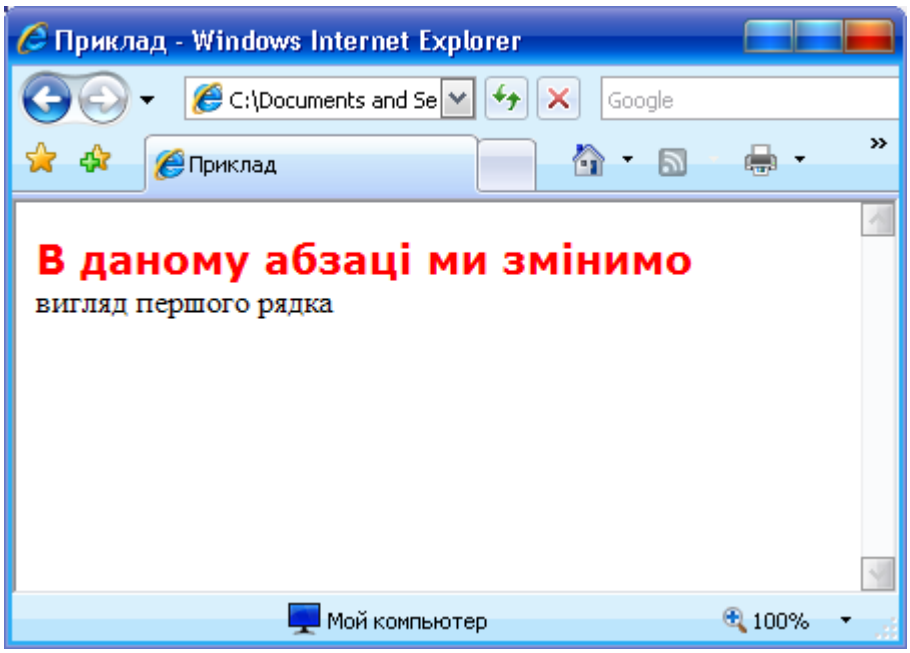

Рис 2.1.5 Приклад зміни першого рядка абзацу

Крім псевдоелементів в CSS є ще псевдокласи.

Псевдокласи

Визначення

Псевдокласом називається клас, який грунтується на інформації, яка не є частиною HTML-документа.

Наприклад, є псевдоклас, який позначає відвідані посилання, проте в коді документа зовсім неможливо відрізнити відвідувані посилання від невідвідуваних. У даному випадку таку інформацію може надати браузер, на основі цих даних і визначається яке це посилання: відвідуване чи ні, тобто ця інформація не міститься в тілі документа.

У CSS-1 є чотири псевдокласи, які позначають невідвідувані посилання, відвідувані посилання, посилання на які наведений курсор миші і активні посилання, тобто ті на які в даний момент натискає відвідувач сайту. Ось найпростіша таблиця стилів, яка містить чотири псевдокласи:

```
A:link {
  color: #00F
}
A:visited {
  color: #CCC
}
A:active {
  color: #F00
}
```

```
A:hover {
  color: #F90
}
```

У даному випадку невідвідуванні посилання на сторінці будуть синього кольору, відвідуванні - сірого, активні - червоного, а при наведенні курсору миші посилання будуть виділені помаранчевим кольором. У HTML є аналог трьох з цих псевдоклассов - це атрибути тега <BODY>:

- LINK - відповідає невідвідуваним посиланням;
- VLINK - відповідає відвідуваним посиланням;
- ALINK - відповідає активним посиланнями.

Проте таким способом можна задати один і той же колір для всіх посилань на сторінці, але ніяк не можна їх диференціювати. А ось за допомогою псевдокласів це робиться дуже легко, тому що разом з псевдокласами можна використовувати і звичайні класи. Наприклад, внизу сторінки часто дублюють навігацію. Колір таких посилань може відрізнятися від кольору основних посилань. Можна написати наступний CSS-код:

```
A.footer:link {
  color: #000
}
A.footer:visited {
  color: #CCC
}
A.footer:active {
  color: #00F
}
A.footer:hover {
  color: #FFF
}
```

Тоді в HTML-коді посилання

```
<A HREF="/" CLASS="footer"> Головна </A>
```

в залежності від статусу буде відображатися саме зазначеними вище кольором.

Крім того, псевдокласи можна використовувати і з ID. Наприклад, так:

```
A#someid:active {
```

```
    color: #F00
}
```

У даному разі колір активних посилань з `ID = "someid"` буде червоним.

А тепер детально розберемо структуру правил, за допомогою яких застосовуються стилі.

2.1.3Синтаксис та структура CSS

Для початку дамо визначення поняттю правила в CSS.

Визначення

Правилом називається структурна одиниця таблиці стилів, яка містить опис стилів для цього елемента.

Правило має структуру, зображену на Рис 2.1.6

Рис 2.1.6 Структура правила CSS

Таким чином, *правило* складається з *селектора*, який завжди розташовується зліва, і *блоку оголошень стилів*, який розміщується у фігурних дужках і слідує безпосередньо за селектором. Кожне оголошення в свою чергу складається з *властивості* і її *значення*. Саме властивості позначають вигляд стилю, який буде застосований до елемента, вибраного за допомогою селектора. . Властивістю може бути колір елемента (`color`), параметри шрифту (наприклад, `font-size`), відступи (наприклад, `padding-left`) та ін.

Правило може містити кілька оголошень. Втім, оголошень може і не бути зовсім, тому конструкція

```
H1{}
```

синтаксично коректна, хоча і ніяк не впливає на відображення елементів `<H1>`.

Якщо оголошень кілька, то всі вони відокремлюються одне від одного крапкою з комою. Після останнього оголошення крапку з комою можна не ставити. Тому стилі:

```
H1 {
  font-family: Verdana;
}
H1 {
  font-family: Verdana
}
```

абсолютно еквівалентні і коректні. Властивість і його значення завжди розділяються двокрапкою. У правилі

```
H1 {
  color: #CCC
}
```

властивістю є колір (`color`), а значенням є завдання цього кольору в RGB (`#CCC`).

В деяких випадках властивість може мати кілька значень. Тоді вони розділяються пробілами. Наприклад, правило

```
H1 {
  font: bold 18px Verdana
}
```

вказує на те, що заголовки `<H1>` треба виводити напівжирним шрифтом Verdana розміром 18 пікселів.

Іноді є необхідність поставити ряд значень, які застосовувалися б у разі неможливості застосування попередніх. Наприклад, можна задати шрифтовий ряд:

```
H1 {
  font-family: Verdana, Tahoma, sans-serif
}
```

У даному прикладі, якщо в операційній системі користувача не встановлений шрифт Verdana, то для відображення заголовків `<H1>` буде використаний шрифт Tahoma. Якщо ж і цей шрифт не встановлений, то буде використаний шрифт, що є для даної операційної системи шрифтом без зарубок по замовчуванню (він позначається ключовим словом `sans-serif`).

Всі пробіли і переходи рядків у блоці оголошень ігноруються, так що для поліпшення сприйняття таблиці стилів можна використовувати будь-яку кількість цих символів. Наприклад, правило

```
DIV {
  color: #000;
  background-color: #DDD;
  font-size: llpx
}
```

цілком аналогічно з правилом

```
DIV {color: #000, background-color: #DDD; font-
size: llpx}
```

Групування селекторів

Часто виникають ситуації, коли до різних елементів застосовуються одні і ті ж стилі. Наприклад, нам треба, щоб текст усередині елементів <P> і <TD> виглядав однаково. Тоді ми створимо щось схоже на це:

```
P {
  font: 12px Verdana, Tahoma, sans-serif
}
TD {
  font: 12px Verdana, Tahoma, sans-serif
}
```

Таку конструкцію можна оптимізувати з допомогою групування.

Визначення

Групуванням називається об'єднання селекторів з однаковими оголошеннями з метою зменшення обсягу коду.

При групуванні селектори повинні розділятися комою:

```
P, TD {
  font: 12px Verdana, Tahoma, sans-serif
}
```

Це правило цілком аналогічно двом правилам з попереднього прикладу. Однак використовувати групування треба обережно, тому що іноді може порушуватися логічність таблиці стилів.

От ми й підійшли до однієї дуже цікавої проблеми.

Каскадування

Спробуємо розібратися чому саме «каскадні» таблиці стилів.

Одному і тому ж елементу можна застосовувати кілька правил. Тому наступна таблиця стилів коректна з точки зору CSS:

```
EM {
  color: #F00
}
EM {
  font: 12px Verdana, sans-serif
}
```

У такому разі до елемента <EM> будуть застосовані обидва правила, тобто текст всередині <EM> буде відображатися на екрані червоним кольором і шрифтом Verdana. Але дуже часто виникають і більш заплутані ситуації. Допустимо, у нас є така таблиця стилів:

```
P {
  color: #000;
  font: 12px Verdana, Tahoma, sans-serif
}
P.smaller {
  color: #333;
  font-size: 10px
}
CODE P {
  color: #00F;
  font-family: "Courier new", monospace
}
```

А в коді сторінки є конструкція:

```
<CODE>
<P> Тут йде код програми </ P>
<P CLASS = "smaller"> Тут що-небудь менш
важливе з коду дрібним шрифтом </P>
</CODE>
```

Виходить що тег <P CLASS = "smaller"> задовольняє всім трьом правилам з таблиці стилів:

- Він є елементом <P>;
- Він відноситься до класу smaller;
- Він вкладений в елемент <CODE>.

Слід розібратися, які саме правила застосовувати до даного елементу? Зрозуміло, що він не може одночасно мати три різних кольори і різний розмір шрифту. Для вирішення цієї проблеми ввели механізм каскадування, який функціонує за наступним алгоритмом.

Інтерпретатор таблиці стилів (іншими словами, веб-браузер) спочатку знаходить в таблиці всі оголошення стилів для цього елемента. Якщо селектор відповідає елементу, то оголошення застосовується. Припустимо, написано в таблиці стилів `H4` (`color: green`) і якщо в коді зустрінеться елемент `<H4>`, то до нього буде застосовано оголошення `color: green`. У нашому прикладі всі селектори відповідають елементу `<P>`, так що на цьому кроці алгоритму конфлікт не усувається. Якщо ж на елемент немає стилів, то вони успадковуються від попереднього елемента. Наприклад, у коді:

```
P {
  color: #AAA
}
...
<P> Текст з <EM> виділеним </ EM> словом </ P>
```

елемент `<EM>` буде сірого кольору, хоча явно це в стилях не зазначалося. Тим не менше, елемент `<EM>` є нащадком елемента `<P>`, тобто він вкладений в елемент `<P>`, і з цієї причини успадкує батьківський колір.

Якщо ж на батьківський елемент теж не написані стилі, то використовуються значення по замовчуванню.

Знайдені оголошення сортуються за ступенем важливості. У CSS існує ключове слово: `!important`:, яке дозволяє збільшувати важливість даного стилю. Приклад:

```
P {
  color: #CCC !important;
  font: 12px Verdana, Tahoma, sans-serif
}
```

У деяких браузерах можна налаштовувати свої власні таблиці стилів, які переписуються авторськими стилями. Проте користувач може помітити деякі оголошення як важливі, тоді вони перевизначать авторські стилі. Але якщо автор помітив оголошення словом

!important, то вони в будь-якому випадку перевизначать стиль, встановлений користувачем.

Далі знайдені оголошення сортуються по параметру, що називається *специфічністю*. Специфічність обчислюється таким чином: рахується кількість ID у селекторі (a), рахується кількість класів в селекторі (b), рахується кількість імен елементів в селекторі (c). Значення специфічності виводиться шляхом об'єднання трьох значень abc, тобто фактично це специфічність селектора. Розглянемо приклади наведені в Таблиця 2.1.2

Таблиця 2.1.2 Приклади обчислення специфічності різних селектор

Селектор	Обчислення	Специфічність
P.smaller {}	a = 0; b = 1; c = 1	11
CODE P {}	a = 0; b = 0; c = 2	2
layer {}	a = 1; b = 0; c = 0	100
DIV#layer P.smaller {}	a = 1; b = 1; c = 2	112
# Layer #inlayer P	a = 2; b = 0; c = 1	201

Правила, що пов’язані з більш специфічним селектором, будуть перевизначати правила, пов'язані з менш специфічним селектором. Якщо ж специфічність буде однакова, то до елемента буде застосований той стиль, який описаний пізніше.

Повернемося до нашого прикладу:

```
P {
  color: #000;
  font: 12px Verdana, Tahoma, sans-serif
}
P.smaller {
  color: #333;
  font-size: 10px
}
CODE P {
  color: #00F;
  font-family: "Courier new", monospace
```

```
}
```

У коді селектор `P {}` має специфічність 1, селектор `P.smaller {}` має специфічність 10 і селектор `CODE P {}` має специфічність 2. Звідси, найважливішими будуть оголошення стилів у правилі селектором `P.smaller {}`, потім у правилі з селектором `CODE P {}`, а потім уже в правилі з селектором `P {}`. Таким чином, в коді

```
<CODE>
<P> Тут йде код програми </ P>
<P CLASS = "smaller"> Тут що-небудь менш
важливе з коду дрібним шрифтом </P>
</CODE>
```

тег `<P CLASS = "smaller">` буде відображений сірим кольором `#333` і шрифтом розміром 10 пікселів. А накреслення шрифту `Courier New` він успадкує від правила `CODE P {}`.

Власне, в цьому і полягає механізм каскадування.

Визначення

Наслідування - це механізм, який дозволяє елементам-нащадкам мати ті самі стилі, що й у елемента-предка.

Узагальнюючи, можна дати чітке визначення поняття "каскадування".

Визначення

Каскадування-це метод визначення ваги або важливості конкретного оголошення, який усуває конфлікти, що виникають у випадку декількох оголошень для одного і того самого елемента.

2.1.4Одиниці вимірювань

Розглянемо, як задаються значення кольорів, лінійних розмірів і URL. Почнемо з лінійних розмірів.

Одиниці довжини

Всі одиниці довжини служать для завдання вертикальних або горизонтальних розмірів.

Саме значення довжини будується наступним чином: спочатку йде знак - або +, причому знак + ставиться по замовчуванню і його можна опускати, потім слідує чисельне значення, а в кінці *ідентифікатор розмірності*, що представляє собою двохлітерну абревіатуру (його, до речі, теж можна опускати, якщо чисельне значення дорівнює

нулю). Треба сказати, що не всі властивості можуть мати негативні значення.

Всі одиниці довжини діляться на два великі класи: *відносні* та *абсолютні*. Відносні одиниці встановлюють розмір відносно іншої довжини. Їх усього три:

- `px`. `1px` дорівнює одній точці на екрані монітора, яку зазвичай називають пікселем;
- `em`. `1em` дорівнює розміру (висоті) шрифту цього елемента, тобто, якщо елемент `<H1>` має розмір шрифту 18 пікселів, то правило:

```
H1 {padding-top: 2em}
```

означає, що верхній відступ елементів `<H1>` дорівнює 2x18 = 36 пикселів;

- `ex`. `1ex` дорівнює висоті малої літери "x" у шрифті. Саме по цій букві формується оптична висота рядка. Зазвичай вона відповідає половині висоти шрифту, проте в деяких шрифтах це співвідношення змінюється. Треба сказати, що особливої різниці між `em` і `ex` немає, але в деяких випадках користуватися тими чи іншими одиницями зручніше. Крім того, для властивості `font-size` величина `em` або `ex` визначається по батьківському елементу. Наприклад, у коді:

```
P {
  font-size: 12px
}
EM {
  font-size: 1.5em
}
...
<P>Текст з <EM>виділенням</EM></P>
```

Текст в елементі `<P>` буде відображений шрифтом розміром 12 пікселів, а текст в елементі `<EM>` шрифтом розміром в 1,5 рази більше, тобто 18 пікселів.

Абсолютні одиниці довжини мають фіксований розмір. Їх усього п'ять видів:

- `in`. `1in` дорівнює одному дюйму, який у свою чергу дорівнює 2,54 см;

- pt. lpt дорівнює 1/72 дюйма, ця величина називається друкарським пунктом і прийшла в CSS з поліграфії, де дуже часто кегль шрифту вказується в пунктах;
- pc. lpc дорівнює 12pt або 1/6 дюйма, ця величина називається списом (теж з поліграфії);
- mm. lmm дорівнює одному міліметру метричної системи одиниць;
- cm. lcm дорівнює одному сантиметру (теж з метричної системи одиниць).

Треба сказати, що використовувати абсолютні величини треба тільки в тому випадку, коли точно відомі параметри пристрою виводу. Наприклад, екрани моніторів можуть мати зовсім різні розміри. Тому в таблицях стилів, які використовуються саме для виведення на екран, краще вживати відносні одиниці довжини. Абсолютні одиниці довжини можна застосовувати, скажімо, при виведенні сторінки на друк.

Крім того, в CSS в якості одиниці вимірювання можна користуватися відсотками від будь-якої величини. Без відсотків абсолютно не обійтися при "гумовії" верстці.

Позначення кольору

У CSS існує кілька способів задання кольору. Перш за все, колір можна задавати з допомогою ключових слів. Їх 16 і прийшли вони з палітри VGA. Наприклад, синій колір задається так:

```
H1 {
  color: blue
}
```

Дуже популярною є кольорова схема RGB, де всі кольори описуються ступенем насиченості червоного, зеленого і синього компонентів. Насиченість компонентів вказується числами в шістнадцятковому форматі. Наприклад, білий колір позначається #FFFFFF, чорний - #000000, червоний - #FFCC00, сірий - #666666. Крім того, є скорочений запис кольорів. Наприклад, білий - #FFF, червоний - #FC0. Як бачимо, замість шести цифр можна записати три, причому вони дублюються. Зрозуміло, що такий скорочений запис можливий тільки в тому випадку, коли цифри в кожному компоненті кольору однакові.

На практиці майже завжди користуються саме такими позначеннями кольорів, але існує ще визначення кольору у вигляді функцій:

- `H1 {color: rgb(0,0,255)}` - заголовки будуть виведені синім кольором;
- `H1 {color: rgb(0, 100%, 0)}` - заголовки будуть зеленими.

Задання URL

У деякі властивості треба вказувати шлях до файлу. Наприклад, у властивості `background-image` треба вказувати шлях до графічного файлу, який повинен бути фоном. Робиться це в такий спосіб:

```
BODY {
 background-image:
url{http://www.artel.by/i/backl.gif)
}
```

Формат універсального локатора ресурсу такий: спочатку йде позначення функції `url`, а потім у круглих дужках вказується шлях до файлу Можна вказувати абсолютні шляхи, а можна і відносні, причому шлях буде визначатися щодо таблиці стилів, а не щодо документа. Так що, якщо ви підключаєте до даного документу зовнішню таблицю стилів за допомогою тега <LINK>, то шлях буде визначатися щодо місцезнаходження на сервері даної таблиці стилів, а якщо ви використовуєте в документах вбудовані таблиці стилів між тегами `<STYLE> </ STYLE>` , то в цьому випадку шлях буде визначатися щодо документа, тому що фізичне місцезнаходження таблиці стилів і документу збігається (вони знаходяться в одному і тому ж файлі).

У всіх інших випадках як значення властивості задаються ключові слова, які у кожному випадку свої.

2.2 Властивості CSS.

2.2.1Колір і фон

Можна встановлювати колір тексту, фону і рамок в елементі.

color

Задає колір елемента. Можна застосовувати до всіх елементів без винятку.

Як значення вказується колір в одному з доступних форматів. Наприклад, такі правила абсолютно ідентичні і встановлюють червоний колір логічного виділення в тексті:

```
EM {
  color: red
}
EM {
  color: #F00
}
EM {
  color: rgb(255,0,0)
}
```

Приклад

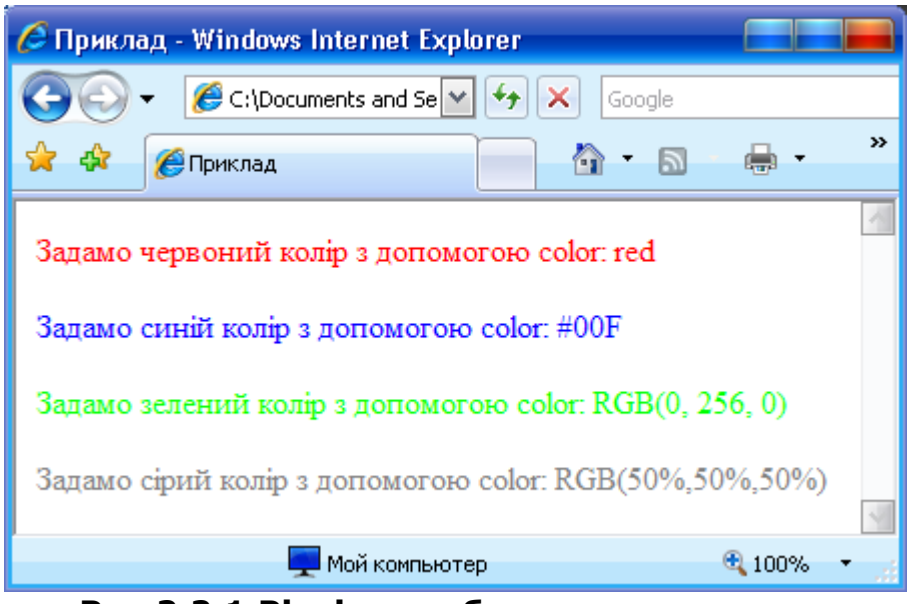

Рис 2.2.1 Різні способи задання кольору

```
P.red {
  color: red
}
P.blue {
```

```
  color: #00F
}
P.green {
  color: RGB(0, 256, 0)
}
P.gray {
  color: RGB(50%, 50%, 50%)
}
...
<P CLASS="red">Задамо червоний колір з
допомогою color: red</P>
<P CLASS="blue">Задамо синій колір з допомогою
color: #00F</P>
<P CLASS="green">Задамо зелений колір з
допомогою color: RGB(0, 256, 0)</P>
<P CLASS="gray">Задамо сірий колір з допомогою
color: RGB(50%,50%,50%)</P>
```

background-color

Задає колір фону елемента.

Як значення вказується колір в одному з доступних форматів або ключове слово `transparent`, яким позначається прозорий фон. По замовчуванню фон встановлений як `transparent`.

Приклад

```
P.white {
  background-color: green;
  color: #FFF
}
P.yellow {
  background-color: blue;
  color: yellow
}
...
<P CLASS="white">Білий текст на зеленому фоні
</P>
<P CLASS="yellow">Жовтий текст на синьому фоні
</P>
```

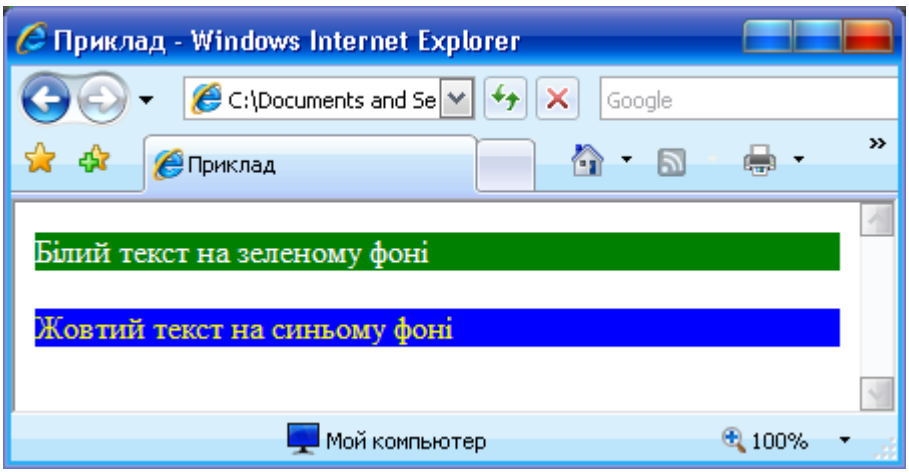

Рис 2.2.2 Зміна кольору фону

background-image

Задає графічне фонове зображення елементу.

Значення:

- URL графічного файлу;
- `none` ключове слово, що позначає відсутність фонового зображення.

Ясна річ, що немає сенсу спеціально вказувати `background-image: none`, оскільки це значення у всіх елементів встановлено по замовчуванню.

Взагалі, якщо вказується фонове зображення, то рекомендується вказувати і фоновий колір, тому що малюнок може не завантажитися або користувач може відключити завантаження зображень. У цьому випадку текст може погано читатися на фоні, заданому по замовчуванню, так що краще вказувати прийнятний фон явно.

Приклад

```
P {
  background-image: url(mal.gif);
  color: #FFF
}
...
<P> Білий текст на фоновому малюнку </P>
```

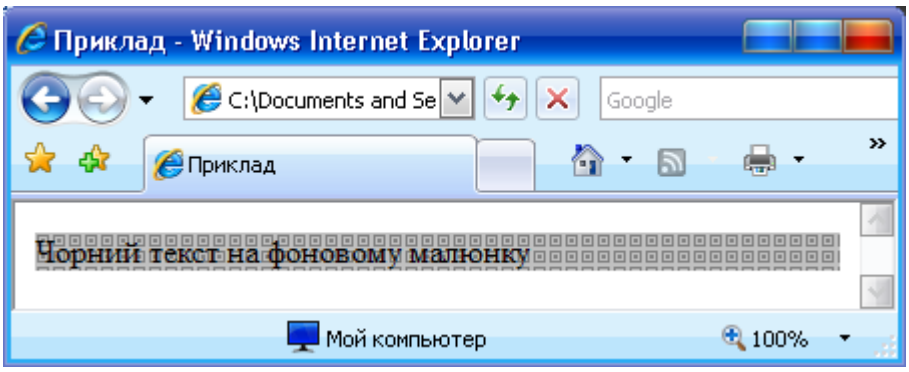

Рис 2.2.3 Використання фонового малюнку

background-repeat

Для заданого фонового зображення ця властивість визначає, *чи буде це зображення повторюватися*, і якщо буде, то яким чином.

Значення:

`repeat` - зображення повторюється по горизонталі і по вертикалі;
`repeat-x` - зображення повторюється тільки по горизонталі;
`repeat-y` - зображення повторюється тільки по вертикалі;
`no-repeat` - зображення не повторюється.

Приклад

```
P.nrp {
  background-image: url(mal.gif);
  background-repeat: no-repeat
}
P.xrp {
  background-image: url(mal.gif);
  background-repeat: repeat-x
}
P.yrp {
  background-image: url(mal.gif);
  background-repeat: repeat-y
}
P.rp {
  background-image: url(mal.gif);
  background-repeat: repeat
}
...
<P CLASS="rp"> Текст на фоновому малюнку, що
повторюється </P>
<P CLASS="xrp"> Текст на фоновому малюнку, що
```

```
повторюється по горизонталі </P>
<P CLASS="yrp"> Текст на фоновому малюнку, що
повторюється по вертикалі </P>
<P CLASS="nrp"> Текст на фоновому малюнку, що
не повторюється </P>
```

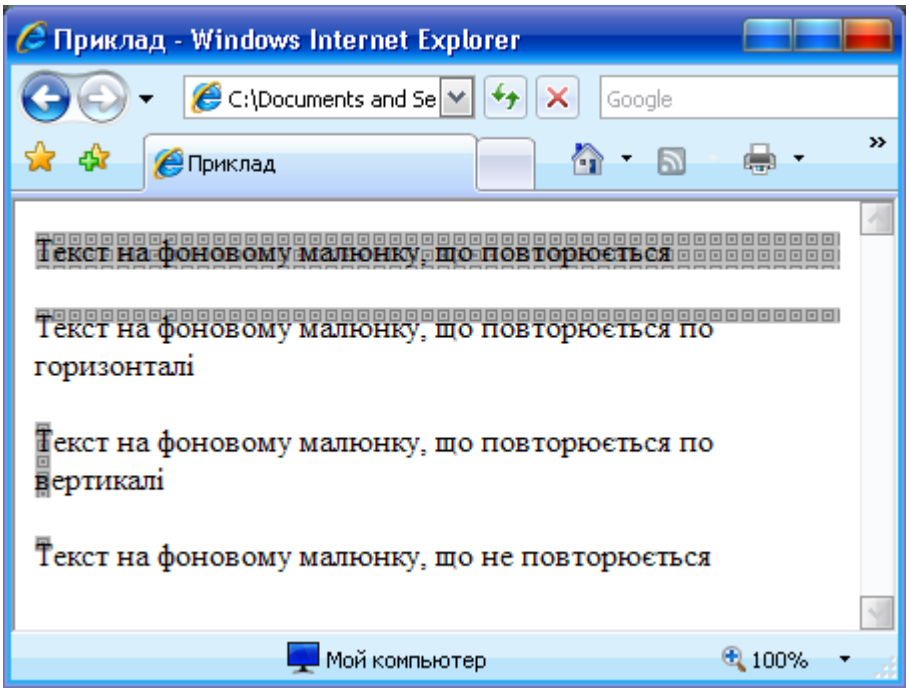

Рис 2.2.4 Зміна властивості повторюваності фонового малюнку

background-attachment

Визначає, *чи буде переміщатися фон разом з усім вмістом сторінки при прокручувані* чи ні. Взагалі коли фон сторінки не переміщуєся, це трохи незвично для відвідувача сайту і іноді викликає роздратування. Так що слід застосовувати дану властивість дуже обережно.

Значення:

- `fixed` - фон буде залишатися нерухомим, а вміст сторінки буде перемішатися відносно нього;
- `scroll` - фон буде переміщуватися разом з рештою вмісту. Значення по замовчуванню.

Приклад

```
P {
  background-image: url(img/bg.gif);
  background-repeat: no-repeat)
  background-attachment: fixed
```

```
}
...
<P> Текст на фоновому малюнку, що не
повторюється і зафіксований</P>
```

background-position

Задає позиціонування фонового зображення. З допомогою цієї властивості можна зміщувати фонове зображення відносно лівого верхнього кута елементу.

Властивість має два параметри: перший визначає зміщення по вертикалі, другий - по горизонталі.

Значення можна вказувати як позитивне, так і негативне. Наприклад, правило

```
P {
  background-image: url (img/bg.gif);
  background-position:-12px 50px
}
```

зміщує фонове зображення на 12 пікселів вліво і на 50 пікселів вниз від лівого верхнього кута елементу `<P>`. Крім того, можна вказати відсоткові співвідношення. Відсотки обчислюються щодо ширини і висоти блоку елемента. Наприклад, правило

```
P {
  background-position: 20% 40%
}
```

зміщує фонове зображення на 20% вправо і на 40% вниз від лівого верхнього кута блоку елемента `<P>`. Значенням по замовчуванню є 0% 0%, що відповідає розташуванню зображення у верхньому лівому куті блоку.

Крім того, можна замість чисельних значень вказувати вирівнювання щодо елемента. Так, для вирівнювання по вертикалі можна використовувати три ключові слова:

- `top` - вирівнювання по верхньому краю;
- `center` - вирівнювання по центру;
- `bottom`-вирівнювання по нижньому краю.

Для вирівнювання по горизонталі можна використовувати ключові слова:

- `left` - вирівнювання по лівому краю;

- `center` - вирівнювання по центру;
- `right` - вирівнювання по правому краю.

Таким чином, правило

```
P {
  background-position: 0% 0%
}
```

еквівалентно правилу

```
P {
  background-position: top left
}
```

Приклад

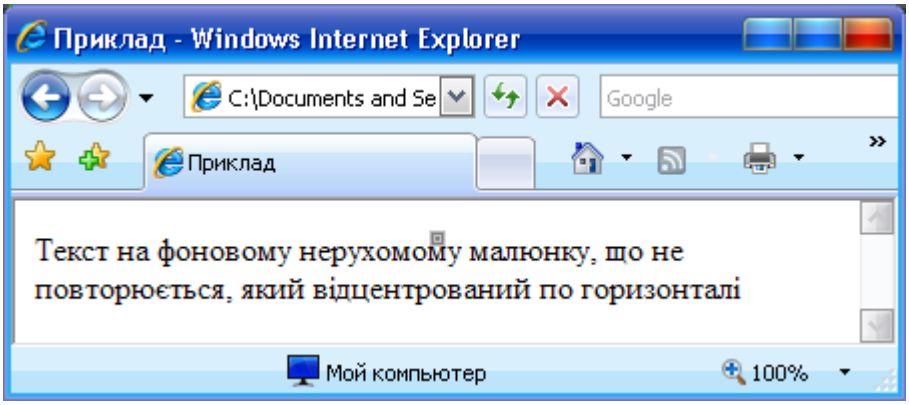

Рис 2.2.5 Використання позиціонування фонового малюнку

```
P {
  background-image: url(mal.gif);
  background-repeat: no-repeat;
  background-position: top center
}...
<P> Текст на фоновому нерухомому малюнку, що не
повторюється, який відцентрований по
горизонталі </P>
```

Скорочена форма запису властивостей фону

Існує скорочена форма запису властивостей фону. Останній приклад можна коротко записати наступним чином:

```
P {
  background: url(img/bg.gif) no-repeat fixed
top center
}
```

тобто формат скороченого запису такий:

```
background: {<background-color> || <background-image> || <background-repeat> || <background-attachment> || <background-position>}
```

У трикутних дужках <> містяться властивості, а знак || позначає пробіл. Причому порядок перерахування властивостей у скороченій формі запису не має значення.

2.2.2Шрифт

Мова HTML може лише встановлювати накреслення шрифту, визначати розміри шрифту, і кілька стильових параметрів, таких як насиченість і нахил. Стандарт CSS-1 недалеко пішов в цьому напрямку від HTML: значно збільшився контроль над розміром шрифту і насиченістю, а також з'явилася можливість виводити текст малими та великими літерами.

font-family

Задає власне шрифт.

Визначення

Шрифт (гарнітура) - це набір символів, які об'єднані спільними стильовими ознаками. Причому сукупність стильових ознак є унікальною.

Як параметр задається ім'я шрифту або назва сімейства. Причому можна вказувати декілька імен через кому, формуючи список альтернативних шрифтів. Тоді у разі, якщо на комп'ютері користувача невстановлений шрифт, який стоїть у списку першим, для відображення тексту, то буде використовуватися другий шрифт зі списку. Якщо ж і його немає. то третій і т.д. Що стосується сімейства шрифтів, то вони можуть бути наступними:

`serif` - шрифти з зарубками, такі як Times New Roman та Garamond;

`sans-serif` - рублені шрифти, такі як Verdana, Arial та Tahoma;

`monospace` - моноширинні шрифти, такі як Courier New;

`cursive` - курсивні шрифти, такі як Zapf-Chancery;

`fantasy` - декоративні шрифти.

Останні два сімейства використовуються вкрай рідко. Треба сказати, що рекомендується сімейство шрифтів вказувати в якості останньої альтернативи тоді, якщо на комп'ютері користувача не

встановлено жодного зі списку, то буде використаний шрифт даного сімейства, який для даного браузера є шрифтом по замовчуванню.: Дизайн сторінки постраждає незначно.

Приклад

```
P.fss {
  font-family: Verdana, Tahoma, sans-serif
}
P.fsr {
  font-family: Times New Roman, serif
}
P.fms {
  font-family: Courier New, monospace
}
...
<P CLASS="fss">Текст виведений шрифтом
Verdana</P>
<P CLASS="fsr">Текст виведений шрифтом Times
New Roman</P>
<P CLASS="fms">Текст виведений шрифтом Courier
New</P>
```

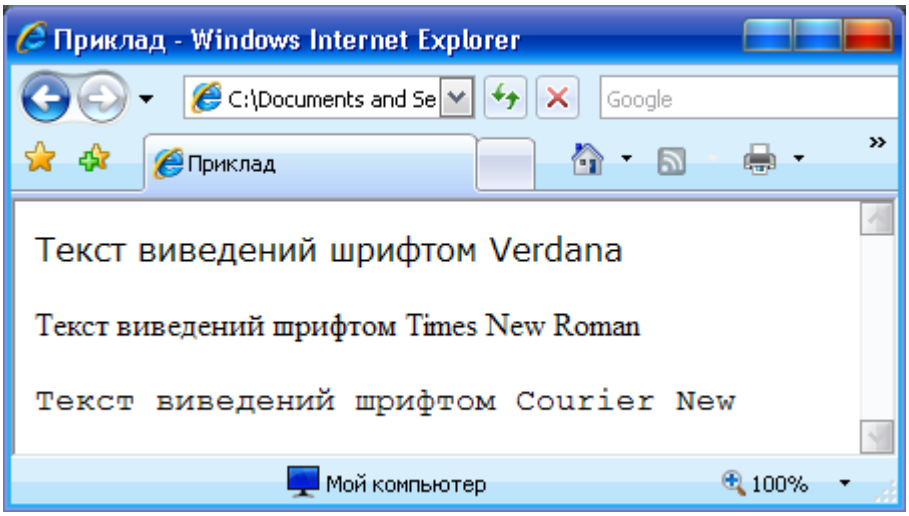

Рис 2.2.6 Використання різних шрифтових гарнітур

font-style

Задає стиль шрифту.

Значення

- `normal` - звичайний;
- `italic` - курсивний;
- `oblique` - похилий.

Курсивний варіант шрифту повинен бути встановлений на комп'ютері по замовчуванню, інакше він буде емулюватися простим нахилом наявного шрифту, тобто `italic` буде замінятися `oblique`.

За допомогою даної властивості можна прибрати нахил тексту всередині елементів `<I>` і `<EM>`, так що можна робити абсолютно різні види виділень.

Приклад

```
P {
  font-family: Verdana, Tahoma, sans-serif;
  font-style: italic
}
...
<P> Текст виведений курсивний шрифтом Verdana
</P>
```

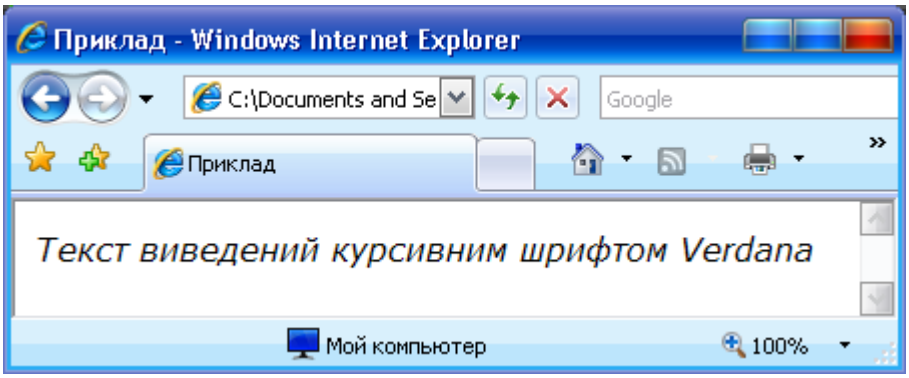

Рис 2.2.7 Використання курсивного шрифта

font-variant

Дозволяє вибирати з двох варіантів виведення *шрифту: звичайними або малими прописними літерами*.

Значення:

- `normal` – звичайний шрифт;
- `small-caps` – капітель- шрифт, в якому всі малі літери замінені на малі прописні. Малі прописні відрізняються від звичайних прописних дещо зміненими пропорціями і дещо зменшеним розміром.

Взагалі малими прописними можна виводити заголовки, але це вже залежить від дизайну сайту.

Приклад

```
P {
```

```
  font-variant: small-caps
}
...
<P> Текст виведений малими прописними літерами
</P>
```

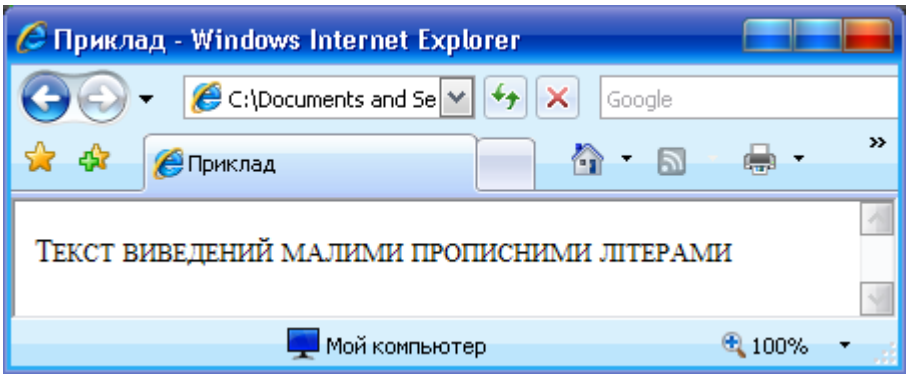

Рис 2.2.8 Використання капітелі

font-weight

Визначає насиченість шрифту.

Існує чисельний ряд для насиченості шрифту: 100, 200, 300, 400, 500, 600, 700. 800, 900. Відразу скажу, що в браузерах такою багатою градації немає, так що доводиться задовольнятися малим. Крім того, двом числах даного ряду відповідають ключові слова:

- 400 - `normal` (нормальний);
- 700 - `bold` (напівжирний).

Тому правила

```
P{
  font-weight: 700
}
```

і

```
P {
  font-weight: bold
}
```

еквівалентні.

Є ще два ключових слова, які задають насиченість відносно батьківського елемента:

- `lighter` - насиченість шрифту буде менша, ніж у батьківського елемента;
- `bolder` - насиченість шрифту буде більша, ніж у батьківського елемента.

Приклад

```
P {
  font-weight: bold
}
...
<P> Текст виведений напівжирним шрифтом</P>
```

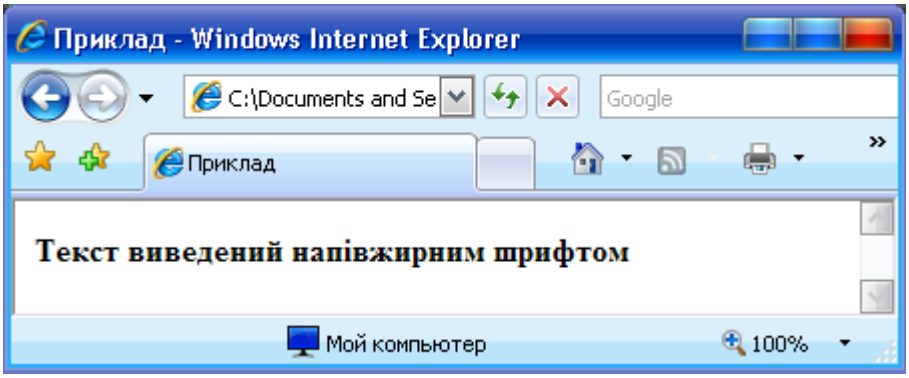

Рис 2.2.9 Використання напівжирного шрифту

font-size

Визначає розмір (кегль) шрифту.

Існує безліч способів задати розмір шрифту. Перший - за допомогою ключових слів:

- `xx-small`;
- `x-small`;
- `small`;
- `medium`;
- `large`;
- `x-large`;
- `xx-large`.

Якій величині відповідає кожне ключове слово - залежить від браузера і від операційної системи. Немає однозначного стандарту, і з цієї причини використовувати ключові слова досить складно. Наприклад, в Internet Explorer 5.0 ключове слово medium відповідає розміру 13,5 пунктів

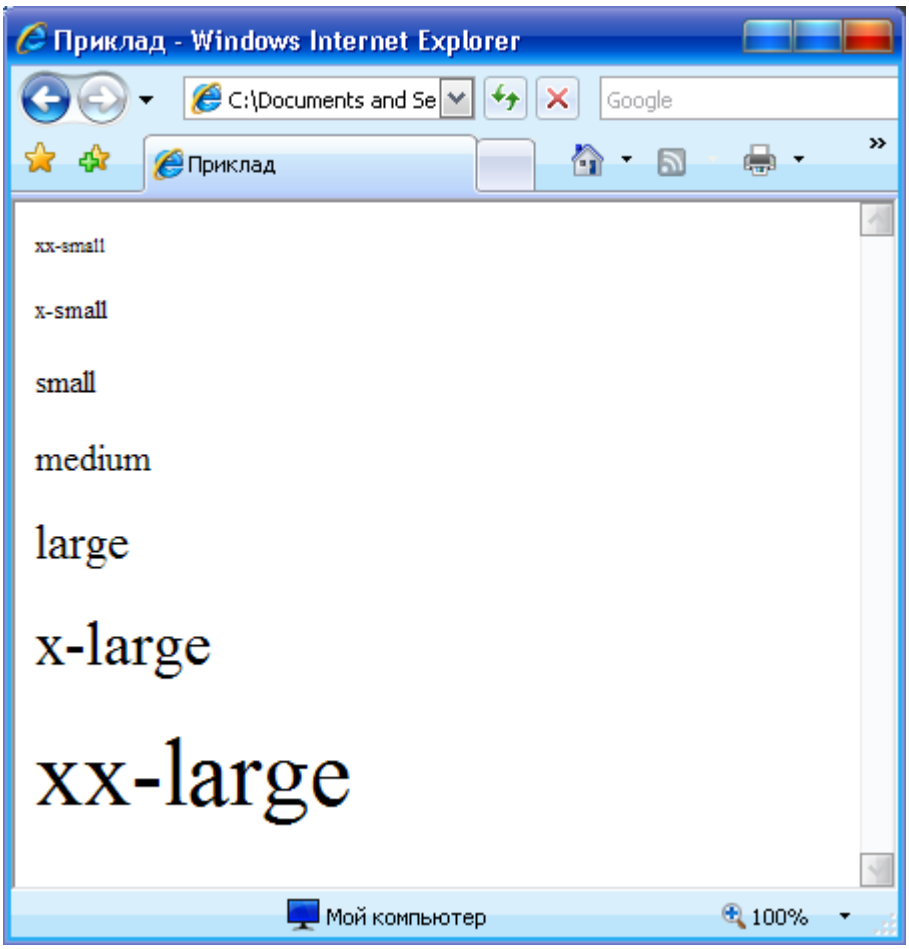

Рис 2.2.10 Ключові слова для розміру шрифтів

Крім того, є два ключових слова, які дозволяють вказувати розмір шрифту відносно батьківського елементу:

- `smaller` - шрифт буде менше, ніж у батьківського елементу;
- `larger` - шрифт буде більше, ніж у батьківського елементу.

Окрім ключових слів можна задавати розмір шрифту в будь-яких доступних одиницях довжини: пунктах, дюймах, пікселях, сантиметрах і міліметрах.

Розмір шрифту, заданого за допомогою ключових слів CSS, співпадає з розміром шрифту, заданого за допомогою HTML. Треба відзначити, що в HTML базовим є розмір 3, тоді як в CSS базовим є значення `medium`.

Приклад

```
P.fs1 {
  font-size: 10px
}
P.fs2 {
  font-size: 15px
}
```

```
P.fs3 {
  font-size: 25px
}
...
<P CLASS="fs1"> Шрифт розміром 10 пікселів</P>
<P CLASS="fs2"> Шрифт розміром 15 пікселів</P>
<P CLASS="fs3"> Шрифт розміром 25 пікселів</P>
```

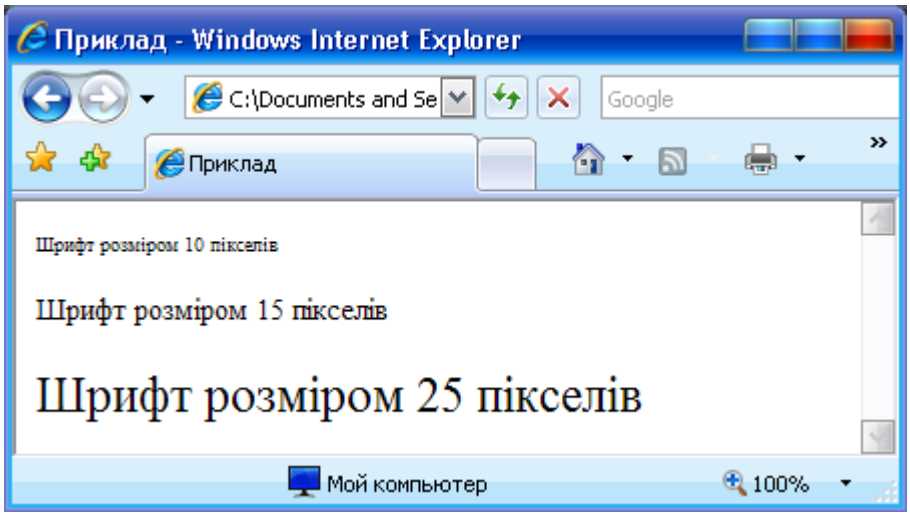

Рис 2.2.11 Використання різних розмірів шрифту

Скорочена форма запису властивостей шрифту

Як і у випадку з фоном, для всіх властивостей шрифту існує скорочений запис, в який можна об'єднати всі перераховані вище властивості. Наприклад, наступний набір оголошень

```
P {
  font-style: italic;
  font-variant: small-caps;
  font-size: lem;
  font-family: Verdana, sans-serif
}
```

можна замінити одним:

```
P {
  font; italic small-caps lem Verdana, sans-
serif
}
```

Формат правила `font` наступний:

```
font: [<font-style> || <font-variant> || <font-
weight>]? <font-size>[<line-height>]? <font-
```

```
faice>
```

Квадратні дужки [] показують групування, а знак питання після такої групи означає, що властивості з цієї групи є необов'язковою. Отже, необов'язковими в даному випадку є властивості font-style, font-variant, font-weight, line-height. Таким чином, скороченим записом стилів для шрифту повинні бути обов'язково вказані розмір і сімейство шрифта:

```
P {
  font: lem Verdana
}
```

Не можна користуватися властивістю font і вказувати тільки розмір або тільки гарнітуру шрифту. Це означає, що наступні правила не коректні:

```
P {
  font: Verdana, sans-serif
}
P {
  font: lem
}
```

Про властивість пізніше line-height

2.2.3Властивості тексту

Інформація на HTML-сторінках зазвичай представлена в текстовому вигляді, так що контроль над текстовими блоками надзвичайно важливий для зручного представлення інформації. У HTML текст можна вирівнювати по горизонталі і вертикалі за допомогою атрибутів ALIGN І VALIGN, а також застосовувати деякі елементи оформлення, такі як перекреслений і підкреслений текст. У HTML не можна змінювати трекінг, відстань між словами, висоту рядка і абзацний відступ, але все це можна робити за допомогою каскадних таблиць стилів.

word-spacing

Дозволяє *встановлювати інтервали між словами*.

Будь-яке значення з розмірністю довжини, як позитивне, так і негативне. При негативному значенні слова можуть накладатися одне на одного, так що при його використанні треба бути обережним.

Приклад

```
P.ws1 {
  word-spacing: -5px
}
P.ws2 {
  word-spacing: 0
}
P.ws3 {
  word-spacing: 3em
}
...
<P CLASS="ws1"> Негативне значення інтервалу
між словами -5px</P>
<P CLASS="ws2"> Стандартне значенням інтервалу
між словами</P>
<P CLASS="ws3"> Інтервал між словами 3em</P>
```

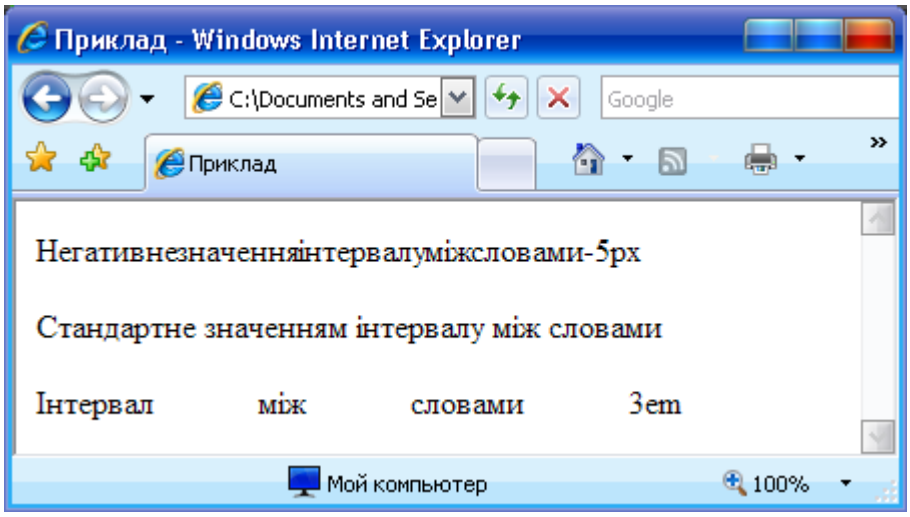

Рис 2.2.12 Різні значення інтервалу між словами

white-space

Параметр `white-space` встановлює, як відображати пробіли між словами. У звичайних умовах будь-яку кількість пробілів в коді HTML показується на веб-сторінці як один. Винятком є тег `<PRE>`, текст що помістили в цей контейнер виводиться з усіма пробілами, як було відформатовано користувачем. Таким чином, `white-space` імітує роботу тега `<PRE>`, але на відміну від нього не змінює шрифт на моноширинний.

Значення

- normal - текст у вікні браузера виводиться як зазвичай, переноси рядків встановлюються автоматично.
- nowrap - перенесення рядків в коді HTML ігноруються, весь текст відображається одним рядком, разом з тим, додавання тега
 переносить текст на новий рядок.
- pre - текст показується з урахуванням всіх пробілів і переносів, як вони були додані розробником в коді HTML. Якщо рядок виходить дуже довгим і не поміщається у вікні браузера, то буде додана горизонтальна лінія прокручування.

letter-spacing

Дозволяє *встановлювати величину трекінгу* (інакше кажучи, відстань між буквами)..

Будь-яке значення з розмірністю довжини, як позитивне, так і негативне. При негативному значенні букви можуть накладатися одна на одну, так що при його використанні треба бути обережним.

Приклад

```
P.ls1 {
  letter-spacing: -2px
}
P.ls2 {
  letter-spacing: 0
}
P.ls3 {
  letter-spacing: 1em
}
...
<P CLASS="ls1">Негативне значення інтервалу між
буквами -2px</P>
<P CLASS="ls2">Стандартне значенням інтервалу
між буквами</P>
<P CLASS="ls3">Інтервал між буквами 1em</P>
```

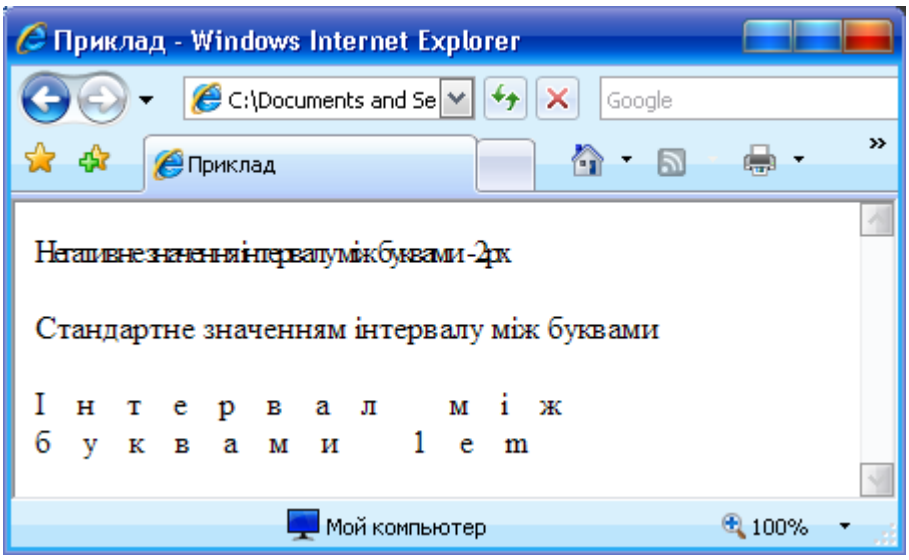

Рис 2.2.13 Різні значення інтервалу між буквами

text-decoration

Дозволяє *оформляти текст*.

Значення:

- `none` - виводить звичайний текст без всякого оформлення;
- `underline` - виводить підкреслений текст;
- `overline` - виводить "надкреслений" текст;
- `line-through` - виводить перекреслений текст;
- `blink` - виводить мигаючий текст.

Властивість `text-decoration` зазвичай використовують для того, щоб зробити посилання не підкресленим. Беремо селектор посилання з псевдокласcом і прописуємо йому оголошення `text-decoration: none`:

```
A: link, A: visited, A: active {
  text-decoration: none
}
```

Приклад

```
P.td1 {
  text-decoration: none
}
P.td2 {
  text-decoration: underline
}
P.td3 {
  text-decoration: overline
```

```
}
P.td4 {
  text-decoration: line-through
}
P {
  text-decoration: none
}
...
<P CLASS="td1"> Текст без оформлення </P>
<P CLASS="td2"> Підкреслений текст </P>
<P CLASS="td3"> Надкреслений текст </P>
<P CLASS="td4"> Перекреслений текст </P>
```

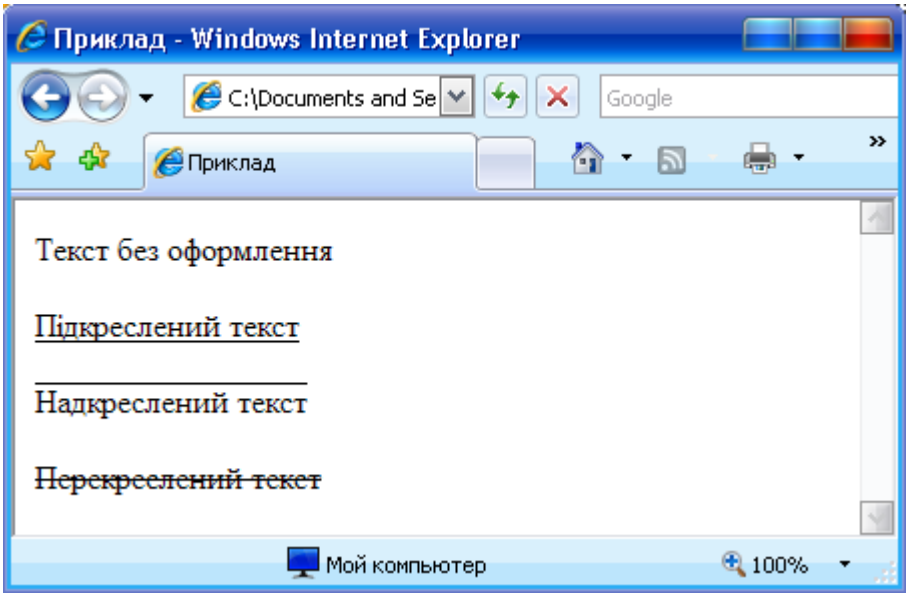

Рис 2.2.14 Різні значення оформлення тексту

vertical-align

Дозволяє встановлювати *вертикальне вирівнювання тексту*.

Вертикальне вирівнювання може бути відносно батьківського елементу і відносно лінії форматування. При вирівнюванні відносно батьківського елементу можуть бути наступні значення:

- `baseline` - суміщає середню лінію елемента і середню лінією батьківського елемента;
- `sub` - робить елемент підрядковим (аналог тега <SUB>);
- `super` - робить елемент надрядкових (аналог тега <SUP>);
- `text-top` - суміщає верхній край елемента з верхнім краєм шрифту батьківського елемента;

- middle - суміщає середню лінію елемента (звичайно зображення) з рівнем "середній рівень плюс половина висоти x-height батьківського елемента";
- text-bottom - суміщає нижній край елемента з нижнім краєм шрифту батьківського елемента.

При вирівнюванні відносно лінії форматування можуть бути наступні значення:

- bottom - суміщає нижній край елемента з нижнім краєм самого низького елемента в лінії;
- top - суміщає верхній край елемента з верхнім краєм самого високого елемента в лінії.

Крім того, як значення можна використовувати процентні співвідношення.

Приклад

```
B{
  font-weight: normal;
  height: 30px;
}
.va1 {
  vertical-align: baseline
}
.va2 {
  vertical-align: sub
}
.va3 {
  vertical-align: super
}
.va4 {
  vertical-align: text-top
}
.va5 {
  vertical-align: middle
}
.va6 {
  vertical-align: text-bottom
}
.va7 {
  vertical-align: bottom
```

```
}
.va8 {
  vertical-align: top
}
...
<B CLASS="va1">baseline</B>
<B CLASS="va2">sub</B>
<B CLASS="va3">super</B>
<B CLASS="va4">text-top</B>
<B CLASS="va5">middle</B>
<B CLASS="va6">text-bottom</B>
<B CLASS="va7">bottom</B>
<B CLASS="va8">top</B>
```

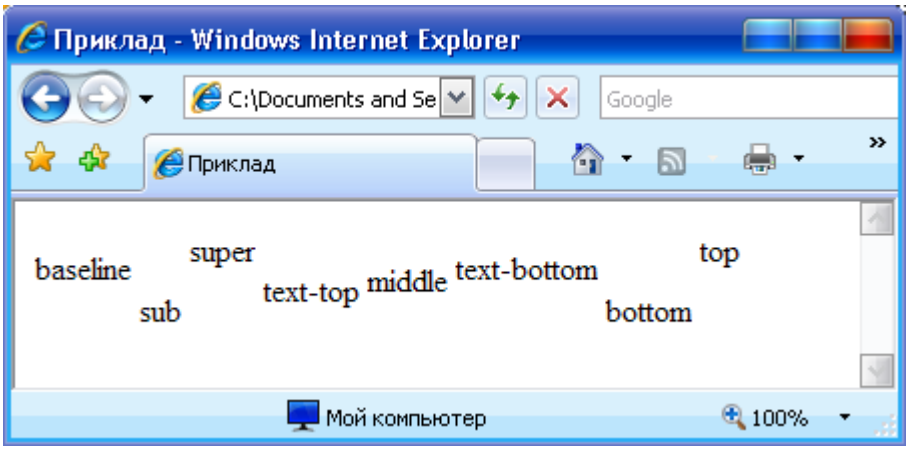

Рис 2.2.15 Різні значення вертикального вирівнювання тексту

text-align

Дозволяє встановлювати *горизонтальне вирівнювання тексту* всередині елемента (виключку).

Значення:

- `left` - вирівнювання по лівому краю елемента;
- `center` - вирівнювання по центру елемента;
- `right` - вирівнювання по правому краю елемента;
- `justify` - вирівнювання по ширині елемента.

Приклад

```
P {
  font-size: 17px
}
P.ta1 {
```

```
  text-align: left
}
P.ta2 {
  text-align: center
}
P.ta3 {
  text-align: right
}
P.ta4 {
  text-align: justify
}
...
<P CLASS="ta1"> Абзац тексту з горизонтальним
вирівнюванням по лівому краю</P>
<P CLASS="ta2"> Абзац тексту з горизонтальним
вирівнюванням по центру</P>
<P CLASS="ta3"> Абзац тексту з горизонтальним
вирівнюванням по правому краю</P>
<P CLASS="ta4"> Абзац тексту з горизонтальним
вирівнюванням по ширині елементу</P>
```

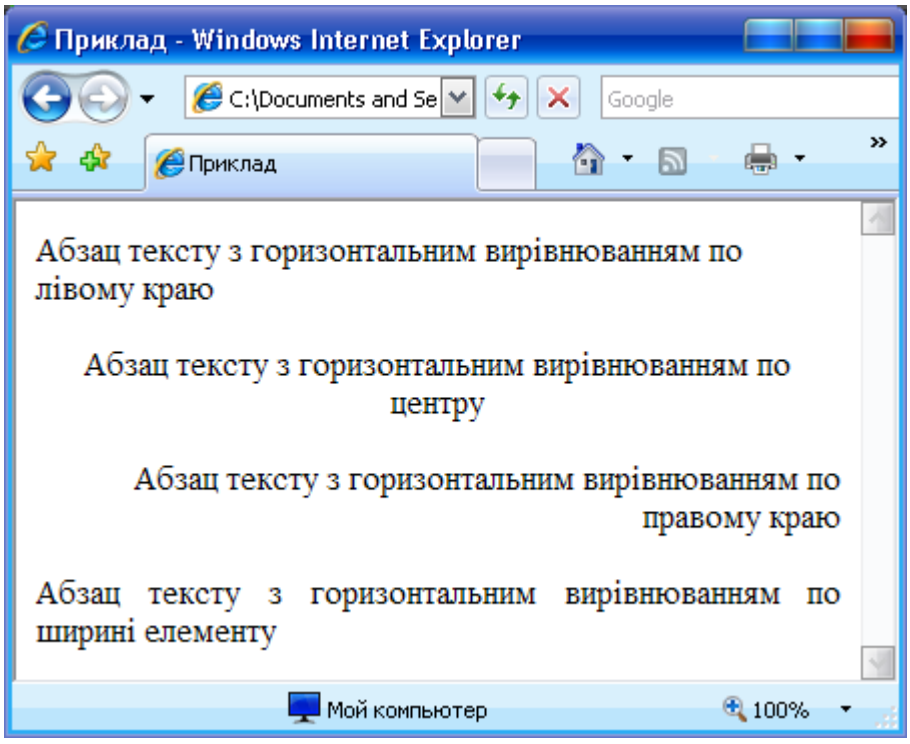

Рис 2.2.16 Різні значення вертикального вирівнювання

text-transform

Дозволяє трансформувати наявний текст. Значення:

- `capitalize` - перша буква кожного слова в елементі стане великою;
- `uppercase` - всі слова в елементі будуть виведені прописними літерами;
- `lowercase` - всі слова в елементі будуть виведені малими літерами;
- `none` - текст в елементі виводиться без трансформацій (значення за замовчуванням).

Приклад

```
P.tt1 {
  text-transform: none
}
P.tt2 {
  text-transform: capitalize
}
P.tt3 {
  text-transform: uppercase
}
P.tt4 {
  text-transform: lowercase
}
...
<P CLASS="tt1"> Текст в елементі виводиться без
трансформацій </P>
<P CLASS="tt2"> Перша буква кожного слова в
елементі стане великою</P>
<P CLASS="tt3"> Всі слова в елементі будуть
виведені прописними літерами</P>
<P CLASS="tt4"> Всі слова в елементі будуть
виведені малими літерами</P>
```

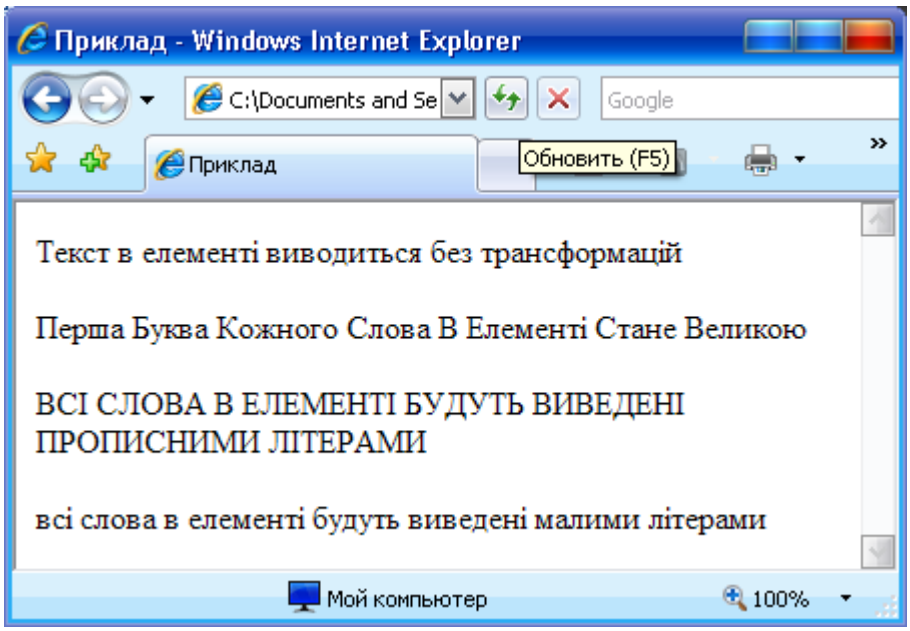

Рис 2.2.17 Різні значення трансформації тексту

text-indent

Дозволяє встановлювати величину відступу перед першим рядком абзацу.

Будь-яке значення з розмірністю довжини, як позитивне, так і негативне. Крім того, можна використовувати відсотки, які обчислюються щодо ширини батьківського елементу.

Приклад

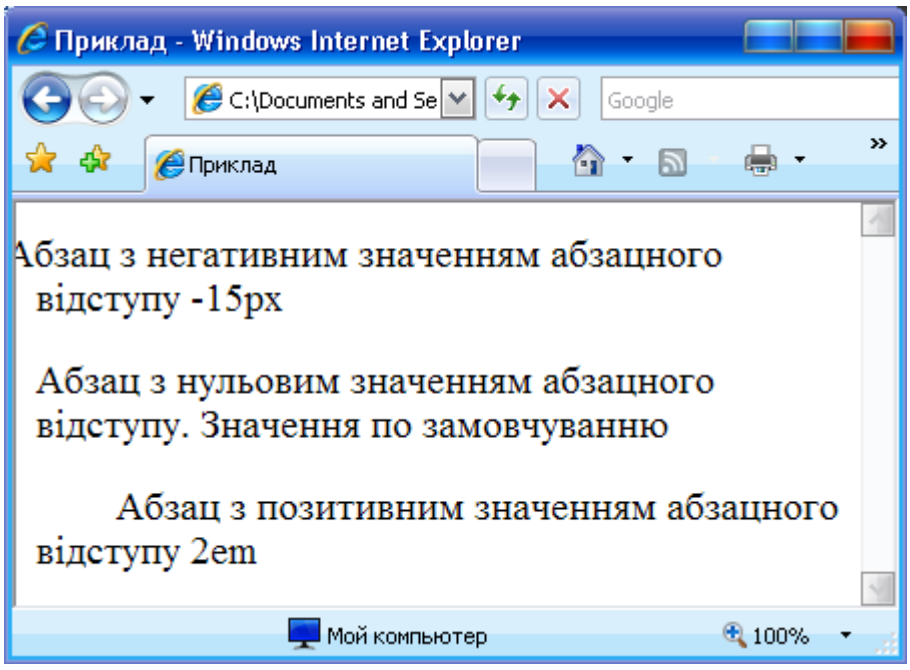

Рис 2.2.18 Різні значення абзацного відступу

```
P {
  font-size: 20px
}
```

```
P.ti1 {
  text-indent: -15px
}
P.ti2 {
  text-indent: 0
}
P.ti3 {
  text-indent: 2em
}
...
<P CLASS="ti1"> Абзац з негативним значенням
абзацного відступу -15px </P>
<P CLASS="ti2"> Абзац з нульовим значенням
абзацного відступу. Значення по замовчуванню
</P>
<P CLASS="ti3"> Абзац з позитивним значенням
абзацного відступу 2em</P>
```

line-height

Дозволяє встановлювати величину відстані між рядками (інтерльяж).

Будь-яке значення з розмірністю довжини, але тільки позитивне. Можна використовувати відсотки, які обчислюються відносно розміру шрифту даного елемента. Крім того, можна використовувати безрозмірні числа - у цьому випадку висота рядка визначається як висота шрифту, помножена на дане число. До речі кажучи, безрозмірні числа є оптимальним варіантом для встановлення висоти рядка.

Приклад

```
P {
  font-size: 20px
}
P.lh1 {
  line-height: 15px
}
P.lh2 {
  line-height: 25%
}
P.lh3 {
```

```
  line-height: 2
}
...
<P CLASS="lh1"> Задання міжрядкового інтервалу
різними способами</P>
<P CLASS="lh2"> Задання міжрядкового інтервалу
різними способами </P>
<P CLASS="lh3"> Задання міжрядкового інтервалу
різними способами</P>
```

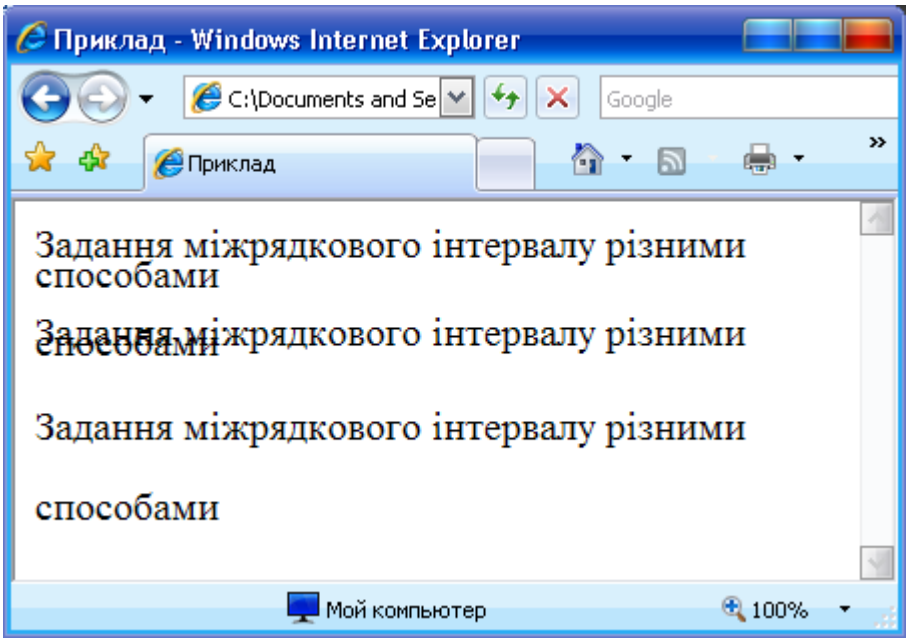

Рис 2.2.19 Різні способи задання міжрядкового інтервалу

2.3 Блокова модель

Блок складається з так званої тематичної частини, яка може бути оточена відступами (`padding`), рамками (`border`) і полями (`margin`). Причому і відступи, і рамки, і поля можуть бути різними з чотирьох сторін (зверху, справа, знизу і зліва). Кожен блок може мати фіксовану ширину (`width`) і висоту (`height`)

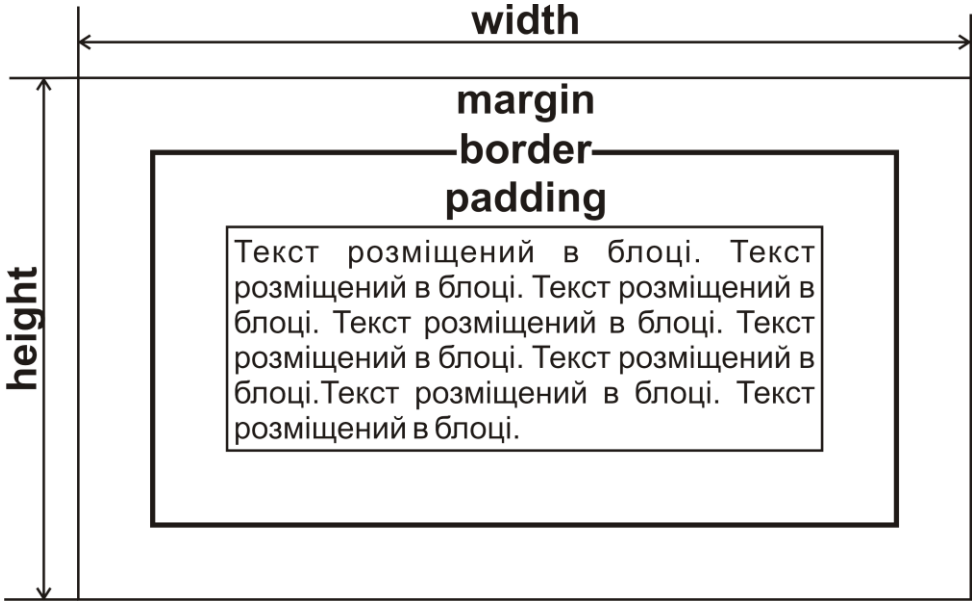

Рис 2.3.1 Структура блоку в CSS

Спочатку розглянемо властивості відступів, полів і рамок.

2.3.1Властивості відступів, полів і рамок

margin-top, margin-right, margin-bottom і margin-left

Дозволяють *встановити величину верхнього, правого, нижнього або лівого поля елемента* відповідно.

Значення:

- будь-яке значення з розмірністю довжини, як позитивне, так і негативне;
- відсотки, які обчислюються відносно ширини батьківського елементу;
- ключове слово `auto`.

Приклад

```
IMG {
  margin-top: 15px;
  margin-bottom: 0px;
  margin-left: 20px;
```

```
  margin-right: 5px;
}
...
<P>Блок складається з так званої тематичної
частини, яка може бути оточена <img
src="rys.gif">відступами (padding), рамками
(border) і полями (margin).</P>
```

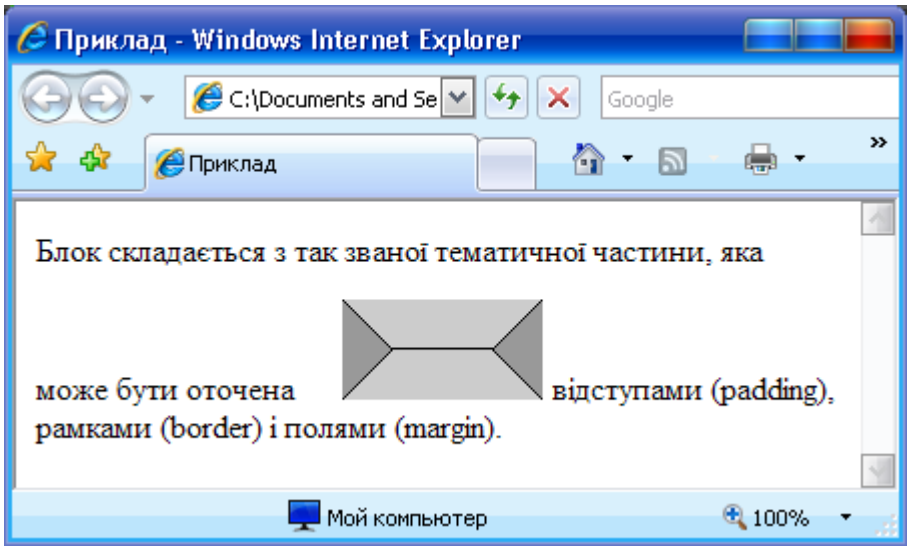

Рис 2.3.2 Приклад блоку з різними полями

margin

Дозволяє у скороченій формі задавати величини всіх полів елемента. Значення абсолютно аналогічні можливим значенням у попередніх властивостях.

Формат у даної властивості досить різноманітний. Можна встановити всі чотири значення, можна тільки одне, тоді це значення буде застосовано до всіх чотирьох полів. Можна два значення, тоді перше з них буде застосовано до верхнього і нижнього полів, а друге - до лівого і правого полів. Можна три. тоді перше значення буде застосовано до верхнього поля, друге до правого і лівого полів, а третє - до нижнього. Якщо будь-яке значення пропущено, то воно береться рівним значенню з протилежного боку.

Таблиця 2.3.1 Приклади скороченою запису стилів для полів елемента <p>

Повна форма	Скорочена форма

P {
margin-top: 5px;
margin-bottom:
5px;
	P {
	margin: 5px;
	}

margin-left: 5px; margin-right: 5px; }	

P {
margin-top: 5px;
margin-bottom:
5px;
margin-left:
10px;
margin-right:
10px;
}
	P {
	margin: 5px 10px;
	}

padding-top, padding-right, padding-bottom і padding-left

Дозволяють встановити *величину верхнього, правого, нижнього або лівого відступу* елемента відповідно.

Будь-яке значення з розмірністю довжини, але тільки позитивне, а також відсотки, які обчислюються відносно ширини батьківського елемента.

Приклад

```
P {
  padding-top: 0px;
  padding-bottom: 30px;
  padding-left: 20px;
  padding-right: 15px;
  border: 1px solid black;
}
...
<P>Блок складається з так званої тематичної
частини, яка може бути оточена відступами
(padding), рамками (border) і полями
(margin).</P>
```

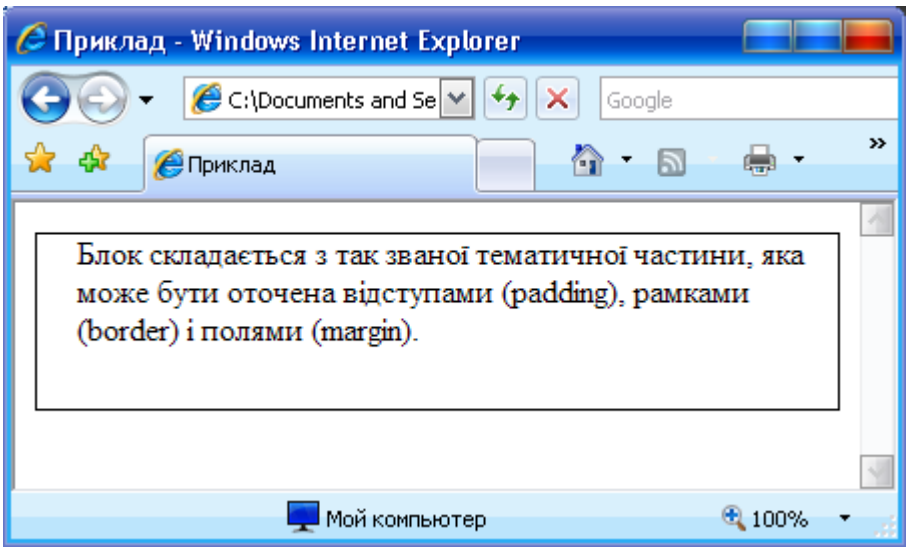

Рис 2.3.3 Приклад блоку з різними відступами

padding

Дозволяє у скороченій формі задавати величини всіх відступів елемента. За своїм форматом цілком аналогічно властивості `margin`.

border-top-width, border-right-width, border-bottom-width і border-left-width

Дозволяють встановлювати *ширину верхньої, правої, нижньої або лівої рамки* відповідно.

Будь-яке позитивне значення з розмірністю довжини. Крім того, можна використовувати ключові слова:

- `thin` - тонка рамка;
- `medium` - середня по ширині рамка;
- `thick` - широка рамка.

Чисельні значення ключових слів залежать від конкретного браузера, так що краще вказувати ширину рамки безпосередньо в одиницях довжини.

Приклад

```
P {
  border-style: solid;
  border-color: black;
  border-top-width: 1px;
  border-bottom-width: 0;
  border-left-width: thick;
  border-right-width: 3px;
}
...
```

```
<P>Блок складається з так званої тематичної
частини, яка може бути оточена відступами
(padding), рамками (border) і полями
(margin).</P>
```

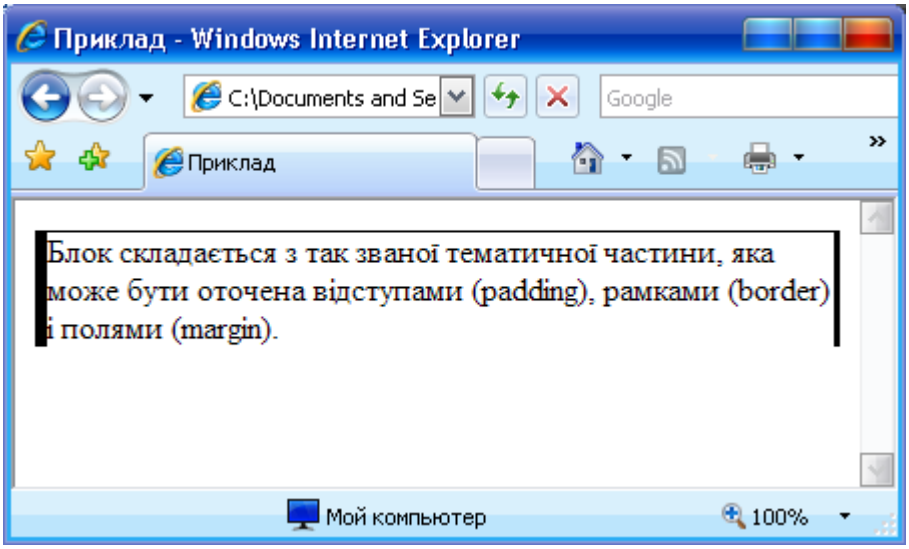

Рис 2.3.4 Приклад рамки блоку з різною товщиною

border-width

Дозволяє у скороченій формі задавати ширину всіх рамок. Значення аналогічні можливим значенням у нескороченій формі запису ширини рамок. Формат такий же, як у властивостей `margin` і `padding`.

border-color

Дозволяє *встановлювати колір рамок*. Будь-які колірні значення. Формат такий же, як у властивості `border-width`

Зауважимо, що якщо колір рамки не вказаний явно за допомогою властивості `border-color`, то рамка буде мати колір, встановлений за допомогою властивості `color`.

Приклад

```
P {
  border-style: solid;
  border-width: 5px;
  border-top-color: yellow;
  border-bottom-color: red;
  border-left-color: blue;
  border-right-color: green;
}
```

```
...
<P>Блок складається з так званої тематичної
частини, яка може бути оточена відступами
(padding), рамками (border) і полями
(margin).</P>
```

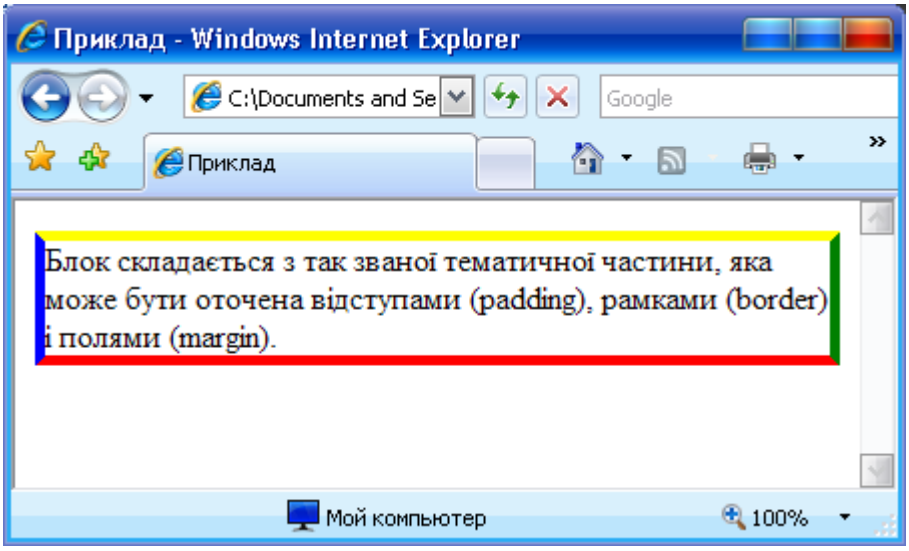

Рис 2.3.5 Приклад рамки блоку з різним кольором

border-style

Дозволяє встановлювати стиль рамки. Формат такий же, як у властивості `border-width` і `border-color`

Значення:

- `none` - рамки відсутні зовсім (це значенням по замовчуванню);
- `dotted` - рамка у вигляді пунктирної лінії;
- `dashed` - рамка у вигляді переривчастої лінії;
- `solid` - рамка у вигляді суцільної лінії;
- `double` - рамка у вигляді подвійної лінії (ширина ліній і відстані між ними напряму залежать від значення властивості `border-width`);
- `groove` - тривимірна заглиблена рамка;
- `ridge` - тривимірна опукла рамка;
- `inset` - рамка у вигляді заглиблення;
- `outset` - рамка у вигляді опуклості.

Як бачимо, можливих значень багато. Однак багато з них не підтримуються деякими браузерами.

Приклад

```
P {
```

```
  border-width: 7px;
  border-color: black;
}
P.dotted {
  border-style: dotted
}
P.dashed {
  border-style: dashed
}
P.solid {
  border-style: solid
}
P.double {
  border-style: double
}
P.groove {
  border-style: groove
}
P.ridge {
  border-style: ridge
}
P.inset {
  border-style: inset
}
P.outset {
  border-style: outset
}
...
<P CLASS="dotted">dotted</P>
<P CLASS="dashed">dashed</P>
<P CLASS="solid">solid</P>
<P CLASS="double">double</P>
<P CLASS="groove">groove</P>
<P CLASS="ridge">ridge</P>
<P CLASS="inset">inset</P>
<P CLASS="outset">outset</P>
```

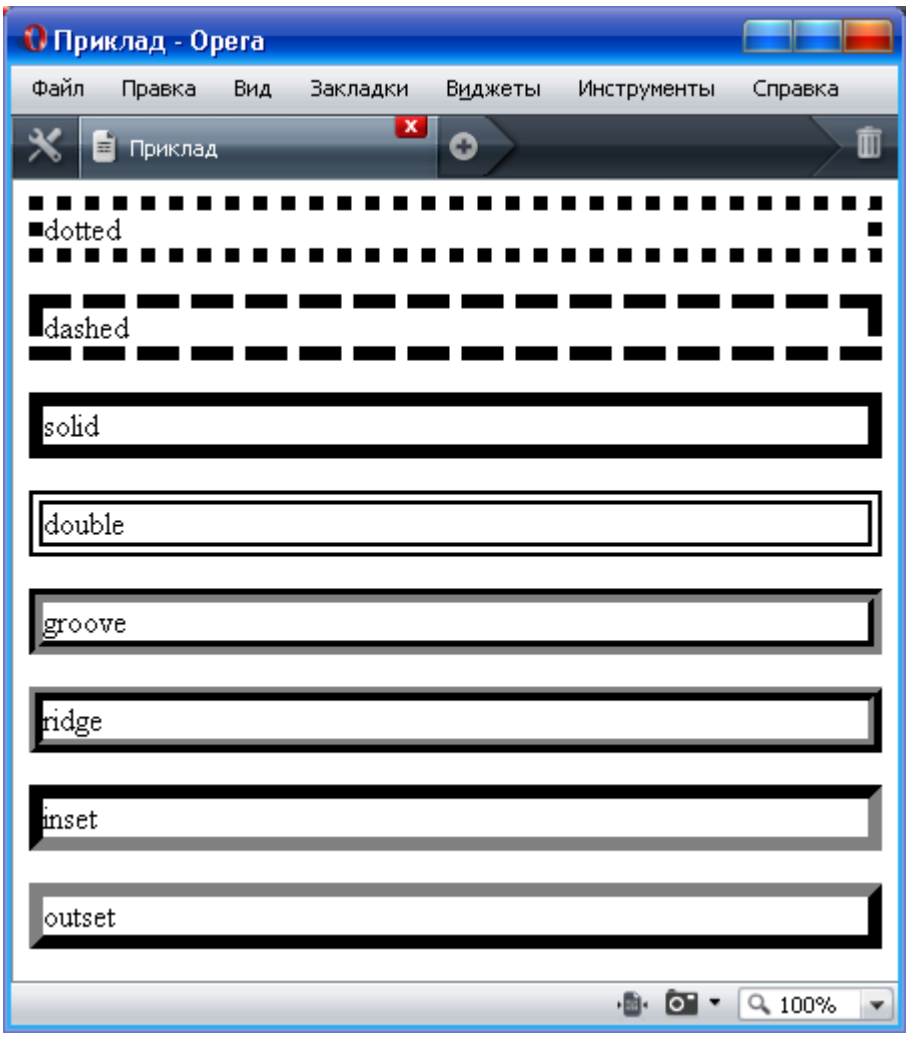

Рис 2.3.6 Приклад різних стилів рамки блоку

border-top, border-right, border-bottom і border-left

Дозволяють встановити властивості верхньої, правої, нижньої або лівої рамки відповідно у скороченій формі запису.

Формат властивостей наступний:

```
border-top: <border-top-width> || <border-
style> || <border-color>
```

Таблиця 2.3.2 Приклади скороченого запису стилю для верхньої рамки елемента

Повна форма	Скорочена форма

P {
border-top-width: 2px;
border-style: solid;
border-color: #000
	P {
	border-top: 2px solid #000
	}

}	

border

Дозволяє встановити властивості рамки у скороченій формі запису.

Основна відмінність від властивостей `margin` і `padding` в тому, що не можна ставити різні значення для рамок з різних сторін, тобто встановлене значення застосовуються до всіх чотирьох сторін елементу.

Таблиця 2.3.3 Приклади скороченого запису стилю для рамки елемента

Повна форма	Скорочена форма

P {
border-top: 2px solid #CCC;
border-right : 2px solid #CCC;
border-bottom: 2px solid #CCC;
border-left : 2px solid #CCC
}
	P {
	border: 2px solid #CCC
	}

border-collapse

Дана властивість використовується для таблиці або табличних блоків.

Встановлює, як відображати межі навколо клітинок таблиці. Цей параметр відіграє роль, коли для клітинок встановлена рамка, тоді в місці стику клітинок вийде лінія подвійної товщини. Додавання значення `collapse` змушує браузер аналізувати подібні місця в таблиці і прибирати в ній подвійні лінії. При цьому між клітинками залишається тільки одна межа, що одночасно належить обом клітинкам. Те ж правило дотримується і для зовнішніх меж, коли навколо самої таблиці додається рамка.

Значення

- `collapse` - лінія між осередками відображається тільки одна.
- `separate` - навколо кожної клітинки відображається своя власна рамка, в місцях зіткнення осередків показуються відразу дві лінії.

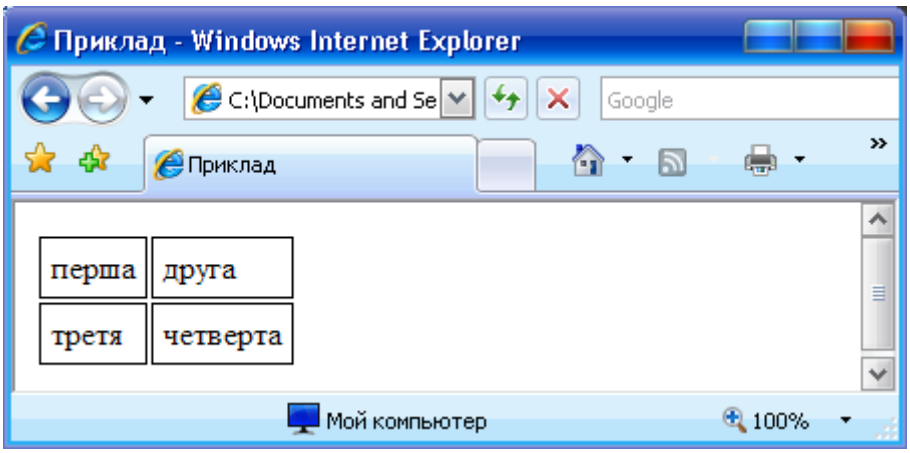

Рис 2.3.7 Вигляд таблиці з оголошенням border-collapse: separate

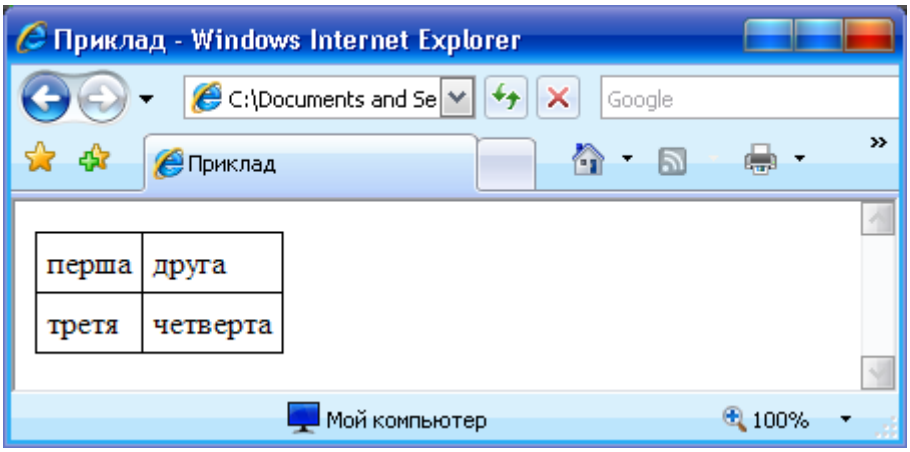

Рис 2.3.8 Вигляд таблиці з оголошенням border-collapse: collapse

2.3.2Поняття блоку

Блок складається з контентної частини, яка може бути оточена відступами, рамками і полями. Кожен блок може мати фіксовану ширину (width) і висоту (height). Як видно з малюнка, загальна ширина контейнера складається з ширини окремих компонентів:

```
Ширина контейнера =margin-left +border-left +padding-left +width+padding-right +border-right +margin-left
```

По замовчуванню і поля, і відступи, і рамки відсутні, тобто мають величину, рівну нулю. У цьому випадку ширина контентної частини рівна ширині всього блоку. Якщо ж і властивість `width` не встановлена, то ширина контентної частини визначається контейнером. Будь-який елемент має контейнер.

Визначення

Контейнер елементу - це елемент блокового рівня, що є найближчим предком даного елементу.

Наприклад:

```
<BODY>
  <P> Елемент P є предком елементу BODY </P>
</BODY>
```

В даному випадку для елементу <P> контейнером є елемент <BODY>. Якщо не задавати явно ширину блоку, утвореного елементом <P>, і не задавати поля, відступи і рамки, то ширина блоку, утвореного елементом <P>, визначатиметься шириною елементу <BODY>. Відомо що ширина блоку, утвореного елементом <BODY>, рівна розміру вікна браузера. Тоді виходить, що якщо для <BODY> всі поля встановлені в нуль, то ширина блоку елементу <P> буде рівна ширині вікна браузера.

Для яснішого представлення суті контейнера приведемо ще один приклад з вкладеними блоками. Спочатку створимо перший блок, який буде мати чорну рамку завтовшки один піксель і ширину 40% від ширини вікна браузера. Крім того, в даному блоці правий і лівий відступи будуть рівні 2, а верхній і нижній - 1. Код, який створює такий блок, буде наступним:

```
#blockl {
  border:lpx solid #000;
  padding:lem 2em;
  width:40%
}
...
<DIV id-"blockl">
  Це контент блоку
</DIV>
```

Вигляд цього коду в браузері на Рис 2.3.9

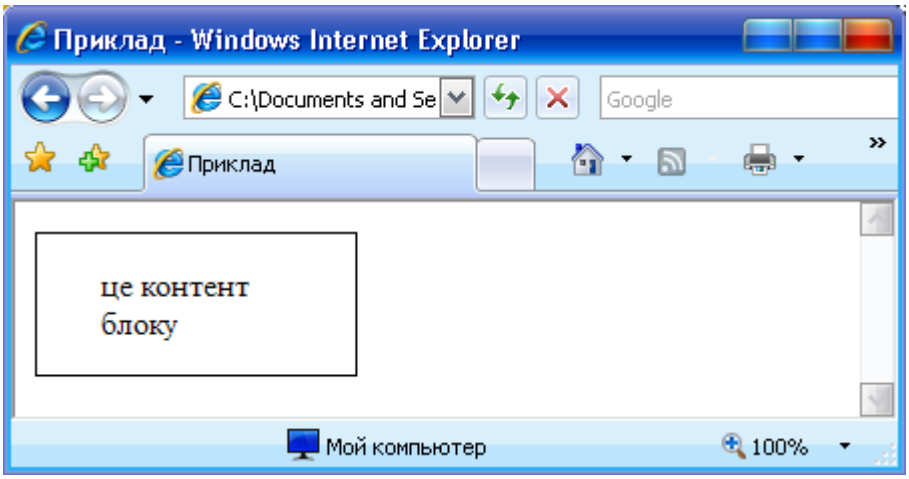

Рис 2.3.9 Простий блок з рамкою

Створимо другий блок, вкладений в перший. Цей блок буде без відступів, з рамкою завтовшки два пікселя, причому ширина блоку явно не задана:

```
#block1 {
  border: 1px solid #000;
  padding: 1em 2em;
  width:40%
}
#block2 {
  border: 2px solid #000
}
<DIV id="block1">
  Це контент першого блоку
  <DIV id="block2">
    Це другий блок, вкладений в перший
  </DIV>
</DIV>
```

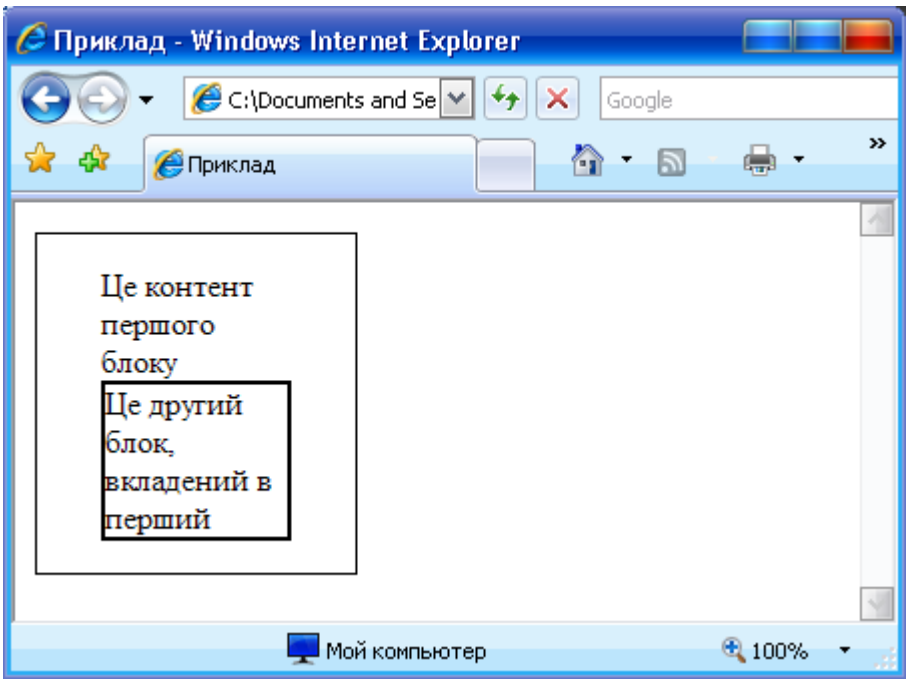

Рис 2.3.10 Вкладені блоки і їх розміщення

Рамка першого блоку має товщину один піксель. Правий і лівий відступи мають ширину `2em`., а нижній і верхній – `1em`. Вмістом першого блоку є фраза `"це контент першого блоку"` і другий блок. У другому блоці відступів немає, тому що в таблиці стилів явно вони не вказані, а по замовчуванню відступи рівні нулю, з цієї причини контент другого блоку впритул прилягає до рамки. Зверніть увагу на те, що ширина другого блоку не вказана, а контейнером його є перший блок. Тому ширина другого блоку рівна ширині контентної частини першого блоку.

2.3.3Види блоків

Введемо декілька визначень.

Визначення

Елемент називатимемо блоковим, якщо він візуально форматується у вигляді структурної одиниці, що є абзацом, заголовком або іншою структурною одиницею.

У HTML такі елементи зазвичай починаються і закінчуються символом переходу рядка (наприклад, елементи `<P>`, `<H1>`, `<TABLE>`). Отже, блокові елементи не можуть розташовуватися в одному рядку.

Визначення

Елемент називатимемо рядковим, якщо він не формує структурних одиниць і виводиться лінійним рядком.

У HTML це звичайно елементи форматування тексту (наприклад, елементи <B>, <EM>, <CODE>). Природно, два рядкові елементи можуть розташовуватися у одному рядку.

Відмітимо, що *блок* і *блоковий елемент* це не одне і те ж. Насправді два вищенаведені визначення однаково відносяться до блоків. Інакше кажучи, блоки бувають *структурними* (породжувані блоковими елементами) і *рядковими* (породжувані рядковими елементами).

Структурні і рядкові блоки повинні розрізнятися між собою тільки тим, що перші утворюють структурні одиниці, а другі – ні. В дереві документів рядкові елементи будуть вкладені в блоки.

2.3.4Ширина і висота блоку.

Нам необхідно створити не просто довільні блоки, а блоки певного розміру із заданою шириною і висотою. У CSS загальна ширина блоку складається з ширини контентної частини і сумарної ширини полів, рамок і відступів. Ширина контентної частини задається властивістю width. Висота контентної частини задається властивістю height.

Для маніпулювання висотою і шириною блоку є ще декілька цікавих властивостей із специфікації CSS-2:

- min-width - задає мінімальну ширину блоку;
- max-width - задає максимальну ширину блоку;
- min-height - задає мінімальну висоту блоку;
- max-height - задає максимальну висоту блоку.

З підтримкою цих властевостей в деяких браузерів є проблеми

2.3.5Типи блоків

У глобальній системі класифікацій блоки бувають тільки двох видів: структурні і рядкові. Проте і ті, та інші мають безліч типів.

У CSS тип блоку визначається властивістю display. Це властивість в специфікації CSS-1 може приймати чотири значення: none, inline, block, listitem. Проте в специфікації CSS-2 значень значно більше, і деякі браузери їх в тому або іншому ступені підтримують. Проте, підтримка ця часто некоректна і неповна.

display:none

Відключає відображення елементу в браузері. У CSS існує властивість visibility, за допомогою якої теж можна приховати елемент, проте різниця між цими властивостями є. Річ у тому, що при

використанні оголошення visibility:hidden елемент просто стає невидимим, але присутній на сторінці і займає певне місце, тоді як при використанні оголошення display:none елемент взагалі не присутній на сторінці, він віддаляється з нормального потоку, згідно якому браузер форматує сторінку.

За допомогою JavaScript можна динамічно показувати і приховувати елементи, так що є можливість організувати нескладне ієрархічне меню, підрозділи якого будуть розгортатися при натисненні на посилання розділу і згортатися назад, якщо на посилання натиснути ще раз. Таке меню значно прискорює переміщення по сайту, та і користувачі до нього звикли, тому що по своїй суті воно нічим не відрізняється від дерева файлів в провіднику ОС Windows.

display:block

Елемент з таким оголошенням форматуватиметься, як структурний блок. Взагалі багато елементів по замовчуванню мають оголошення display:block, наприклад <H1>, <P>, <TABLE>. Використовувати це оголошення доцільно тільки разом з оголошенням display:none. Дійсно, його можна застосувати для відновлення елементу на сторінці, який до цього був прихований за допомогою display:none. Звичайно, можна робити рядкові елементи блоковими, але особливого сенсу в цьому немає.

display:inline

Елемент з таким оголошенням форматуватиметься як рядковий блок. Так само багато елементів по замовчуванню мають оголошення display:inline, наприклад <I>, <CITE>, <EM>, <STRONG>. Використовувати значення inline можна для того, щоб перетворити блокові по замовчуванню елементи у рядкові елементи. Наприклад, можна використовувати заголовки шостого рівня таким чином:

```
BODY {
  font: 1em Verdana,sans-serif
}
P.in {
  display: inline
}
H6 {
```

```
  font: bold lem Verdana;
  display: inline
}
...
<H6> Заголовок: </H6>
<P CLASS="in" > Текст підрозділу починається в
тому ж рядку </P>
```

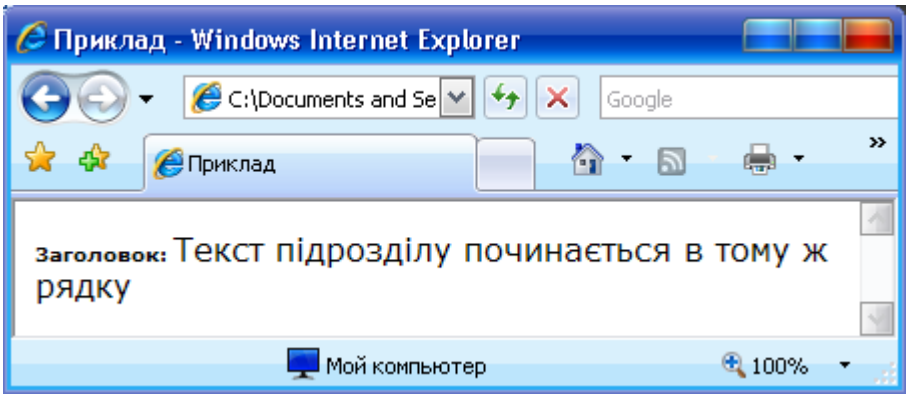

Рис 2.3.11 Заголовок шостого рівня стає рядковим

display:list-item

Робить елемент частиною списку, тобто спочатку до кожного абзацу додається кружечок або будь-який інший маркер, що позначає пункт списку. За допомогою такого оголошення можна будь-який HTML-елемент перетворити на список. Наприклад, абзац:

```
P {
  display:list-item
}
```

Проте ніякої особливої цінності це не має, тому що в HTML є елементи `<UL>`, `<OL>`, `<LI>`, які і задають списки. Отже набагато логічніше робити список саме засобами HTML, а не засобами CSS, Інша справа, що в CSS є декілька властивостей, які значно збільшують контроль над представленням списків.

list-style-type

Дозволяє задавати маркер списку, тобто те, що виводитиметься перед кожним пунктом списку. Може приймати наступні значення:

- `disc` - зафарбований кружечок (значення по замовчуванню);
- `circle` - незафарбований кружечок;
- `square` - зафарбований квадратик;
- `decimal` - арабські цифри (1,2,3 і т.д.):

• lower-roman - римські цифри, що позначаються маленькими буквами (i, ii, iii, iv);
• upper-roman - римські цифри, що позначаються великими буквами (I, II, III, IV);
• lower-alpha - маленькі латинські букви (a, b, c);
• upper-alpha - великі латинські букви (A, B, C);
• none - відключає маркери.

Як маркер можна використовувати будь-який малюнок, і для цього служить ще одна властивість.

list-style-image

Як параметр використовується URL до зображення, яке можна зробити маркером. Наприклад, маркер збережено в файл arrow.gif. Тоді для всіх списків задати маркер у вигляді цієї стрілки можна так:

```
LI {
  list-style-image:url(arrow.gif)
}
```

Є ще властивість.

list-style-position

Дозволяє встановлювати обтікання маркерів. Може приймати два значення: inside і outside (Рис 2.3.12).

```
.in {
  list-style-position: inside
}
.ou {
  list-style-position:outside
}
...
<UL>
<LI CLASS="in"> Приклад різноманітного
обтікання маркерів списків, що змінює
властивість list-style-position
<LI CLASS="ou"> Приклад різноманітного
обтікання маркерів списків, що змінює
властивість list-style-position
</UL>
```

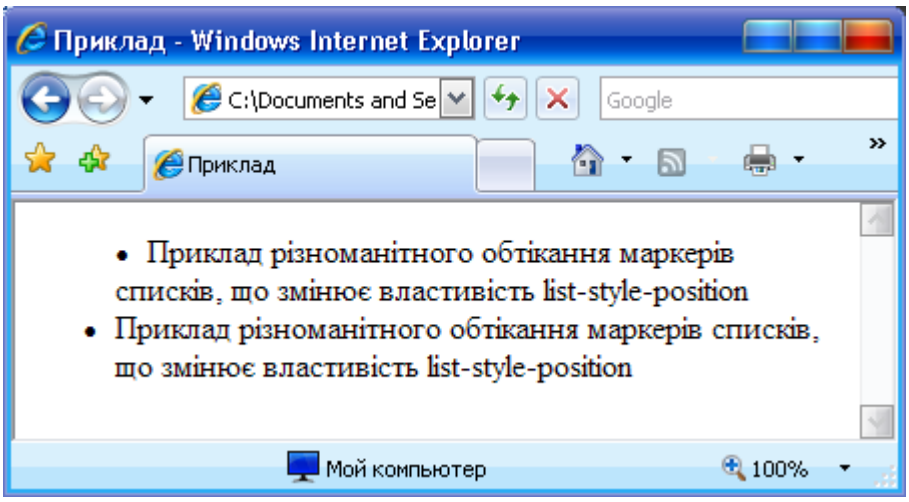

Рис 2.3.12 Різні способи обтікання списків

Проблем з цією властивістю у браузерів немає, так що нею можна користуватися сміливо.

list-style

Крім того, є властивість `list-style`, яка служить для скороченого запису стилів. Формат у неї такий:

```
list-style:<keyword> <position> <url>
```

Наприклад, список з маркерами `disc` і обтіканням `inline` можна записати так:

```
LI{
  list-style-type:circle;
  list-style-position:inline
}
```

А можна і так:

```
LI {
  list-style:circle inline
}
```

2.3.6Відображення блоків

Для повноти картини залишилося розглянути корисні властивості із специфікації CSS-2, які дозволяють добитися деяких ефектів при створенні блоків.

visibility

За допомогою даної властивості можна робити блоки видимими і невидимими. Причому робити це можна динамічно, що і

застосовується успішно при виготовленні так званих випадних меню (drop-down menu).

Значення •

- `visible` — елемент видимий;
- `hidden` — елемент невидимий;
- `collapse` — служить для приховування колонок і рядів в таблиці.

Ще раз повторимо головну відмінність між оголошеннями `visibility:hidden` і `display:none`. Воно полягає в тому, що при використанні властивості `visibility` блок фізично залишається на сторінці, тоді як при використанні `display` блок віддаляється з потоку форматування, тобто його взагалі немає на сторінці.

overflow

Відповідає за відображення блоку, якщо контент не поміщається повністю в межі блоку. Може приймати чотири значення. Зробимо блок шириною 200 пікселів, заввишки 50 пікселів і наповнимо його текстом.

```
#block {
  border: 1px solid #000;
  width: 200px;
  height: 50px;
  padding: 1em;
}
...
<DIV ID="block">
Відповідає за відображення блоку, якщо контент
не поміщається повністю в межі блоку. Може
приймати чотири значення.
</DIV>
```

Тепер будемо встановлювати різні значення властивості `overflow: visible`

В цьому випадку вміст блоку повинен перекривати блок і залишатися видимим. (Рис 2.3.13)

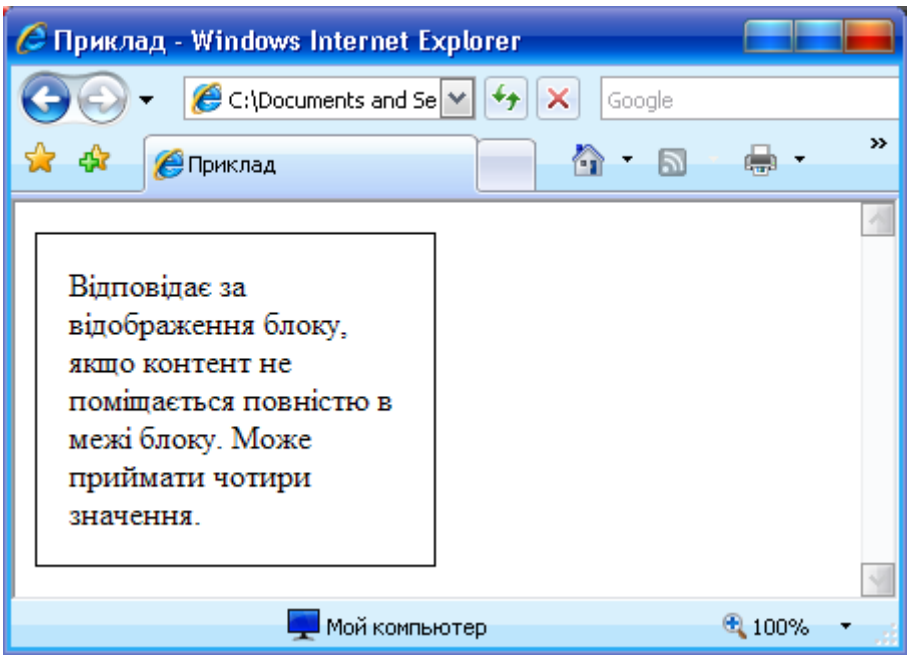

Рис 2.3.13 Відображення блоку з оголошенням overflow: visible

`overflow: hidden`

В цьому випадку контент, який перевищує розміри блоку, нещадно обрізається і користувач його не бачить. Взагалі застосування даного оголошення особливого сенсу не має. Якщо усередині блоку знаходиться текст, то він обов'язково повинен бути доступний для прочитання (Рис 2.3.14).

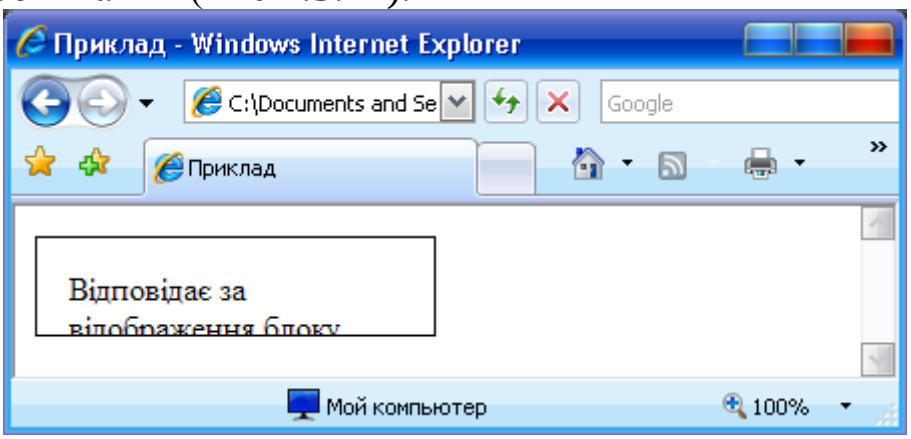

Рис 2.3.14 Відображення блоку з оголошенням overflow: hidden

`overflow: scroll`

В цьому випадку блок повинен забезпечуватися горизонтальними і вертикальними смугами прокрутки. При цьому користувач зможе переглянути весь вміст блоку. Як це виглядає видно з Рис 2.3.15.

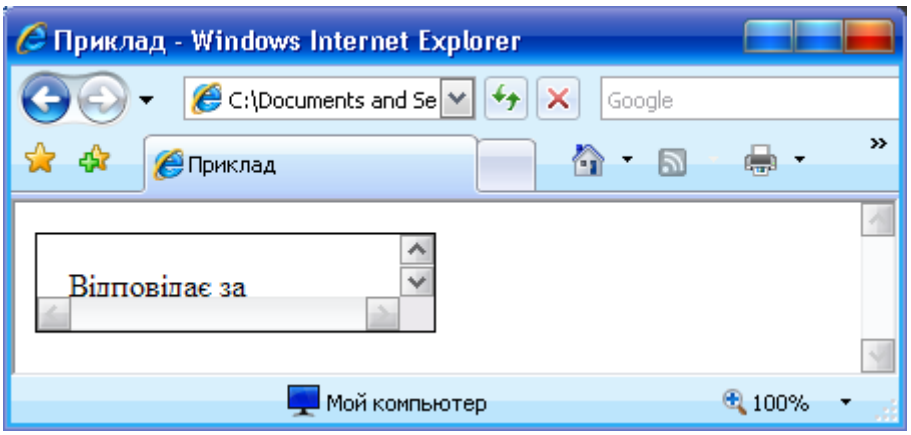

Рис 2.3.15 Відображення блоку з оголошенням overflow: scroll

`overflow: auto`

Воно забезпечує блок тільки тією смугою прокрутки, яка необхідна. У нашому прикладі висота контента перевищує висоту блоку і потрібна тільки вертикальна смуга прокрутки. Цього можна досягти, встановивши значення `auto` у властивості `overflow` (Рис 2.3.16).

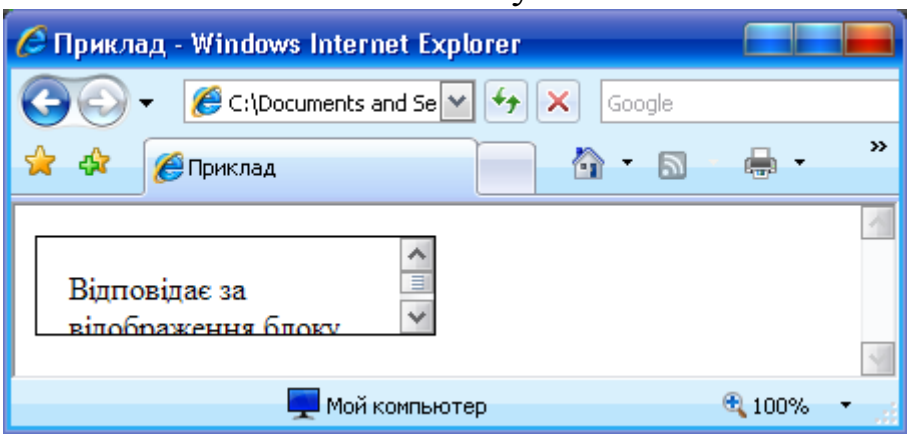

Рис 2.3.16 Відображення блоку з оголошенням overflow: auto

Застосовувати значення `scroll` і `auto` можна у тому випадку, коли треба заощадити місце на сторінці. Наприклад, так можна організувати подачу новин.

Кожна новина поміщається в окремий блок з оголошенням `overflow:auto`.

Висота блоку підбирається так, щоб на виду був заголовок новини або ж короткий анонс, а решта тексту знаходитиметься нижче.

clip

Параметр `clip` визначає область позиціонованого елемента, в якій буде показано його вміст. Все, що не поміщається в цю область, буде обрізане і стає невидимим. На даний момент єдино доступна форма області - прямокутник. Все інше залишається тільки в мріях.

Параметр `clip` працює тільки для абсолютно позиційований елементів.

Формат запису `clip: rect (Y1, X1, Y2, X2) | auto` В якості аргументів використовується відстань від краю елемента до області обрізки, яка задається в одиницях CSS - пікселі (px), відсотки (%) та ін Якщо край області потрібно залишити без змін, слід поставити параметр `auto`. Координати Y1, X1 визначають вертикальну і горизонтальну координату верхнього лівого кута прямокутника, а Y2, X2 правого нижнього.

Приклад

```
#block {
  position: absolute;
  clip: rect(5px, 180px, auto, 40px);
  width: 200px;
  border: 1px solid black;
  padding: 10px;
}
...
<DIV ID="block">
Відповідає за відображення блоку, якщо контент
не поміщається повністю в межі блоку. Може
приймати чотири значення.
</DIV>
```

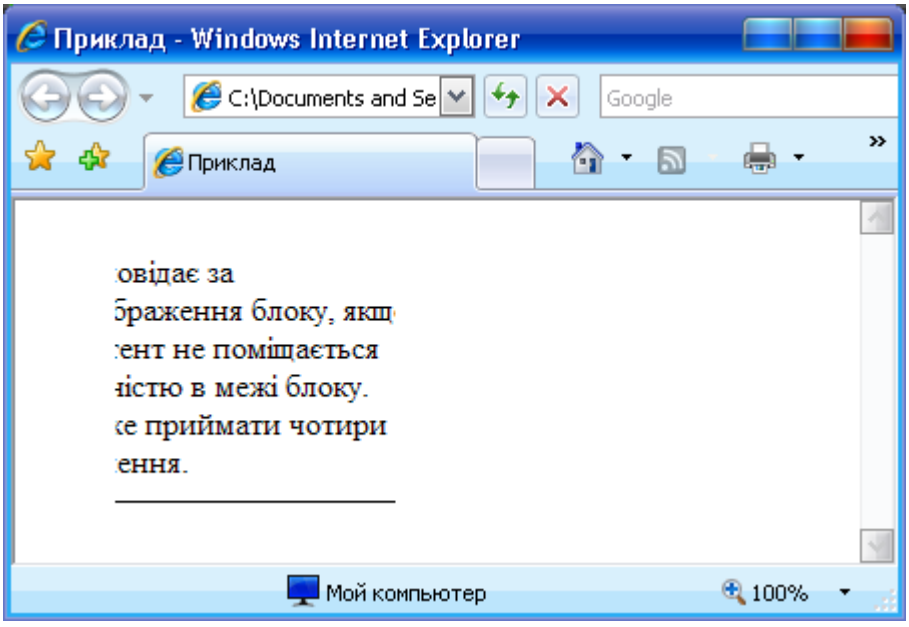

Рис 2.3.17 Приклад використання властивості clip

2.4 Позиціонування

Позиціонування є самим багатообіцяючим розділом CSS. Вся справа в тому, що використання таблиць для позиціонування елементів, як це робиться засобами HTML, є штучним прийомом. Спочатку таблиці для цього не призначалися, але верстальникам мимоволі довелося користуватися саме ними, оскільки ь HTML більше немає ніяких способів реалізувати складне форматування сторінки. Проте у таблиць є свої недоліки.

- По-перше, таблиці не дуже компактні з точки зору коду.
- По-друге, таблиці досить повільно відображаються.
- По-третє, вміст таблиці розташовується, як правило, досить нелогічно Скажімо, шапка сторінки може бути закодована за допомогою трьох таблиць
- По-четверте, при використанні таблиць ускладнюється правка коду статичних сторінок і шаблонів через змішування структури документа і його вмісту.
- По-п'яте, таблична верстка дуже нетривіальна і всім її тонкощам доводиться вчитися досить довго.

Природно, переваги у них також є:

- Таблиці дозволяють довільним чином позиціонувати елементи;
- Таблична технологія верстки добре освоєна і по ній написано багато книг, так що вчитися можна ефективно, хоча сама таблична верстка не стає від цього простішою;
- Таблиці практично однаково відображаються всіма браузерами.

Тепер розглянемо переваги верстки за допомогою блоків і CSS-позиціонування.

- Значно покращується логічність коду. Це досягається завдяки декількох причин. Кожен блок має унікальне ім'я. Природно, блоку можна дати ім‘я що максимально відповідає йому. За рахунок поділу візуального представлення та структури документа, безпосередньо з HTML-коду зникнуть багато атрибутів, так що код стане прозорішим і чистішим.
- Зменшується загальний обсяг коду. Щоправда, для дуже маленьких і простих сторінок обсяг може й збільшуватися, але якщо врахувати, що зовнішні CSS-файли зазвичай кешуються браузерами, то час повторного завантаження сторінки все одно зменшується. Власне, час завантаження і є визначальний фактор, тоді як обсяг коду вторинний.

Значно збільшується контроль над блоками. Змінювати структуру таблиці складніше, ніж міняти стилі для блоків.

Що стосується недоліків, то вони теж є.

- Браузери вельми неоднаково підтримують CSS-позиціонування. Створювати блоки з розрахунком на різні браузери не легко, але на щастя з кожною новою версією браузерів проблем стає менше.
- CSS-позиціювання - це абсолютно новий напрямок.
- Не кожен макет можна зверстати на основі CSS. Це дуже рідкісні винятки і часто можна зробити невеликі правки в дизайні, але, тим не менше, факт залишається фактом.

Відразу слід відмітити, що в стандарті CSS-1 позиціонування вкрай нерозвинене, так що для повноцінного CSS-позиціонування необхідна хоча б часткова підтримка браузерами стандарту CSS-2.

Отже, для початку систематизуємо все те, що може впливати на розташуваня блоків на сторінці. Виявляється, лише три речі.

- Схема позиціювання. Вона буває чотирьох видів: *нормальний потік*, *відносне позиціонування*, *абсолютне позиціювання*, *плаваюча модель*.
- Розташування блоків в тілі документа. Якщо конкретизувати, то в деяких випадках велике значення має відносне положення блоків один відносно іншого. Найменше значення це має при абсолютному позиціонуванні, а при нормальному потоці це фактично визначає вигляд документа.
- Зовнішні параметри, такі як розмір вікна браузера. Вони критичним чином впливають на "гумову" верстку, так що дизайн сторінки може сильно постраждати при занадто маленькому розмірі вікна. Насправді нічого дивного і страшного в цьому немає, треба просто спочатку розробляти сторінку для найбільш популярної роздільної здатності моніторів, а вже потім підганяти під інші розміри

Розглянемо детальніше схеми позиціонування.

2.4.1Нормальний потік

Давайте згадаємо, як браузер обробляє дерево документа. Є два типи блоків: структурні і рядкові. Правила форматування для них відрізняються.

- Структурні блоки форматуються наступним чином: браузер розміщує їх безпосередньо один за одним по вертикалі, тобто

два структурних блоки не можуть перебувати на одній горизонтальній лінії. За допомогою полів (margins) можна задавати відстань між блоками по вертикалі.

- Рядкові блоки форматуються так: браузер розпорядженні їх горизонтально один за одним. Один рядок називається лінійним блоком, якщо рядкові блоки не можуть вміститися в одному лінійному, то вони розміщаються в кількох лінійних блокам, які слідують один за одним по вертикалі, тобто в кілька рядків.

Таким чином, нічого складнішого однієї колонки за допомогою нормального потоку зробити неможливо.

Найпростіший приклад - це кілька абзаців з малими блоками.

```
#first {
  padding: 10px;
  border: 1px solid #000;
  width: 90%
}
#second {
  border: 1px solid #000;
  width: 50%
}
EM{
  border: 1px dashed #AAA;
}
...
<DIV id="first">
  Перший блок <EM>нормального</EM>
<EM>потоку</EM> з рядковими блоками елементами
легкого виділення
</DIV>
<DIV id="second">
  Другий блок нормального потоку
</DIV>
```

В браузері цей код буде виглядати так як на Рис 2.4.1

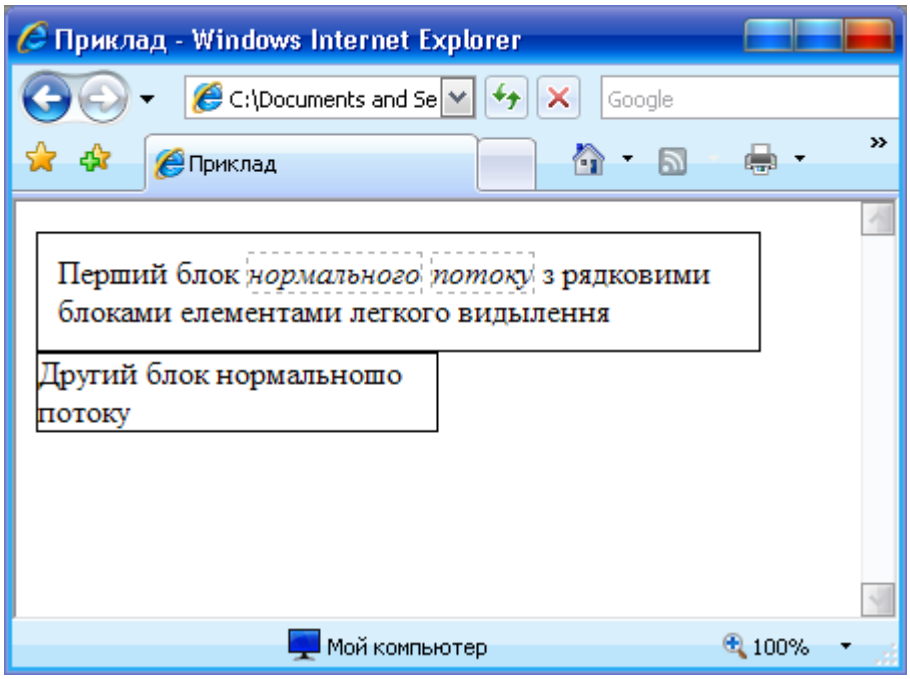

Рис 2.4.1 Обробка блоків браузером в нормальному потоці

2.4.2Відносне позиціонування

Відносне позиціонування функціонує за наступним алгоритмом. Спочатку положення елемента розраховується згідно нормального потоку (воно називається нормальним положенням) Потім елемент змішається відносно нормального положення. Величина зміщення задається з допомогую наступних властивостей:

- `left` - встановлює зміщення від лівого краю контейнера;
- `right` - встановлює зміщення від правого краю контейнера;
- `top` - встановлює зміщення від верхнього краю контейнера;
- `bottom` - встановлює зміщення від нижнього краю контейнера.

Значення для цих властивостей можна вказувати в будь-яких одиницях довжини та у відсотках. Як працює відносне позиціонування, добре видно з наступного прикладу. Створимо три зовсім однакових блоки:

```
P{
  border: 1px solid #000
}
...
<P>Перший блок</P>
<P>Другий блок</P>
<P>Третій блок</P>
```

Зовнішній вигляд блоків показано на Рис 2.4.2

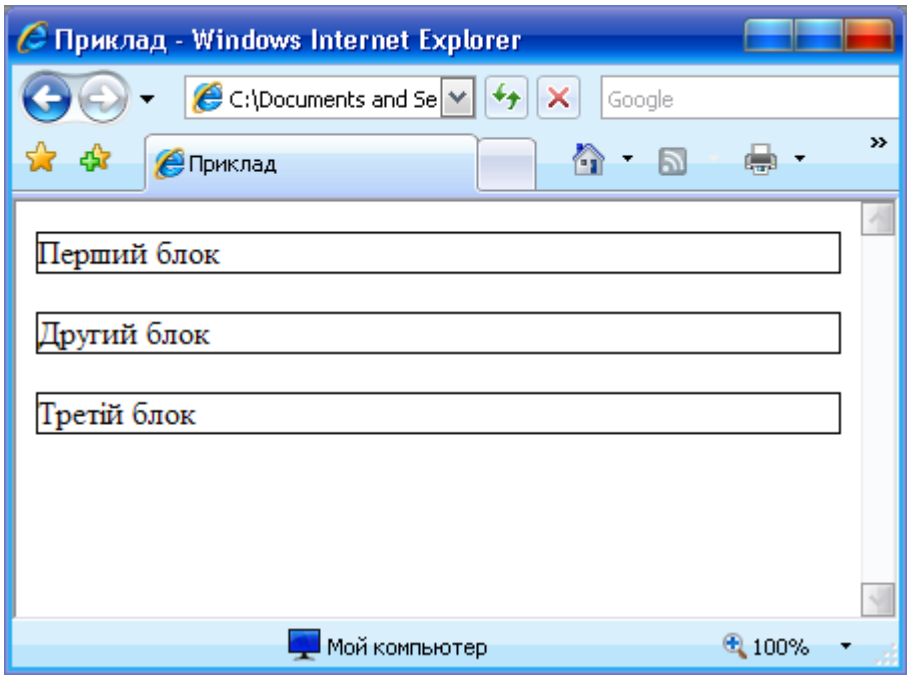

Рис 2.4.2 Три блоки в нормальному потоці

А зараз другий блок відносно спозиціонуємо, змістимо його на 23 пікселя від верхнього краю.

Для цього присвоїмо другому блоку `ID = "rel"`, і код буде таким:

```
P {
  border: 1px solid #000
}
#rel {
  position: relative;
  top: 23px
}
...
<P>Перший блок</P>
<P ID="rel">Другий блок</P>
<P>Третій блок</P>
```

На Рис 2.4.3 видно як зміниться вигляд в браузері

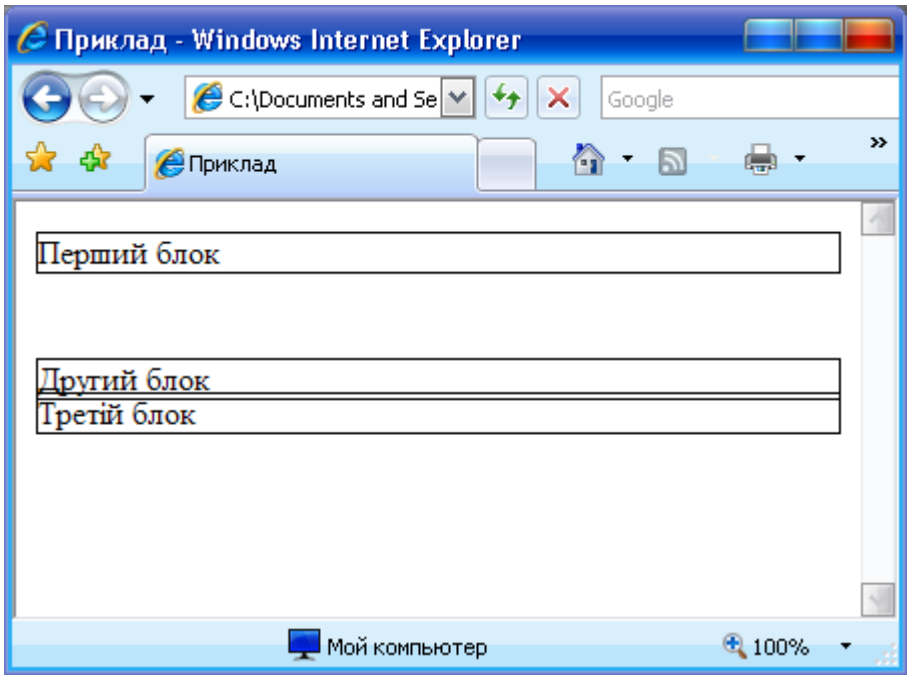

Рис 2.4.3 Три блоки в нормальному потоці. Другий блок зміщений на 23 пікселя вниз

Бачимо, що другий блок просто змістився на 23 пікселя вниз, внаслідок чого сталося накладення другого і третього блоків, але становище третього блоку на сторінці не змінилося. Отже, відносне позиціонування безпосередньо впливає тільки на сам позиціонований блок, але не впливає на всі інші блоки Це важливе правило, яке слід пам'ятати.

Відносне позиціонування лише дозволяє зміщувати блоки, і все. Теоретично ми можемо розташувати блоки як захочемо, якщо точно задані їх висота і ширина

Однак є дві причини, які цьому перешкоджають.

- Фіксувати висоту блоку практично нереально. Природно, що тексти в блоках можуть бути довгими або короткими, так що висота деяких блоків буде змінюватися. При цьому розташовані нижче блоки просто опускаються вниз. Дизайн повинен бути зроблений так, правильно реагувати за розумної висоті блоку.
- Браузери абсолютно по-різному інтерпретують складне відносне позиціонування. Особливо коли брати три і більше блоків.

Верстальники газет і журналів звикли що блоки інформації розміщаються туди де потрібно не залежно один від одного. Саме з такою метою і розроблено в CSS-2 абсолютне позиціонування.

2.4.3Абсолютне позиціонування

Якщо у відносному позиціонуванні елемент лише зміщувався відносно нормального положення, але залишався в нормальному

потоці, то при абсолютному позиціонуванні елемент повністю вивертається з нормального потоку і вже потім позиціонується у відповідності з заданими оголошеннями в таблиці стилів. Це дуже принципова відмінність і з нього безпосередньо випливають два наслідки.

- Якщо блок абсолютно позиціонується, то в HTML-коді він може розміщатися в будь-якому місці (якщо не брати в розрахунок вкладені блоки, з ними дещо інакше), тому що він як би знаходиться поза нормальним потоком. При відносному позиціонуванні положення блоку безпосередньо впливало на місце блоку на екрані.
- Очевидно, властивості `top`, `left`, `right`, `bottom` функціонують інакше, ніж при відносному позиціонуванні. Тоді вони задавали величину зміщення відносно положення елемента в нормальному потоці, зараз елемент вивертається з нормального потоку, так що контейнером для нього завжди буде або вікно браузера, або позиціонований блок (тобто блок з оголошенням `position:relative`, `position:absolute`).

Зробити блок абсолютно позиціонованм можна з допомогою оголошення `position:absolute`. Для того щоб зрозуміти відмінності між відносним та абсолютним позиціонуванням, розглянемо приклад. Візьмемо два блоки шириною 30% і висотою 20% від ширини екрана. Обидва будуть відносно позиціоновані. Перший блок (one) буде зміщений вниз на 10%, другий (two) теж буде зміщений вниз на 10%, але, крім того, ще і вліво на 30%.

```
DIV {
  border: 1px solid #000
}
#one {
  width: 30%;
  height: 20%;
  position: relative;
  left: 0;
  top: 10%
}
#two {
  width: 30%;
  height: 20%;
  position: relative;
```

```
    left: 30%;
    top: 10%
  }
  ...
  <DIV ID="one">Перший блок</DIV>
  <DIV ID="two">Другий блок</DIV>
```

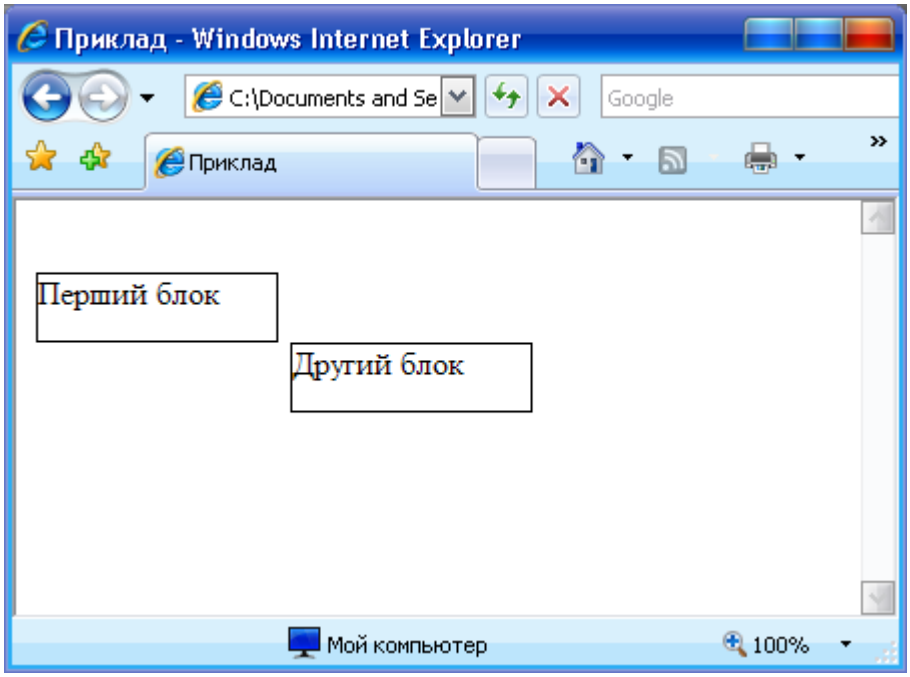

Рис 2.4.4 Два блоки позиціоновані відносно

Зрозуміло, в цьому випадку положення блоків у коді важливо, так що блок `one` йде першим, а блок `two` - другим У результаті блоки будуть розташовані драбинкою, як показано на Рис 2.4.4

А зараз просто замінимо в коді оголошення `position:relative` на `position:absolute` і поміняємо місцями блоки `one` і `two` в коді. Все решту залишимо без змін.

З Рис 2.4.5 бачимо, що вигляд повністю змінився

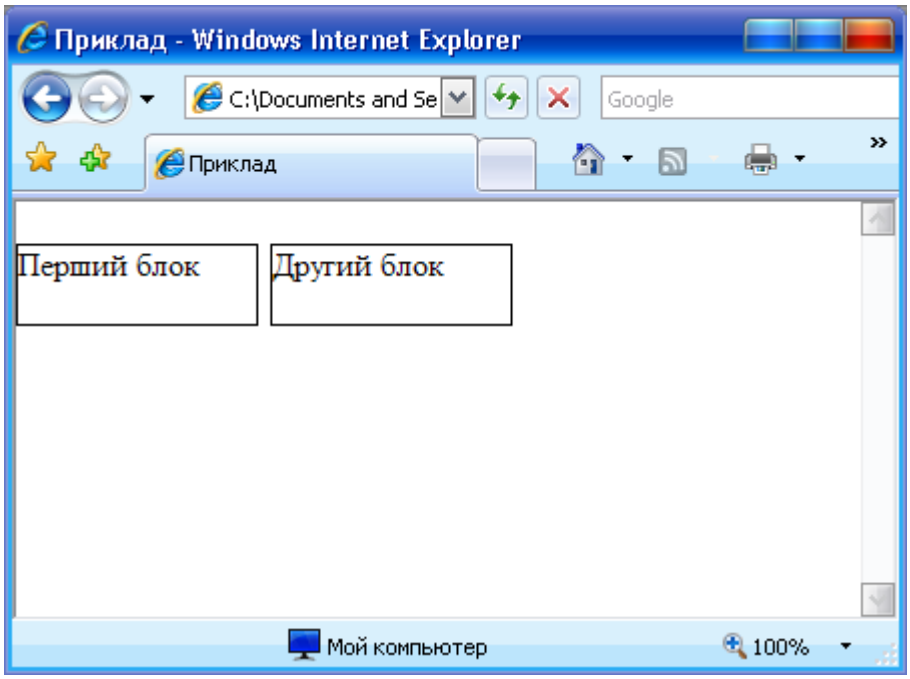

Рис 2.4.5 Два блоки позиціоновані абсолютно

Як бачимо, перестановка в HTML-коді блоків, створених за допомогою елементів <DIV>, абсолютно не вплинула на розташування блоків у вікні браузера. Інакше кажучи при абсолютному позиціонуванні елементи повністю вириваються з нормального потоку, а точне розташування блоку у вікні браузера задається безпосередньо в стилях. На основі цього можна вивести очевидне правило.

Правило

При абсолютному позиціонуванні розташування блоку, який є прямим нащадком елемента <BODY>, У HTML-коді документа не впливає на розміщення цього блоку у вікні браузера

Абсолютне позиціювання є непоганим кандидатом на роль схеми для верстки сайтів. За допомогою абсолютного позиціонування можна легко розміщувати блоки на одній горизонтальній лінії та контролювати їх позицію з точністю до одного пікселя.

Слід розглянути поняття контейнер для абсолютно позиціонування елемента, тому що воно трохи відрізняється від звичайного представлення контейнера. Це істотно для вкладених блоків. Треба сказати, що для вкладених блоків абсолютне позиціювання рідко застосовується відразу до обох.

Визначення

Контейнером для абсолютно позиціонованого елемента може бути лише абсолютно або відносно позиціонований елемент або ж безпосередньо вікно браузера.

Таким чином, абсолютно позиціоновані елементи першого рівня вкладеності мають один і той же контейнер - вікно браузера.

Що стосується стабільності та адекватності, то абсолютне позиціонування однозначний лідер по цих параметрах. Воно дуже просте, його легко вивчити і серйозних багів воно не має. Так що для нескладних сайтів з фіксованою шириною ця схема є найбільш підходящою.

2.4.4Фіксоване позиціонування

Крім усього іншого, є цікавий підвид абсолютного позиціонування. Назвемо його фіксованим позиціонуванням. Особливість його в тому, що фіксований елемент завжди буде присутній в області перегляду, тобто при прокручувані він все рівно буде на тому ж місці. Це дуже схоже на фрейми, при використанні яких ми теж можемо прокручувати вміст одного фрейму, тоді як вміст другого фрейму буде на виду. Але у фреймів безліч недоліків, тоді як у фіксованого позиціонування їх набагато менше.

Ця схема включається за допомогою оголошення `position: fixed`

Найбільшою проблемою є те, що фіксоване позиціонування не підтримується браузером InternetExplorer. Тут вони розглядаються так наче вони знаходяться в нормальному потоці. З цієї причини навряд чи слід користуватися фіксованим позиціонуванням, але про таку можливість знати потрібно. Розглянемо приклад:

```
DIV {
  border: 1px solid #000
}
#one {
  width: 30%;
  height: 20%;
  position: fixed;
  left: 0;
  top: 10%
}
#two {
  width: 31%;
  height: 20%;
  position: absolute;
  left: 30%;
```

```
  top: 10%
}
...
<DIV ID="two">Крім усього іншого, є цікавий
підвид абсолютного позиціонування. Назвемо його
фіксованим позиціонуванням. Особливість його в
тому, що фіксований елемент завжди буде
присутній в області перегляду, тобто при
прокручувані він все рівно буде на тому ж
місці.
</DIV>
<DIV ID="one">Перший блок</DIV>
```

Як ми бачимо на рисунку Рис 2.4.6 частина тексту другого блоку прокручена а перший блок залишився на цьому ж місці.

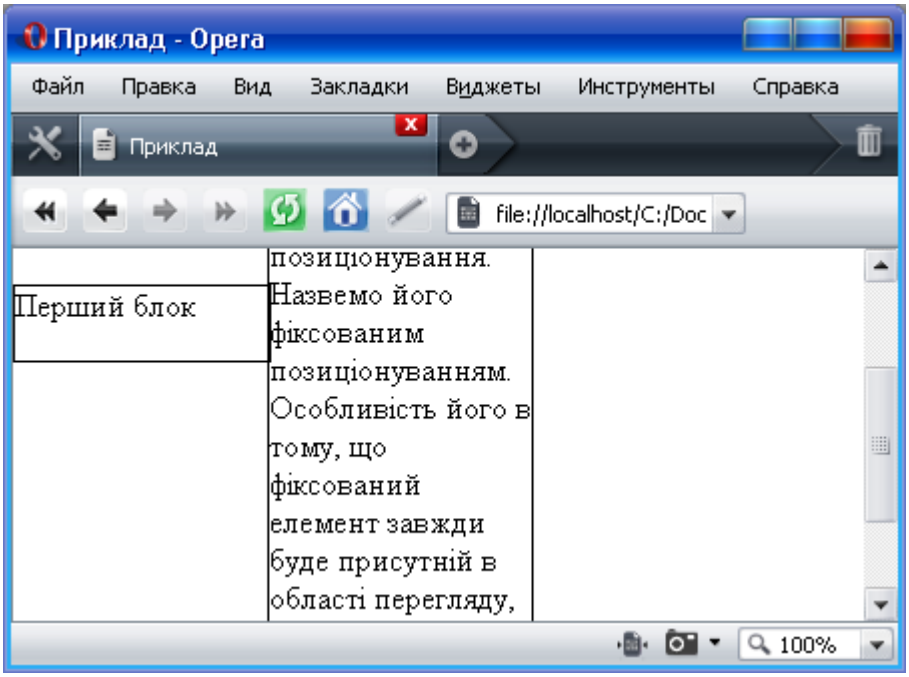

Рис 2.4.6 Коректне опрацювання фіксованого позиціонування баузер Opera

На рисунку Рис 2.4.7 видно не коректну роботу фіксованого позиціонування в InternetExplorer.

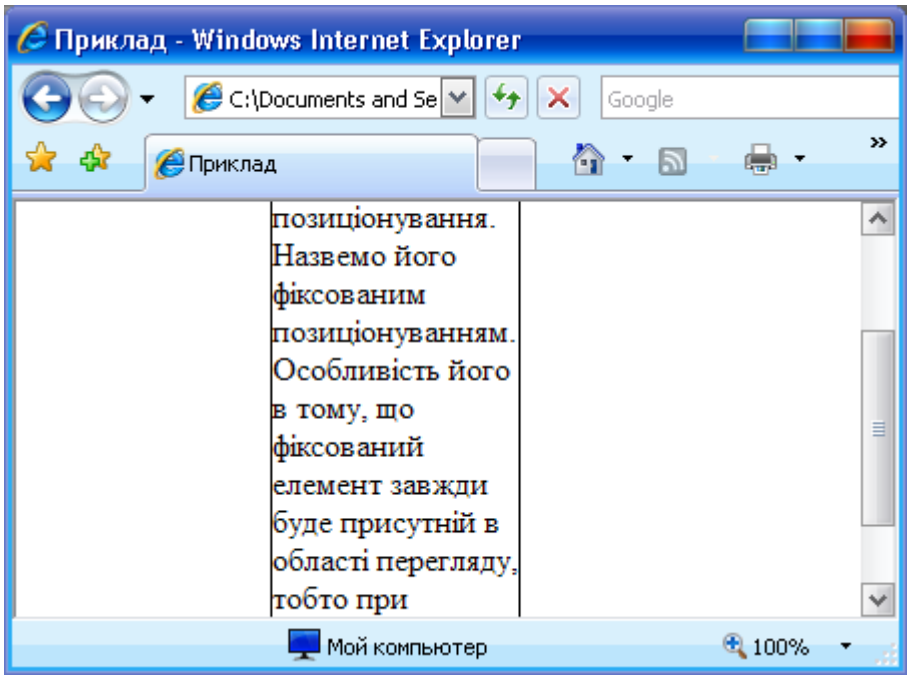

Рис 2.4.7 Некоректне опрацювання фіксованого позиціонування браузер InternetExplorer

2.4.5Плаваюча модель

Вона принципово відрізняється від абсолютного позиціонування тим, що тут немає такої точності і не можна до пікселя позиціонувати блоки. Однак без неї не обійтися в тому випадку, якщо необхідна "гумова" верстка сайту. Крім того, плаваюча модель підходить для верстки в кілька колонок. Власне кажучи, саме для цього вона і задумувалась.

Отже, якщо елемент позначений як плаваючий, то він вивається з нормального потоку (як і при абсолютному позиціонуванні) і "прилипає" до правого або лівого краю контейнера, а текст і графіка обтікають його. Якщо елемент "прилипає" до лівого краю, то текст обтікає його справа, і навпаки. Плаваюча модель для елемента встановлюється за допомогою властивості `float`, що може приймати три значення: `left`, `right` і `none`. При оголошенні `float:left` елемент "прилипає" до лівого краю контейнера, а при оголошенні `float:right`, відповідно, до правого. Якщо ж вказано `float:none`, то елемент не переміщується.

Розглянемо найпростіший приклад. Створимо блок, який буде "прилипати" до правого краю контейнера, а текст - обтікати його зліва. Блок матиме сірий фон, товсту чорну рамку і відступи з усіх сторін для кращого сприйняття тексту. Код блоку буде таким:

```
#right {
```

```
  background: #CCC;
  width: 200px;
  border: 5px solid #000;
  padding: 1em;
  float: right
}
...
<DIV ID="right">
  Цей блок буде прилипати до правого краю
</DIV>
<P>Розглянемо найпростіший приклад. Створимо
блок, який буде "прилипати" до правого краю
контейнера, а текст - обтікати його зліва. Блок
матиме сірий фон, товсту чорну рамку і відступи
з усіх сторін для кращого сприйняття тексту.
Код блоку буде таким:</P>
```

В результаті в браузері ми побачимо Рис 2.4.8

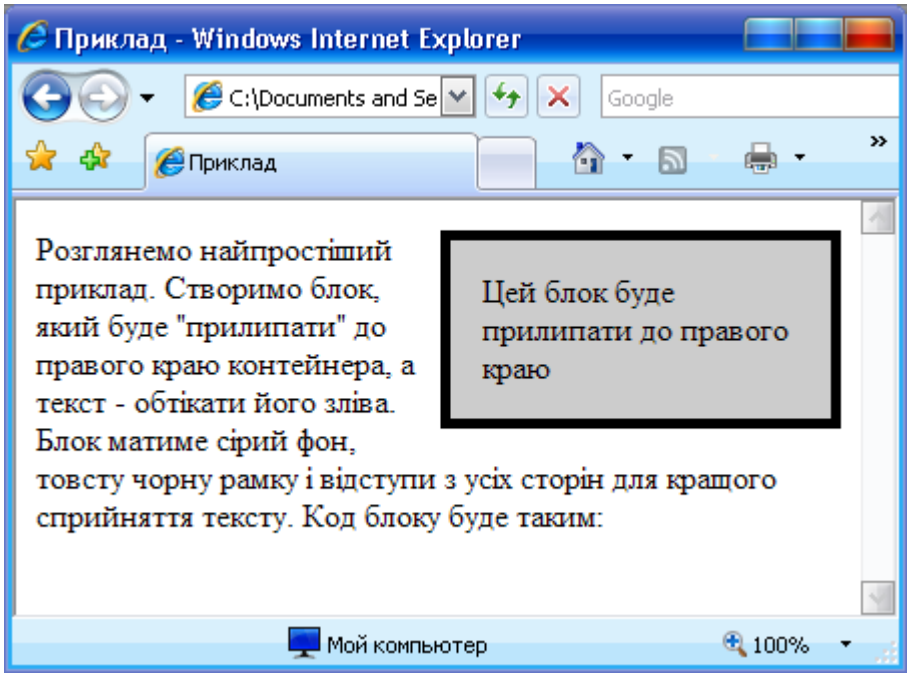

Рис 2.4.8 Простий плаваючий блок

Звернемо увагу на наступне правило.

Правило

Для плаваючих блоків треба завжди вказувати ширину за допомогою властивостей `width`. Тому що в іншому випадку блок буде розтягуватися, щоб вмістити в себе весь контент.

У HTML був атрибут ALIGN у елемента <IMG>, так що можна було заставляти текст обтікати малюнки. За допомогою CSS можна легко зробити так, щоб обтікались блоки тексту. На основі цього можна робити врізки, в які поміщати підказки, відступи від тексту, корисні поради - тоді сторінка буде виглядати набагато цікавіше і різноманітніше. Читати текст на екрані монітора взагалі важче ніж текст, надрукований на папері, так що будь-яке удосконалення, полегшує процес читання.

Для подальшого вивчення плаваючою моделі розглянемо одну проблему. На Рис 2.4.8текст обтікає блок. Іноді абсолютно необхідно, щоб текст починався безпосередньо під блоком

Наприклад, йде у вас текст, оточуючий врізку, розділ з заголовком (Рис 2.4.9). Код такий:

```
#left {
 background: #CCC;
 width: 120px;
 border: 5px solid #000;
 padding: 1em;
 float: left
}
...
<DIV ID="left">
 Цей блок буде прилипати до лівого краю
</DIV>
<P>Розглянемо найпростіший приклад. Створимо
блок, який буде "прилипати" до правого краю
контейнера, а текст - обтікати його зліва.
</P>
<H3>Заголовок</H3>
<P>Розглянемо найпростіший приклад. Створимо
блок, який буде "прилипати" до правого краю
контейнера, а текст - обтікати його зліва.
</P>
```

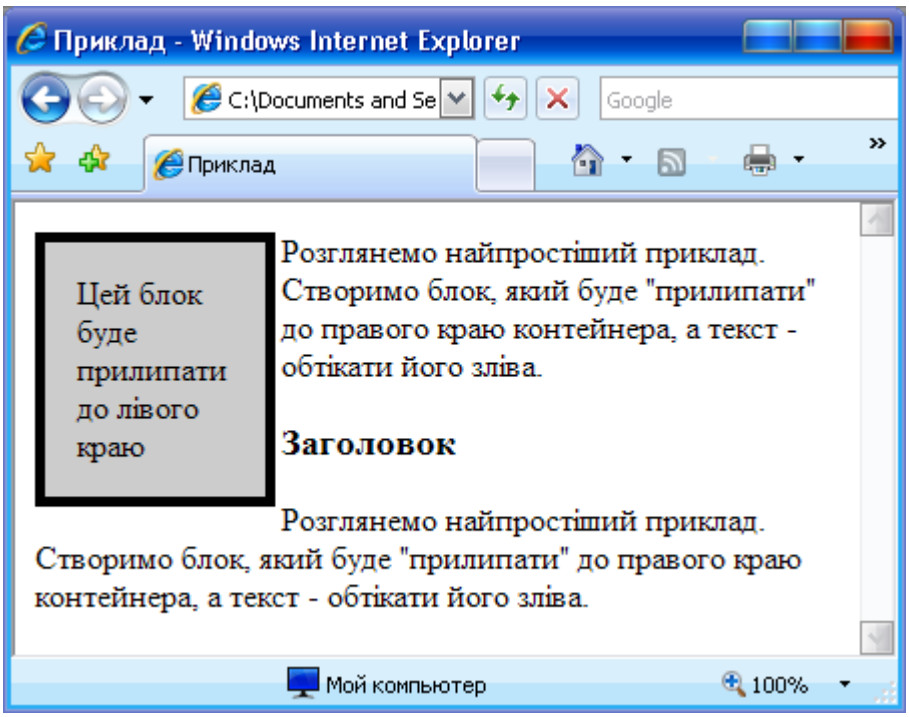

Рис 2.4.9 Плаваючий блок і обтікаючий його заголовок

Заголовок має розташовуватися безпосередньо після малюнка, а не праворуч від нього. І це можна зробити за допомогою властивості `clear`, яка приймає наступні значення:

- `left` - блок переміщається нижче всі плаваючих блоків з оголошенням `float:left`;
- `right` - блок перемішається нижче всіх плаваючих блоків з оголошенням `float:right`;
- `both` - блок переміщається нижче всіх плаваючих блоків;
- `none` - обмежень на обтікання немає.

Тому для того, щоб заголовок починався після врізки, нам треба дописати для нього оголошення `clear: left`:

```
H3 {
  clear: left
}
```

І картина прийме необхідний вигляд, показаним на Рис 2.4.10.

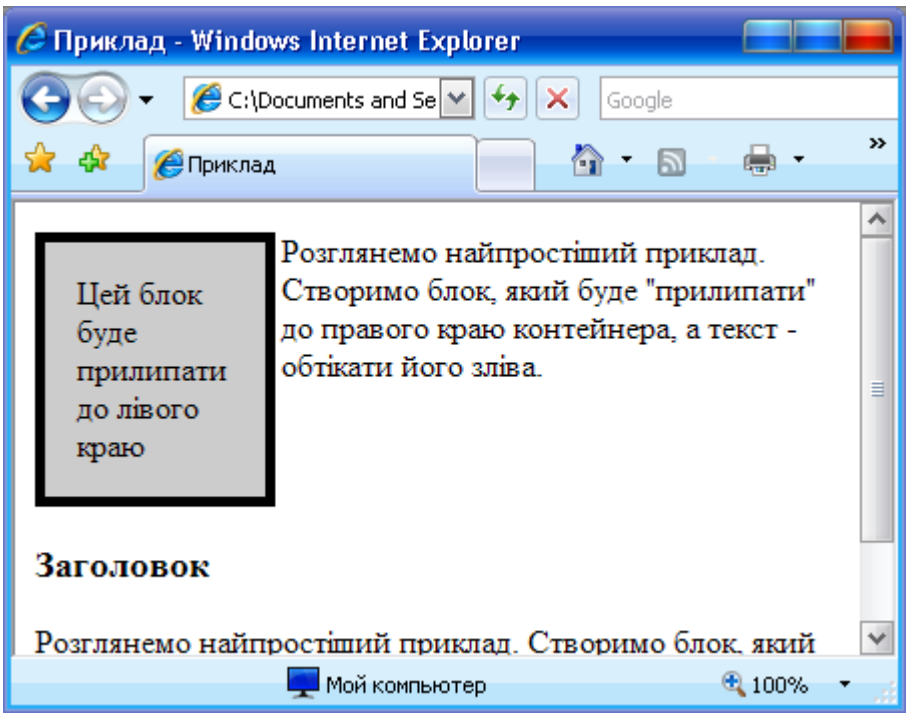

Рис 2.4.10 Блок заголовку з застосованою до нього властивістю clear

Тепер розглянемо випадок коли для обох блоків написано оголошення float: left.

Правило

Якщо поточний плаваючий блок є лівостороннім (тобто має оголошення float: left) і існує ще хоча б один лівосторонній плаваючий блок, який був згенерований у вихідному документі раніше за нього, то лівий зовнішній край поточного блоку повинен знаходитися праворуч від правого зовнішнього краю попереднього блоку, або верхній його край повинен бути нижче ніж нижній край попереднього блоку

Виходить, що блоки будуть розташовуватися на одній горизонталі в тому випадку. якщо їх сумарна ширина менше або дорівнює ширині контейнера.

Створимо три абсолютно однакові блоки:

```
#left {
  background: #CCC;
  width: 28%;
  height: 90%;
  padding: 2%;
  border: 1px solid #000;
  float: left
}
#center {
```

```
  background: #CCC;
  width: 28%;
  height: 90%;
  padding: 2%;
  border: 1px solid #000;
  float: left
}
#right {
  background: #CCC;
  width: 28%;
  height: 90%;
  padding: 2%;
  border: 1px solid #000;
  float: left
}
...
<DIV id="left">
  <B>Ліва колонка</B> А також трохи тексту.
</DIV>
<DIV id="center">
  <B>Центральна колонка</B> Тут потрібно значно
більше тексту, щоб провірити довжину.
</DIV>
<DIV  id="right">
  <B>Права колонка</B>. І також трохи тексту.
</DIV>
```

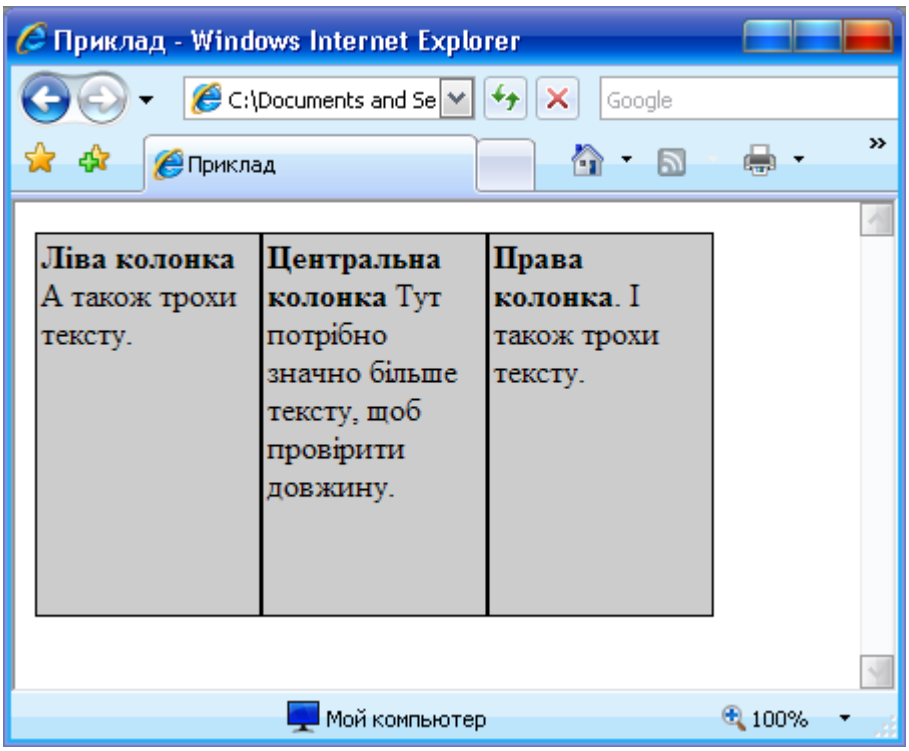

Рис 2.4.11 Три плаваючі блоки

2.4.6Шари

Будь-який блок має координати на площині екрану, проте в CSS-2 є і третя координата, яка описує положення блоку в просторі. Що таке координати `x` і `y` на екрані монітора - пояснювати не треба, а от що таке координата `z` мабуть варто пояснити. Уявіть, що у вас є декілька прозорих плівок розміром з екран. Якщо намалювати щось на кожній плівці і прикріпити їх до екрану. Кожна плівка представляє. собою окремий шар, а віддаленість її від поверхні монітора і буде характеризуватися координатою `z`. Як бачите, вона досить специфічна, тому її безглуздо висловлювати в пікселях або будь-яких інших одиницях довжини, тобто має значення тільки відносне положення шару: який. з них ближче до поверхні, а який дальше. Саме так і реалізована координатна вісь `z` в CSS-2, а положення шару задається властивістю `z-index`, що може приймати тільки цілочисельні значення. Чим вище значення. тим далі від поверхні екрана розташовується шар. Наприклад

```
#deep {
  width: 220px;
  color: yellow;
  font-size: 50px;
  font-family: "Arial Black";
  position: absolute;
```

```
  z-index: 1;
}
#high {
  width: 220px;
  color: red;
  font-size: 15px;
  font-family: "Arial";
  position: absolute;
  z-index: 2;
}
...
<DIV id="deep">
  ТЕКСТ
</DIV>
<DIV id="high">
  текст текст текст текст текст текст текст
текст текст текст текст текст текст текст
</DIV>
```

Результат показано на Рис 2.4.12

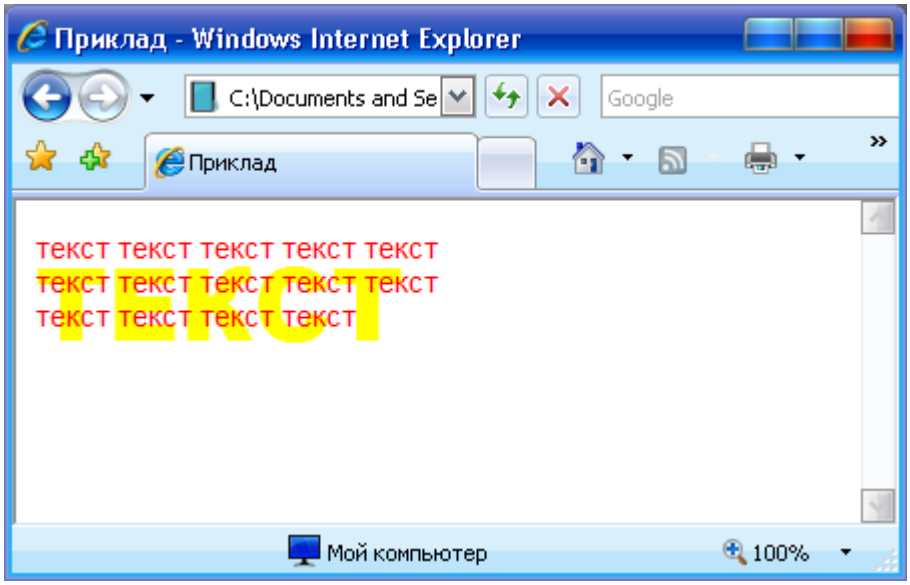

Рис 2.4.12 Приклад використання шарів

Шар `high` буде далі від поверхні, тобто ближче до користувача. Прицьому вміст цього шару буде перекривати вміст шару `deep`.

Шари можна використовувати якщо ви хочете створити анімацію на основі DHTML. При створенні різного роду випадаючих меню теж треба вказувати z-index, щоб меню було розташовано максимально ближче до користувача. Тобто поверх всіх інших шарів, інакше його просто не буде видно.

2.5 Контрольні запитання

- Які існують типи таблиць стилів?
- Для чого призначений атрибут class елементів розмітки?
- Для чого призначений атрибут style елементів розмітки?
- Як можна використовувати атрибут id для установки CSS-властивостей?
- Для чого призначений тег <STYLE>?
- Як можна використовувати тег <LINK> для установки CSS-властивостей?
- Які властивості таблиць стилів (CSS) призначені для застосування жирного шрифту і курсиву?
- Які властивості таблиць стилів (CSS) є альтернативою використання тега <FONT>?
- Які властивості таблиць стилів (CSS) призначені для управління вертикальними і горизонтальними розмірами елементів розмітки?
- Які властивості таблиць стилів (CSS) призначені для управління рамкою елементів розмітки?
- Які властивості таблиць стилів (CSS) призначені для управління зовнішнім і внутрішнім відступами елементів розмітки?
- Які властивості таблиць стилів (CSS) призначені для горизонтального і вертикального вирівнювання тексту елементів розмітки?Для чого призначені html-дескриптори?

2.6 Перелік навчально-методичної літератури.

1. Основи веб-дизайну / Пасічнік О.Г., Пасічнік О.В., Стеценко І.В. – Видавнича група BHV 2009, . – 336с.
2. Основи будування сайтів / В. Манако, Д. Манако, О.Данилова, О.Войченко – Шкільний світ 2006, . – 120с.
3. Мастерская CSS / Энди Бадд, Камерон Молл, Саймон Коллизон – Вильямс 2007, . – 272с.
4. Спецификация CSS 3 / W3C – 1999.

.

3 Мова JavaScript

Мова JavaScript забезпечує програмування клієнтських сценаріїв. З її допомогою можна створювати різноманітні варіанти меню та навігаційних панелей, відкривати додаткові вікна з заданими параметрами, реалізовувати інтерактивні тести для самоперевірки, і т.д. Цікавим є використання та програмування різних декоративних ефектів.

Будучи цілком повноцінною алгоритмічною мовою, JavaScript позбавлена можливостей роботи з файловою системою. Це цілком природне обмеження продиктовано міркуваннями безпеки клієнтського комп'ютера.

JavaScript є інтерпретованою мовою (для алгоритмічних мов подібного типу часто вживають ще й термін «мова сценаріїв»). Задані в текстовому форматі оператори JavaScript аналізуються браузером і або виконуються в міру інтерпретації, або у відповідь на настання певних подій. Сценарії JavaScript можуть бути збережені у зовнішніх файлах. Приєднуючи ці файли до веб-документів можна використовувати загальні бібліотеки, що описують глобальні змінні, об'єкти та функції

Строго кажучи, JavaScript не є справжньою об'єктно-орієнтованою мовою, однак об'єктний тип даних в ній визнач

ено. Більш того, основну силу даній мові надає об'єктна модель, що описує специфічну множину об'єктів, що відповідають елементам HTML-документа.

3.1 Основні конструкції JavaScript

3.1.1Зауваження до синтаксису

- JavaScript – мова яка залежить від регістру. Чутливі до регістру ідентифікатори змінних, імена функцій, міток, ключові слова. Усі ключові слова використовують нижній регістр.
- Ідентифікатори можуть містити символи ASCII, цифри, символ підкреслення `"_"` і символ долара `"$"`. Перший символ не повинен бути цифрою. Як ідентифікатори не можна використовувати ключові і зарезервовані слова.
- Оператори розділяються крапкою з комою, яку можна опустити, якщо оператор закінчується символом нового рядка.
- Коментарі:

```
// однорядковий коментар
/* ...
багаторядковий
коментар
*/
```

Коментарі зручно використовувати і не «за прямим призначенням», а для того, щоб візуально відокремити один від одного фрагменти коду. Наприклад, так:

```
// ----------------------------------------
// вийшов зручний горизонтальний розділювач
```

- Змінні не мають строгої типізації. Оголошення відбувається за допомогою оператора `var`, який можна опускати, за винятком оголошення локальних змінних в тілі функції. Можливо оголошення з одночасною ініціалізацією, наприклад:

```
var s = 123
```

3.1.2Типи даних

В JavaScript підтримує такі типи даних.

Числа

Числа представляються у форматі з плаваючою крапкою довжиною 64 розряду (в тому числі цілі). Наприклад,

```
123
3.14169256
1e-12
015 // вісімкове число починається з 0
0xFC // шістнадцяткове число починається з 0x
```

Булеві величини

Як і в інших мовах програмування, булеві величини визначаються ключовими словами `true` і `false`.

Рядкові величини

Рядкові величини беруться в одинарні або в подвійні лапки. Причому текст, що містить одинарні лапки може містити подвійні і навпаки. Всередині рядка можуть міститися спецсимволи перед ними стоїть символ `"\"`. Розглянемо їх значення.

Таблиця 3.1.1 Спецсимволи JavaScript

Код	Значення
\b	видалення останнього передуючого символу (backspace)
\f	подача сторінки (feed)
\n	символ нового рядка (new)
\r	повернення каретки (return)
\t	символ табуляції (tab)
\'	'
\"	"
\\	\
\ааа	символ у кодуванні Latin-1, заданий трьома вісімковими символами ааа
\хаа	символ у кодуванні Latin-1, заданий двома шістнадцятковими символами аа
\символ	для всіх символів, відмінних від перерахованих вище, трактується як вказаний символ

Для чого може знадобитися остання конструкція? Наприклад, старі браузери некоректно сприймали в рядках деякі літери кирилиці, видаючи при цьому повідомлення про помилки сценарію. І для того, щоб, скажімо, в рядку `s` містилося слово `"філологія"`, доводилося виконувати присвоювання `s = '\ філологія'`, після чого символ `"ф"` трактувався інтерпретатором правильно.

JavaScript має ряд функцій для роботи з рядками.

Масиви

Масив JavaScript являє собою об'єкт, доступ до значень якого здійснюється за допомогою індексів - цілих чисел, починаючи з нуля. Оголошується за допомогою конструктора `new`, що створює новий екземпляр об'єкта `Array`.

```
teachers = new Array (); // оголошення без
ініціалізації
teachers[0] = 'Pankratov'; // ініціалізація 1-
го елемента
teachers[1] = 'Zakharkina'; // ініціалізація 2-
го елемента
days = new Array('Monday', 'Tuesday',
'Wednesday'); // оголошення з ініціалізацією
```

JavaScript надає ряд функцій для роботи з масивами.

Об'єкти

В JavaScript визначений ряд внутрішніх об'єктів для роботи з рядками, числами, датами і т.д., але особливий інтерес для веб-програміста представляє об'єктна модель DOM (Document Object Model), що описує специфічну множину об'єктів, що відповідають елементам HTML-документа. З кожним об'єктом пов'язані властивості, методи (функції, визначені для об'єкта) і події.

При зверненні до властивостей і методів об'єктів використовується так звана крапкова нотація, тобто ім'я змінної об'єктного типу відділяється від імені властивості і методу крапкою.

Наведемо інтуїтивно зрозумілий приклад. Головним в ієрархії об'єктів DOM є вікно браузера - об'єкт `window`. З цим об'єктом пов'язаний ряд методів і велика кількість властивостей. Наприклад, в об'єкта `window` є метод `open()`, що відкриває нове вікно з вказаними параметрами. Викликати його можна таким чином:

```
window.open('qq.htm', 'new', 'width = 300,
height = 200, toolbar = 1')
// Відкриється вікно 300x200, в якому зі
стандартних
// інтерфейсних елементів буде тільки панель
інструментів.
```

Ще один інтуїтивно зрозумілий приклад. Об'єкт `screen` дає доступ до параметрів екрану монітора комп'ютера клієнта (зрозуміло, ці параметри можна лише дізнатися, але не змінити). Методи для цього об'єкта не визначені, але визначений ряд властивостей. Найбільш корисні - `width`, `height` (ширина і висота в пікселях), `availWidth` і `availHeight` (доступна ширина і висота в пікселях). Приклад звернення до властивостей:

```
w = screen.width;
h = screen.height;
```

null

Це ключове слово вказує на відсутність значення.

3.1.3Оператори

Основні оператори (англ. operator) у порядку зменшення пріоритету.

Таблиця 3.1.2 Оператори JavaScript

Оператор	Виконувана операція
`.`	доступ до властивості або методу об'єкта
`[]`	доступ до елемента масиву
`()`	виклик функції
`++`	приріст (інкремент)
`--`	зменшення (декремент)
`-`	унарний мінус
`!`	заперечення (логічне NOT)
`delete`	видалення властивостей об'єкта
`new`	створення екземпляра об'єкту
`typeOf`	повертає тип операнда
`void`	повертає невизначенне значення
`*, /, %`	арифметичні множення, ділення, взяття за модулем
`+, -`	арифметичні додавання і віднімання
`+`	зчеплення (конкатенація) строк
`<, <=, >, > =`	менше, менше або дорівнює, більше, більше або дорівнює
`= =, != `	перевірка рівності і нерівності
`= = =,!= =`	перевірка ідентичності (тобто рівності і нерівності без перетворення типу операндів)
`&&`	логічне AND (другий операнд обчислюється, якщо перший дорівнює true)
`||`	логічне OR (другий операнд обчислюється, якщо перший дорівнює false)
`=`	присвоювання
`+=, -=, *=`	присвоювання з операцією

3.1.4Інструкції

Прості і складені інструкції

Простими інструкціями (англ. statement) JavaScript є оператори присвоювання, виклики методів об'єктів, оператори інкремент і декремент і т.д. Як і в інших мовах програмування, в JavaScript можна об'єднувати послідовності інструкцій в блоки, отримуючи складову інструкцію. Для цього послідовність інструкцій береться у фігурні дужки. Надалі під інструкцією розуміється проста або складена інструкція.

Інструкції можуть бути вкладені одна в іншу з будь-яким ступенем вкладеності.

Цикл while з передумовою

```
while (умова)
  інструкція, яка виконується в разі істинності
умови;
```

Під «умовою» тут розуміється вираз, що має значення булевого типу, наприклад

```
X > 112.345
```

або

```
(A != 0) && (b > 1)
```

Цикл while з післяумовою

```
do
  інструкція, яка виконується в разі істинності
умови
while (умова);
```

Цикл for

```
for (ініціалізація лічильника циклу;
   перевірка умови продовження циклу;
   зміна лічильника циклу)
  інструкція;
```

Наприклад, після виконання наступного циклу змінні `x` і `y` отримають значення, рівні відповідно до суми і добутку чисел від `1` до `10`.

```
x = 0;
y = 1;
for (i = 1; i <= 10; i ++) {
  x = x + i;
  y = y * i
};
```

Умовний перехід if

```
if (умова)
  інструкція, яка виконується в разі істинності
умови
else
  інструкція, яка виконується в разі хибності
умови;
```

Наприклад, в результаті виконання наступного фрагменту коду змінна `result` отримає рядкове значення `'відмінно'`.

```
ball = 5;
if (ball == 5)
  result = 'відмінно'
else
  result = 'до відмінної оцінки ви не дотягли';
```

Якщо треба послідовно розглянути виконання кількох умов, можна використовувати вкладені умовні інструкції. У ряді випадків при наявності кількох взаємовиключних умов зручна інструкція `switch`, описана нижче.

Інструкція switch

Якщо варіантів значення умови багато, і конструкція `if` виходить занадто громіздкою, зручна інструкція `switch` наступного формату:

```
switch(вираз)
{
  case значення1 : інструкція; break;
  case значення2 : інструкція; break;
  ...
  case значенняN : інструкція; break;
  default: інструкція по замовчуванню;
}
```

Оператор `break` припиняє виконання інструкції `switch` у випадку знайденої відповідності; якщо його не поставити, будуть виконуватися всі наступні перевірки `case`. Варіант по замовчуванню (default) вказувати необов'язково.

Наприклад, наступний цикл перерветься на другій перевірці та встановить значення `monthName = 'добре'`:

```
ball = 2;
switch(ball)
{
  case 5 : result = 'відмінно'; break;
  case 4 : result = 'добре'; break;
  case 3 : result = 'задовільно'; break;
  case 2 : result = 'погано'; break;
};
```

Інструкції break та continue

Інструкція `break` призводить до виходу з циклу або інструкції `switch`. У разі вкладених циклів вихід відбувається з самого внутрішнього. Формат застосування:

```
break;
```

Інструкція `continue` перериває виконання поточної ітерації циклу і запускає нову. Формат застосування:

```
continue;
```

Інструкція var

Інструкція `var` служить для явного оголошення глобальних і локальних змінних. При оголошенні глобальної змінної з одночасною ініціалізацією цю інструкцію можна опустити. При оголошенні локальної змінної інструкція `var` обов'язкова.

Функції

Функція визначається за допомогою інструкції `function`. Оголошення починається ключовим словом `function`, за яким (не менше ніж через один пробіл) слідують.

- Ім'я функції.

- Необов'язковий список аргументів, вкладений в круглі дужки, що перераховуються через кому. Якщо у функції аргументів немає, круглі дужки все одно обов'язкові.
- Набір інструкцій, що становить тіло функції, вкладений у фігурні дужки. Фігурні дужки обов'язкові, навіть якщо тіло функції представлено однією інструкцією. Інструкції виконуються при виклику функції.

Отже, синтаксис оголошення функцій наступний:

```
function ім'я_функції([арг1, [арг2, [арг3,
...]]])
{
  інструкції_тіла_функції
};
```

Строго кажучи, визначення функції за допомогою інструкції `function` - не єдиний спосіб, хоча в переважній більшості випадків застосовується саме він.

Функції JavaScript можуть бути вкладеними, тобто одна функція може бути визначена в тілі іншого.

У тілі функції може бути присутнім інструкція `return` у форматі

```
return значення;
```

Виконання інструкції `return` припиняє виконання функції і повертає вказане значення.

Наведемо максимально прості приклади функцій, перша з яких не повертає значення, а друга - повертає. Ці функції, як видно, викликаються в різних контекстах.

```
function showMessage(msg)
// виводить вікно попередження з заданим
повідомленням
{
  alert(msg);
};
// ------------------------
function square(a)
// повертає квадрат заданого числа
{
  return(a * a);
};
// --- Ці функції можуть бути викликані так:
```

```
showMessage ('Hello world!');
// буде виведено маленьке віконце з повідомленням
x = 5;
y = square(x);
// змінна y отримає значення 25
```

Функції можуть бути задані рекурсивно, тобто в тілі функції вона може викликати саму себе. Наступна функція являє собою традиційний приклад обчислення факторіалу.

```
function factorial(n)
// обчислює і повертає факторіал свого аргументу
{
  if (n =< 1)
    return 1;
  return x * factorial(x);
};
```

Області видимості змінних

Область видимості змінної - це блок програми, в якому змінна визначена. Глобальна змінна, оголошена за межами будь-якої функції, визначена для всього сценарію. При оголошенні глобальної змінної з одночасною ініціалізацією інструкцію `var` можна опустити.

Змінна, оголошена за допомогою інструкції `var` в тілі функції, є локальною, тобто визначена тільки в тілі функції. Її значення доступне лише при виконанні функції.

Можливий збіг ідентифікаторів глобальних і локальних змінних - синтаксично допустима, але вкрай незручна при відлагодці ситуація. Наведемо приклад.

```
// так можуть бути оголошені глобальні змінні:
// явне оголошення без ініціалізації
var x;
// неявне оголошення з ініціалізацією
a = 123;
// явне оголошення з ініціалізацією
var q = 'hello';
// -------------------------------------
function test ()
{
  var a = 555;
  // це локальна змінна, яка не має ніякого
  //відношення до оголошеної вище глобальної a
```

```
  q = q + 'world';
  // змінилося значення глобальної змінної q
  alert('всередині функції a =' + a);
  // виведе віконце з повідомленням:
  // 'усередині функції a = 555');
}
// ---------------------------------
// перевіримо значення глобальних змінних
// до і після виклику функції:
alert('до виклику функції a =' + a);
  // виведе віконце з повідомленням:
  // 'до виклику функції a = 123 '
test(); // виклик функції
alert('після виклику функції a =' + a);
  // виведе віконце з повідомленням:
  // 'після виклику функції a = 123'
alert('після виклику функції q =' + q);
  // виведе віконце з повідомленням:
  // 'після виклику функції q = Hello world'
```

Незважаючи на примітивність, цей приклад повною мірою демонструє поняття області видимості змінних.

3.2 Вбудовування коду JavaScript в документ HTML

Код JavaScript може бути вбудований в документ HTML наступними основними способами.

Включення фрагментів сценарію всередині елемента script

```
<script type="text/JavaScript">
... інструкції ...
</script>
```

Елементи `script` можуть міститися як у розділі `head`, так і в `body`. При інтерпретації документа вони виконуються послідовно. У `head`, як правило, оголошуються і ініціалізуються глобальні змінні і розміщується опис функцій. У `body` фрагменти сценарію зазвичай реалізують вставку динамічно форматованого вмісту, наприклад, виведення випадкового зображення або дати останньої зміни документа.

Включення файлів зі сценаріями JavaScript

Включаються зовнішні файли, що містять код JavaScript, визначаються в розділі `head` за допомогою того ж елемента `script`, шлях до файлу задається значенням атрибута `src`.

```
<script type = "text/JavaScript"
  src = "myFunc.js">
</script>
```

Включення файлів актуально, якщо одні й ті ж функції, об'єкти, глобальні змінні використовуються не в одному, а в ряді документів сайту. У такому випадку ці файли (зазвичай мають розширення. Js) розміщують у розділі `head` відповідних документів. Це спрощує підтримку сценаріїв і дозволяє прискорити завантаження за рахунок кешування браузером файлу з кодом JavaScript.

Вказання обробника події

Обробники подій, пов'язаних з елементами документа HTML, вказуються як атрибути цих елементів. Список цих атрибутів наведений у розділі "Події JavaScript". Значення цих атрибутів може являти собою будь-яку кількість операторів JavaScript, проте зазвичай

обробники визначаються як функції, описані в елементі `script`. Нижче наведено два умовні приклади.

```
<span onClick="this.style.color='red'">
  Click me!
</span>
```

У відповідь на клацання миші текст "Click me!" стане червоним

```
<img src = "jazz.jpg" id = "jazz"
     onMouseOver = "changeImg('rock.jpg')"
     onMouseOver = "changeImg('jazz.jpg')">
```

При наїзді/з‘їзді курсору миші зображення змінюється; це реалізовано за допомогою деякої функції `changeImg()`.

URL типу JavaScript

У гіперпосиланні можна вказати значення атрибуту `href` як псеводопротокол javascript:, після якого йде список інструкцій. У цьому випадку при виборі гіперпосилання браузер виконує код JavaScript. Наприклад вибір гіперпосилання

```
<a href = "javascript:
      window.open('next.htm', 'newWin',
      'width = 300, height = 200'); void(0)">
  відкриємо нове вікно
</a>
```

приведе до відкриття нового вікна 300x200px і завантаження в нього документа `next.htm`.

При використанні URL типу JavaScript слід пам'ятати, що якщо остання інструкція повертає якесь значення, його рядковий еквівалент буде виведено у поточний документ, замінивши його вміст. Спробуйте, наприклад, у наведеному вище коді прибрати інструкцію `void(0)`. Нове вікно, звісно, відкриється, але оскільки метод `window.open()` повертає значення об'єктного типу, у вихідному документі з'явиться рядок `[object]` або `[object Window]` (в залежності від браузера). Щоб уникнути цього, застосовують інструкцію `void(0)`, що вказує на те що повертається невизначене значення.

Певним недоліком URL типу JavaScript є відображення в статусному не адреси цільового документа, а послідовності інструкцій JavaScript, незрозуміле для більшості користувачів.

3.3 Події JavaScript

В JavaScript визначений ряд подій, пов'язаних з елементами документа. Обробники дають можливість організувати реакцію на виникнення подій зі сценарію. При цьому відповідний обробник вказується як атрибут елемента HTML-документа; значенням цього атрибуту є вираз JavaScript. Наприклад,

- Наїзд і з‘їзд курсору миші на елемент супроводжуються зміною кольору фону.

```
<span
   onmouseOver = "this.style.bgColor =
'#CCCCCC'"
   onmouseOut = "this.style.bgColor =
'#EEEEEE'">
 Наведіть курсор миші
</span>
```

- При спробі користувача закрити вікно і вивантажити документ виводиться повідомлення

```
<body onUnLoad="alert('вікно закривається!'">
```

- Якщо клацнути мишею по зображенню виконується якась функція `showPict()`

```
<img src="cat.jpg" onClick="showPict()">
```

Добра половина обробників підтримуються практично всіма HTML-елементами. Деякі події можна імітувати за допомогою відповідних методів. Нижче наводиться список подій згідно специфікації HTML 4.0 і деякі події. Трактування браузерів може відрізнятися від стандарту і в плані застосування обробника до тих чи інших елементів

Таблиця 3.3.1 Події JavaScript

Обробник події	HTML елементи що його підтримують	Опис	Метод імітації
onAbort	img	Переривання завантаження	

		зображення	
onBlur	a, area, button, input, label, select, textarea	Втрата поточним елементом фокуса, тобто перехід до іншого елементу. Виникає при клацанні кнопкою миші поза елементом або натисканні клавіші табуляції	blur()
onChange	input, select, textarea	Зміна значень елементів форми. Виникає після втрати елементом фокуса, тобто після події blur	
onClick	майже всі	Одинарний клацання (натиснута і відпущена кнопка миші)	
onDblClick	майже всі	Подвійне клацання	
onError	img, window	Виникнення помилки виконання сценарію	
onFocus	a, area, button, input, label, select, texta	Отримання елементом фокусу (клацання мишею на елементі або чергове натискання клавіші табуляції)	focus()

	rea		
onKeyDown	майже всі	Натиснута клавіша на клавіатурі	
onKeyPress	майже всі	Натиснута і відпущена клавіша на клавіатурі	
onKeyUp	майже всі	Відпущена клавіша на клавіатурі	
onLoad	body, frameset	Закінчено завантаження документа	
onMouseDown	майже всі	Натиснуто кнопку миші в межах поточного елемента	
onMouseMove	майже всі	Переміщення курсору миші в межах поточного елемента	
onMouseOut	майже всі	Курсор миші виведений за межі поточного елемента	
onMouseOver	майже всі	Курсор миші наведений на поточний елемент	
onMouseUp	майже всі	Відпущена кнопка миші в межах поточного елемента	
onMove	window	Переміщення вікна	
onReset	form	Скидання даних форми (клацання по кнопці <input type="reset">)	reset()
onResiz	windo	Зміна розмірів	

e	w	вікна	
onSelect	input, textarea	Виділення тексту в поточному елементі	select()
onSubmit	form	Надсилання даних форми (клацнути по кнопці <input type="submit">)	submit()
onUnload	body, frameset	Спроба закрити вікно frameset браузера і вивантаження документа	

Уважно переглянувши таблицю обробників подій, зауважимо, що специфічні події, що застосовуються лише до окремих елементів, в основному відносяться до елементів форм. Дійсно, саме елементи форм становлять особливий клас HTML-елементів, так як їх функціональність прямо пов'язана з роботою сценаріїв (клієнтських чи серверних). Не дивно, що для роботи з ними доведеться обробляти особливі події.

3.4 Об'єктна модель документа

Об'єктна модель документа не є частиною мови JavaScript. Строго кажучи, DOM (Document Object Model) – це інтерфейс прикладного програмування для представлення документа (наприклад, документа HTML, а також інших) і забезпечення доступу до її елементів та інтерактивної зміни їх властивостей. Більш того, DOM надає механізми для зміни самої структури документа (додавання та видалення елементів, зміна їх вмісту). Але це окремий стандарт, в даний час розвивається під егідою W3C.

Проте всі версії мови JavaScript в тій чи іншій мірі підтримують об'єктну модель документа. Ранні версії JavaScript підтримували тільки модель DOM0, яка надає базові можливості. Інтерпретатори JavaScript у сучасних браузерах нехай не повною мірою, але орієнтовані на стандарт W3C DOM2. Принаймні, MS Internet Explorer і Mozilla цілком адекватно підтримують DOM2 (але, на жаль, в дещо різній інтерпретації).

3.4.1Об'єктна модель документа DOM0

Базовий рівень функціональності документа забезпечується об'єктами, що підтримуються навіть найстарішими браузерами (природно, і сучасними теж). Ця ієрархія об'єктів представляє об'єктну модель документів рівня 0 (Document Object Model level0 - DOM0).

У клієнтської частини JavaScript основним об'єктом є window, який посилається на поточне вікно браузера. Інші об'єкти, ієрархія яких тут представлена, є властивостями кореневого об'єкта window. Майже всі ці об'єкти мають багато корисних властивостей, з ними пов'язані властивості та методи, використання яких дозволяє створювати сценарії, що забезпечують необхідну функціональність. Коротко, практично не називаючи конкретних властивостей і методів, у цьому розділі розглянемо лише ряд можливостей, які отримує розробник при використанні об'єктів вищого рівня ієрархії.

- Об'єкт Screen дозволяє дізнатися (але, зрозуміло, не змінити) роздільну здатність клієнтського екрану і глибину кольору. Визначивши роздільну здатність екрану, можна передбачити різні варіанти компонування сторінки, встановлювати розміри і положення нових вікон, що відкриваються зі сценарію.

Методи для цього об'єкта не визначені, але визначений ряд властивостей. Найбільш корисні:

`width` – ширина екрана в пікселях;

`height` – висота екрана в пікселях;

`availWidth` – доступна ширина екрана в пікселях;

`availHeight` – доступна висота екрану в пікселях.

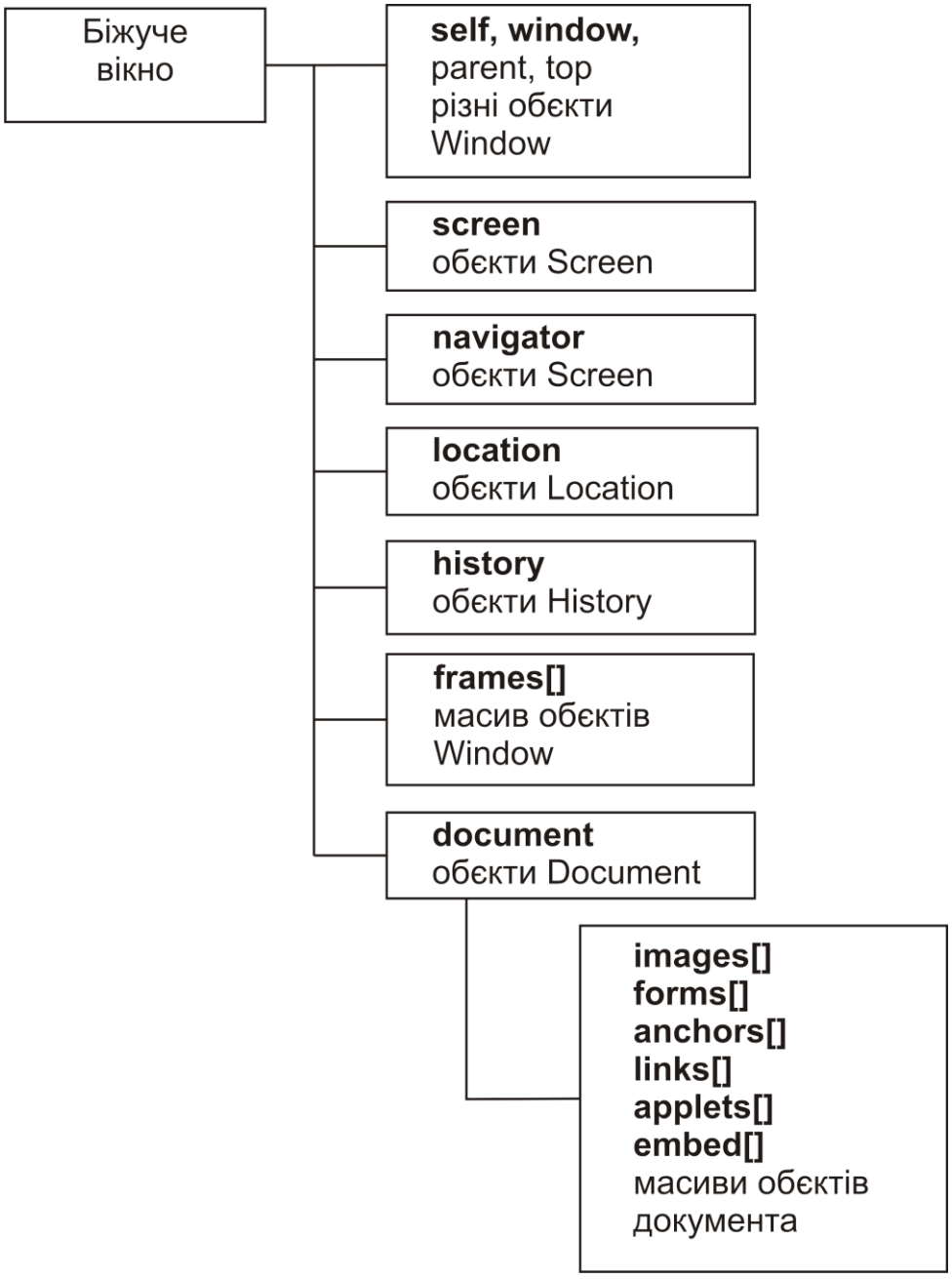

Рис 3.4.1 Обєктна модель DOM0

- Об'єкт `Navigator` дає інформацію про версію браузера. У принципі, це можна використовувати при створенні «браузеронезалежного» сценарію. Однак, найчастіше більш зручний інший підхід, описаний в розділі «Визначення можливостей клієнтського JavaScript».

- Об'єкт `Location` дає доступ до URL документа, що відображається у вікні браузера. Дозволяє визначити повний URL, а також його частини: протокол, доменне ім'я і т.д. На відміну від двох попередніх об'єктів, його властивості доступні не тільки для читання, але і для зміни. Тобто, в залежності від виконання умов, визначених в сценарії, ми можемо завантажити потрібний документ як в поточне вікно або його фрейм, так і в будь-яке з вікон, відкритих з сценарію. Цей об'єкт має і два методи:

`reload()` перезавантажує зазначений як аргумент документ;

`replace()` завантажує вказаний документ, який заміщає поточний в списку історії перегляду.

- Об'єкт `History` має єдину властивість `length` (кількість переглянутих в даному сеансі документів), і три методи, що дозволяють переміщатися по історії перегляду:

`back()` - на один крок назад в історії перегляду;

`forward()` - на один крок вперед з історії перегляду;

`go(n)` - на `n` кроків з історії перегляду (якщо `n>0`, то вперед, якщо `n<0`, то назад).

- Об'єкт `Document`, його властивості та методи надають найбільш багаті можливості для розробника. Наведена тут схема ієрархії об'єктів включає тільки основні властивості цього об'єкта, визначені в базовій об'єктної моделі документа (`Document Object Model Level 0 - DOM0`). На підтримку цих властивостей можна сміливо розраховувати при використанні будь-якого відносно сучасного браузера. Основні властивості і методи об'єкту document обговорюються в окремому розділі.
- Масив `frames[]` дає доступ до документів, завантажених у фрейми.

Треба відзначити, що різні браузери, підтримуючи розглянуту ієрархію, пропонують і додаткові властивості майже для кожного об'єкта.

3.4.2Об'єктна модель документа DOM2

Прийнята в даний час об'єктна модель документа `DOM2` (`Document Object Model level2 - DOM2`) визначена організацією World Wide Web Consorcium (W3C). Вона не розвиває (хоча, природно, підтримує) об'єкти моделі `DOM0`, але пропонує

певну концепцію представлення елементів HTML-документа. Акцент моделі DOM2 - ієрархічна структура документа, в якій кожен елемент є об'єктом (з точки зору алгоритмічного мови) зі своїми властивостями, методами і подіями.

Модель DOM2 відображає реальну ієрархічну структуру документа. Дійсно, в документі HTML є розділи head і body, в кожному з яких своя ієрархія елементів. У розділі head особливої ієрархії немає, там послідовно вказується набір елементів, що безпосередньо не відображаються у вікні браузера. Вони можуть впливати на загальне відображення і функціональність документа (визначення зовнішніх таблиць CSS, JavaScript коди), на ідентифікацію документа (meta-елементи), задавати HTTP-заголовки. Зате розділ body має явну ієрархічну структуру. Розглянемо, наприклад, дуже простий опис HTML-документа:

```
<html>
  <head>
    <title> Приклад </title>
  </head>
<body>
  <h1>Приклад простого документа </h1>
  <p>
    Початок абзацу <strong> виділений текст
</strong> продовження абзацу
  <img src="test.jpg">
  </p>
</body>
</html>
```

Цей документ може бути представлений деревом, що складається з наступних вузлів:

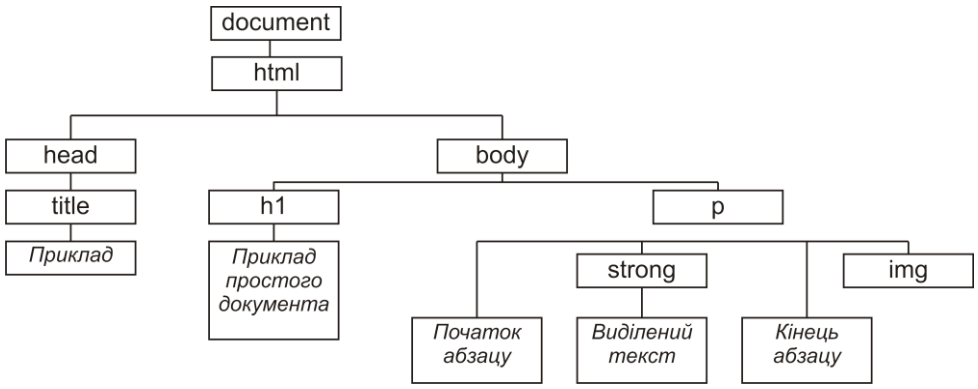

Рис 3.4.2 Приклад дерева документа в моделі DOM2

Приблизно таку ієрархічну модель документа і пропонує стандарт DOM2. Окремого розгляду цього інтерфейсу прикладного програмування в даному розділі немає. Проте відзначимо, що саме звернення до ряду властивостей і методів, представлених в DOM API, дозволяє сценаріями JavaScript забезпечувати необхідну функціональність в типових актуальних завданнях (тим більше, в нетривіальних).

3.4.3Визначення можливостей клієнтського JavaScript

Очевидно, що браузери різних версій і різних виробників відрізняються ступенем підтримки базової версії JavaScript і можливостей DOM. Властивість window.navigator.userAgent дозволяє визначити параметри браузера і організувати адекватне налаштування сценарію.

Однак, у більшості випадків нас цікавить конкретне запитання: чи реалізована в даному браузері потрібна нам можливість. Наведу два приклади: з недавнього минулого актуальні сьогодні.

Приклад з недавнього минулого

Зараз зміна зображень при наведенні курсору миші здається тривіальним декоративним прийомом. Але не так вже давно такий фокус можна було реалізувати лише в браузері Netscape Navigator 3 +. В MS IE 3 ця можливість була відсутня, гірше того - сценарій видавав помилку і повідомляв про це в спливаючому модальному вікні. Причина в тому, що інтерпретатор JavaScript Netscape Navigator 3 + підтримував об'єкт images стандарту DOM0, а MS IE 3 - не підтримував. Мабуть, найбільш елегантний метод з'ясування, чи підтримується об'єкт images, може бути реалізовано в такий спосіб:

```
if (document.images)
{
  опис інструкцій, що забезпечують зміну
зображень
}
else
{
  альтернативний варіант
};
```

Чому працює такий сценарій і не видає повідомлення про помилку? Справа в тому, що звернення до якогось об'єкту як до умови інструкції `if` повертає логічне значення: `true`, якщо об'єкт визначений, і `false` в іншому випадку. Таким чином, спрацьовує потрібний блок умовної інструкції.

Приклад, актуальний сьогодні

Найбільш універсальний і зручний спосіб звернення до елементів документа зі сценарію - використання їх унікальних ідентифікаторів `id`. Для ідентифікації елементів застосовується метод `getElementById()`, не визначений у ряді застарілих браузерів. Не будемо зараз розглядати питання про те, що ж саме зробити, якщо цей метод не визначений. Все залежить від розв'язуваної задачі. Але підхід до визначення, які саме з варіантів ідентифікації вибрати, той же, що і в попередньому прикладі.

```
if (document.getElementById)
{
  опис інструкцій, що використовують метод
  getElementById ()
}
else
{
  альтернативний варіант
};
```

Відзначимо також, що саме факт підтримки тих чи інших властивостей і методів, пов'язаних з моделями `DOM0` і `DOM2`, може служити зручним методом визначення типу браузера і його версії. Подивіться на наступний приклад, в якому глобальним змінним isMozilla, isIE5 і isOpera присвоюються булеві значення, які потім зручно буде використовувати в сценарії в контексті умовних інструкцій.

```
var isMozilla = (document.getElementById &&
          !document.all);
var isIE5 = (document.getElementById &&
          document.all);
var w = navigator.userAgent.toLowerCase();
var opera = (w.indexOf('opera')! = -1);
```

3.4.4Ідентифікація елементів документа

Моделі документа DOM0 і DOM2 надають свої можливості ідентифікації елементів документа. Для чого потрібна ця ідентифікація? Наприклад, для того, щоб при реалізації певної події (`click`, `mouseOver`, `mouseOut`, ...)

- вказаному елементу присвоїти інші властивості стилю: колір, фон, видимість і т.д. (скажімо, при реалізації меню);
- організувати зміну зображень (те, що називається rollover image);
- обробити дані форм на стороні клієнта;
- і т.д.

Коротше кажучи, для того, щоб змінити якісь властивості елемента HTML-документа або застосувати відповідні методи, треба мати механізм, що дозволяє ідентифікувати необхідний елемент. Можливості моделі DOM0 бідніші, але почнемо з них. Зрозуміло, засоби ідентифікації DOM0 поширюються і на модель DOM2, що має додаткові можливості.

Ідентифікація елементів документа в моделі DOM0

Посилання на поточний елемент

У всіх моделях документа при виклику обробника для поточного елемента використовується ключове слово `this`. Наприклад:

```
<span onMouseOver="this.style.color='red'"
      onMouseOut="this.style.color='blue'"
      onClick="this.style.color='springgreen'">
  тест
</span>
```

Тут задані обробники трьох подій для поточного елемента.

Посилання на об'єкти моделі DOM0, вкладені в об'єкт document

Всі ці об'єкти являють собою масиви: масив гіперпосилань `links[]`, масив зображень `images[]`, масив форм `forms[]` і т.д. Індексація масивів починається з 0, тому, наприклад, до і-того зображення документа можна звернутися `images[i-1]`. Очевидно, що така можливість актуальна лише у випадках, коли є сенс перебрати всі елементи цієї групи. Наприклад, перебрати всі елементи форми з метою з'ясувати, які з них є `checkbox`-ами і чи включені вони.

Посилання на ім'я (name) HTML-елемента

Лише кілька HTML-елементів згідно специфікації мають атрибут `name`. Це зображення `img`, а також форми та їх елементи. До цих елементів можна звертатися, використовуючи значення атрибута `name`, причому принаймні в двох варіантах. Нижче наведено приклад звернення до зображення. Нехай зображення в HTML описано так:

```
<img src="view.jpg" name="pict_view">
```

Тоді в сценарії JavaScript це зображення ідентифікується як

```
document.images['pict_view']
```

або

```
document.images.pict_view
```

Перший варіант явно щось нагадує тим, кому знайоме поняття асоціативного масиву. І правильно: всі об'єкти JavaScript по суті є асоціативними масивами, ключі яких збігаються з іменами властивостей. Це і пояснює можливість застосування двох варіантів, наведених вище.

Ідентифікація елементів документа в моделі DOM2

Крім вищезгаданих варіантів (явно обмежених), модель `DOM2` пропонує ще три варіанти ідентифікації елементів документа, основним з яких є наступний:

Ідентифікація по ID

Якщо у HTML-елемента вказаний атрибут `id`, то до нього можна звертатися, використовуючи значення цього атрибута. При цьому застосовується метод `getElementById()`. Зверніть увагу на те, що це - метод об'єкта `document`. Нижче наведено приклад.

```
<span id="q"> тест </span>
<span onClick =
"document.getElementById('q').style.color =
'red'">
клацніть мишкою, і попередній текст стане
червоним
</span>
```

Клацання мишею по елементу, в якому заданий обробник `onClick`, призведе до зміни кольору елементу з `id = "q"`.

Звернення до всіх елементів зазначеного типу

Метод `getElementsByTagName()` об'єкта `document` повертає масив елементів з вказаним ім'ям тега. Наприклад, `getElementsByTagName('h2')` поверне масив всіх заголовків 2-го рівня.

3.4.5Властивості HTML-елементів у DOM2

Інтерфейс DOM2 визначає ряд загальних властивостей елементів, а також набори властивостей, специфічних для елемента документа конкретного типу. Не заглядаючи в довідник, ми можемо бути, принаймні, впевнені в тому, що кожен HTML-елемент має властивості, що відповідають його атрибутами, визначеним у стандарті HTML. При цьому імена атрибутів HTML транслюються в імена властивостей наступним чином:

- Назва атрибуту HTML, що представляє одне слово, транслюється в ім'я властивості без змін. Всі букви в імені властивості маленькі. Виняток: атрибут `class` транслюється в властивість `className`.
- Назва атрибуту HTML, що представляє кілька слів, транслюється в ім'я властивості, у якому всі слова йдуть злитно (без дефісів та інших роздільників). Перші букви другому і наступних властивостей є великими. Наприклад, `maxLength`, `borderColor` і т.д.

Таким чином, всі елементи мають властивості `id`, `style`, `title`, `lang`, `dir` і `className`.

На додаток до цих загальних властивостей, елемент кожного типу має специфічні. Наприклад, для елемента `img` це властивості `src`, `width` і `height`.

Доступ до властивостей елементів документа дозволяє бажаним чином організувати їх інтерактивну зміну.

3.5 Вікна та фрейми. Об'єкт Window

Робота з вікнами (поточними та їх відкривання зі сценарію) і фреймами в JavaScript організовується за однією схемою. І це природно, адже і в повне вікно, і у фрейм (зокрема, вбудований) завантажується окремий документ HTML.

До об'єктів поточного вікна звертаються, використовуючи синоніми `window` або `self`, причому в цьому випадку вказівку вікна можна навіть опускати. Наведемо приклад звернення до властивості `bgcolor` (фоновий колір) об'єкта `Window`. Ця властивість наводиться виключно для прикладу. Використання його в реальних сценаріях абсолютно неактуально, зате зміст зрозумілий, що дозволить зосередитися на форматі звернення до вікна. Для поточного вікна наступні три звернення рівносильні:

- `window.bgcolor`
- `self.bgcolor`
- `bgcolor`

З фреймами, очевидно, ситуація трохи складніша: їх декілька, значить до кожного потрібно особливе звернення. Впевнений, що всі вже здогадалися: до фреймів звертаються за їх іменами (тобто, за заданими в HTML-описі значень атрибута `name`). Якщо визначений набір вкладених фреймів, то можуть бути використані звернення до батьківського фрейму як до об'єкта `parent`. Об'єкт `top` посилається на верхній рівень набору фреймів - поточне вікно.

3.5.1Властивості об'єкта Window

Для об'єкта `Window` визначений ряд властивостей, найбільш корисні та вживані з яких самі є об'єктами. Звернувшись до схеми, наведеної в розділі «Об'єктна модель документа DOM0», можна побачити вищі рівні ієрархії властивостей об'єктного типу. Зауважимо, що об'єкт `Document`, будучи властивістю об'єкта `Window`, сам має властивості об'єктного типу. У свою чергу, ряд його властивостей також представляють собою об'єкти і т.д.

Є в об'єкта `Window` і скалярні властивості, найбільш корисні з яких наведемо:

- `closed` - має значення true, якщо відкрите сценарієм вікно було закрито, і false у противному випадку.
- `status` - текст рядка статусу.

- `defaultStatus` - текст рядка статусу за замовчуванням.
- `name` - рядок, що містить ім'я вікна.
- `opener` - посилання на об'єкт `window`, зі сценарію якого було відкрито поточне вікно.
- `parent` - посилання на об'єкт `window`, що містить поточне вікно або фрейм.

3.5.2Методи об'єкта Window

Об'єкт `Window` має ряд методів, основні з яких:

- `open()` - відкриває нове вікно (див. в окремому розділі).
- `close()` - закриває вікно.
- `resizeTo(w, h)` - змінює розміри вікна до ширини `w` і висоти `h`.
- `resizeTo(x, y)` - змінює розміри вікна на `x` пікселів по ширині і `y` пікселів по висоті.
- `moveTo(x0, y0)` - переміщує вікно таким чином, щоб його координати лівого верхнього кута мали значення `x0` і `y0`.
- `moveBy(x, y)` - переміщує вікно на `x` пікселів по горизонталі і `y` пікселів по вертикалі.
- `focus()` - передає вікна фокус; при цьому вікно переміщується на передній план поверх інших вікон додатків.

Метод open()

Метод `window.open()` викликається з такими аргументами: `window.open(адреса файлу, ім'я вікна, параметри)`

де

- адреса файлу - адреса файлу, що спочатку завантажується в нове вікно;
- ім'я вікна - умовне ім'я, зміст якого буде пояснений нижче.
- параметри - набір елементів нового вікна браузера - рядок, в якому через кому в будь-якому порядку перераховуються пари `параметр = значення`. Якщо якийсь параметр не вказано, буде застосовано значення за замовчуванням.

параметр	значення	опис
`width`	розмір в пікселях	ширина нового вікна

height	розмір в пікселях	висота нового вікна
left	розмір в пікселях	абсциса лівого верхнього кута нового вікна
top	розмір в пікселях	ордината лівого верхнього кута нового вікна
toolbar	1/0/yes/no	вивід панелі інструментів
location	1/0/yes/no	вивід адресного рядка
directories	1/0/yes/no	вивід панелі посилань
menubar	1/0/yes/no	вивід рядка меню
scrollbars	1/0/yes/no	вивід смуг прокручування
resizable	1/0/yes/no	можливість зміни розмірів вікна
status	1/0/yes/no	вивід рядка статусу
fullscreen	1/0/yes/no	вивід на повний екран

Наприклад, при виконанні інструкції

```
window.open('test.htm', 'new',
            'width = 300, height = 200, toolbar
= 1')
```

відкриється вікно 300x200, в якому зі стандартних інтерфейсних елементів буде тільки панель інструментів. Оскільки не вказані координати верхнього лівого кута, розташування вікна буде залежати від замовчувань браузера.

Сенс 2-го аргументу (умовне ім'я вікна) методу `open()` має потребу в поясненні. При послідовному застосуванні методу `open()` з однаковим значенням 2-го аргументу нові вікна не відкриваються, а черговий цільової документ завантажується в раніше відкрите вікно з тим же ім'ям. Якщо ж указується умовне ім'я вікна, що раніше не вказувалось, відкривається нове вікно з вказаними параметрами.

Якщо виклик методу `open()` завантажує новий документ у вже відкрите вікно, це вікно не стає активним. Тобто, якщо воно закрито вікнами інших програм, то не «спливе» поверх інших. Отже, користувач і не помітить, що в це вікно завантажився новий документ. Для того, щоб активізувати вікно з знову завантаженим документом, слід, наприклад, застосувати метод `focus()`, описаний у наступному розділі.

Інші методи

У попередньому розділі ми застосували виклик методу `open()` в процедурному форматі:

```
window.open(список аргументів)
```

З точки зору користувача при цьому відкривається нове вікно з вказаними параметрами, і в нього завантажується цільовий документ. Для програміста ж важливо, що виклик цього методу створює новий об'єкт `Window`. Однак, поки що незрозуміло, яким чином можна реалізувати доступ до цього нового об'єкту зі сценарію.

Проблема вирішується таким чином. Метод `open()` повертає значення об'єктного типу. Таким чином, для того, щоб мати можливість подальшої роботи з вікном, відкритим сценарієм, слід зберегти значення що повертається методом `window.open()` в деякій змінній. Звернення до цієї змінної реалізується доступ до методів і властивостей нового вікна. Саме до неї можна застосовувати перераховані нище методи:

- `close()`
- `resizeTo(w, h)`
- `resizeTo(x, y)`
- `moveTo(x0, y0)`
- `moveBy(x, y)`
- `focus()`

Отже, присвоємо значення що повертається методом `window.open()` деякій змінній `newWin`.

```
newWin = window.open ('test.htm', 'new', 'width
= 300, height = 200);
```

Тепер це вікно можна закрити, перемістити, змінити його розміри зі сценарію, використовуючи відповідні методи. Наведемо приклади звернення до змінної `newWin` у ряді ситуацій.

Приклади

Експеримент. Закриття вікна.

Нове вікно, відкрите зі сценарію, можна закрити, застосувавши до об'єкта `newWin` метод `close()`. Перед виконанням цього методу треба переконатися в тому, що вікно відкрите, наприклад так:

```
if (!newWin)
```

```
{
  alert('Вікно newWin ще не відкрилося')
}
else if (newWin.closed)
{
  alert('Вікно newWin вже закрито')
}
else
{
  newWin.close();
}
```

Експеримент. Зміна параметрів вікна.

Звертаючись до змінної `newWin`, можна отримувати та змінювати властивості нового вікна. При цьому треба переконатися в тому, що вікно відкрите. Наприклад, змінимо текст рядка статусу і колір фону документа за допомогою операторів:

```
if (!newWin)
{
  alert('Вікно newWin ще не відкрилося')
}
else if (newWin.closed)
{
  alert('Вікно newWin вже закрито')
}
else
{
  newWin.status = 'Властивості вікна змінені';
  newWin.document.bgColor = '#CCCCCC';
};
```

Корисна функція. Відкриття нового вікна в центрі екрану.

Необхідність відкрити вікно заданих розмірів - дуже поширена ситуація. Наприклад, треба показати ілюстрацію у збільшеному масштабі. Актуальний приклад - фотоархів. Одне з поширених (і цілком технологічних) рішень полягає у висновку заздалегідь заготовлених мініатюр, клацанням по яких в окремому вікні показується повномасштабне зображення. При цьому, зрозуміло, можливі різні варіанти в рамках дизайнерської концепції.

- Всі зображення однакового розміру (точніше, при попередній обробці фотоматеріалу приведені до однакового розміру). У цьому випадку мініатюри також однакового розміру.
- Зображення різного розміру і різних пропорцій. Але мініатюри представляють собою вирізані і масштабовані фрагменти однакового розміру.
- Зображення різного розміру і різних пропорцій. Мініатюри також мають різні пропорції, але масштабовані таким чином, щоб вписатися в квадрат заданих розмірів. До речі, саме такий варіант реалізований у більшості програм перегляду графічних об'єктів. Так організований і вивід мініатюр в MS Windows «провіднику» в режимі «ескізи сторінок».

Не розглядаючи тут питання підготовки мініатюр і завдання параметрів їх відображення за допомогою CSS, наведемо лише один з варіантів функції, що відкриває вікно заданих розмірів по центру екрана монітора та завантажування в це вікно документ із зазначеної адреси. При цьому припустимо, що в новому вікні будуть відкриватися зображення різного розміру.

Плануючи послідовні відкриття зображень, погодимося, що нові вікна не повинні множитися. Значить, всі вони (див. розділ «Метод open()») повинні мати одне умовне ім'я, що задається як 2-гий аргумент методу `open()`. Як зазначалося вище, в такому випадку необхідно після завантаження нового вмісту активізувати вікно, щоб воно перемістилося на передній план поверх інших вікон додатків. Тут можна застосувати принаймні два підходи:

- Надіслати фокус раніше відкритого вікна і змінити його розміри і положення, щоб за нових розмірів вікно знову розташувалося по центру екрана.
- Закрити вікно з зображенням, якщо воно вже було раніше відкрито, а потім відкрити нове вікно потрібного розміру.

Налаштуйте сценарій, порівняйте результати і зробіть відповідний висновок.

Отже, запропонований варіант фрагмента сценарію та оголошення функції.

```
// Оголошуємо глобальну змінну newWin без
ініціалізації.
// При виконанні функції openWin() їй буде
присвоєно значення
```

```
// об'єктового типу, що мають посилання на нове
вікно.
var newWin;
function openWin (addr, w, h)
// Відкриває вікно з умовною назвою 'new'
шириною w
// і висотою h і завантажує в нього документ з
адресою addr.
// Вікно розташовується по центру вільній
частині екрана.
{
// Закриваємо вікно, якщо воно раніше було
відкрито:
if (newWin) newWin.close();
// Визначаємо координати лівого верхнього кута
вікна, використовуючи
// властивості об'єкта Screen:
var x = (screen.availWidth-w) * 0.5;
var y = (screen.availHeight-h) * 0.5;
// Відкриваємо нове вікно з вказаним
документом:
newWin = window.open(addr, 'new', 'width =' +
         w + ', height =' + h + ', left =' +
         x + ', top =' + y););
```

3.6 Об'єкт Document

Об'єкт `Document` особливо важливий при розробці сценаріїв. Наведена тут схема ієрархії об'єктів включає тільки основні властивості цього об'єкта, визначені в базовій об'єктній моделі документа (Document Object Model Level 0 - DOM0).

На підтримку цих властивостей можна сміливо розраховувати при використанні будь-якого щодо сучасного браузера. Більш пізні стандарти DOM1 і DOM2 представляють документ HTML у вигляді дерева і дають доступ до всіх елементів, незрівнянно розширюючи можливості розробника. Однак, у цьому розділі розглянемо лише базові можливості.

3.6.1Властивості об'єкта Document

Почнемо з скалярних властивостей, загальних для всіх браузерів. Більшість їх доступні як для читання, так і для зміни. Відзначимо, що значення властивостей, пов'язаних з кольором тексту, фону і гіперпосилань, можна змінювати динамічно лише в тих випадках, коли не задані відповідні описи CSS, які мають більший пріоритет. Всі значення властивостей - рядкові.

- `title` - текст заголовка документа (вміст елемента `title`);
- `fgColor` і `bgColor` - колір тексту і колір фону документа;
- `linkColor`, `vLinkColor`, `aLinkColor` - кольори невідвідування, відвіданих і активних гіперпосилань;
- `lastModified` (тільки для читання) - дата зміни документа;
- `referrer` (тільки для читання) - `URL` документа, посилання в якому призвела до завантаження поточного документа;
- `URL` (і застаріле `location`) - `URL` документа.

Більш цікаві й корисні для розробника властивості-об'єкти (властивості-масиви) об'єкта `Document`. Всі вони, природно, мають властивість `length` (кількість елементів у масиві). Більшість властивостей, специфічних для об'єктів, що зберігаються в цих масивах, асоціюються з атрибутами відповідних елементів HTML. Ось лише деякі з них, зрозумілі без пояснень будь-кому, хто знає HTML:

- об'єкт `Form` має властивості `name`, `action`, `method`;
- об'єкт `Anchor` має єдину властивість `name`;
- об'єкт `Link` має властивості `href`, `target`;

- об'єкт Image має властивості src, width, height.

До об'єктів документа, що зберігається в масивах images, forms і applets, а також до елементів форми можна звертатися і на ім'я, якщо в початковому тегу відповідного елементу HTML заданий атрибут name. Нехай, наприклад, в документі описано зображення

```
<img src="images/cat.jpg" id = "cat id"
name="cat_name">
```

і воно є n-м зображенням, яке зустрічається в документі. До цього елементу img можна звернутися по крайній мірі такими способами (див. розділ «Ідентифікація елементів документа»):

- Як до елемента масиву images, використовуючи його індекс (індексація починається з 0):

```
window.document.images[n-1]
```

- Як до елемента масиву images, використовуючи значення атрибута name як ключ масиву:

```
window.document.images['cat_name']
```

- Використовуючи значення атрибута name як властивість об'єкту:

```
window.document.cat_name
```

- Використовуючи значення атрибута id та властивість getElementById:

```
window.document.getElementById('cat_id')
```

3.6.2Методи об'єкта Document

- open() - відкриває новий документ; весь його вміст видаляється.
- close() - закриває раніше відкритий документ.
- write() - записує в документ задану як аргумент рядок.
- writeln() - аналогічний попередньому, але виведений в документ рядок закінчується символом переведення рядка.

Методи write() і writeln() досить корисні і часто використовуються для динамічного формування вмісту документа. Ось як, наприклад, можна включити в документ дату його останньої зміни:

```
<script type="text/JavaScript">
 window.document.write
```

```
(document.lastModified);
</script>
```

3.7 Об'єкт Images

3.7.1Властивості об'єкта Images

Для зміни зображень використовується об'єкт Images, вкладений в об'єкт Document. Кожному HTML-елемента img, зустрічається в документі, зіставляється елемент масиву Images з властивостями src, width, height і так далі (назви властивостей відповідають атрибутів елемента img). Таким чином, рішення задачі зміни зображення зводиться до зміни значення властивості src певного елемента масиву Images.

Отримати доступ до властивостей зображень можна кількома способами. Нехай, наприклад, в документі описано зображення

```
<img src = "images/cat.jpg"
   id = "cat_id" name = "cat_name">
```

і воно є n-м зображенням, яке трапляється в документі. Тоді наступні конструкції, що мають посилання на властивість src зображення рівносильні:

- Звертаємося до елемента масиву images, використовуючи його індекс (індексація починається з 0):

```
window.document.images [n-1].src
```

Цей варіант може бути зручний, коли треба працювати з декількома послідовно розташованими в документі зображеннями.

- Використовуємо значення атрибута id:

```
window.document.cat_id.src
```

- Використовуємо значення атрибута name:

```
window.document.cat_name.src
```

3.7.2Зміна зображень

Отже, зміна зображення зводиться до зміни значення властивості src певного елемента масиву images. Звертатися до цієї властивості можна будь-яким з описаних раніше способів. Наприклад, використовуємо атрибут name елемента img.

Сценарій може працювати у зв'язку з будь-яким з доступних в мові подій миші. Атрибути onMouseOver, onMouseOut, onClick і т.д. в HTML 4.0 і вище можуть застосовуватися з більшістю

елементів. Так що для сучасних браузерів при бажанні виконати скрипт, наприклад, при наведенні курсору миші на зображення, застосовується конструкція вигляду

```
<img src="cat.gif" name="pict"
onMouseOver="…інструкції…">
```

Найчастіше зустрічається два варіанти зміни зображень.

1. Зображення змінюється при наведенні на нього курсору миші.

Нехай у документі є зображення і при наведенні курсору миші хочеться його змінювати зображенням `butterfly1.jpg`.

Для зручності опишемо функцію `changeImg()`:

```
function changeImg(source)
{
  document.pict.src = source + '. jpg';
};
```

і застосуємо її при настанні подій `onMouseOver` і `onMouseOut`:

```
<img src="butterfly.jpg" name="pict" border=0
   onMouseOver="changeImg('butterfly')"
   onMouseOut="changeImg('butterfly1')">
```

2. Зображення змінятся при наведенні курсору миші на інші елементи документа.

Нехай у документі є оригінал:

```
<img src="flowers.jpg" name="pict1" border=0>
```

яке треба змінювати в залежності від того, над яким гіперпосиланням знаходиться курсор миші. Робимо абсолютно аналогічно попередньому варіанту.

Як і раніше, опишемо функцію `changeImg1()`

```
function changeImg1 (source)
{
  document.pict1.src = source + '.jpg';
};
```

В елементі `a` кожному гіперпосиланню, поряд з адресою цільового документа вказуємо атрибути `onMouseOver` і `onMouseOut`. Вийде опис приблизно такого вигляду:

```
<a href = "адрес_цільового_документа"
  onMouseOver = "changeImg1('butterfly1')"
  onMouseOut = "changeImg1('flowers')">
```

Попереднє підвантаження зображень

Для того, щоб ефект зміни зображень проявився відразу після завантаження сторінки, потрібно завантажити зображення, які будуть змінювати вихідні у відповідь на події миші, написавши на початку документа відповідний інструкції. Для попереднього прикладу з метеликами можна зробити це так:

```
but1 = new Image();
but1.src = 'butterfly1.jpg';
but2 = new Image();
but2.src = 'butterfly2.jpg';
but3 = new Image();
but3.src = 'butterfly31.jpg';
```

У цьому випадку зображення будуть завантажені заздалегідь і в потрібний момент будуть взяті з кеша.

Ще трохи про зміну зображень

У будь-якому сценарії, зокрема і при організації зміни зображень, треба намагатися зробити код раціональним. Ось дуже простий, але цілком ефектний приклад: при наведенні курсору миші на мініатюри змінюються збільшеним зображення. Файли зображень знаходяться в каталозі `Images` і називаються 1.jpg, 2.jpg і 3.jpg. Файли мініатюр називаються так само і знаходяться в підкаталозі Images/Thumbs.

Зрозуміло, що потрібно виконати попереднє підвантаження і написати функцію, що реалізує зміну зображень.

На початку сценарію визначимо змінну

```
imageDir = 'Images/'
```

і розглянемо три варіанти.

1 варіант

Функція:

```
function changeImg (imgAddr)
{
  document.bigImg.src = imgAddr
}
```

Виклик функції (наприклад, для 1-ої мініатюри):

```
<img src="Images/Thumbs/1.jpg" width="50"
height="50"
   onMouseOver="changeImg('Images/1.jpg')">
```

2 варіант

Користуючись тим, що імена файлів зображень представляють послідовність 1.jpg, 2.jpg ... і знаходяться в каталозі `imageDir`. Функція злегка змінилася:

```
function changeImg (imgAddr)
{
  document.bigImg.src = imageDir + imgAddr + '.
jpg'
}
```

Виклик функції вийшов коротше і зрозуміліше:

```
<img src="Images/Thumbs/1.jpg" width="50"
height="50"
  onMouseOver="changeImg(1)">
```

3 варіант

А якщо до того ж динамічно сформувати послідовності мініатюр (адже їх могло б бути багато), вийде зовсім компактно. Користуючись тим, що імена файлів зображень представляють послідовність 1.jpg, 2.jpg, виведемо зображення в циклі, використовуючи метод `write()` об'єкта `document`.

```
for (i = 1; i <= 3; i + +)
  document.write (
    '<img src = "Images/Thumbs/' +
    i + '. jpg "width =" 50 "height =" 50 "' +
    'onMouseOver = "changeImg('+ i +')">');
```

3.8 Форми і елементи форм

3.8.1Завдання форм та їх елементів в HTML

Елементи форм знайомі всім користувачам сучасної глобальної мережі. Це поля введення тексту і пароля, стандартні кнопки, радіокнопки-перемикачі, «прапорці», випадаючі списки і т.д. Найбільш очевидні варіанти застосування: введення ключових слів в пошукових системах, робота з електронною поштою через веб-інтерфейс, реєстрація на сайті, веб-анкети, online-тести. Відразу зауважимо, що використання форм передбачає інтерактивність і, отже, забезпечення функціональності неминуче пов'язано з програмуванням (клієнтським чи серверним). У більшості випадків форми використовуються для передачі даних на сервер, однак і на стороні клієнта є завдання, в яких зручно застосування форм (наприклад, календар або калькулятор).

Роль HTML полягає в описі необхідних елементів і компонування їх на сторінці. Всі атрибути форм та їх елементів, описаних за допомогою HTML, транслюються у відповідні властивості об'єктів DOM і використовуються при створенні сценаріїв JavaScript.

Форми

Описи керуючих елементів форм, покликаних забезпечувати необхідну функціональність, повинні бути розташовані в контейнері `form`, атрибути якого істотні на етапі розробки сценаріїв.

Елемент form

Основні атрибути елемента `form`:

- `name` - назва форми; використовується в клієнтських і серверних сценаріях для ідентифікації вкладених керуючих елементів
- `action` - адреса файлу серверного сценарію, який буде обробляти заповнену і передану форму
- `method` - метод передачі даних сервера (по замовчуванню get)
- `enctype` - тип вмісту, який використовується для відправки форми на сервер (по замовчуванню application/x-www-form-urlencoded)
- `accept-charset` - список кодувань символів введених даних, які будуть оброблятися сервером

- target - назва вікна або фрейму для завантаження документу, згенерованого сценарієм на підставі прийнятих з форми даних (по замовчуванню _self, тобто результат обробки форми завантажується в це ж вікно або фрейм)
- onSubmit - сценарій JavaScript, що виконується перед відправленням даних форми на сервер
- onReset - сценарій JavaScript, що виконується при скиданні значень елементів форми в значення по замовчуванню

Уважно прочитавши описи атрибутів, зауважимо, що більшість з них актуальні при обробці даних форми серверним сценарієм. Три атрибута мають значення по замовчуванню (що, взагалі кажучи, не характерно для HTML).

Всі перераховані вище атрибути формально не є обов'язковими, проте

- при роботі на стороні клієнта необхідно вказати ім'я форми name, щоб мати можливість звертатися до елементів форми з сценарію JavaScript;
- відправлення даних форми на сервер вимагає, як мінімум, вказівки атрибуту action, що визначає серверний сценарій обробки.

Зміст, принаймні, двох атрибутів необхідно пояснити більш детально.

Атрибут method

Атрибут method має два основних можливих значення: get (по замовчуванню) і post.

Сенс цих значень наступний. При передачі даних методом get дані форми відправляються на сервер в заголовку запиту, а при використанні методу post - в тілі запиту. Передача текстових даних може здійснюватися будь-яким з цих методів. А ось бінарні дані можуть бути відправлені тільки методом post. Це відбувається у випадку завантаження файлу не сервер (всім зрозумілий приклад - додаток до електронного листа). До речі, в цьому випадку необхідно вказати enctype = "multipart/form-data" (див. далі зауваження до атрибуту enctype).

Отже, за винятком випадку передачі бінарних даних (тільки методом post) однаковим чином можна застосовувати обидва можливих методу. На складність розробки серверного сценарію, що

приймає дані з форми, це жодним чином не впливає. Який же метод кращий? Пропоную два підходи для вирішення цього питання. По-перше, має сенс уважно розглянути наявні в мережі ресурси, переконатися, що в переважній більшості випадків застосовується метод `get`, і вважати це негласним емпіричним правилом. З іншого боку, можна замислитися, який варіант зручніше користувачеві інформаційного ресурсу.

При передачі методом `get` користувач має можливість бачити дані форми в адресному рядку. Символи, відмінні від стандартної латиниці кодуються. Наприклад, пробілу відповідає код %20. Кожен з нас постійно спостерігає таку ситуацію, працюючи з пошуковими системами. Задамо, наприклад в Google ключове словосполучення «мова HTML» і побачимо в адресному рядку:

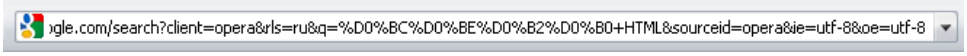

Рис 3.8.1 Адресний рядок при пошуці в Google

Слово «мова» закодовано послідовністю `%D0%BC%D0%BE%D0%B2%D0%B0`, а «HTML» передається так як є. Одержавши такий запит, відповідна серверна програма звернеться до бази даних, згенерує і відправить браузеру список результатів пошуку.

При передачі методом `post` дані форми відправляються на сервер. При цьому користувач не бачить в адресному рядку нічого «зайвого». Проте з'являються інші проблеми: спроба оновити сторінку викликає малозрозуміле повідомлення: «... оновлення сторінки неможливо без повторної відправки даних ...».

Мабуть, слід вибирати заданий по замовчуванню метод `get` у всіх випадках, крім передачі на сервер бінарних даних.

Атрибут enctype

Цей атрибут має два можливих значення:

- application/x-www-form-urlencoded (по замовчуванню)
- multipart/form-data

Перше значення використовується в абсолютній більшості випадків. Немає сенсу вказувати його явно - воно і так передбачається по замовчуванню. Другий же тип кодування (`multipart/form-data`) вказується в єдиному випадку: при завантаженні на сервер бінарного файлу. При цьому обов'язковим є завдання атрибуту `method = "post"`.

Елементи форм. Загальне представлення

Численні елементи форм можна умовно розбити на кілька груп. Всі елементи, природно, мають різну функціональність і різний зовнішній вигляд. Очевидно, що у кожного з обговорюваних елементів є певна кількість атрибутів (які якраз і забезпечують функціональність і, меншою мірою, параметри зовнішнього відображення). Але в цьому розділі ми відзначимо лише абсолютно необхідні атрибути, щоб хоча б побачити, як виглядають ті чи інші елементи форм. При роботі з формами, як на стороні клієнта, так і за допомогою серверних сценаріїв, очевидна необхідність вказівки інших атрибутів, але про них розмову в наступному розділі. Тут же просто познайомимося з різноманітними за зовнішнім виглядом і функціональнісцю елементами. У наведених нижче прикладах написи на елементах форми, які виглядають як кнопки, чисто умовні (зрозуміло, їх можна визначити за допомогою відповідних атрибутів).

Відразу відзначимо синтаксичний момент.

- Усі елементи `input` не мають вмісту і, відповідно, при їх заданні не вказується кінцевий тег.
- Інші елементи форм (`textarea`, `select`, `option`) є контейнерами і мають кінцевий тег. Для елемента `option` кінцевий тег може бути опущений.

Елементи input

Елементи `input`, залежно від значення атрибуту `type`, не тільки виглядають, але і функціонують принципово по-різному. Більшість з них можуть бути рівним чином використані як в клієнтських, так і в серверних сценаріїв. Представимо їх у вигляді таких груп.

Кнопки

```
<input type = "submit">
```

Відправляє дані з форми на сервер (передбачається, що серверна програма їх зможе прийняти і правильно обробити).

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

Очищає форму, відновлюючи значення по замовчуванню.

```
<input type = "button">
```

Виглядає як системна кнопка, але натискання на неї всього лише запускає клієнтський сценарій JavaScript (цього ж ефекту можна і

досягти використанням обробника onClick для будь-якого елемента HTML-документа).

```
<input type = "image" src = "адреса графічного
файла">
```

Як і кнопка типу `submit`, відправляє дані з форми на сервер, але виглядає нестандартним чином. Зовнішній вигляд цього елемента задається зображенням, адресу якого потрібно вказати як значення атрибуту `src`.

Текстові поля

```
<input type="text">
```

Однорядковий поле введення тексту.

```
<input type="password">
```

Поле вводу пароля.

Перемикачі

```
<input type="checkbox">
```

Перемикач (незалежний перемикач). Перемикачі можуть бути відзначені незалежно один від одного.

```
<input type="radio">
```

Радіокнопки (перемикач з залежною фіксацією). Виходячи з функціональності цього елемента, має сенс говорити не про одну, а про групу радіокнопок. Груп у формі може бути декілька, але в кожній групі в поточний момент тільки одна з радіокнопок може бути вибрана.

Поле вибору файла

```
<input type="file">
```

Дозволяє вибрати файл для завантаження на сервер.

Приховане текстове поле

```
<input type="hidden">
```

Дозволяє передавати дані, не введені користувачем, а отримані в результаті роботи сценарію.

Багаторядкові текстове поле textarea

```
<textarea>
...
```

```
текст за замовчуванням
...
</textarea>
```

Випадаючий список select і його елементи option

```
<select>
  <option> Москва </option>
  <option> Санкт-Петербург </option>
  <option> Новгород </option>
  <option> Южно-Сахалінськ </option>
</select>
```

Атрибути елементів форм

Як і переважна більшість елементів HTML, всі елементи форм мають спільні атрибути `id`, `style`, `class`, `title`.

Атрибут `type` елементу `input` обговорювалося в попередньому розділі.

Перерахуємо інші атрибути, починаючи з тих, що застосовуються з усіма елементами форм, і закінчуючи специфічними для окремих елементів.

З усіма елементами форми можна вказувати атрибути `name`, `disabled`, `tabindex`, `accesskey`.

- `name` - умовне ім'я елемента форми. Використовується клієнтськими і серверними сценаріями для звернення до елемента.
- `disabled` - завдання цього атрибуту робить елемент тимчасово недоступним. При необхідності це значення можна змінити зі сценарію.
- `tabindex` - порядок в послідовності переходу за елементами за допомогою клавіші табуляції.
- `accesskey` - клавіша доступу до елемента.

Примітка

Атрибут `name` служить для ідентифікації елементів форм при зверненні до них зі сценарію. Отже, значення цього атрибута повинно бути унікальним для кожного елемента. У відношенні елементу `input` типу `radio` є нюанс: унікальним повинно бути ім'я групи залежних радіокнопок. Власне, єдине ім'я групи і дозволяє браузеру трактувати радіокнопки як залежні.

Далі наведені додаткові атрибути, специфічні для окремих елементів форм.

Атрибути елементів input різних типів

- `value` - початкове значення. Атрибут обов'язковий для радіокнопок і перемикачів. Сенс його для різних елементів форм різний.

Для текстових елементів це текст по замовчуванню (якщо атрибут не вказаний, текстове поле пусте).

Для кнопок значення `value` задає напис на кнопці.

Задане поля для вибору файла значення `value` ігнорується сучасними браузерами з міркувань безпеки. Так що, фактично, можна вважати, що елемент `<input type="file">` не має атрибута `value`.

- `size` - для короткого текстового поля і поля введення пароля задає ширину в символах.
- `maxLength` - для короткого текстового поля і поля введення пароля задає максимальну кількість символів.
- `readonly` - вказівка цього атрибута для короткого текстового поля і поля введення пароля робить текст тимчасово недоступним для зміни. При необхідності це значення можна змінити зі сценарію.
- `checked` - вказівка цього атрибута для прапорця або радіокнопки робить відповідний елемент обраним по замовчуванню.

Атрибути елемента textarea

Крім перерахованих вище загальних атрибутів, елемент `textarea` має ще два, які задають його розміри:

- `rows` - ширина текстового поля в символах.
- `cols` - висота текстового поля в символах.

Атрибути елемента select

Елемент `select` служить контейнером для елементів списку `option`. Крім перерахованих вище загальних атрибутів, у цього елемента є наступні атрибути:

- `size` - кількість одночасно видимих елементів при нерозкритому стані списку (по замовчуванню 1).

- `multiple` - вказівка цього атрибуту дозволяє множинний вибір (по замовчуванню вибирається єдиний елемент).

Атрибути елемента option

Елемент `option` оформляє елемент списку. Природно наявність атрибуту, який вказує, якщо треба, якийсь елемент списку обраний по замовчуванню:

- `selected` - вказівка цього атрибуту робить відповідний елемент списку обраних по замовчуванню.

3.8.2Робота з формами в JavaScript

В JavaScript визначений ряд корисних властивостей і методів для роботи з формами та її елементами. Крім того, передбачені специфічні обробники подій, що забезпечують функціональність сценарію. Всі значення властивостей мають рядковий тип.

Як і в загальному випадку трансляції атрибутів елемента HTML в властивості об'єкта моделі DOM, HTML-атрибутів форм та їх елементів зіставлені відповідні властивості об'єктів JavaScript. Крім того, передбачені й інші властивості, корисні для застосування в сценаріях.

Ідентифікації елементів форм

Істотним моментом програмування форм є ідентифікація їх елементів. Розгляньте ще раз ієрархічну схему об'єктів розділу «Об'єктна модель документа DOM0». Масив форм `forms[]` вкладений в об'єкт `document`; елементи форм є вкладеними об'єктами самої форми; елементи списку є вкладеними об'єктами об'єкта `select`.

Згадаймо про те, що об'єкти JavaScript по суті являють собою асоціативні масиви з рядковими ключами - іменами властивостей.

Розглянемо приклад звернення до елементів форми зі сценарію. Зрозуміло, приклад умовний, у ньому представлені три характерні елемента, звернення до яких зі сценарію принципово по-різному. Точніше, за типом ідентифікації елементи форми можна розділити на три групи:

- радіокнопки;
- елементи списку;
- інші елементи форм.

Неважко здогадатися про причини саме такого поділу. На відміну від інших елементів форм, групи радіокнопок і елементи списку представляють собою масиви. А масив елементів списку до того ж є вкладеним по відношенню до об'єкта `select`.

Нехай в документі HTML задана наступна форма (це міг би бути фрагмент online-анкети). З групи «інших елементів форм» у ній представлено однорядкове текстове поле вводу. Вказані мінімально необхідні атрибути.

```
<form name="f">
  Ваше ім'я: <br>
  <input type="text" name="yourName">
  Ваша стать: <br>
  <input type="radio" name="sex" value="male">
чоловічий <br>
  <input type="radio" name="sex"
value="female"> жіночий <p>
  Місто: <br>
  <select name="town">
    <option value="msk"> Москва </option>
    <option value="spb"> Санкт-Петербург
</option>
    <option value="other"> інший </option>
  </select>
</form>
```

У всіх випадках при ідентифікації елементів форм використовуються імена (значення атрибута `name`) заданих елементів HTML. На ім'я ідентифікується і сама форма; для нашої форми це звернення

```
document.f
```

Ідентифікація більшості елементів форми

Групу «інших елементів форм» у нашій формі представляє текстове поле `yourName`. Варіанти ідентифікації:

```
document.f.yourName
document.f['yourName']
```

Ідентифікація груп радіокнопок

У нашій формі є група з двох радіокнопок з ім'ям `sex`. Група радіокнопок являє собою масив, в якому до окремої кнопки з

індексом i (індексація починається з нуля) можна звертатися в такий спосіб:

```
document.f.sex[i]
document.f['sex'][i]
```

Ідентифікація елементів списку

У нашій формі представлений список з ім'ям town. Його елементи являють собою масив options[]. До елементу цього масиву з індексом i (індексація починається з нуля) можна звертатися в такий спосіб:

```
document.f.options[i]
document.f['options'][i]
```

Універсальна ідентифікація

Є й альтернативна можливість ідентифікації. Всі елементи форми представлені масивом elements[], в якому містяться в порядку їх оголошення в документі HTML.

Властивості та методи форми

Для форм в JavaScript визначені наступні властивості, частина з яких представляє собою трансляцію атрибутів HTML елементу form.

властивість	атрибут HTML	Опис
name	name	Назва форми
action	action	Адреса файлу серверного сценарію, який буде обробляти заповнену і передану форму
method	method	Метод передачі даних серверу
encoding	enctype	Тип вмісту, який використовується для відправки форми на сервер
target	target	Ім'я вікна або фрейму для завантаження документа, згенерованого сценарієм на підставі прийнятих з форми даних
length	-	Кількість елементів форми
elements[]	-	Масив елементів форми

Для форми визначені два методи, емулює натиснення на кнопки типу submit і reset:

- submit()

- `reset()`

Властивості елементів форм

Для елементів форм в JavaScript визначений ряд властивостей, частина з яких представляє собою трансляцію атрибутів відповідних HTML елементів.

Загальні властивості

Для всіх типів елементів форм визначені наступні властивості.

властивість	**атрибут HTML**	**Опис**
`name`	`name`	Назва елемента форми
`value`	`value`	Рядок, визначальне значення, що наведено елементом і/або використовувана серверним сценарієм, обробляють отримані їх форми дані. Цю властивість задано для всіх елементів форм, крім `select`
`disabled`	`disabled`	Булеве значення, що визначає доступність елемента в поточному контексті
`form`	-	Назва форми

Елементи input

Для елементів input на додаток до властивостей, загальних для всіх елементів форм, визначені властивості:

властивість	**атрибут HTML**	**тип елемента input**	**Опис**
`type`	`type`	все	Тип елементу `input`
`defaultValue`	-	`text`, `password`	Текст, що спочатку відображаються в елементі форми
`checked`	-	`radio`, `checkbox`	Булеве значення визначає відзначений в даний момент елемент форми

defaultChecked	checked	radio, checkbox	Булеве значення, визначає чи відзначений елемент форми по замовчуванню

Елемент textarea

Крім властивостей, загальних для всіх елементів форм, для елемента `textarea` визначено властивість `defaultValue` - текст, спочатку відображаються в багаторядкової текстовому полі.

Елемент select

Крім властивостей, загальних для всіх елементів форм, для елемента `select` визначені властивості

властивість	Опис
length	Кількість елементів списку
options[]	Масив елементів списку
selectedIndex	Номер вибраного в поточний момент елемента списку

Елемент option

Крім властивостей, загальних для всіх елементів форм, для елемента `option` визначені властивості

властивість	Опис
text	Рядок, що задає текст елемента списку
selected	Булева значення, що визначає вибраний в даний момент елемент списку
defaultSelected	Булева значення, що визначає обраний елемент списку по замовчуванню

Обробники подій і методи елементів форм

Для різних елементів форм визначений ряд специфічних подій. З цими подіями пов'язані методи - імітатори подій.

Обробник подій	Елементи форм що підтримуються	Опис	Метод ініціалізації
onBlur	практично всі	Втрата поточним	blur()

		елементом фокуса, тобто перехід до іншого елементу. Виникає при клацанні кнопкою миші поза елементом або натисканні клавіші табуляції	
onChange	практично всі	Зміна значення елементів форми. Виникає після втрати елементом фокуса, тобто після події blur	
onClick	практично всі	Одинарне клацання (натиснута і відпущена кнопка миші)	
onDblClick	практично всі	Подвійне клацання	
onFocus	практично всі	Отримання елементом фокусу (клацнути мишею на елементі або чергове натискання клавіші табуляції)	focus()
onKeyDown	текстові елементи:	Натиснуто клавішу на	

	input типу text і password, textarea	клавіатурі	
onKeyPress	текстові елементи: input типу text і password, textarea	Натиснуто і відпущено клавішу на клавіатурі	
onKeyUp	текстові елементи: input типу text і password, textarea	Відпущено клавішу на клавіатурі	
onMouseDown	практично всі	Натиснуто кнопку миші в межах поточного елемента	
onMouseMove	практично всі	Переміщення курсора миші в межах поточного елемента	
onMouseOut	практично всі	Курсор миші виведений за межі поточного елемента	
onMouseOver	практично всі	Курсор миші наведений на поточний елемент	
onMouseUp	практично всі	Відпущено кнопку миші в межах поточного	

		елемента	
`onReset`	практично всі	Скидання даних форми (клацання по кнопці типу `reset`	`reset()`
`onSelect`	практично всі	Виділення тексту в поточному елементі	`select()`
`onSubmit`	практично всі	Надсилання даних форми (клацання по кнопці типу `submit`)	`submit()`

3.9 Контрольні запитання

- В чому різниця між мовами Java і JavaScript?
- До якого типу мов відноситься JavaScript - це алгоритмічна мова, мова скриптів або мова розмітки?
- Які існують способи оформлення програм на JavaScript?
- Яким чином єдиний файл *. Js визначає представлення текстів у кількох Web - сторінках?
- Чи припустимо застосування декількох файлів *. Js для одного сайту?
- В чому переваги і недоліки JavaScript?

3.10 Перелік навчально-методичної літератури.

1. Основи веб-дизайну / Пасічник О.Г., Пасічник О.В., Стеценко І.В. — К.: Видавнича група BHV, 2009. — 336 с.
2. Фленеган Д. JavaScript. Санкт-Петербург: O'Reilly, 2004, 955 с.
3. Мак-Федріс П. Використання JavaScript. М.-СПб-Київ, 2002, 895 с.
4. Спейнаур С., Екштейн Р. Справочник вебмастера. Санкт-Петербург: O'Reilly, 2001, 604 с.
5. Розділ JavaScript на citforum.ru http://www.citforum.ru/internet/javascript/ Технічна документація, статті, підручники, переклади.
6. Розділ JavaScript на eManual.ru http://www.emanual.ru/cgi-bin/show.pl?51 Технічна документація, статті, підручники, переклади.
7. JavaScript без кордонів http://javascripts.boom.ru/ Довідники, статті, колекції скриптів.
8. Javascripts.ru http://www.javascripts.ru Довідники, статті, колекції скриптів.
9. Бібліотека кодів Javascript http://www.txm.ru/js2/ Колекції скриптів.
10. JavaScript Documentation http://developer.netscape.com/docs/manuals/index.html?content=javascript.html
11. Документація по Client-Side JavaScript (англійською мовою). JavaScript 1.2, 1.3, 1.4, 1.5. Довідники. DOWNLOAD. Інструменти, статті, Plug-ins.
12. JavaScript-1.5 Reference http://www.mozilla.org/js/spidermonkey/apidoc/complete-frameset.html Документація по Client-Side JavaScript 1.5 (англійською мовою).
13. JavaScript Developer Central http://developer.netscape.com/tech/javascript/index.html Документація з JavaScript від Netscape (англійською мовою).
14. JavaScript Guide http://developer.netscape.com/docs/manuals/communicator/jsguide4/index.htm
15. Документація по ядру і розширень JavaScript (англійською мовою).

Предметний покажчик